

Optimised Grid-Partitioning for Block Structured Grids in Parallel Computation

Vom Fachbereich Mathematik
der Technischen Universität Darmstadt
zur Erlangung des Grades eines
Doktors der Naturwissenschaften
(Dr. rer. nat.)
genehmigte Dissertation

von
Dipl.-Math. Daniel Junglas
aus Freiburg im Breisgau

Referent:	Prof. Dr. A. Martin
Koreferent:	Prof. Dr. M. Schäfer
Tag der Einreichung:	27. April 2006
Tag der mündlichen Prüfung:	12. Dezember 2006

Darmstadt 2007
D17

Abstract

Simulation of turbulent flows in complex geometries is nowadays usually performed by application of grid-based algorithms on parallel computers. In this approach it is not only important to have a clever discretisation and an appropriate grid. One must also use (or develop) algorithms that are convergent and numerically stable. As a final ingredient to success the different work packages of the simulation must be distributed over the set of available processors. This distribution must be performed so as to optimally exploit the computational resources provided by processors, thereby minimising simulation time.

Our work will focus on this last optimisation problem and develop models as well solution algorithms for it. To this end we assume that a block structured as well as a suitable simulation algorithm are fixed and look for an optimal mapping of blocks to processors. We restrict ourselves to block structured grids for two reasons: one the one hand, this class of grids is most widely used in simulation of turbulent flows in complex geometries. On the other hand block structured grids can be partitioned into a relatively small number of blocks and the mapping problem can be restricted to the set of blocks. This last aspect allows application of integer programming methods to find optimal mappings. As opposed to standard approaches in the literature, we not only aim at balancing computational load over the processors, but also consider communication overhead induced by data dependencies between blocks mapped to different processors. The communication model we apply is an exact representation of the restrictions our hardware imposes on inter-processor communication.

Seeking to minimise simulation time leads to a highly complex combinatorial optimisation problem the solution of which is the aim of our work. To this end we formulate the problem as integer program. Since it is a new problem that has – to the best of our knowledge – not been investigated in the literature we do not stick with a single optimisation model. Instead, we propose different formulations and consider tradeoffs between them. Finally, we investigate the polyhedra defined by the various integer programs in order to implement the valid and facet-defining inequalities found in Branch-and-Cut algorithms. As we cannot expect to solve large problem instances by integer programming methods in an acceptable amount of time, we also develop several local-search heuristics that produce good solutions in a reasonable amount of time.

Computational results show that our models and solution algorithms – both of which are dedicated to a certain hardware model – are highly superior to generic approaches described in the literature. We were thus able to improve the usage of CPU resources during simulation of turbulent flows in complex geometries. Let us finally remark that our approach is not limited to this concrete application of finite-element or finite-volume procedures. Instead it can be applied in any situations where small or block structured grids arise and the hardware model is at least similar to the one assumed in our work.

Zusammenfassung

Die Simulation turbulenter Strömungen in komplexen Geometrien erfolgt heute meistens durch den Einsatz von gitterbasierten Lösungsalgorithmen auf Parallelrechnern. Dabei kommt es nicht alleine darauf an, eine möglichst geschickte Diskretisierung bzw. ein möglichst geeignetes Gitter auszuwählen oder numerisch stabile und konvergente Algorithmen zu entwickeln. Insbesondere müssen die einzelnen Arbeitspakete der Simulation so auf die gegebenen Prozessoren verteilt werden, dass die Rechnerressourcen optimal ausgenutzt werden, d.h. die Simulation in minimaler Zeit abläuft.

Die Arbeit konzentriert sich auf dieses letzte Optimierungsproblem und entwickelt dafür Modelle sowie Lösungsmethoden. Dazu nehmen wir an, dass für die Simulation bereits ein blockstrukturiertes Gitter gewählt wurde und suchen nach einer optimalen Verteilung der einzelnen Gitterelemente auf die vorhandenen Prozessoren. Die Beschränkung auf blockstrukturierte Gitter erfolgt, da dies erstens der in der Simulation turbulenter Strömungen in komplexen Geometrien am häufigsten eingesetzte Gittertyp ist und zweitens Gitter dieses Typs in eine überschaubare Menge von Blöcken zerlegt werden können, sodass zur Berechnung einer optimalen Verteilung des Gitters Methoden der ganzzahligen Programmierung verwendet werden können. Im Gegensatz zu den üblichen Ansätzen aus der Literatur achten wir bei Zuordnung von Arbeitspaketen an Prozessoren nicht nur auf eine gleichmäßige Verteilung der Rechenlast, sondern betrachten auch exakt den Zusatzaufwand, der sich für ein bestimmtes Hardwaremodell durch Datenaustausch zwischen verschiedenen Prozessoren während der Simulation ergibt.

Diese Fragestellung führt uns auf ein hochkomplexes kombinatorisches Optimierungsproblem, dessen Lösung das Ziel unserer Arbeit ist. Zu diesem Zweck formulieren wir das Problem als ganzzahliges lineares Programm. Da es sich um ein in der Praxis unseres Wissens bisher nicht untersuchtes Problem handelt, beschränken wir uns nicht auf eine Formulierung, sondern stellen verschiedene Formulierungen vor und wägen diese gegeneinander ab. Schließlich untersuchen wir die durch die ganzzahligen Programme definierten Polyeder, um die daraus gewonnenen in einem Branch-and-Cut Algorithmus zu implementieren. Da für große Probleminstanzen eine Lösung mit Methoden der ganzzahligen Programmierung nicht zu erwarten ist, entwickeln wir außerdem effiziente Heuristiken, die gute, aber nicht notwendig optimale Zuordnungen in akzeptabler Zeit liefern.

Unsere Rechenergebnisse am Schluß der Arbeit zeigen, dass sowohl unsere Modelle als auch die implementierten Verfahren, die speziell auf die vorliegende Hardware abgestimmt sind, den Methoden aus der Literatur weit überlegen sind. Wir konnten so einen Beitrag zur besseren Ausnutzung von Rechnerressourcen bei der Simulation turbulenter Strömungen in komplexen Geometrien leisten. Abschließend bleibt anzumerken, daß unsere Ergebnisse nicht auf diese konkrete Anwendung von Finite-Elemente- oder Finite-Volumen-Verfahren beschränkt sind. Vielmehr lassen sich unsere Algorithmen überall dort einsetzen, wo entweder kleine oder blockstrukturierte Gitter auf einer zumindest ähnlichen Hardware verwendet werden.

Acknowledgements

First and foremost I want to thank Prof. Dr. Alexander Martin. Without your support and your faith in me I would have never been able to write this thesis.

I would also like to thank the DFG (the German Research Society) which gave the financial support for Research Training Course 853 – Modelling, Simulation and Optimisation of Engineering Applications. This thesis is a consequence of the research work in this course. I am thankful to Dörte Sternel and Prof. Dr. Michael Schäfer for posing the problem handled in this thesis and providing me with the required data to perform computational experiments.

Furthermore I would like to thank my officemates in Darmstadt and my fellow members of research unit Discrete Optimisation at Darmstadt University of Technology, especially Ursula Röder for taking care of me as well as Susanne Moritz and Markus Möller for proofreading this thesis and many (off-topic) discussions.

I want to thank all my relatives from the Bornscheuer and Junglas families for their ongoing support. It is quite a pleasure to be part of these great families. Special thanks go to Sybille and Karl-Dieter Bornscheuer (not only) for calling me time and again, thereby asserting that I was keeping up the work; to Else Junglas for telling everyone that I was graduating, thus making it impossible to quit; to Ursula and Jürgen Faust for proofreading this thesis and inviting me to funny parties; and finally to my parents Martina and Mario Junglas for many ways of backing me up.

I also want to thank my daughter Anna-Lena and my girlfriend Katja for supporting me and believing in me. Thank you also for your endurance during the last months of this thesis.

Although I tried hard to list everybody who supported me during the last three years I have almost certainly missed some people. “Thank you” to all of you, and please take my apologies for not listing you here explicitly.

Contents

1	Introduction	13
2	Notation and Definitions	15
2.1	Discrete Optimisation	15
2.2	Graphs	16
2.2.1	Connectivity	19
2.2.2	Edge-Colouring	20
2.3	Polyhedra	28
2.4	Relaxations and Branch-and-Cut	32
3	Graph-Partitioning with Communication Overhead	37
3.1	A counter-example for the edge-cut metric	43
4	Related Work	47
4.1	Graph-Partitioning Problems	47
4.2	Communication Optimisation	49
4.3	Multiple Knapsack and General Assignment	50
5	Individual Models	53
5.1	Block-Mapping Models	53
5.1.1	Naive Block-Mapping	54
5.1.2	Representative Block-Mapping	64
5.1.3	Mapping with Slots	71
5.2	Edge-Colouring Models	80
5.2.1	Naive Edge-Colouring	80
5.2.2	Representative Edge-Colouring	87
5.2.3	Edge-Colouring with Matchings	93
6	Joined Models	99
6.1	Capacitated or Uncapacitated Processors?	100
6.2	Naive Model	102
6.2.1	LP Relaxation	106
6.2.2	Valid Inequalities	107

6.3	Representative Model	110
6.3.1	LP Relaxation	113
6.3.2	Valid Inequalities	115
6.4	Slot Model	118
6.4.1	LP Relaxation	122
6.4.2	Polyhedral Analysis	123
6.4.3	Valid Inequalities	124
6.5	(SLOT) on four Processors	126
6.6	(SLOT) on many Processors	128
6.6.1	Valid Inequalities	130
6.7	Symmetry in MIP Models	131
6.7.1	Variable Sorting	132
6.7.2	Variable Fixing	133
6.8	Comparison of Models	135
6.8.1	Symmetry	135
6.8.2	Model Size	136
6.8.3	LP Relaxations	137
6.8.4	Conclusion	137
6.9	Alternate Architectures	138
6.9.1	Non-symmetric Data Exchange	138
6.9.2	Heterogeneous Processors	139
6.9.3	Heterogeneous Network Connections	139
6.9.4	Non-Connected Processors	140
6.9.5	Routed Communication	141
7	Bounds	143
7.1	Bounds for the Number of Colours Required	144
7.1.1	Lower Bounds on $\Delta(M_P)$	145
7.1.2	Lower bounds on $ E_P $	147
7.2	Objective Function Bounds	147
7.2.1	A Distinct Processor	148
7.3	Bounding Processor Size	149
8	Connectivity in (SLOT) and its relatives	151
8.1	Edge-Connected Covers	152
8.1.1	Tree Cover Inequality	152
8.1.2	k -Connected Cover Inequality	152
8.1.3	k -Connected q -Cover Inequality	153
8.1.4	Sparser Variants of Cover Inequalities	154
8.1.5	Per-Processor Version of Connected Cover Inequalities	155
8.2	Separating Valid Inequalities	157
8.2.1	Separating Tree Covers	158

8.2.2	Separating Biconnected Covers	160
8.2.3	Separation of k -Connected Cover Inequalities	163
8.2.4	Handling k -Connected q -Cover Inequalities	164
8.3	Knapsack Cover Inequalities	164
9	Heuristics	167
9.1	Heuristics for Initial Assignment	168
9.1.1	Algorithms Aiming at Balanced Processor	169
9.1.2	Assignment Algorithms with Communication Support	172
9.2	Local Search Heuristics	176
9.3	Tabu Search	186
9.4	Edge-Colouring the Processor Multigraph	189
9.5	Algorithm for a Feasible Starting Solution	192
9.6	Rounding Heuristics	193
9.6.1	Probabilistic Rounding	194
9.6.2	Prioritised Rounding	195
9.7	Heuristic Constraints	196
9.7.1	Connected Processors	196
9.7.2	Easily Colourable Processor Multigraphs	197
10	Computational Results	201
10.1	Branching Strategies	201
10.1.1	SOS Branching	202
10.1.2	Branching on Virtual Variables	203
10.1.3	SOS Branching on Matchings	204
10.2	Test-Bed Instances	205
10.3	Preprocessing	207
10.4	Parameter Setup	208
10.5	Local Search Heuristics	208
10.5.1	Four Processors	209
10.5.2	More than four Processors	216
10.6	Branch-and-Cut Algorithms	218
10.6.1	Computation Time	218
10.6.2	Results	219
10.6.3	Negative Results	226
10.7	Algorithms from the Literature	228
11	Conclusions and Outlook	233
A	Further Computational Results	237
A.1	Initial Assignment on four Processors	237
A.2	Local Improvement on four Processors	240
A.3	Local Improvement on eight Processors	245

B Separation of Blossom-Inequalities	249
Bibliography	251

List of Figures

2.1	A star around u and a triangle e, f, g	18
2.2	A multigraph on 4 nodes and all its matchings.	20
2.3	Different formulations of the set-covering problem illustrated.	21
2.4	The Petersen graph, a graph with $\chi^*(G) = 3$ but $\chi'(G) = 4$	21
2.5	Equivalence classes of matchings in the example multigraph.	24
2.6	Bad example for Vizing's bound.	26
3.1	Structured and unstructured grids [164].	38
3.2	A grid-graph arises from a grid.	39
3.3	A processor multigraph arises from a mapping.	41
3.4	Different mappings yield different processor multigraphs	42
3.5	A two-dimensional block-structured grid interpreted as graph.	43
3.6	Mapping the example grid to four processors.	44
3.7	Mapping the example grid to four processors.	44
5.1	Mapping a grid graph in model (SLOTMAP).	72
6.1	Different configurations for edges of largest multiplicity	126
6.2	1-, 2- and 3-dimensional hypercubes.	140
6.3	Different communication paths in hypercubes.	141
8.1	A tree in which each edge is oriented away from u	156
8.2	Tree augmented to a 2-edge connected subgraph.	161
9.1	Three different classes of moves.	178
10.1	Test-Bed instances.	206

List of Tables

6.1	Number of pairwise disjoint maximum cardinality matchings in K_n	128
6.2	Variables and constraints in (NAIVE) and (SLOT*)	130
6.3	Size of model instances	137
10.1	Test-Bed instances.	205
10.2	Initial assignment algorithms on Grid-1 and 4 processors.	209
10.3	Initial assignment algorithms on Grid-2 and 4 processors.	210
10.4	Initial assignment algorithms on Grid-3 and 4 processors.	210
10.5	Initial assignment algorithms on Grid-4 and 4 processors.	210
10.6	Local improvement on Grid-1 and 4 processors.	212
10.7	Local improvement on Grid-2 and 4 processors.	213
10.8	Local improvement on Grid-3 and 4 processors.	214
10.9	Local improvement on Grid-4 and 4 processors.	215
10.10	Local improvement algorithms on Grid-1 and 8 processors.	217
10.11	Local improvement algorithms on Grid-2 and 8 processors.	217
10.12	Local improvement algorithms on Grid-4 and 8 processors.	218
10.13	Lower bound by LP relaxation in non-augmented models.	220
10.14	Branch-and-Cut performance on four processors.	221
10.15	Branch-and-Cut performance on four processors.	222
10.16	Branch-and-Cut performance on four processors.	222
10.17	Branch-and-Cut performance on eight processors	223
10.18	Results for (SLOT*) on four processors.	224
10.19	Results for (SLOT*) on eight processors.	225
10.20	Mapping grids to a two-dimensional hypercube.	226
10.21	Colours (communication rounds) in optimal mappings.	230
10.22	Maximum processor load in optimal mappings.	230
10.23	Milliseconds per iteration using optimal mappings.	231
A.1	Initial assignment algorithms on Grid-5 and 4 processors.	237
A.2	Initial assignment algorithms on Grid-6 and 4 processors.	238
A.3	Initial assignment algorithms on Grid-7 and 4 processors.	238
A.4	Initial assignment algorithms on Grid-8 and 4 processors.	239
A.5	Initial assignment algorithms on Grid-9 and 4 processors.	239

A.6	Local improvement on Grid-5 and 4 processors.	240
A.7	Local improvement on Grid-6 and 4 processors.	241
A.8	Local improvement on Grid-7 and 4 processors.	242
A.9	Local improvement on Grid-8 and 4 processors.	243
A.10	Local improvement on Grid-9 and 4 processors.	244
A.11	Local improvement algorithms on Grid-5 and 8 processors.	245
A.12	Local improvement algorithms on Grid-6 and 8 processors.	246
A.13	Local improvement algorithms on Grid-7 and 8 processors.	246
A.14	Local improvement algorithms on Grid-8 and 8 processors.	247
A.15	Local improvement algorithms on Grid-9 and 8 processors.	247

List of Algorithms

1	Basic Edge-Colouring Scheme	25
2	Cutting Plane Algorithm	32
3	Branch-and-Bound	34
4	Branch-and-Cut	35
5	Grid-Based Simulation	38
6	Separation of violated Tree Cover Inequalities	159
7	Augment a tree to biconnectivity	162
8	Iteratively filling up processors	169
9	Assign blocks to leastly loaded processor	170
10	Assign blocks in a circular fashion	171
11	Small Degree Assignment	172
12	Small Multicut Assignment	173
13	Connected Subgraphs	174
14	General Local Search Algorithm	176
15	Local Search Loop	178
16	Initialise Arrays	180
17	Update Arrays	181
18	Initialise <code>mul</code> and <code>mmul</code>	184
19	Update <code>mul</code> and <code>mmul</code>	185
20	Tabu Search	187
21	Incremental Edge-Colouring	189
22	Edge-Colouring with Matchings	191
23	Feasible Solution	192

List of Integer Programming Models

Naive block-mapping	55
Representative block-mapping	66
Block-Mapping with slots	72
Explicit edge-colouring	81
Representative edge-colouring	88
Edge-colouring with matchings	94
Naive formulation of (OGPC)	104
Representative formulation of (OGPC)	112
Slot-based formulation of (OGPC)	120
Slot-based formulation for four processors	127
Slot-based formulation for many processors	128
Minimal degree of biggest processor	146
Lower bound by skewed bipartitioning	149
Maximum number of elements per processor	150
Bipartite processor multigraph	197
Optimal solutions to state-of-the-art algorithms	228

Chapter 1

Introduction

“Would you tell me, please, which way I ought to go from here?”

“That depends a good deal on where you want to get to.”

— *Lewis Carroll, Alice in Wonderland*

The numerical solution and simulation of problems in computational fluid dynamics (CFD) by means of flow-solvers requires the application of parallel computers. The efficient use of these parallel computers with distributed memory demands that the computational load of the simulation is spread over the processors in a way that balances the computational effort well and also keeps the cost for inter-processor communication small.

As in many other problems in scientific computing [105, 144] the task of optimally distributing computational load over a set of processors can be modelled by means of a graph: A node (or vertex) in the graph represents a computation and an edge between two nodes indicates data dependency between these nodes. In order to efficiently solve the problem instance given by such a graph, this graph must be partitioned into subsets where each subset has approximately the same size and the cost of the resulting communication between nodes in different subsets is small. The number of subsets into which the graph is to be divided is bounded from above by the number of available processors.

Traditionally, this optimisation problem is solved as an instance of the *graph-partitioning problem* (see e. g., [105]): Distribute the graph such that each processor has equal load and the number of edges that have their endpoints on different processors is minimal. This model enforces a perfect balancing of the computational load and assumes that the communication requirement of a mapping is proportional to the number of edges between different processors (because communication effort between nodes on the same processor can be neglected). The set of edges with endpoints on different processors is commonly called *multi-cut* and the assumption that communication effort is proportional to the cardinality of this set is called the *edge-cut metric*.

Although widely used [93, 105], the traditional graph-partitioning model with edge-cut metric has two severe drawbacks:

- The model is not suitable for problems in which multiple objective functions or additional constraints must be considered [105, 102, 101, 103, 83].
- For most hardware architectures, the edge-cut metric is not appropriate to account for the communication effort induced by a graph-partitioning [80, 81, 83]. Other metrics exist that measure the communication overhead more appropriately [83].

In this work, we will concentrate on the second drawback of the traditional model: the edge-cut metric. We will propose another way to estimate the communication effort that stems from a given graph-partitioning. This new metric will correctly reflect the communication effort for a certain hardware architecture.

On the other hand, the ultimate goal is neither to obtain a partitioning that results in a perfect load-balancing nor to find a minimal communication schedule. Instead we want to find a load-balancing that allows a communication schedule such that the running time of the whole simulation process is minimised. To this end, we will assume that we can associate execution times with each node in the graph and can also estimate the communication time between two processors with a sufficient accuracy. The problem itself naturally decomposes into two subproblems: Assigning nodes to processors and finding an optimal communication schedule for a given assignment. The first one is a Multiple Knapsack-/General Assignment-type problem while the special hardware under consideration in this work renders the second one an edge-colouring problem in multigraphs. Unfortunately, it turns out that we cannot solve both problems sequentially but must optimise them simultaneously.

An outline of this thesis is as follows: In the next chapter we introduce several notions that are required to formalise description. In Chapter 3 we describe the optimisation problem to be solved in detail and in Chapter 4 we will review literature related to this problem. Chapters 5 and 6 are dedicated to integer programming formulations of our problem and subproblems. In Chapter 7 we present several bounds that can be used to enhance and strengthen the problem formulation as integer programming model. Chapter 8 contains further valid inequalities for our models and algorithms to separate them. Since not all problem instances can be solved exactly, we present in Chapter 9 several local-search heuristics to solve our problem. This chapter also describes several heuristic assumptions that can be made to simplify the problem and thus make it more tractable. In Chapter 10 we present computational results for the methods described in this thesis. The thesis ends with Chapter 11 where we summarise our work and point out several directions that seem promising for future research.

Chapter 2

Notation and Definitions

Never express yourself more clearly than you think.
— *N. Bohr*

So as to make this work more self-contained, we provide preliminary material on discrete optimisation, graphs, and polyhedra. Concepts and notation not listed or explained here can be found in pertinent textbooks about the respective topics [17, 29, 132, 152, 54]. We especially recommend [64] for the unexperienced reader to obtain an introduction into the realm of algorithmic analysis and \mathcal{NP} -completeness. Our notation closely follows [152].

2.1 Discrete Optimisation

The aim of Discrete Optimisation is to find an optimal element among a finite (but usually gigantic sized) set. One prominent example of discrete optimisation problems is to find the shortest among all possible routes between two cities.

Since discrete optimisation problems are special cases of the general optimisation problem we start by defining the latter:

Definition 1 (General Optimisation Problem). *Let S be a set and (T, \leq) a partially ordered set, i. e., one with $t \leq s$, $s \geq t$ or $s = t$ for all $s, t \in T$. If $f : S \rightarrow T$ is a map then the general optimisation problem is to either find $s^* \in S$ with $f(s^*) \geq f(s)$ for all $s \in S$ or $s^* \in S$ with $f(s^*) \leq f(s)$ for all $s \in S$. The first problem is called Maximisation Problem and the second one Minimisation Problem. We write*

$$\max_{s \in S} f(S) \quad \text{respectively} \quad \min_{s \in S} f(S) \tag{2.1}$$

and call f the objective function.

In this work we will usually have $T = \mathbb{N}$ or $T = \mathbb{R}$. Together with $T = \mathbb{Z}$ these are the typical examples for T . The set S is usually not enumerated but implicitly described as a set of points that satisfy certain properties. The devices used to describe the points in S can be used to further classify the set of optimisation problems. For example:

1. Let $g_i, i \in \{1, \dots, m\}$ and $h_j, j \in \{1, \dots, p\}$ be continuous (differentiable) functions from \mathbb{R}^n to \mathbb{R} . Then we call (2.1) with

$$S = \{x \in \mathbb{R}^n | g_i(x) \leq 0, h_j(x) = 0\}$$

Non-Linear Optimisation Problem.

2. If f and S are both convex then (2.1) is called *Convex Optimisation Problem*.
3. If f and S are both linear then (2.1) is called *Linear Optimisation Problem*.

A *Discrete Optimisation Problem* is a linear optimisation problem with $S \subseteq \mathbb{R}^n$ and the additional restriction that for a nonempty set $I \subset \{1, \dots, n\}$ the coefficients s_i are integral for all $i \in I$ and all $s \in S$. Linear and Discrete Optimisation Problems can both be described in a very compact fashion:

Definition 2 (Mixed Integer Linear Program). *Let $c \in \mathbb{R}^n$, $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$ and $p \in \{0, \dots, n\}$. Then*

$$\begin{aligned} \min \quad & c^T x \\ Ax \leq & b \\ x \in & \mathbb{Z}^p \times \mathbb{R}^{n-p} \end{aligned} \tag{2.2}$$

is called Mixed Integer Linear Program or Mixed Integer Program (MIP). If $p = n$ it is called Integer Program (IP) and if $p = 0$ it is called Linear Program (LP). The special case where $p = n$ and $x \in \{0, 1\}^n$ in problem (2.2) is called Binary Program (BIP).

By definition Mixed Integer and Binary Programs are discrete optimisation problems. Notice that it is no loss of generality if we require the first p coefficients x_1, \dots, x_p to be integral, for we may always permute coefficients appropriately.

2.2 Graphs

An *undirected graph* is a pair $G = (V, E)$ where V is a finite set and E is a family of unordered pairs from V . The elements of V are called *vertices* or *nodes* and the elements of E are called *edges*. We also write $V(G)$ and $E(G)$ to denote the sets of nodes and edges

in G . We usually define $n := |V|$ and $m := |E|$. An edge $\{u, v\} \in E$ is often abbreviated uv (or vu). For brevity we write $V - v$ instead of $V \setminus \{v\}$ and $E - e$ instead of $E \setminus \{e\}$. If we have $u = v$ for an edge $e = uv$ then e is called a *loop*. In this work we will consider only *loopless* graphs, that is from now on all graphs are silently assumed to have no loops. Two edges $uv, st \in E$ are *parallel* if $\{u, v\} = \{s, t\}$. The *multiplicity* of an edge $e \in E$ is the number of edges parallel to e (including e) and is abbreviated by $\mu(e)$, i. e.,

$$\mu(uv) := |\{st \in E : \{u, v\} = \{s, t\}\}|.$$

Two edges $uv, st \in E$ are *incident* if $\{u, v\} \cap \{s, t\} \neq \emptyset$ and an edge $uv \in E$ is also called *incident* to its both *endnodes* u and v . The set of edges incident at a node u is denoted by $\delta_G(u)$, i. e.,

$$\delta_G(u) := \{e \in E : e \cap \{u\} \neq \emptyset\}.$$

The number of edges incident to a node u is the *degree* of this node in G and denoted by $\deg_G(u)$ (we drop the index graph if the respective graph is understood from the context). For $U \subseteq V$ the set of all edges with exactly one endpoint in U is denoted by $\delta(U)$. Two nodes $u, v \in V$ are *adjacent* if $uv \in E$. The set $N(u) := \{v \in V : uv \in E\}$ contains all nodes adjacent to $u \in V$ and is called the *neighbourhood* of u (in G). For a subset $F \subseteq E$ of edges, $\delta(F)$ denotes the set of all edges that have exactly one endpoint in the set of nodes incident to F . For a node $u \in V$ we denote by $\bar{N}(u)$ the set of all non-neighbours of u , i. e., the set of all nodes, that are not adjacent to u . Similarly, $\bar{N}(e)$ for $e \in E$ is the set of all edges not incident to e .

The *maximum degree* and *maximum multiplicity* of a graph are defined as

$$\begin{aligned} \Delta(G) &:= \max_{v \in V} \deg_G(v) \text{ (maximum degree),} \\ \mu(G) &:= \max_{e \in E} \mu_G(e) \text{ (maximum multiplicity).} \end{aligned}$$

In this work we assume that nodes are always numbered from 0 to $n - 1$ and edges are always numbered from 0 to $m - 1$. The number of a node or edge is also referred to as the node's or edge's *label* or *index*. We will use both, the node itself and the label of the node, to refer to a node or to its label, thus using both notions as synonyms.

A graph G with $\mu(G) = 1$ contains no parallel edges and is called *simple*. Graphs with maximum multiplicity at least 1 are called *multigraphs*. In order to distinguish both kinds of classes we will sometimes write G_s for simple graphs and G_m for multigraphs. Notice that according to our definition a simple graph is also a multigraph. So all results for multigraphs are valid for simple graphs as well, but not vice versa.

The simple graph $G = (V, E)$ that contains all possible edges, i. e., $E = \{\{u, v\} : u \neq v\}$ is called the *complete graph* on $n = |V|$ nodes and abbreviated by K_n . Other (sub)graphs important for this thesis are stars and triangles (see Figure 2.1). A *star* $S = (V_S, E_S)$ is a graph in which all edges are incident to a distinct node $u \in V_S$, i. e., $e \cup \{u\} = \{u\}$ for all



Figure 2.1: A star around u and a triangle e, f, g .

$e \in E_S$. A *triangle* is a set of exactly three edges e, f, g that are pairwise incident but do not form a star.

If $G_m = (V, E_m)$ is a multigraph, then we define $\mathfrak{s}(G_m) := (V, E_s)$ with

$$E_s = \{\{u, v\} : u, v \in V, u \neq v, u \text{ and } v \text{ are adjacent in } G_m\}$$

and call $G_s = \mathfrak{s}(G_m)$ the *simple graph underlying* G_m . In a more illustrated explanation, $\mathfrak{s}(G_m)$ is the simple graph that arises when we replace in G_m every family of parallel edges by a simple edge.

A *node-weighting* of a graph is a function $w : V \rightarrow \mathbb{R}$ that assigns to each node $u \in V$ a *weight* $w(u) \in \mathbb{R}$. Likewise we define an *edge-weighting* as a function $w : E \rightarrow \mathbb{R}$ that assigns to each edge $e \in E$ a weight $w(e) \in \mathbb{R}$. A graph together with an node- or edge-weighting is called a *node- or edge-weighted graph*. If the type of weights is understood from the context, we simply say *weighted graph*. Many results for weighted graphs are only valid if all weights are non-negative. All weight functions in this thesis will have this property and we will not emphasise in theorems or the like that the weights are non-negative.

For a node set $U \subseteq V$ the term $E_G[U]$ denotes the set of all edges (in G) with both endpoints in U . Similarly, for $F \subseteq E$ the term $V_G[F]$ denotes the set of all endpoints of edges in F . A *subgraph* $H = (U, F)$ of $G = (V, E)$ is a graph such that $U \subseteq V$ and $F \subseteq E[U]$. For $U \subseteq V$ and $F \subseteq E$ the subgraphs $G[U] := (U, E[U])$ and $G[F] := (V[F], F)$ are called the *node-* and *edge-induced* subgraphs of G .

In a *directed graph* (or *digraph* for short) the edges in E are assumed to be *ordered* pairs and are usually written as $(u, v) \in E$. The edges of a directed graph are also called *arcs*. For a node $u \in V$ in a directed graph $G = (V, E)$ we define $\delta_G^+(u) := \{(v, w) \in E : v = u\}$ as the set of edges *leaving* node u and $\delta_G^-(u) := \{(v, w) \in E : w = u\}$ as the set of edges *entering* node u . We set $\delta_G(u) = \delta_G^+(u) \cup \delta_G^-(u)$ and $\deg_G(u) := |\delta_G^+(u) \cup \delta_G^-(u)|$. In a directed graph, two edges $(u, v), (s, t) \in E$ are only parallel if $u = v$ and $s = t$, i. e., the edges (u, v) and (v, u) are *not* parallel (they are *anti-parallel* instead). Most of this thesis deals with undirected graphs so we will assume that in the following a graph is undirected unless it is explicitly specified to be directed.

2.2.1 Connectivity

A *walk* in an undirected graph $G = (V, E)$ is a sequence $P = (v_0, e_1, v_1, \dots, e_k, v_k)$ where $k \geq 0$, v_0, v_1, \dots, v_k are vertices in V and $e_i \in E$ is an edge connecting v_{i-1} and v_i . If all vertices in a walk are distinct the walk is called a *path*. Notice that in a path also all edges are distinct. Two paths P and Q are called *edge-disjoint* if they share no edges and *node-disjoint* if they share no nodes. If two paths have the same start and end nodes but do not share other nodes, they are called *internally node-disjoint*. The *length* of a path or walk is the number of edges in it, or in edge-weighted graphs the sum of the edge-weights of its edges. A path $P = (v_0, e_1, v_1, \dots, e_k, v_k)$ together with the additional edge v_0v_k is called a *cycle* of length $k + 1$. In this work we will often consider paths and cycles simply as a set of edges and will not explicitly specify the nodes on the path or cycle.

A path $P = (v_0, e_1, v_1, \dots, e_k, v_k)$ is said to *connect* the two nodes v_0 , and v_k , and an undirected graph $G = (V, E)$ is *connected* if there is a path between any two nodes in G . A connected subgraph $H = (U, F)$ of $G = (V, E)$ with $U = V$ is called a *spanning* subgraph of G . We call a graph *k-edge-connected* (*k-node-connected*) if there are k edge-disjoint (internally node-disjoint) paths between any two nodes in G . In this work we will consider only edge-connectivity and will thus drop the prefix “edge-”. We also refer to 2-edge-connected graphs as biconnected graphs. A *k-edge-connected component* in a graph $G = (V, E)$ is an inclusion-wise maximal k -edge-connected subgraph $C = (V_C, E[V_C])$ of G .

A 1-edge-connected graph that does not contain a cycle is called a *tree*. In a tree $T = (V, E)$ each pair of nodes $u, v \in V$ is connected by a *unique* path in E . A tree with n nodes contains exactly $n - 1$ edges.

An interesting kind of paths are *shortest paths*. In an unweighted graph a shortest u - v -path is a u - v -path of minimum cardinality and in an edge-weighted graph a shortest u - v -path such that the sum of edge-weights on the path is minimal. In any unweighted graph a shortest path between two nodes can be found in linear time [16, 129] and in a graph with non-negative edge-weights a shortest path between two nodes can be found in time $\mathcal{O}(n^2)$ by Dijkstra’s method [40, 118].

An *edge-cut* in an undirected graph $G = (V, E)$ is a set $F \subseteq E$ of edges such that $G \setminus F$ becomes disconnected. The edges in F are commonly referred to as *cut-edges*. A minimum edge-cut (or *min-cut* for short) is an edge-cut of minimal cardinality and is denoted by $\delta(G)$. It is clear that in an k -connected graph a minimum edge-cut contains k edges. Again, in a weighted graph a min-cut is a cut for which the sum of edge-weights on the cut-edges is minimal. We can find a min-cut in G in time $\mathcal{O}(n^3)$ if G is either unweighted or if all edges have rational weight [106, 119, 159].

A *k-partitioning* of a graph is a surjective map $p : V \rightarrow \{1, \dots, k\}$, i. e., a map such that the pre-image $p^{-1}(i)$ is non-empty for all $i \in \{1, \dots, k\}$. For a given partitioning p , the

sets $p^{-1}(i)$ are called *partitions* and an edge $uv \in E$ is said to *cut* by the partitioning if $p(u) \neq p(v)$. The set of all edges cut by p is the *multicut* under p .

2.2.2 Edge-Colouring

An *edge-colouring* of a graph $G = (V, E)$ is a map $\mathbf{c} : E \rightarrow C$ which assigns to each edge $e \in E$ a colour $\mathbf{c}(e) \in C$ such that no two incident edges receive the same colour. In this work, we usually assume that $C \subseteq \mathbb{N}$ and number its elements sequentially starting from zero. The minimal cardinality of the colour set C for which such a mapping exists is called the *chromatic index* of the graph and denoted by $\chi'(G)$. In an edge-colouring the set of all edges that receive the same colour $c \in C$ is called the *colour class* of c .

Edge-Colouring as Set-Covering Problem

Given a finite set X and a collection $\mathcal{S} = \{S_1, \dots, S_k\} \subseteq 2^X$ of subsets of X , the *set-covering problem* asks for a selection $\mathcal{S}' \subseteq \mathcal{S}$ such that

$$\bigcup_{S \in \mathcal{S}'} S \supseteq X \tag{2.3}$$

and $|\mathcal{S}'|$ is minimal. A set \mathcal{S} that satisfies (2.3) is said to *cover* X , hence the name set-covering problem. In some cases there are also weights w_S associated with each $S \in \mathcal{S}$ and the aim is to find a set \mathcal{S}' of minimum weight that covers X .

The edge-colouring problem on graphs can easily be phrased as a set covering problem. To this end, recall that in an edge-colouring all edges of a common colour are pairwise non-incident. In graph theory a set of pairwise non-incident edges in a graph is called a *matching*. For a graph $G = (V, E)$ and an edge $e \in E$ we denote by $\mathcal{M}(G)$ the set of all matchings in G (see Figure 2.2) and by $\mathcal{M}(G, e)$ the set of all matchings in G that contain (also *cover*) e . Similarly we denote by $\mathcal{M}(G, u)$ the set of all matchings in G that *match* node $u \in V$, i. e., all matchings that contain an edge incident to u . The maximum cardinality of a matching

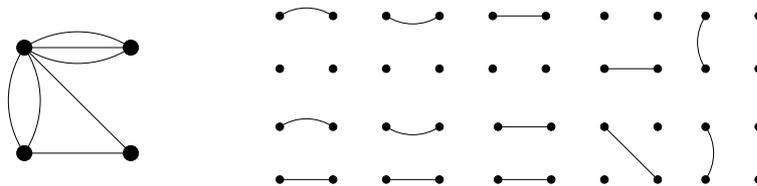


Figure 2.2: A multigraph on 4 nodes and all its matchings.

in G is called the *matching number* of G and is denoted by $\nu(G)$.

To define the edge-colouring problem for a graph $G = (V, E)$ as set-covering problem we set $X = E$ and $\mathcal{S} = \mathcal{M}(G)$. It is then clear that the colour classes of an edge-colouring yield a set-covering for E and vice versa. Also obvious is the fact that a minimal edge-colouring yields a minimal set-covering and vice versa. More formally, we have

$$\chi'(G) = \min \left\{ \sum_{M \in \mathcal{M}(G)} \lambda_M : \sum_{M \in \mathcal{M}} \lambda_M \chi^M = \mathbb{1}, \lambda_M \in \{0, 1\} \right\} \quad (2.4)$$

where χ^M denotes the incidence vector of M in \mathbb{R}^E . The interpretation here is that for a matching $M \in \mathcal{M}(G)$ we have $\lambda_M = 1$ if and only if M is used in the covering of E (Figure 2.3(a)). Relaxing the integrality condition on λ in (2.4) leads to the *fractional*

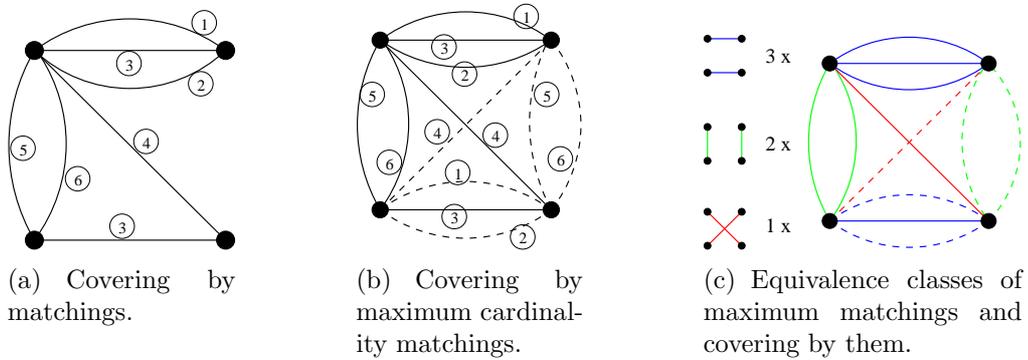


Figure 2.3: Different formulations of the set-covering problem illustrated.

chromatic index of a graph $G = (V, E)$:

$$\chi^{*}(G) = \min \left\{ \sum_{M \in \mathcal{M}(G)} \lambda_M : \sum_{M \in \mathcal{M}} \lambda_M \chi^M = \mathbb{1}, \lambda_M \in [0, 1] \right\} \quad (2.5)$$

It is clear that $\chi^{*}(G) \leq \chi'(G)$ and the Petersen graph (Figure 2.4) is an example for a graph in which fractional chromatic index and chromatic index do not coincide. An equivalent

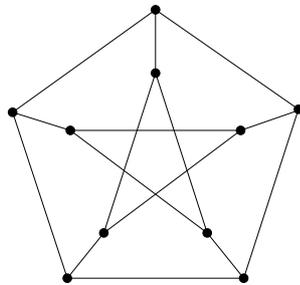


Figure 2.4: The Petersen graph, a graph with $\chi^{*}(G) = 3$ but $\chi'(G) = 4$.

definition for the fractional chromatic index is as follows: For a graph $G = (V, E)$ define

$$\Gamma(G) := \max \left[\frac{E[U]}{\left\lfloor \frac{|U|}{2} \right\rfloor} \right], \quad (2.6)$$

where the maximum is taken over all non-singleton subsets $U \subseteq V$ with $|U|$ odd. Then we obviously have

$$\chi'(G) \geq \Gamma(G) \quad (2.7)$$

and $\chi^{*}(G) = \max\{\Delta(G), \Gamma(G)\}$ (see [152]).

Since $\chi^{*}(G) \leq \chi'(G)$ for all graphs, the fractional chromatic index yields a lower bound for the chromatic index. Plantholt [138, 139] proved that

$$\chi'(G) \leq \chi^{*}(G) + \left\lceil \frac{n}{8} \right\rceil - 1 \quad (2.8)$$

which means that for any multigraph $G = (V, E)$ we have $\chi^{*}(G) = \chi'(G)$ unless $|V| > 8$. In a followup paper [142] the authors also identified the multigraphs with $|V| \in \{9, 10\}$ for which $\chi' = \chi^{*} + 1$, thus completing the analysis for multigraphs with at most 10 nodes. Goldberg [66], Andersen [3] and Seymour [154] conjectured that given $\chi^{*}(G)$ the computation of $\chi'(G)$ is only a decision problem. They claimed

Conjecture 1. *Every multigraph $G = (V, E)$ satisfies*

$$\chi'(G) \leq \max\{\Gamma(G), \Delta(G) + 1\}.$$

As $\chi^{*}(G) = \max\{\Delta(G), \Gamma(G)\}$ this conjecture would imply that the chromatic index of any multigraph can be approximated by the fractional chromatic index within a constant error of one.

Kahn [97] showed that Conjecture 1 is true asymptotically and Schrijver [152] proved that the separation problem over the polytope defined by (2.5) is solvable in polynomial time (see Section 2.3 for details about polyhedra). Thus – by the equivalence of separation and optimisation [76] – the fractional chromatic index can be determined in polynomial time.

Recall that we set $X = E$ and $\mathcal{S} = \mathcal{M}(G)$ to phrase the edge-colouring problem on multigraphs as an set-covering problem. One drawback of this formulation is the big size of $\mathcal{M}(G)$:

Theorem 2 (Maximum matchings in complete graphs). *Let $G = K_n$ be the complete graph on n nodes. Then G has*

$$\prod_{i=0}^{\lfloor (n-1)/2 \rfloor} (2i + 1)$$

pairwise disjoint matchings of maximum cardinality.

Proof. The proof is by induction, the base case $n = 2$ being clear.

If n is even, fix a node $u \in G$. By induction hypothesis, for any edge $uv \in G$ the graph $G \setminus \{u, v\}$ has $\prod_{i=0}^{\lfloor (n-3)/2 \rfloor} (2i+1)$ different maximum cardinality matchings. Combining each of these matchings with uv yields a maximum cardinality matching in G . Moreover, each edge $uv \in G$ results in a set of different matchings, thus we have

$$(n-1) \cdot \prod_{i=0}^{\lfloor (n-3)/2 \rfloor} (2i+1) = \prod_{i=0}^{\lfloor (n-1)/2 \rfloor} (2i+1)$$

maximum cardinality matchings in G .

Now assume that n is odd and consider a node $u \in G$. The graph $G \setminus u$ has $\prod_{i=0}^{\lfloor (n-2)/2 \rfloor} (2i+1)$ maximum cardinality matchings by induction hypothesis. For each $u \in G$ these matchings are different and we therefore have

$$n \cdot \prod_{i=0}^{\lfloor (n-2)/2 \rfloor} (2i+1) = \prod_{i=0}^{\lfloor (n-1)/2 \rfloor} (2i+1)$$

different maximum cardinality matchings in G . □

Notice that Theorem 2 only counts the number of maximum cardinality matchings. The number of matchings in K_n is even bigger. As first attempt to reduce the cardinality of \mathcal{S} in the set-covering formulation, we may restrict ourselves to maximum cardinality matchings (Figure 2.3(b)). To this end, let $\mathcal{M}^*(G)$ denote the set of all maximum cardinality matchings in G . Obviously,

$$\chi'(G) = \min \left\{ \sum_{M \in \mathcal{M}^*(G)} \lambda_M : \sum_{M \in \mathcal{M}^*} \lambda_M \chi^M \geq \mathbb{1}, \lambda_M \in \{0, 1\} \right\} \quad (2.9)$$

and

$$\chi^*(G) \leq \min \left\{ \sum_{M \in \mathcal{M}^*(G)} \lambda_M : \sum_{M \in \mathcal{M}^*} \lambda_M \chi^M \geq \mathbb{1}, \lambda_M \in [0, 1] \right\}. \quad (2.10)$$

The above inequality is true since using only maximum cardinality matchings is equivalent to requiring $\lambda_M = 0$ in (2.5) for all non-maximum cardinality matchings in G and this can of course not decrease the value of the minimum.

In a further attempt to decrease the cardinality $|\mathcal{S}|$ of the set \mathcal{S} in the set-covering formulation we define an equivalence relation \sim on the set $\mathcal{M}^*(G)$ (Figure 2.5):

$$M_1 \sim M_2 \iff \text{The simple graphs given by } M_1 \text{ and } M_2 \text{ are equal.}$$

Graph theory experts note that we require the graphs to be equal and not just isomorphic! This is not strictly required but simplifies reasoning and notation. Let $[M]$ denote the equivalence class of $M \in \mathcal{M}^*(G)$ and denote by $[\mathcal{M}^*(G)]$ the set of equivalence classes of

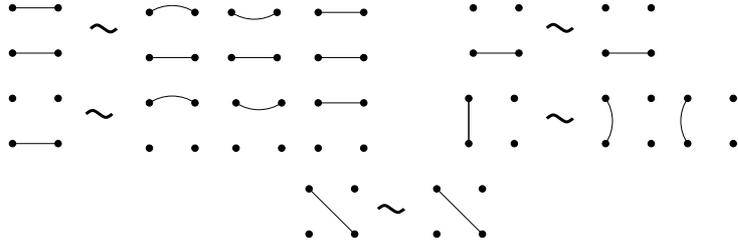


Figure 2.5: Equivalence classes of matchings in the example multigraph.

maximum cardinality matchings under relation \sim . It is clear that for simple graphs we have $[M] = \{M\}$ and $[\mathcal{M}^*(G)] = \bigcup_{M \in \mathcal{M}^*(G)} \{\{M\}\}$. For multigraphs however, the set $[\mathcal{M}^*(G)]$ is usually considerably smaller than $\mathcal{M}^*(G)$. In order to state the edge-colouring problem as set-covering problem in terms of $[\mathcal{M}^*(G)]$ we must slightly adjust the definition in (2.4). To this end we say that $[M]$ *covers* an edge $e \in E$, if $[M]$ contains a matching that covers e and denote by $[\mathcal{M}^*(G, e)]$ the set of all equivalence classes that cover e . With these definitions we have (Figure 2.3(c))

$$\chi'(G) = \min \left\{ \sum_{[M] \in [\mathcal{M}^*(G)]} \lambda_{[M]} : \sum_{[M] \in [\mathcal{M}^*(G, e)]} \lambda_{[M]} \geq \mu(e), e \in E, \lambda_M \in \mathbb{N} \right\} \quad (2.11)$$

and similarly for the fractional chromatic index.

Apart from reducing the size of \mathcal{S} , equivalence relation \sim offers another advantage. Obviously the representatives for $[\mathcal{M}^*(G)]$ can be chosen from the representatives of $[\mathcal{M}^*(K_{|V|})]$ for any multigraph $G = (V, E)$. We can thus consistently represent all maximum cardinality matchings that can exist in a multigraph on n nodes by $[\mathcal{M}^*(K_n)]$. We will see later, that the multigraph to be edge-coloured is not always known beforehand and in these cases $[\mathcal{M}^*(K_n)]$ will come in quite handy. For ease of exposition we slightly abuse notation and write $[M] \in [\mathcal{M}^*(n)]$ and $[M] \in [\mathcal{M}^*(n, e)]$ instead of $[M] \in [\mathcal{M}^*(K_n)]$ and $[M] \in [\mathcal{M}^*(K_n, e)]$.

Bounds for the Chromatic Index and Edge-Colouring Algorithms

Determining the chromatic index of a multigraph is \mathcal{NP} -complete [94], i. e., there are no efficient algorithms known that apply to any graph. It is thus important to have (tight) bounds for the chromatic index which can be computed in polynomial time and to have good approximation algorithms that run in polynomial time.

Apart from the trivial lower bound $\chi'(G) \geq \Delta(G)$ we stated in the previous section that $\chi'(G) \geq \Gamma(G)$. For general graphs, these are the only known lower bounds on the chromatic index. However, for some graphs (such as triangles or stars), these bounds are tight.

An edge-colouring \mathbf{c} is called *partial*, if it is only defined on a subset $F \subseteq E$ of edges,

i. e., if some edges are allowed to be uncoloured. For any edge-colouring, we say that a colour is *missing* from a vertex, if no edge incident to the vertex receives this colour. Most edge-colouring algorithms are variants of the following general scheme:

Algorithm 1 *Basic Edge-Colouring Scheme*

Input: A multigraph $G = (V, E)$.

Output: An edge-colouring $\mathfrak{c} : E \rightarrow \mathbb{N}$ for G .

```

1  Choose a set  $C$  of potential colours available.
2  For  $uv \in E$  Do
3    If there is a colour  $c \in C$  missing from  $u$  and  $v$  Then
4      Set  $\mathfrak{c}(uv) = c$ .
5    Else
6      Change colours in  $G$  such that a colour  $c'$  is missing at  $u$  or  $v$ .
7      Set  $\mathfrak{c}(uv) = c'$ .
8    End If
9  End For
10 Return  $\mathfrak{c}$ .
```

It is clear that the algorithm never uses more than $|C|$ colours to edge-colour G . The most important steps in Algorithm 1 are steps 1 and 6. The smaller the set C of potential colours is chosen, the more complicated will the *recolouring* in step 6 be. The most simple choice for C is $C = E$. In this case, there always is a colour c that is missing from both endpoints of uv in step 3 and we never need to recolour any edges. However, one can do much better than this:

Theorem 3 (Shannon [155]). *For any multigraph $G = (V, E)$ Algorithm 1 can be implemented with $|C| = \lfloor 3\Delta(G)/2 \rfloor$ in time $\mathcal{O}((n + \Delta)m)$ and using space $\mathcal{O}(n + m)$. Thus,*

$$\chi'(G) \leq \left\lfloor \frac{3\Delta(G)}{2} \right\rfloor \quad (2.12)$$

for any multigraph.

Theorem 4 (Vizing [160]). *For any multigraph $G = (V, E)$ Algorithm 1 can be implemented with $|C| = \Delta(G) + \mu(G)$ in time $\mathcal{O}((n + \Delta)m)$ and using space $\mathcal{O}(n + m)$. Thus,*

$$\chi'(G) \leq \Delta(G) + \mu(G) \quad (2.13)$$

for any multigraph.

The latter bound is especially interesting for simple graphs because it implies $\chi'(G) \in \{\Delta, \Delta + 1\}$ if G is simple. So for simple graphs the edge-colouring problem is just a decision problem and the chromatic index can be approximated with an absolute error no

greater than one in polynomial time. A bound similar to (2.13) was obtained by Ore [136], who proved that

$$\chi'(G) \leq \max_{u \in V} (\deg(u) + \max_{v \in N(u)} \mu(uv)) \quad (2.14)$$

for any multigraph $G = (V, E)$.

To the best of our knowledge the approximation guarantee of Vizing's bound (and algorithm) has not been established explicitly, but is no better than 2. This can be seen by the following example: Consider a cycle $C = (V, E)$ on four nodes with one chord $e = uv$ (see Figure 2.6) and let all edges be simple except e which has multiplicity $\mu(e) = k$. Then Vizing's upper

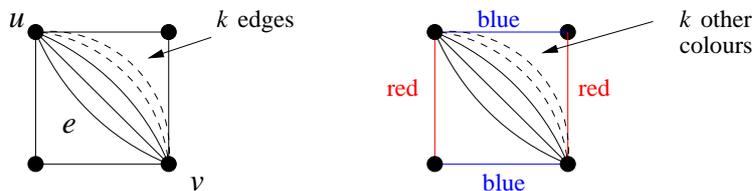


Figure 2.6: Bad example for Vizing's upper bound: The chromatic index is $k + 2$ but the upper bound yields $2k + 2$.

bound suggests $\chi'(C) \leq 2k + 2$ but in fact $\chi'(C) = k + 2$. Thus the ratio between the upper bound and the real chromatic index is asymptotically 2 for this example. The same example shows that the approximation guarantee of Shannon's bound (algorithm) is no better than 1.5.

Using more and more complicated recolouring operations in step 6 of Algorithm 1 several authors obtained in a sequence of papers better and better approximation guarantees:

Nishizeki and Sato [134]	$(5\chi'(G) + 2)/4$
Goldberg [67], Hochbaum, Nishizeki and Shmoys [92]	$(9\chi'(G) + 6)/8$.
Kashiwagi and Nishizeki [133]	$(11\chi'(G) + 8)/10$.

All the algorithms can be implemented to run in time $\mathcal{O}((n + \Delta)m)$ and use $\mathcal{O}(n + m)$ space. Caprara and Rizzi [24] suggested a preprocessing technique that is suitable for any of the above algorithms and improves the performance guarantee of [133] to $1.1\chi'(G) + 0.7$ for example.

Sanders and Steurer [149] took another approach to the edge-colouring problem on multigraphs. Their algorithm is not based on Algorithm 1, but instead attempts to produce a *balanced* partial colouring and then uses Vizing's algorithm to colour the remaining edges. For $\varepsilon > 0$ the algorithm described in [149] has approximation guarantee $(1 + \varepsilon)\chi'(G) + \mathcal{O}(1/\varepsilon)$ and runs in time $\mathcal{O}((n + \Delta) \cdot \text{poly}(1/\varepsilon))$. This algorithm allows for an arbitrarily small factor in the approximation guarantee at the cost of a higher additive term in it.

Although edge-colouring multigraphs is \mathcal{NP} -hard in general, there are polynomial (sometimes even linear) time algorithms to edge-colour certain classes of graphs:

- Bipartite graph can be coloured in polynomial time [152]. We will explain bipartite graphs in detail in Chapter 9.
- Series-parallel multigraphs [162, 135] can be edge-coloured in linear time (see [135] for a definition of series-parallel multigraphs).
- The nearly complete graph $K_{2m+1} - m$ (which arises from the complete graph K_{2m+1} by removing m arbitrary edges) has chromatic index $\chi'(K_{2m+1} - m) = 2m$ and can be optimally edge-coloured in time $\mathcal{O}(m^2)$ [145].

Also known are various bounds for $\chi'(G)$ for multigraphs $G = (V, E)$ where the proofs of the bounds are not constructive, i. e., do not yield a polynomial time algorithm. For example, regular multigraphs of sufficiently high degree are 1-factorizable [141, 143] which implies that $\chi'(G) = \Delta(G)$ for these graphs (it is nevertheless \mathcal{NP} -complete to determine the chromatic index of regular multigraphs in general [61]). $\chi'(G) = \Gamma(G)$ is also true for *nearly bipartite* graphs [140], i. e., graphs that can be made bipartite by removing a vertex. If in the multigraph $G = (V, E)$ each cycle of length greater than 2 contains a simple edge, then $\chi'(G) \leq \Delta(G) + \lceil \sqrt{\mu(G)} \rceil$ [78]. For multigraphs on a sufficiently large number n of nodes we have $\chi'(G) \leq \Gamma(G) + 1 + \sqrt{n \log n / 10}$, i. e., the difference between $\chi'(G)$ and its upper bound is eventually sublinear [137].

For simple graphs there are some algorithms that do not have a performance guarantee, but seem to work pretty well in practice. These algorithms include Ant Systems and Evolutionary Algorithms [91] as well as Parallel Algorithms [120, 75].

For any kind of edge-colouring algorithm, the following theorem immediately implies pre-processing techniques that can be applied in order to reduce the size of the multigraph to be coloured.

Theorem 5. *If $G = (V, E)$ is a multigraph then the following facts are true:*

1. *If $e \in E$ is universal (i. e., e is incident to all edges in E) then $\chi'(G) = \chi'(G - e) + 1$.*
2. *For $e \in E$ let $\bar{N}(e) = \{f \in E : e \cap f = \emptyset\}$ denote the set of edges that are not incident to e . If $\bar{N}(e)$ is a matching, then $\chi'(G) = \chi'(G - \bar{N}(e))$.*

By recursively applying this theorem we may reduce the multigraph G until it contains no more universal edges and $\bar{N}(e)$ contains at least two edges and is not a matching for all $e \in E$. Theorem 5 will also be helpful when proving certain statements about the representative model described in Section 6.3 below.

Proof of fact 1. Obviously $\chi'(G - e) \leq \chi'(e)$. Assume $\chi'(G - e) < \chi'(G) - 1$. Then edge-colour $G - e$ using $\chi'(G - e)$ colours and use one additional colour for e to obtain an edge-colouring of G . This colouring uses fewer than $\chi'(G)$ colours, a contradiction. Now assume $\chi'(G - e) = \chi'(G)$. In this case, a minimal edge-colouring of $G - e$ uses $\chi'(G - e) = \chi'(G)$ colours and can be extended to an edge-colouring of G only by using a new colour for e , thus $\chi'(G - e) = \chi'(G) + 1$ which is again a contradiction. This leaves $\chi'(G) = \chi'(G - e) + 1$ which is the claim of fact 1. \square

Proof of fact 2. Assume that $\bar{N}(e)$ is a matching. Since all these edges are mutually non-incident, they may receive the same colour in an edge-colouring of G . Moreover, by definition none of these edges are incident to e , so an edge-colouring on $G - \bar{N}(e)$ can be extended to G by assigning to $f \in \bar{N}(e)$ the same colour as to e . This proves that $\chi'(G - \bar{N}(e)) \geq \chi'(G)$. To see that $\chi'(G - \bar{N}(e)) \leq \chi'(G)$ observe that by deleting the edges in $\bar{N}(e)$, each edge-colouring of G yields an edge-colouring of $G - \bar{N}(e)$. This concludes the proof. \square

2.3 Polyhedra

Before we plunge into the details of the theory of polyhedra, let us recall some basic facts from linear algebra that will be useful in this section (and the rest of this thesis).

A *vector* (or *point*) in \mathbb{R}^n is an n -tuple with components from \mathbb{R} . Unless otherwise specified, we understand vectors as column-vectors, i. e.,

$$x = \begin{pmatrix} x_1 \\ \vdots \\ x_n \end{pmatrix} \in \mathbb{R}^n.$$

For row vectors we write $x^T = (x_1, \dots, x_n)$. A vector $x \in \mathbb{R}^n$ is greater or equal to a vector $y \in \mathbb{R}^n$ if it is componentwise greater or equal to y , i. e., if $x_i \geq y_i$ for $i = 1, \dots, n$. Likewise, it is strictly greater if it is componentwise strictly greater. Similarly, a vector $x \in \mathbb{R}^n$ is greater or equal (strictly greater) than a scalar $r \in \mathbb{R}$ if each component of x is greater or equal (strictly greater) than r . The *scalar product* between two vectors $x, y \in \mathbb{R}^n$ is defined as $x^T y = \sum_{i=1, \dots, n} x_i y_i$ and the (Euclidean) norm of $x \in \mathbb{R}^n$ is $\|x\| = \sqrt{x^T x}$. Distinguished vectors are

$$\begin{aligned} e_j &:= \text{the } j\text{-th canonical basis vector} \\ \mathbb{1} &:= \sum_{i=1, \dots, n} e_i = (1, \dots, 1)^T \text{ (the all-1 vector)} \\ 0 &:= (0, \dots, 0)^T \text{ (the all-0 vector)}. \end{aligned}$$

For a finite set S and $s \in S$ we denote by

$$\hat{x}_s := (\hat{x}_1, \dots, \hat{x}_{|S|})^T \in \mathbb{R}^S \quad \text{with} \quad \hat{x}_i = \begin{cases} 1 & \text{if } i = s, \\ 0 & \text{otherwise} \end{cases}$$

the *incidence vector* of $s \in S$.

The set $\mathbb{R}^{m \times n}$ is the set of all matrices with m rows, n columns and entries from \mathbb{R} . For $A \in \mathbb{R}^{m \times n}$ we write $A = (a_{ij})_{\substack{i=1, \dots, m \\ j=1, \dots, n}}$ and call the a_{ij} the *coefficients* of the matrix A . Notice that a vector is a special kind of matrix, namely one with only one column (column vector) or only one row (row vector).

A set $X = \{x_1, \dots, x_k\}$ of vectors is called *linearly independent* if

$$\sum_{i=1, \dots, k} \lambda_i x_i = 0, \lambda_i \in \mathbb{R}$$

implies $\lambda_i = 0$ for all $i \in \{1, \dots, k\}$. A set X of vectors that is not linearly independent, is called *linearly dependent*. Notice that in \mathbb{R}^n a set of at least $n + 1$ vectors is always linearly dependent.

Similarly, a set $X = \{x_1, \dots, x_k\}$ of vectors is called *affinely independent*, if

$$\begin{aligned} \sum_{i=1, \dots, k} \lambda_i x_i &= 0 \text{ and} \\ \sum_{i=1, \dots, k} \lambda_i &= 0 \end{aligned}$$

implies $\lambda_i = 0$ for all $\lambda \in \mathbb{R}^k$. Again a set of vectors is *affinely dependent* if it is not affinely independent. In \mathbb{R}^n an affinely independent set of vectors cannot contain more than $n + 1$ vectors.

For $S \subseteq \mathbb{R}^n$ we call the cardinality of the inclusionwise biggest set of linearly (affinely) independent subsets of S the (affine) rank of S . The rank of S is denoted by $\text{rg}(S)$ and the affine rank of S by $\text{arg}(S)$. The *dimension* $\dim(S)$ of a set S is defined to be one less than its affine rank, i. e., $\dim(S) = \text{arg}(S) - 1$. The rank of a matrix $A \in \mathbb{R}^{m \times n}$ is defined as the rank of the set of its rows. This rank is always equal to the rank of the set of columns. The rank of a matrix $A \in \mathbb{R}^{m \times n}$ never exceeds $\min\{m, n\}$. If the rank is equal to this number the matrix is said to have *full rank*.

If $X = \{x_1, \dots, x_k\}$ and $y = \sum_{i=1, \dots, k} \lambda_i x_i$ with $\lambda_i \in \mathbb{R}$ and $x_i \in \mathbb{R}^n$, then y is called *linear combination* of X . If additionally $\lambda \geq 0$ and $\sum_{i=1, \dots, k} \lambda_i = 1$ the combination is called *convex*. A (linear or convex) combination is called *pure* if $\lambda_i \neq 0$ for $i = 1, \dots, k$. Given a non-empty set $S \subseteq \mathbb{R}^n$, the *linear hull* and *convex hull* of S – denoted by $\text{lin}(S)$ and $\text{conv}(S)$

respectively – are the sets of all vectors that are linear or convex combinations of vectors from S . By definition, we have $\text{lin}(\emptyset) = \{0\}$ and $\text{conv}(\emptyset) = \emptyset$. A subset $S \subseteq \mathbb{R}^n$ is called *linear space* if $S = \text{lin}(S)$ and *convex set* if $S = \text{conv}(S)$.

A subset $S \subseteq \mathbb{R}^n$ is called

hyperplane if there exist $a \in \mathbb{R}^n \setminus \{0\}$ and $\alpha \in \mathbb{R}$ such that $S = \{x \in \mathbb{R}^n : a^T x = \alpha\}$,

halfspace if there exist $a \in \mathbb{R}^n \setminus \{0\}$ and $\alpha \in \mathbb{R}$ such that $S = \{x \in \mathbb{R}^n : a^T x \leq \alpha\}$,

polyhedron if there exists a matrix $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$ such that $S = \{x \in \mathbb{R}^n : Ax \leq b\}$.

In this case we often write $S = P(A, b)$.

A *polytope* is a polyhedron P that is bounded, i. e., for which exists $B \in \mathbb{R}$ such that $\|x\| \leq B$ for all $x \in P$. Notice that a polyhedron is always a convex set. With a linear program $\min\{c^T x : x \in \mathbb{R}^n, Ax \leq b\}$ we associate the polyhedron $P(A, b)$ and say that the linear program *defines* this polyhedron. Notice that $P(A, b)$ is independent of the objective function $c^T x$. As opposed to linear programs, a mixed integer program is described by a system of linear inequalities $Ax \leq b$ and integrality constraints $x \in \mathbb{Z}^p \times \mathbb{R}^{n-p}$ with $p > 0$. This leads to an alternate definition of polyhedra associated with mixed integer programs. We set $S := \{x \in \mathbb{R}^n : Ax \leq b, x \in \mathbb{Z}^p \times \mathbb{R}^{n-p}\}$. Then the polytope defined by $Ax \leq b, x \in \mathbb{Z}^p \times \mathbb{R}^{n-p}$ is $P := \text{conv}(S)$, i. e., the convex hull of all points in S . This definition is equivalent to the previous one since it can be shown [29] that there exists $D \in \mathbb{R}^{m \times n}$ and $d \in \mathbb{R}^m$ such that $\text{conv}(S) = P(D, d)$. Moreover, any polytope $P(A, b) \subseteq \mathbb{R}^n$ obviously is equal to the convex hull of all the points contained in it. Consequently, a polytope can be equivalently described as either the convex hull of its points or as a set of points that satisfy a system of linear inequalities. In either case, the polytope defined by a (mixed integer) linear program is independent of the objective function.

As we will see shortly, it is often useful to determine the dimension of a polyhedron $P \subseteq \mathbb{R}^n$. For this purpose, we have

Theorem 6 (Dimension of polyhedra [152]). *If $P \subseteq \mathbb{R}^n$ is a polyhedron, then $\dim(P) = k$ if either of the following is true:*

- *P contains $k + 1$ affinely independent points, but no set of $k + 2, \dots, n + 1$ affinely independent points (by definition).*
- *A minimal equation system for P contains exactly $n - k$ equations. A minimal equation system for $P = P(A, b)$ is a system $Dx = d$ of linear equations with the following properties:*
 - *D has full rank.*

- $Dx = d$ holds true for all $x \in P(A, b)$.
- If $d^T x = \delta$ is an equation that is satisfied by all points $x \in P(A, b)$ and that is not contained in $Dx = d$, then $d^T x = \delta$ is a linear combination of equations from $Dx = d$.

Theorem 6 implies that if we can show that the only equation satisfied by all $x \in P$ is the trivial equation $0^T x = 0$, then P is full-dimensional.

As implied by Theorem 6, we will prove statements about the dimension of polyhedra by exhibiting sets of affinely independent points in the polyhedra. For this purpose, the following well-known lemma is helpful.

Lemma 7. *If $x = (x_1 \dots x_n) \in \mathbb{R}^n$ is a vector and $I \subseteq \{1, \dots, n\}$, then*

$$\{x\} \cup \bigcup_{i \in I} \{x + e_i\}$$

is a set of affinely independent vectors.

Let $P \subseteq \mathbb{R}^n$ be a polyhedron. An inequality $a^T x \leq \alpha$ with $a \in \mathbb{R}^n \setminus \{0\}$ and $\alpha \in \mathbb{R}$ is a *valid inequality* for P , if $a^T x \leq \alpha$ for all $x \in P$. A subset $F \subseteq P$ is called a *face* of P if there is a valid inequality $d^T x \leq \delta$ such that $F = \{x \in P : d^T x = \delta\}$. In this case inequality $d^T x \leq \delta$ is said to define or induce face F . Notice that by definition a face of P is again a polyhedron. A face F of P is *proper* if $F \neq P$ and *non-trivial* if $F \neq \emptyset$ and $F \neq P$. An inclusionwise maximal non-trivial face F of P is called *facet*. As each non-trivial face is the intersection of facets [152], it is clear that the facets of P yield a minimal description of P by inequalities. Due to this fact, facets play a crucial role in the analysis of the facial structure of a polyhedron. In order to prove that an inequality induces a facet of a polyhedron, there are two basic criteria:

Theorem 8 (Facets of polyhedra [132, 152]). *If $P \subseteq \mathbb{R}^n$ is a polyhedron and $d^T x \leq \delta$ is a valid inequality for P , then the face F induced by $d^T x \leq \delta$ is a facet of P if*

1. *for all faces $F' := \{x \in P : b^T x = \beta\}$ of P with $F \subseteq F'$ there is $a \in \mathbb{R}$ such that $b = a \cdot d$ and $\beta = a \cdot \delta$ or*
2. $\dim(\{x \in P : d^T x = \delta\}) = \dim(P) - 1$.

When it comes to optimisation, faces of polyhedra also play an important role, for the following fact is well known and easily established.

Theorem 9. *Let $\min\{c^T x : x \in \mathbb{R}^{n-p} \times \mathbb{Z}^p, Ax \leq b\}$ be a (mixed integer) linear program and P the polyhedron associated with it. If the program has an optimal solution, then it has an optimal solution lying on a face of P .*

It can even be shown, that if a (mixed integer) linear program has an optimal solution, then it has one that is a vertex (a zero-dimensional face) of the associated polyhedron.

2.4 Relaxations and Branch-and-Cut

Recall the (mixed) integer programming problem from Section 2.1:

$$\begin{aligned} & \text{minimise} && c^T x \\ & Ax &\leq & b \\ & x &\in & \mathbb{Z}^p \times \mathbb{R}^{n-p} \end{aligned} \tag{2.15}$$

where $c \in \mathbb{R}^n$, $A \in \mathbb{R}^{m \times n}$, $b \in \mathbb{R}^m$ and $p > 0$. This problem is \mathcal{NP} -complete in general, which is easily seen by transformation to the satisfiability problem. However, if we had $p = 0$ then (2.15) could (in theory) be solved in polynomial time by the ellipsoid method [108, 109] and (in practice) efficiently by the simplex method [32].

Consider the two related polyhedra

$$\begin{aligned} P_{\text{MIP}} &= \{x \in \mathbb{R}^n : Ax \leq b, x \in \mathbb{Z}^p \times \mathbb{R}^{n-p}\} \\ P_{\text{LP}} &= \{x \in \mathbb{R}^n : Ax \leq b, x \in \mathbb{R}^n\}. \end{aligned}$$

and the corresponding optimisation problems

$$\begin{aligned} z_{\text{MIP}}^* &= \min\{c^T x : x \in P_{\text{MIP}}\} && \text{(MIP)} \\ z_{\text{LP}}^* &= \min\{c^T x : x \in P_{\text{LP}}\}. && \text{(LP)} \end{aligned}$$

Polyhedron P_{LP} arises from P_{MIP} by relaxing the condition $x \in \mathbb{Z}^p \times \mathbb{R}^{n-p}$ to $x \in \mathbb{R}^n$, thus turning the corresponding optimisation problem from a mixed integer program (MIP) into a linear program (LP). For this reason (LP) is often called the *LP relaxation* of (MIP) (sometimes also P_{LP} is called the LP relaxation of P_{MIP} as well). Since $P_{\text{MIP}} \subseteq P_{\text{LP}}$ it is clear that $z_{\text{LP}}^* \leq z_{\text{MIP}}^*$. In other words z_{LP}^* gives a lower bound on z_{MIP}^* . Moreover, if $x^* = \operatorname{argmin}\{c^T x : x \in P_{\text{LP}}\}$ is a vertex of P_{LP} (remember that if there is an optimal solution, then there is always one at a vertex), but not a vertex of P_{MIP} then – by the famous Hahn-Banach Theorem – there exists a hyperplane $a^T x = \alpha$ that separates P_{MIP} and x^* . Then $a^T x \leq \alpha$ (or $a^T x \geq \alpha$) is valid for P_{MIP} but not valid for P_{LP} , hence $x^* \notin P_{\text{LP}} \cap \{x \in \mathbb{R}^n : a^T x \leq \alpha\}$. Geometrically, the hyperplane defined by $a^T x = \alpha$ *cuts off* vertex x^* from P_{MIP} and is therefore called a *cutting plane* (often the same term is used for inequality $a^T x \leq \alpha$ itself). A cutting plane $a^T x \leq \alpha$ that separates a point x^* from P_{MIP} is said to be *violated* by x^* .

These considerations give rise to an iterative optimisation algorithm (see also [69, 70, 71]) for solving $z^* = \min\{c^T x : Ax \leq b, x \in \mathbb{Z}^p \times \mathbb{R}^{n-p}\}$:

Algorithm 2 *Cutting Plane Algorithm*

```

1  Let  $P_{\text{LP}} = \{x \in \mathbb{R}^n : Ax \leq b, x \in \mathbb{R}^n\}$ .
2  While  $x^* = \operatorname{argmin}\{c^T x : x \in P_{\text{LP}}\} \notin \mathbb{Z}^p \times \mathbb{R}^{n-p}$  Do
3    Find a violated cutting plane  $a^T x \leq \alpha$  and set
       $P_{\text{LP}} = P_{\text{LP}} \cap \{x \in \mathbb{R}^n : a^T x \leq \alpha\}$ .
4  End While
5  Return  $x^*$  and  $c^T x^*$ .

```

Gomory [69, 70] proved that the number of iterations in Algorithm 2 is finite. It might however be exponential, so Algorithm 2 is not a polynomial time algorithm.

Another way to solve (MIP) with the aid of (LP) is enumeration: we enumerate all points in \mathbb{Z}^p and determine for each point the respective objective function value of (MIP). More formally, we compute

$$z^* = \min_{y \in \mathbb{Z}^p} \min\{c^T x : x \in P_{\text{LP}}, x_i = y_i, i = 1, \dots, p\}. \quad (2.16)$$

This also yields the optimal solution but requires us to enumerate a potentially infinite set. So we do not perform this enumeration explicitly. Instead, we start by solving (LP) to optimality. Assume that x^* is the optimal solution to (LP). If $x_i \in \mathbb{Z}$ for all $i \in \{1, \dots, p\}$ we have found the optimal solution for (MIP). Otherwise we pick an index $j \in \{1, \dots, p\}$ such that $x_j^* \notin \mathbb{Z}$. It is clear that the optimal solution to (MIP) is contained in either

$$\{x \in P_{\text{LP}} : x_j \geq \lceil x_j^* \rceil\} \quad \text{or} \quad \{x \in P_{\text{LP}} : x_j \leq \lfloor x_j^* \rfloor\}.$$

We therefore create the following two subproblems:

$$z^* = \min\{c^T x : x \in P_{\text{LP}}, x_j \geq \lceil x_j^* \rceil\} \quad (2.17)$$

$$z^* = \min\{c^T x : x \in P_{\text{LP}}, x_j \leq \lfloor x_j^* \rfloor\}. \quad (2.18)$$

We recursively solve these problems with the same strategy. In case the optimal solution x^* for a subproblem satisfies $x_i \in \mathbb{Z}$ for $i = 1, \dots, p$, we have found a feasible solution. However, this solution need not be optimal since some other subproblem may have a better feasible solution. The optimal solution is the minimum over all feasible solutions found. At first glance, this strategy is still exhaustive, i. e., all points in \mathbb{Z}^p are tested eventually. However, this can be remedied by the following two observations:

1. If a subproblem turns out to be infeasible (the respective polyhedron is empty) then no further subproblems need to be generated from this problem since they would be infeasible as well.

2. Let u be an upper bound for the objective function value, i. e., $u \geq \min\{c^T x : x \in P_{\text{MIP}}\}$. If a subproblem has an optimal objective function value greater than u , then neither the subproblem nor any subproblem generated from it can contain the optimal solution. Hence we stop to recurse on them.

In practice, these two pruning strategies render the potentially exhaustive enumeration scheme feasible. We define

Algorithm 3 *Branch-and-Bound*

- 1 Initialise a problem list L with (LP).
- 2 Set $u = \infty$ (or to a finite value that is a known upper bound on the objective function value).
- 3 Set $z_{\text{best}} = \infty$ and $x_{\text{best}} = \text{NIL}$.
- 4 **While** $L \neq \emptyset$ **Do**
- 5 Choose a problem P from L , remove it from L and solve it to optimality. Let x_P^* denote the optimal point and z_P^* the objective function value.
- 6 **If** P is infeasible **Then Goto** 4.
- 7 **Else If** $x_P^* \in P_{\text{MIP}}$ and $z_P^* < z_{\text{best}}$ **Then** set $z_{\text{best}} = z_P^*$, $x_{\text{best}} = x_P^*$, $u = \min\{u, z_P^*\}$ and remove from L all problems that have objective function value greater than z_P^* .
- 8 **Else If** $z_P^* > u$ **Then Goto** 4.
- 9 **Else** pick an index $j \in \{1, \dots, p\}$ with $x_{P,j}^* \notin \mathbb{Z}$ and add the following two problems to L :

$$\min\{c^T x : x \in P, x_j \geq \lceil x_{P,j}^* \rceil\}, \quad \min\{c^T x : x \in P, x_j \leq \lfloor x_{P,j}^* \rfloor\}.$$
- 10 **End While**
- 11 **Return** x_{best} and z_{best} .

The name Branch-and-Bound stems from the fact that the list L is usually kept as binary tree (the root node being the LP-relaxation (LP)) and that creating two subproblems of P in step 9 means adding two branches to the node storing problem P . The term “bound” simply refers to the fact that branches are created by bounding variables. The best solution found so far (x_{best} in Algorithm 3) is called the *incumbent solution* (or simply incumbent for short). Remember that Algorithm 3 is still potentially exhaustive, but the pruning rules in steps 6 and 8 perform quite well in practice.

The last algorithm described in this section is *Branch-and-Cut*. This algorithm is based on Branch-and-Bound but uses cutting planes to strengthen the LP-relaxations found during the enumeration. In step 5 of Branch-and-Bound we not simply solve the subproblem P to

optimality. Instead we solve it to optimality and then repeatedly find violated cutting planes, add them to P and reoptimise P . This usually increases the objective function value of P and/or changes the number of fractional variables in the optimal solution to P . By doing so, we can prune much more subproblems from L . Notice that adding all violated cutting planes to P would result in a simple cutting plane algorithm, which would in turn lead to a large number of iterations. We thus do not add all violated cutting planes but only a limited number. This limit may either be an upper limit on the number of cutting planes added per iteration or a more abstract criterion such as the minimal amount by which adding the cutting plane changes the objective function value. The set of cutting planes added usually depends on the structure of the optimisation problem to solve. A formal description of the Branch-and-Cut algorithm is as follows:

Algorithm 4 *Branch-and-Cut*

- 1 Initialise a problem list L with (LP).
- 2 Set $u = \infty$ (or to a finite value that is a known upper bound on the objective function value).
- 3 Set $z_{\text{best}} = \infty$ and $x_{\text{best}} = \emptyset$.
- 4 **While** $L \neq \emptyset$ **Do**
- 5 Choose a problem P from L , remove it from L and solve it to optimality. Let x_P^* denote the optimal point and z_P^* the objective function value.
- 6 **While** the cutting plane limit for this iteration is not reached, P is feasible and $z_P^* \leq u$ **Do**
- 7 Find violated cutting planes $a_i^T x \leq \alpha_i$ and add them to P .
- 8 Reoptimise P to obtain x_P^* and z_P^* .
- 9 **End While**
- 10 **If** P is infeasible **Then Goto** 4.
- 11 **Else If** $x_P^* \in P_{\text{MIP}}$ and $z_P^* < z_{\text{best}}$ **Then** set $z_{\text{best}} = z_P^*$, $x_{\text{best}} = x_P^*$, $u = \min\{u, z_P^*\}$ and remove from L all problems that have objective function value greater than z_P^* .
- 12 **Else If** $z_P^* > u$ **Then Goto** 4.
- 13 **Else** pick an index $j \in \{1, \dots, p\}$ with $x_{P,j}^* \notin \mathbb{Z}$ and add the following two problems to L :

$$\min\{c^T x : x \in P, x_j \geq \lceil x_{P,j}^* \rceil\}, \quad \min\{c^T x : x \in P, x_j \leq \lfloor x_{P,j}^* \rfloor\}.$$
- 14 **End While**
- 15 **Return** x_{best} and z_{best} .

The key to success of this algorithm is the loop in steps 6 through 9. Efficient reoptimisation in step 8 can usually be implemented by means of the dual simplex algorithm [117], so the

most time-critical ingredient is separation of violated cutting planes. Here it is not vital to only separate them quickly but also to separate those that have great impact on the objective function value or the integrality of the optimal solution at the current node.

Another way to reduce the running time of a Branch-and-Cut algorithm are good upper bounds: The smaller the initial upper bound u is, the more problems can be pruned. It is thus also important to find such good bounds before the algorithm is started. One way to find these bounds is to construct feasible solutions with a small objective function value by heuristics. A great part of this thesis is devoted to developing this kind of heuristics. Another common way to improve the upper bound u is to generate feasible solutions with the help of the optimal fractional solution at some node in the Branch-and-Cut tree. This can be done for example by rounding all fractional variables and checking whether the resulting point is contained in P_{MIP} .

One advantage of Branch-and-Bound and Branch-and-Cut algorithms is that they can give some proof of quality for the incumbent solution x : If L is the list of unsolved subproblems and $z_{\text{LP}}^*(P)$ is the optimal solution of the LP relaxation for $P \in L$, then $\min_{P \in L} z_{\text{LP}}^*(P)$ is a lower bound on the best possible objective function value. If $z^*(x)$ is the objective function value of the incumbent solution, then $z^*(x) - \min_{P \in L} z_{\text{LP}}^*(P)$ yields an upper bound on the maximum absolute error that we get if we accept x as optimal solution.

Nowadays Branch-and-Cut algorithms with decent separation algorithms are among the most successful solution strategies for general and specific (mixed) integer programming problems [59, 126]. The algorithms usually perform the better the more of the structural information about the optimisation problems and associated polyhedra is incorporated into them.

Chapter 3

Graph-Partitioning with Communication Overhead

The significant problems we face cannot be solved
by the same level of thinking that created them.
— *Albert Einstein*

In Computational Fluid Dynamics (CFD) one considers the simulation of fluid systems. In order to perform such simulations several classes of (partial differential) equations must be solved over the domain on which the simulation is defined. Solution of these equations by means of a computer requires discretisation of the domain using Finite Difference, Finite Element, or Finite Volume methods [19, 52]. The concrete discretisation of a domain is called a *grid* for this domain. Such a grid usually comprises of single points in 2D, called *grid points*, or small volumes in 3D, called *control volumes*.

There are three kinds of grids that are commonly used in Computational Fluid Dynamics: structured grids, unstructured grids and block structured grids. The simplest one are *structured grids* (Figure 3.1(a)). In these grids all grid elements are alike (the grid is *regular*) and can be easily stored in computer memory and handled during computation. Unstructured grids can be used only on simple geometries such as cylindrical tubes.

To the contrary an *unstructured grid* contains grid elements of different shape and size (the grid is *irregular*). These grids can be adapted to complex geometries and also allow different degrees of discretisation depending on the place of the grid elements. Figure 3.1(b) shows an example of an unstructured grid for an airfoil. It is obvious that the discretisation is more fine-grained in the direct vicinity of the airfoil than it is at further distance. Due to their irregularity unstructured grids are much more difficult to describe, store and handle.

A compromising class of grids are *block structured* grids. To obtain such a grid the domain

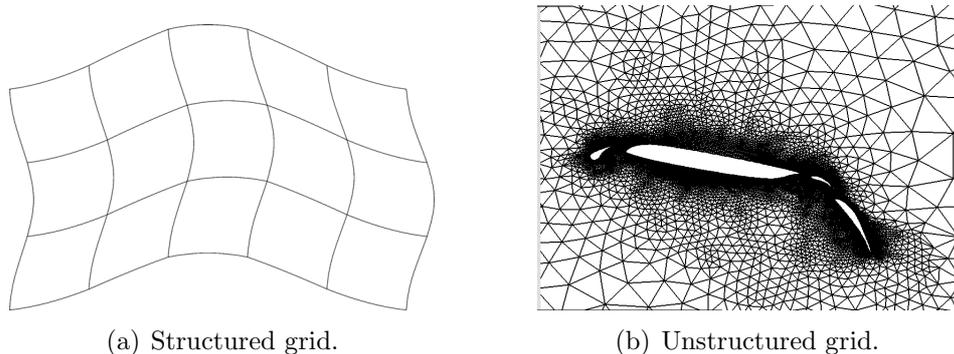


Figure 3.1: Structured and unstructured grids [164].

is first divided into a set of unstructured blocks. In different blocks then structured grids of different kinds are used. This approach restricts the disadvantages of unstructured grids to the blocks.

We will not go into further details about grids, grid generation or actually simulating a turbulent flow and refer the interested reader to [19, 52]. Instead we will focus on the following problem that arises when a grid-based simulation is to be solved on parallel computers: Each grid element must be assigned to a (unique) processor that performs the computations associated with it. In order to achieve minimal simulation time we should equally balance the computational load associated with the grid elements over all processors. However, we must also consider data dependency between grid elements: adjacent grid elements usually require data exchange between each other during the simulation. While this data is easily exchanged between grid elements on the same processor, its exchange between processors may introduce significant communication overhead. When assigning grid elements to processors we must thus not only balance computational load but also keep communication overhead small.

We assume in the following that we are given a block structured grid and that all grid elements belonging to the same block must be handled on the same processor. We must thus distribute the blocks into which the grid decomposes over the set of available processors. To this end we represent the grid by an undirected, node-weighted *grid graph* $G = (V, E)$ as follows: for each block in the grid we introduce a node $u \in V$ with weight (or size) $\kappa_u \in \mathbb{N}$ that counts the number of control volumes in the respective block. Since there is a one-to-one correspondence between blocks in the grid and nodes in the grid graph, we will often use these two notions interchangeably. Blocks that are adjacent (in n -dimensional grids these are usually blocks that share an $(n - 1)$ -dimensional face) must exchange data during the simulation. To represent this data dependency we introduce an edge $uv \in E$ for any pair of adjacent blocks in the grid (see Figure 3.2). Mapping the grid to the available processors is then equivalent to mapping the grid graph to them.

The simulation itself is assumed to be an algorithm of the following structure:

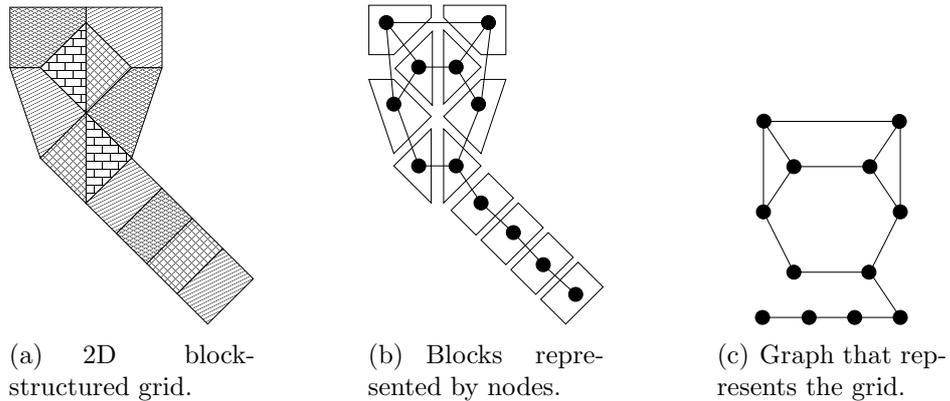


Figure 3.2: The grid-graph arising from a block-structured grid.

Algorithm 5 *Grid-Based Simulation*

- 1 Initialise the simulation.
- 2 **While** the simulation is not finished **Do**
- 3 Perform computations on the parallel machines.
- 4 Exchange data between processors as required.
- 5 **End While**
- 6 Collect results and return them.

In step 3 each of the processors performs the arithmetic operations that are associated with the blocks mapped to it. We call the operations associated with this step the *arithmetic part* of the simulation. In step 4 the processors exchange data to satisfy data dependencies between adjacent blocks. This step is the *communication part* of the simulation. Moreover, we assume that between steps 3 and 4 the processors are *synchronised*, i. e., no processor can start data exchange between blocks unless the last processor has finished its arithmetic computations in step 3. In a typical simulation the loop in Algorithm 5 is carried out several millions of times. It is hence crucial to keep the running time for one iteration of this loop small.

We call a function $\mathbf{m} : V \rightarrow P$ that maps each node $u \in V$ of the grid graph to a processor $p = \mathbf{m}(u) \in P$ a *block-to-processor-mapping* or often simply a *mapping*. For a fixed mapping, we call edges $uv \in V$ *inter-processor* if $\mathbf{m}(u) \neq \mathbf{m}(v)$ and *intra-processor* otherwise. In a grid graph $G = (V, E)$, an edge $uv \in E$ is *cut* with respect to \mathbf{m} , if $\mathbf{m}(u) \neq \mathbf{m}(v)$, i. e., if uv is inter-processor. The set of edges cut with respect to \mathbf{m} is called the *multicut* (with respect to \mathbf{m}). Obviously, a mapping is a partitioning of the set of blocks and vice versa. That is why we (and others) refer to the optimal block-mapping problem as graph-partitioning problem.

The optimal partitioning of a grid graph heavily depends on the hardware on which the

simulation is to be executed. We assume that we have a set $P = \{p_0, \dots, p_{|P|-1}\}$ of processors available. All processors are the same and since memory is limited they have an upper limit K on the number of control volumes they can handle. In order to measure the running time of one iteration of the simulation we assume that it takes a processor t_a milliseconds per iteration to perform the arithmetic operations for one control volume that is mapped to it. For a grid graph $G = (V, E)$ and a mapping $\mathbf{m} : V \rightarrow P$ we denote by $\mathbf{m}^{-1}(p)$ the pre-image of p under \mathbf{m} and define

$$b(\mathbf{m}) := \max_{p \in P} \sum_{u \in \mathbf{m}^{-1}(p)} \kappa_u. \quad (3.1)$$

Then the time required to perform step 3 of Algorithm 5 is $b(\mathbf{m}) \cdot t_a$. It is clear that $b(\mathbf{m}) \geq \lceil \sum_{v \in V} \kappa_v / |P| \rceil$. A mapping \mathbf{m} *perfectly balances the load* of a grid graph if $b(\mathbf{m}) = \lceil \sum_{v \in V} \kappa_v / |P| \rceil$. In this case we also say that the load is perfectly balanced. Notice that the existence of a perfect balancing depends on the grid graph $G = (V, E)$ as well as on the number $|P|$ of processor to which we map.

To account for the overhead introduced by inter-processor communication in step 4 of Algorithm 5 we make the following assumptions (a communication model with similar assumptions is commonly used for the optimal data migration problem [79]):

- The time required to satisfy data dependencies between blocks that are mapped to the same processor can be neglected.
- When two adjacent blocks u and v that are mapped to different processors must exchange data, we must establish a communication channel, exchange data over this channel and release the channel afterwards. The bottleneck in this sequence of operations is acquisition of the communication channel. Once the channel has been acquired, the time required for data exchange and release of the channel can be neglected. We therefore consider only the time that it takes to set up a communication channel between two processors and denote this time by t_c . Notice that we assume that this time is equal for all processor pairs, i. e., we assume homogeneous network connections.

For the minimisation of communication overhead two other hardware properties are important: first we assume that each processor is connected *directly* to all of the other processors. So there is no need to route communication requests through processors other than the endpoints of the request. The second hardware property to consider is that each processor can have at most one active communication channel. In other words, if processor p_1 is currently exchanging data with processor p_2 , it cannot send or receive data to or from any other processor. This limitation requires us to not only keep the amount of inter-processor communication small, but also to schedule this communication cleverly, so as to minimise the overhead incurred by it.

Given a mapping \mathbf{m} for a grid graph $G = (V, E)$, the *processor multigraph* $M_P(\mathbf{m}) = (V_P, E_P)$ (or simply M_P if the mapping is understood from the context) is the multigraph arising from

G by identifying all nodes that are mapped to the same processor under \mathbf{m} and deleting loops (see Figure 3.3). Notice that for this graph $V_P = P$, i. e., the nodes in M_P are precisely the processors. Observe that different mappings give rise to different multigraphs, see Figure 3.4. The nodes in V_P represent the processors and the edges in E_P the inter-processor edges (the multicut) under \mathbf{m} . All the data dependencies corresponding to these edges must be satisfied in step 4 of Algorithm 5. Since each processor can only have one communication channel at a time, this means that we must decompose E_P into a set of edge-sets that contain only pairwise non-incident edges. In other words, we must decompose M_P into matchings which is the same as edge-colouring it. If c is the number of colours used in such a colouring, then the communication overhead in step 4 of Algorithm 5 is $c \cdot t_c$ since we consider only the setup time for the communication channels. We call the colour classes in an edge-colouring of M_P (*communication*) *rounds*. So if we want to minimise the time required for step 4 in Algorithm 5, we must use a minimal number of rounds, i. e., a minimal edge-colouring.

Summarising the above definitions and considerations, we define the following optimisation problem:

For a grid graph $G = (V, E)$ and a processor set P , both as described above, the *Optimal Graph-Partitioning Problem with Communication Overhead* (OGPC) is to find a mapping \mathbf{m}^* such that

$$t_a \cdot b(\mathbf{m}^*) + t_c \cdot \chi'(M_P(\mathbf{m}^*)) = \min \{t_a \cdot b(\mathbf{m}) + t_c \cdot \chi'(M_P(\mathbf{m})) : \mathbf{m} \text{ is a mapping}\}.$$

(OGPC) is a rather complex combinatorial optimisation problem and – as we will see later – solving it requires a considerable amount of time. This implies a tradeoff between the time required to (approximately) solve (OGPC) and time that the optimal mapping \mathbf{m}^* saves us per iteration when compared to non-optimal but easily obtainable mapping \mathbf{m}' . The time required for solving (OGPC) is especially important if the grid is dynamically adapted: in this case we must (re)solve (OGPC) for the adapted grid. In other words, each dynamic grid adaption commits us to start a time consuming optimisation algorithm to find a new optimal

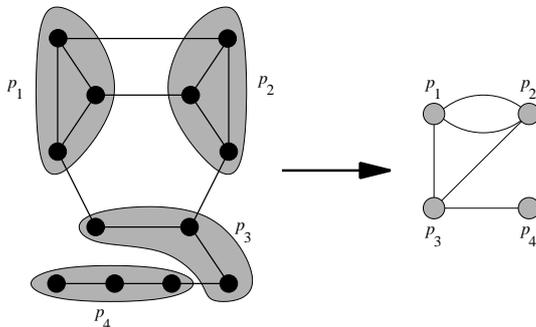


Figure 3.3: The processor multigraph arising from a block-to-processor mapping.

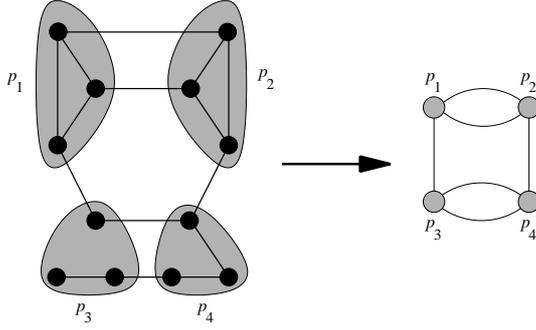


Figure 3.4: Different mappings of the same grid graph lead to different processor multigraphs.

(or at least good) partitioning. This additional overhead for repartitioning grids renders our approach infeasible for algorithms using dynamically adapted grids. We assume in this work that the grid over which the simulation is defined is either not adapted at all or adapted in a predictable fashion that can be taken into account in the initial partitioning. Under this assumption, we need to partition the grid only *once*, namely right before the simulation starts. This allows us to directly trade off the time required to obtain \mathbf{m}^* and the time this mapping saves us per iteration of Algorithm 5.

For ease of exposition in the following chapters we define for a grid graph $G = (V, E)$ and a processor set P

$$\begin{aligned}
\kappa_V &:= \sum_{u \in V} \kappa_u \\
\kappa^*(V, |P|) &:= \min\{b(\mathbf{m}) : \mathbf{m} \text{ maps } V \text{ to } P\} \\
\kappa(U) &:= \sum_{u \in U} \kappa_u \quad \text{for } U \subseteq V \\
\kappa(H) &:= \sum_{u \in V_H} \kappa_u \quad \text{for a subgraph } H = (V_H, E_H) \text{ of } G \\
\kappa(F) &= \sum_{u \in V[F]} \kappa_u \quad \text{for } F \subseteq E \\
\kappa(p) &= \text{number of control volumes mapped to } p \in P.
\end{aligned}$$

κ_V specifies the total number of control volumes in the grid graph. The term $\kappa^*(V, |P|)$ yields the minimal number of control volumes on the most heavily loaded processor in any mapping of V to P . Observe that $\kappa^{*'}(V, |P|) := \lceil \kappa_V / |P| \rceil$ is a trivial lower bound for $\kappa^*(V, |P|)$. If V and P are understood from the context, we simply write κ^* and $\kappa^{*'}$. The terms $\kappa(U)$, $\kappa(H)$ and $\kappa(F)$ all yield the number of control volumes in subsets U , H and F of G . Moreover, we call a bijective function $\sigma : \{0, \dots, |V| - 1\} \rightarrow V$ an *increasing block-ordering* if $i > j$ implies $\kappa_{\sigma(i)} \geq \kappa_{\sigma(j)}$. A block-ordering is *decreasing* if $i > j$ implies $\kappa_{\sigma(i)} \leq \kappa_{\sigma(j)}$.

Observe that in principle we are not limited to block-structured grids: For any grid, we can

define a grid graph by introducing a node for each grid-point and connecting adjacent grid-points by an edge. However, combining multiple grid-points (control volumes) into blocks drastically reduces the complexity of (OGPC) for the respective grid. Only for grid graphs of moderate size, integer programming methods as described in this thesis are feasible. In the future however, powerful computers may allow us to apply the strategies discussed here to any grid graph. Also other hardware architectures than the one above are possible. We will discuss this issue in detail in Section 6.9 below.

3.1 A counter-example for the edge-cut metric

Given that efficient and fast algorithms for grid-partitioning under the edge-cut metric exist, why do we bother with introducing more complex formulations such as (OGPC)? The answer is, that the edge-cut metric miserably fails to correctly account for communication overhead for certain hardware architectures [80, 81], especially the one we described in above. In the following, we give a small example in which the optimal solution to (OGPC) contains even more edges than the optimal solution under the edge-cut metric.

Assume we have four processors and the 2-dimensional block-structured grid depicted in Figure 3.5(a). As usual, we represent each block by a node and connect two nodes by an edge if the corresponding blocks have a common face (Figure 3.5(b) and Figure 3.5(c)). We

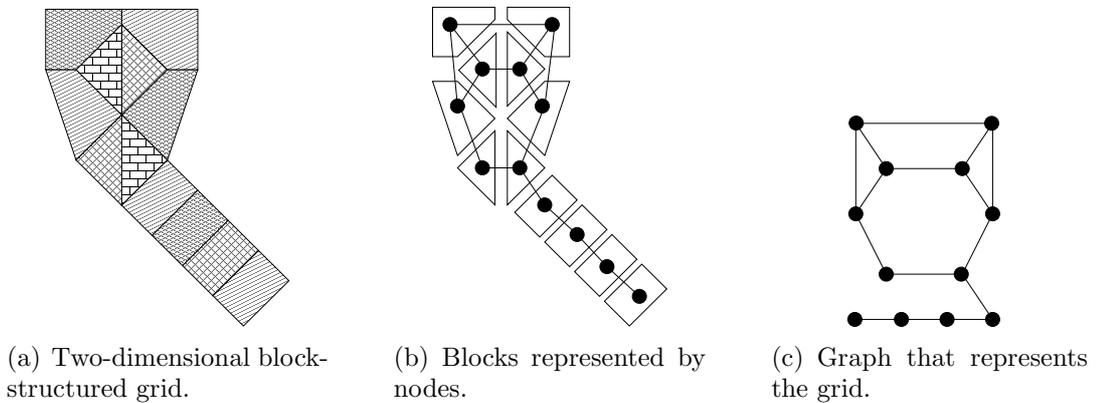


Figure 3.5: A two-dimensional block-structured grid interpreted as graph.

assume that all blocks in the grid have the same computational load and that each processor can bear at most three of them.

Figure 3.6 shows a mapping of nodes to processors that is optimal with respect to the edge-cut metric: Each processor bears exactly three nodes, so the computational load is perfectly balanced. Moreover, we have 5 edges in the multicut, which is minimal in this case.

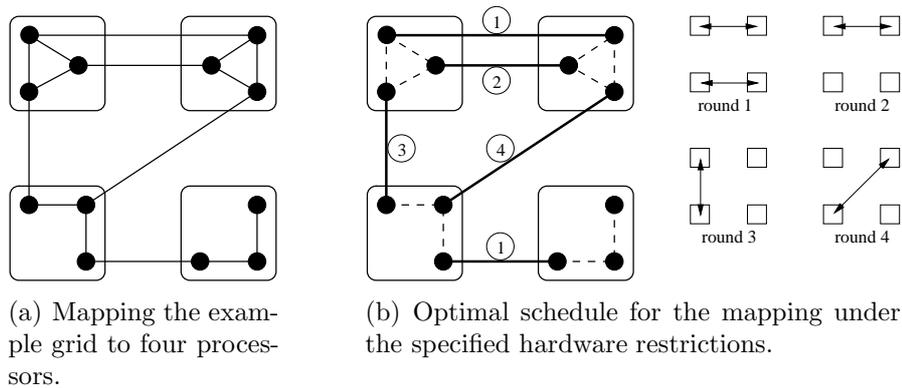


Figure 3.6: Mapping the example grid to four processors: Optimal for edge-cut metric but suboptimal for the specified hardware architecture.

Figure 3.6 also shows an optimal communication schedule for the mapping that is valid under the assumed hardware restrictions: Communication decomposes into 4 rounds and in each round each processor exchanges data with at most one other processor. It can be shown that we cannot do better than 4 rounds with the mapping at hand.

However, if we slightly change the processor mapping, we are able to save one communication round. Look at Figure 3.7. Here we again have three nodes per processor, i. e., the computational load is again perfectly balanced. On the other hand, we are now able to sat-

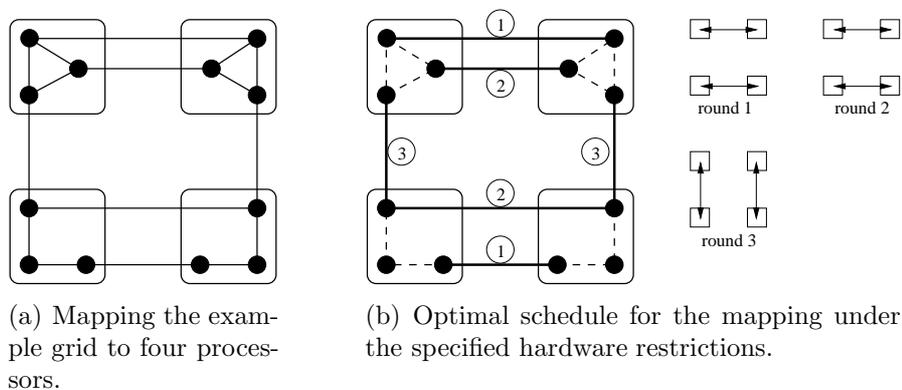


Figure 3.7: Mapping the example grid to four processors: Now optimal for the described hardware architecture.

isfy communication requirements in only 3 rounds. And this although the multicut contains even *more* edges than before!

The above example shows that

1. the edge-cut metric may indeed yield suboptimal partitionings if we assume a hardware

like the one described above and

2. the node-mapping severely influences the quality of the communication schedule.

The first point justifies this work and the second point makes clear that we cannot use a two-stage approach in which we first map and then schedule. Instead we must consider both problems simultaneously and find a good mapping that allows a good schedule as well.

Chapter 4

Related Work

Careful! – we don't want to learn from this!
— *Calvin and Hobbes*

Problem (OGPC) has to the best of our knowledge not yet been studied in the literature. It is however closely related to several graph-partitioning problems. The two subproblems – mapping blocks and finding a minimal communication schedule – that make up (OGPC) are themselves related to the Multiple Knapsack and General Assignment Problems, as well as to colouring problems in graphs. In this chapter, we provide a review of relevant literature concerning these three fields.

4.1 Graph-Partitioning Problems

In order to efficiently execute scientific simulations based on grids on parallel architectures, the underlying grid must be distributed among the available processors such that the computational load on each processor is approximately equal and interprocessor communication is small. For adaptive computations also the effort for repartitioning and redistribution must be minimised, but we do not consider this here. It is well known [144] that this optimisation problem can be phrased as a graph-partitioning (and repartitioning) problem.

The graph-partitioning problem (see [21, 93] and references therein) asks for a partitioning of the nodes of a graph into k partitions, such that the sum of nodes in each partition is equal and the edge-cut is minimal. The input graph may also be weighted in which case the sum of node weights in each partition should be equal and the sum of edge-weights of edges between partitions is to be minimised [47, 93]. Although this problem is \mathcal{NP} -hard to solve in general [63], there are efficient algorithms that work well in practice [82, 83, 144,

105, 99, 11, 10, 12, 13, 53, 100]. The problem is usually considered either in a static or a dynamic context [82]. In the static context [53, 85] only a single partitioning is required and this partitioning is valid for the whole simulation. In the dynamic context [10, 11, 12, 13] however, the grid must be repartitioned and redistributed during the simulation. This means that as opposed to the static model there are additional costs for repartitioning the grid and/or redistributing data over the processors, see e. g., [104] and references therein. In both contexts partitionings are usually determined by a recursive multisection scheme that is based on coordinate bisection, inertial bisection or spectral bi-, quadri- and octasection [82, 86]. Recursive bisection is the most simple and fastest approach and is therefore often used in the dynamic context. The static context allows a little more time for determining a good partitioning and thus renders more complicated multisection approaches feasible. Another way to improve static partitionings is the multilevel paradigm introduced by Hendrickson and Leland [84]. Multilevel algorithms start by contracting (*coarsening*) the grid-graph until a sufficiently simple graph is reached which can be partitioned efficiently. This graph is partitioned and the partitioning is propagated back to the original graph. Optionally, intermediate partitions may be improved by local search algorithms [101, 83, 85, 82].

Apart from heuristic algorithms there is also some work on the combinatorial structure of the related polyhedra as well as on (LP based) algorithms to optimally solve the problem. Due to the nature of these algorithms, they can only be used in the static context were a relative large amount of time is available for solving the optimisation problem.

In [93] Holm and Sørensen considered an integer programming formulation for the graph-partitioning problem on node- and edge-weighted graphs. They paid special attention to the symmetry inherent to their formulation and suggested several preprocessing techniques to remedy the problems incurred by this symmetry. They also analysed the gap between the optimal integral and the optimal fractional solution and found that this gap is always the largest one possible.

The same problem was considered by Ferreira and others [47, 48] but this time the facial structure of the polytope defined by the integer program was under investigation. The authors derived several classes of facet-defining inequalities for the underlying polyhedron and developed an efficient Branch-and-Cut algorithm based on these cuts.

Minimising the edge cut while keeping the load perfectly balanced is often referred to as single-objective and single-constraint problem. As opposed to this, there are also more general instances of the graph-partitioning problem that allow multiple constraints [101, 103, 99] and/or multiple objectives [102, 89]. Single-objective and single-constraint models are usually employed for static simulations while multi-constraint and multi-objective models find application in (adaptive) multi-phase and multi-physics simulations [82, 83, 105, 99].

Even more general is the graph-partitioning problem considered in [87]. Here the authors assume that each vertex in the graph to be partitioned has a desire d_k to be in partition k

and the aim is to maximise the weighted sum of satisfied desires.

4.2 Communication Optimisation

In conjunction with the graph-partitioning problem the traditional assumption is that the communication demands are given by the edges running between partitions and that the communication overhead is proportional to the number of these edges (the size of the edge-cut). However, this does not model hard- and software restrictions with a sufficient accuracy such that it seems not quite sensible to use the edge-cut metric to estimate communication overhead [80, 81]. In [85] Hendrickson and Leland suggest alternate metrics to measure the communication overhead, where each of these metrics is designed for a certain parallel architecture. Hendrickson and Pinar [88] also discussed the problem of finding an optimal partitioning for certain hardware architectures if the partitioning is fixed.

When it comes to data transfer, not only the network connections between processors are limited. In some cases the processor memories are quite small as well and care must be taken how data is exchanged between processors. This additional memory constraint is considered in [88].

Graph colouring formulations are widely used in communication and schedule optimisation (see e. g., [28, 15, 79, 111]). However, most of these formulations are concerned with colouring the nodes of a graph such that adjacent nodes have different colours. As the edge-colouring problem [94], this problem is \mathcal{NP} -hard [63].

From a polyhedral point of view, several formulations of the node-colouring problem are known (see [22] for an overview). Each of these formulations gives rise to a different polytope and all these polytopes were investigated by various authors [22, 38, 27]. In all cases the authors were able to derive the dimension of the polytope as well as several classes of facet-defining inequalities, usually defined by special subgraphs such as holes or cliques. Based on the polyhedral results, Branch-and-Bound-, Branch-and-Cut- and Column-Generation-algorithms [38, 39, 128] were designed that efficiently solve the node-colouring problem and outperform purely combinatoric algorithms (see [39]).

There is also a wide variety of heuristics available for the node-colouring problem. They range from neighbourhood based local search strategies [6, 116], over Tabu-Search techniques [90] to Genetic and Evolutionary Algorithms [62, 28]. Some of these heuristics perform pretty well on certain classes of graphs. However, Rothe [147] proved that the node-colouring problem is \mathcal{NP} -hard on these special classes of graphs as well.

Compared to formulations based on node-colouring problems, edge-colouring formulations are rather rare in communication and schedule optimisation literature. Basically, edge-

colouring a graph is equivalent to node-colouring its linegraph, but this equivalence does not work well in practice: First of all, the linegraph contains usually much more edges than the graph itself. Secondly, the efficiency of node-colouring algorithms is mostly based on certain substructures (cliques for example) of the graph to be coloured and it is not easy to transform information about these substructures to the linegraph or vice versa. Nevertheless, there is some recent research about node-colouring linegraphs [114].

In a series of papers [41, 42, 43] Durand, Jain and Tseytlin define an edge-colouring problem to optimise inter-processor communication for parallel computing in a client-server-like architecture. They also provide some heuristics to solve the problem posed. Also the online-version of the edge-colouring problem has been investigated and it has been proven by Bar-Noy, Motwani and Naor [9, 8] that for this problem the greedy algorithm has performance ratio 2 and that this is optimal.

Another application where edge-colouring formulations are used to optimise communication schedules is data migration on parallel disks [68, 148, 111, 4, 79]. In this problem a set of data items is stored on several disks and some of the items must be moved in a minimal amount of time. As each disk can only be sender or receiver of an item at a time and only participate in the transfer of one item at a time, the required communication can be described by a multigraph that has the disks as nodes and contains an edge for each required transfer. A minimal time transfer schedule is then given by a minimal edge-colouring of this multigraph. Observe that this problem is similar to (OGPC). More applications of the edge-colouring problem can be found in [54].

Polyhedral considerations for edge-colouring problems are even rarer than their applications in communication and schedule optimisation. To the best of our knowledge, the only people to investigate a solution based on polyhedral methods were Nemhauser and Park [131] – apart from some basic considerations [152]. In their paper Nemhauser and Park designed a Branch-and-Cut-and-Price algorithm to solve the edge-colouring problem on simple graphs. Their algorithm worked especially well on 3-regular graphs and could be theoretically extended to k -regular graphs as well.

For a detailed discussion about the edge-colouring problem itself and related literature, please refer to Section 2.2.2 above.

4.3 Multiple Knapsack and General Assignment

As stated earlier, the problem of mapping grid elements to processors while meeting capacity limits can be viewed as a special instance of the Multiple Knapsack or the General Assignment Problem. Since this work is mainly concerned with polyhedral theory and integer programming, we focus on literature targeting these two aspects of Multiple Knapsack

and General Assignment. More detailed literature surveys about algorithmic aspects can be found in [125] (Multiple Knapsack) and [25] (General Assignment).

In the Multiple Knapsack Problem we have a set I of items and a set J of knapsacks. Each item $i \in I$ has a size or weight w_i and a profit p_i and the capacity of a knapsack $j \in J$ is limited to K_j . The aim is to assign items to knapsacks such that the capacity requirements are met and the profit of the assigned items is maximal. In [50] the authors investigated the structure of the polytope arising from an integer programming formulation of the Multiple Knapsack Problem. They found several facets like minimal-cover- and $(1, d)$ -configuration-inequalities that were carried over from the single knapsack problem [161, 146]. They also exhibited several facet-defining inequalities that involve multiple knapsacks and are not generalisations of facets from the single-knapsack case. Based on these facial results, the same authors designed an efficient Branch-and-Cut algorithm that proved well in practice [49, 51]. Other authors considered more restrictive variants of the Multiple Knapsack Problem: In [34] the items assigned to the same knapsack must satisfy an additional colour constraint. Based on [50, 49] the authors (re)proved several facial results and designed an efficient Branch-and-Cut algorithm. Another interesting restriction of the Multiple Knapsack Problem is that for each knapsack we have an additional limit on the *number* of items that can be assigned to it (independent of their weights). The polyhedral aspects of this problem were studied in [36] and several facet-defining inequalities were derived.

The General Assignment Problem is a generalisation of the Multiple Knapsack Problem. Here the weights and profits of items are a function of the knapsacks, i. e., we have weights $w_{i,j}$ and $p_{i,j}$ for each item i and each knapsack j . In most cases, one also wants to assign all items and instead of the profit one considers the cost incurred by an assignment and attempts to minimise that. In order to formulate this problem as integer programming problem, there are two different approaches. In the first and most widespread approach one introduces a binary decision variable for each item/knapsack pair. The polytope arising from this formulation was examined in [73, 72]. Here the authors found several facet-defining inequalities among which we again have the $(1, d)$ -configuration inequalities already known from the Multiple Knapsack Problem. A different approach is taken in [150]. Here the author gives a *disaggregated* formulation of the problem that uses the incidence vector of all feasible assignments and describes a Branch-and-Price algorithm to solve the problem. Other common approaches to solve the problem are to relax the capacity constraints [146] or the requirement that all items must be (uniquely) assigned [35].

Chapter 5

Individual Models

If I have a thousand ideas and only one turns
out to be good, I am satisfied.
— *Alfred Bernhard Nobel*

As stated earlier, (OGPC) combines two different optimisation problems. The first problem is to map a set of blocks to processors such that the maximum number of control volumes on a processor is minimal. The second problem asks for a minimal edge-colouring in a multigraph. In this chapter we will describe integer programming formulations for each of them. To the best of our knowledge, none of the models presented here has been studied in specialised literature explicitly. However, some of the models are very similar to models from the literature and in these cases our proofs are simply adopted from the respective papers.

5.1 Block-Mapping Models

In order to formulate the block-mapping problem as an integer programming model, we assume that we are given a grid graph $G = (V, E)$, block sizes κ_u for $u \in V$ and a processor set P to which we map. Notice that when considering the block-mapping problem as a stand-alone problem we may assume unlimited processor capacities. This is because an optimal solution to the problem will yield a mapping in which the maximum capacity required will be minimal.

Each of the models described here will contain an integer variable b that counts the number of control volumes on the most heavily loaded processor. This variable may in practice be chosen to be continuous, since all block sizes κ_u are integral and so will be b in any optimal solution.

5.1.1 Naive Block-Mapping

Our first model for the block-mapping problem is very similar to Multiple Knapsack and General Assignment models [50, 73, 72, 35]. However, as opposed to these models from the literature our aim is not to maximise profit or minimise cost of mapped blocks. Instead we want to map all blocks and minimise the overall maximum of control volumes mapped to a processor. This difference dramatically changes the structure of the polyhedron associated with the integer programming formulation.

To obtain an integer programming formulation for a grid graph $G = (V, E)$, we introduce binary decision variables

$$x_{u,p} = \begin{cases} 1 & \text{if block } u \text{ is mapped to processor } p \text{ and} \\ 0 & \text{otherwise} \end{cases} \quad \text{for all } (u, p) \in V \times P. \quad (5.1)$$

In other words, we have for each block $u \in V$ one variable $x_{u,p}$ for each of the $|P|$ different target processors to which it may be mapped. We call this modelling strategy *naive* as it naively translates each decision to be taken into a binary variable.

In addition to these binary variables we use variable b that was mentioned before and impose the following constraints:

1. Each block $u \in V$ must be mapped to a processor, i. e., $\sum_{p \in P} x_{u,p} = 1$. For ease of exposition we require only

$$\sum_{p \in P} x_{u,p} \geq 1$$

for all $u \in V$. This means that a block may be mapped to more than one processor. As our objective is to minimise maximum load the following is obvious: if a block u is mapped to multiple processor p_1, \dots, p_k in an optimal solution then we may freely choose any of these processor to handle u . No matter which one we choose, the maximum processor load will not be bigger than the optimal one.

2. The number of control volumes on a processor $p \in P$ is given by $\sum_{u \in V} \kappa_u x_{u,p}$. As we want b to be at least as big as all the processor loads, we add constraint

$$\sum_{u \in V} \kappa_u x_{u,p} \leq b$$

for all $p \in P$. Minimising b then yields that b is equal to the load on the largest processor.

3. As b is at least as big as the biggest processor, minimising b is equivalent to minimising the maximum load of a processor.

Collecting 1 through 3 into an integer programming model yields

(NAVMAP)	minimise b	(5.2a)
	$\sum_{p \in P} x_{u,p} \geq 1 \quad u \in V$	(5.2b)
	$\sum_{u \in V} \kappa_u x_{u,p} \leq b \quad p \in P$	(5.2c)
	x binary.	(5.2d)

Observe that model (NAVMAP) does not depend on the edges of the grid graph, but only on the nodes in G and their respective block sizes.

Let

$$P_{\text{nav}}^{\text{map}}(G) := \text{conv} \left\{ (x, b) \in \{0, 1\}^{V \times P} \times \mathbb{N} : (x, b) \text{ satisfies (5.2b) and (5.2c)} \right\}$$

denote the convex hull of feasible solutions to (NAVMAP).

Since the variant of (NAVMAP) that requires equality in (5.2b) is similar to Multiple Knapsack and General Assignment polyhedra studied in the literature [51, 50, 73, 37], we also consider

$$P_{\text{nav}}^{\text{map},=} (G) := \text{conv} \left\{ (x, b) \in \{0, 1\}^{V \times P} \times \mathbb{N} : (x, b) \text{ feasible for (NAVMAP), } \sum_{p \in P} x_{u,p} = 1 \right\},$$

the polyhedron defined by the convex hull of all feasible solutions to (NAVMAP) that satisfy (5.2b) at equality. Notice that this polyhedron defines a face of $P_{\text{nav}}^{\text{map}}(G)$. Where possible we will analyse both polyhedra simultaneously and will prove statements for either of the two. To simplify notation, we also introduce the vector X_p which is the incidence vector of the feasible solution in which each block is mapped to exactly one processor and all blocks are mapped to processor p .

The single-processor variant of (NAVMAP) was investigated in [121, 122] where Marchand and Wolsey considered¹

$$\text{conv} \{ (y, b) \in \{0, 1\}^n \times \mathbb{R}_+ : \sum_{u \in V} \kappa_u y_u \leq s + b \} \quad (5.3)$$

¹actually they called the real constant b and the single (continuous) variable s

where s is a nonnegative real constant. The authors proved that the polyhedron defined by (5.3) is full-dimensional and that $b \geq 0$, $y_u \geq 0$ and $y_u \leq 1$ are facet-defining for it. Moreover, if $\kappa_v \leq \sum_{u \in V} \kappa_u - s$ for all $v \in V$ then also $\sum_{u \in V} \kappa_u y_u \leq s + b$ is facet-defining. Apart from these facets, the authors also obtained other classes of facets by different lifting approaches.

Dimension and Facets of $P_{\text{nav}}^{\text{map}}(G)$ and $P_{\text{nav}}^{\text{map},=}(G)$

We start the polyhedral investigation of $P_{\text{nav}}^{\text{map}}(G)$ and $P_{\text{nav}}^{\text{map},=}(G)$ by exhibiting their dimensions, assuming that $|P| > 1$.

Theorem 10. *If $G = (V, E)$ is a graph on n nodes, then*

- $P_{\text{nav}}^{\text{map}}(G)$ is full-dimensional, i. e., $\dim(P_{\text{nav}}^{\text{map}}(G)) = n|P| + 1$, and
- $\dim(P_{\text{nav}}^{\text{map},=}(G)) = n|P| + 1 - n$.

Proof. First of all, observe that both polyhedra are subsets of $\mathbb{R}^{n|P|+1}$, thus their dimensions do not exceed $n|P| + 1$.

We start with $P_{\text{nav}}^{\text{map}}(G)$. To show that this polyhedron is full-dimensional we prove that the only equation satisfied by all points $(x, b) \in P_{\text{nav}}^{\text{map}}(G)$ is $0^T x + 0b = 0$ (see Theorem 6). Assume that all points in $P_{\text{nav}}^{\text{map}}(G)$ satisfy an equality $\lambda x + \mu b = \beta$ with $(\lambda, \mu) \neq 0$. Since (X_p, κ_V) and $(X_p, \kappa_V + 1)$ are both feasible solutions we conclude that $\mu = 0$. Moreover, for $p, k \in P$ with $p \neq k$ we have that $(X_p, \kappa_V) \in P_{\text{nav}}^{\text{map}}(G)$ and $(X_p + \hat{x}_{u,k}, \kappa_V) \in P_{\text{nav}}^{\text{map}}(G)$ for all $u \in V$ and thus $\lambda_{u,k} = 0$ for all $u \in V$ and all $k \in P$. Thus $(\lambda, \mu) = 0$ and the only equality satisfied by all points in $P_{\text{nav}}^{\text{map}}(G)$ is $0^T x + 0b = 0$ which proves the claim.

To show that $\dim(P_{\text{nav}}^{\text{map},=}(G)) = n|P| + 1 - n$ we exhibit a minimal equation system of rank n (see Theorem 6). All points in $P_{\text{nav}}^{\text{map},=}(G)$ satisfy $\sum_{p \in P} x_{u,p} = 1$ for all $u \in V$. As these n equations are linearly independent, it is clear that $\dim(P_{\text{nav}}^{\text{map},=}(G)) \leq n|P| + 1 - n$. To see equality we show that the equations $\sum_{p \in P} x_{u,p} = 1$ for $u \in V$ define a minimal equation system for $P_{\text{nav}}^{\text{map},=}(G)$. Suppose that $\lambda x + \mu b = \beta$ is another equation that is satisfied by all points in $P_{\text{nav}}^{\text{map},=}(G)$. Since (X_p, κ_V) and $(X_p, \kappa_V + 1)$ are both contained in $P_{\text{nav}}^{\text{map},=}(G)$, we conclude $\mu = 0$. Moreover, for $u \in V$ and $p, k \in P$ we have $(X_p, \kappa_V) \in P_{\text{nav}}^{\text{map},=}(G)$ and $(X_p - \hat{x}_{u,p} + \hat{x}_{u,k}, \kappa_V) \in P_{\text{nav}}^{\text{map},=}(G)$. Thus $\lambda_{u,p} = \lambda_{u,k}$ for fixed u and $p, k \in P$. Consequently, $\lambda x + \mu b = \beta$ is a linear combination of $\sum_{p \in P} x_{u,p} = 1$ for all $u \in V$ and the claim is proved. \square

We now prove that many of the inequalities that occur in formulation (NAVMAP) define facets.

Theorem 11. *The upper bound inequality*

$$x_{u,p} \leq 1 \quad u \in V, p \in P \quad (5.4)$$

defines a facet of $P_{\text{nav}}^{\text{map}}(G)$. If $|P| = 2$ it also defines a facet of $P_{\text{nav}}^{\text{map},=}(G)$.

If furthermore $|P| \geq 3$ then also the lower bound inequality

$$x_{u,p} \geq 0 \quad u \in V, p \in P \quad (5.5)$$

defines a facet for $P_{\text{nav}}^{\text{map}}(G)$ as well. Inequality (5.5) defines a facet of $P_{\text{nav}}^{\text{map},=}(G)$ if $|P| \geq 2$.

Proof. To show that $x_{u,p} \leq 1$ defines a facet of $P_{\text{nav}}^{\text{map}}(G)$, we show that the face induced by this inequality is not a proper subset of a facet of $P_{\text{nav}}^{\text{map}}(G)$ (see Theorem 8). Fix $u_0 \in V$ and $p_0 \in P$ and let

$$F' := \{(x, b) \in P_{\text{nav}}^{\text{map}}(G) : x_{u_0, p_0} = 1\} \subseteq \{(x, b) \in P_{\text{nav}}^{\text{map}}(G) : \lambda x + \mu b = \beta\} =: F,$$

where F is the facet containing F' . Since $(X_{p_0}, \kappa_V) \in F'$ and $(X_{p_0}, \kappa_V + 1) \in F'$ we immediately get $\mu = 0$. Moreover, since also $(X_{p_0} + \hat{x}_{v,k}, \kappa_V) \in F'$ for all $v \in V$ and $k \neq p_0$ we conclude $\lambda_{v,k} = 0$ if $k \neq p_0$. Finally, we observe that for $k \neq p_0$ the points $(X_k + \hat{x}_{u_0, p_0}, \kappa_V)$ and $(X_k + \hat{x}_{u_0, p_0} + \hat{x}_{v, p_0}, \kappa_V)$ are both contained in F' for $v \neq u_0$ and thus $\lambda_{v, p_0} = 0$ for $v \neq u_0$. So the only entry in λ that is non-zero may be λ_{u_0, p_0} which concludes the proof.

We next show that $x_{u_0, p_0} \leq 1$ is facet-defining for $P_{\text{nav}}^{\text{map},=}(G)$ if $|P| = 2$ by exhibiting $\dim(P_{\text{nav}}^{\text{map},=}(G)) = n(|P| - 1) + 1 = n + 1$ affinely independent points in $P_{\text{nav}}^{\text{map},=}(G)$ (see Theorem 8). Since $|P| = 2$ we have exactly two processors available, say p_0 and p_1 . For a fixed $v_0 \in V$ with $v_0 \neq u_0$ consider the following set of n points that are feasible for $P_{\text{nav}}^{\text{map},=}(G)$:

$$\begin{aligned} X &:= (X_{p_0}, \kappa_V) \\ X_v &:= (X_{p_0} - \hat{x}_{v, p_0} + \hat{x}_{v, p_1}, \kappa_V) \quad \text{for } v \neq u_0, \\ X'_{v_0} &:= (X_{p_0} - \hat{x}_{v_0, p_0} + \hat{x}_{v_0, p_1}, \kappa_V + 1). \end{aligned}$$

Assume there are scalars $a \in \mathbb{R}$, $a_v \in \mathbb{R}$ for $v \neq u_0$ and $a'_{v_0} \in \mathbb{R}$ for $v_0 \neq u_0$ such that

$$aX + \sum_{v \neq u_0} a_v X_v + a'_{v_0} X'_{v_0} = 0 \quad \text{and} \quad (5.6)$$

$$\sum_{v \neq u_0} a_v + a'_{v_0} = 0. \quad (5.7)$$

Looking at the entries for x_{v, p_1} for $v \neq v_0, u_0$ in (5.6) immediately yields $a_v = 0$ in these cases. The entries for variable b then yield $\kappa_V a + \kappa_V a_{v_0} + (\kappa_V + 1) a'_{v_0} = 0$, which — together

with (5.7) — implies that $a'_{v_0} = 0$. The entries for x_{v_0, p_1} in (5.6) yield $a_{v_0} + a'_{v_0} = 0$. Thus $a_{v_0} = 0$ (and by (5.7) also $a = 0$) and the claim is proved.

To show that $x_{u, p} \geq 0$ is facet-defining for $P_{\text{nav}}^{\text{map}}(G)$ if $|P| \geq 3$ fix $u_0 \in V$ and $p_0 \in P$ and assume that

$$F' := \{(x, b) \in P_{\text{nav}}^{\text{map}}(G) : x_{u_0, p_0} = 0\} \subseteq \{(x, b) \in P_{\text{nav}}^{\text{map}}(G) : \lambda x + \mu b = \beta\} =: F,$$

where F is the facet containing F' . For $k \neq p_0$ points (X_k, κ_V) and $(X_k, \kappa_V + 1)$ are both contained in F' , showing that $\mu = 0$. If $v \neq u_0$ then $(X_k, \kappa_V) \in F'$ and $(X_k + \hat{x}_{v, p_0}, \kappa_V) \in F'$, thus $\lambda_{v, p_0} = 0$ for $v \neq u_0$. If not only $v \neq u_0$, but also $k \neq p_0$ we have $(X_{p_0} - \hat{x}_{u_0, p_0} + \hat{x}_{u_0, k}, \kappa_V) \in F'$ and $(X_{p_0} - \hat{x}_{u_0, p_0} + \hat{x}_{u_0, k} + \hat{x}_{v, k}, \kappa_V) \in F'$, showing that $\lambda_{v, k} = 0$ for $v \neq u_0$ and $k \neq p_0$. As $|P| \geq 3$ we can choose $k_1, k_2 \in P$ with $k_i \neq p_0$. We get $(X_{k_1}, \kappa_V) \in F'$ and $(X_{k_1} + \hat{x}_{u_0, k_2}, \kappa_V) \in F'$. This yields $\lambda_{u_0, k} = 0$ for $k \neq p_0$ and the claim is proved.

We see that $x_{u_0, p_0} = 0$ defines a facet of $P_{\text{nav}}^{\text{map}, =}(G)$ by proving that the polyhedron

$$F' := \{(x, b) \in P_{\text{nav}}^{\text{map}, =}(G) : x_{u_0, p_0} = 0\}$$

has dimension $n|P| - n$. We do so by showing that the system of $n + 1$ linear equations

$$\sum_{p \in P} x_{u, p} = 1 \quad u \in V, \quad (5.8)$$

$$x_{u_0, p_0} = 0. \quad (5.9)$$

is a minimal equation system for F' . Obviously, these equations are linearly independent if $|P| \geq 2$ (the left-hand side matrix is easily transformed into upper triangular form by column-permutations). Assume that there is another linear equation $\lambda x + \mu b = \beta$ that is satisfied by all points in F' . For $k \neq p_0$ we have $(X_k, \kappa_V) \in F'$ and $(X_k, \kappa_V + 1) \in F'$ and conclude $\mu = 0$. If furthermore $v \neq u_0$, point $(X_k - \hat{x}_{v, k} + \hat{x}_{v, p_0}, \kappa_V)$ is also contained in F' , showing that $\lambda_{v, k} = \lambda_{v, p_0}$ for $v \neq u_0$.

If $|P| = 2$, then obviously

$$\begin{pmatrix} \lambda_{u_0, p_0} \\ \lambda_{u_0, p_1} \end{pmatrix} = \lambda_{u_0, p_1} \begin{pmatrix} 1 \\ 1 \end{pmatrix} + (\lambda_{u_0, p_0} - \lambda_{u_0, p_1}) \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

and $\lambda x + \mu b = \beta$ is a linear combination of equations (5.8) and (5.9).

If $|P| \geq 3$ we may choose $k_1, k_2 \in P$ with $k_1, k_2 \neq p_0$. Points (X_{k_1}, κ_V) and $(X_{k_1} - \hat{x}_{u_0, k_1} + \hat{x}_{u_0, k_2}, \kappa_V)$ are then both contained in F' , showing that $\lambda_{u_0, k_1} = \lambda_{u_0, k_2}$ for $k_1, k_2 \in P$ with $k_1, k_2 \neq p_0$. Thus

$$\begin{pmatrix} \lambda_{u_0, p_0} \\ \lambda_{u_0, p_1} \\ \vdots \\ \lambda_{u_0, p_{|P|-1}} \end{pmatrix} = \lambda_{u_0, p_1} \mathbf{1} + (\lambda_{u_0, p_0} - \lambda_{u_0, p_1}) \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix}$$

and again $\lambda x + \mu b = \beta$ is a linear combination of equations (5.8) and (5.9). \square

Theorem 12. *The mapping inequality*

$$\sum_{p \in P} x_{u,p} \geq 1 \quad u \in V \quad (5.10)$$

defines a facet of $P_{\text{nav}}^{\text{map}}(G)$.

Proof. Fix some $u_0 \in V$ and let

$$F' := \left\{ (x, b) \in P_{\text{nav}}^{\text{map}}(G) : \sum_{p \in P} x_{u_0,p} = 1 \right\} \subseteq \{(x, b) \in P_{\text{nav}}^{\text{map}}(G) : \lambda x + \mu b = \beta\} =: F,$$

where F is the facet containing F' . Since $(X_p, \kappa_V) \in F'$ and $(X_p, \kappa_V + 1) \in F'$ for any $p \in P$, we conclude $\mu = 0$. Moreover, for $v \neq u_0$ and $k \neq p$ the point $(X_p + \hat{x}_{v,k}, \kappa_V)$ is also contained in F' , hence $\lambda_{v,k} = 0$ for $v \neq u_0$ and $k \neq p$. As p was arbitrary, we conclude $\lambda_{v,k} = 0$ for all $k \in P$ and $v \neq u_0$.

To show that $\lambda_{u_0,p} = \lambda_{u_0,k}$ for all $p, k \in P$ observe that $(X_p, \kappa_V) \in F'$ and $(X_p - \hat{x}_{u_0,p} + \hat{x}_{u_0,k}, \kappa_V) \in F'$ for $p, k \in P$. Thus $\lambda_{u_0,p} = \lambda_{u_0,k}$ for all $p, k \in P$ and the claim is proved. \square

Notice that (5.10) can never define a facet of $P_{\text{nav}}^{\text{map},=}(G)$ since it is satisfied at equality by all points in $P_{\text{nav}}^{\text{map},=}(G)$.

Theorem 13. *The capacity inequality*

$$\sum_{u \in V} \kappa_u x_{u,p} \leq b \quad p \in P \quad (5.11)$$

defines a facet of $P_{\text{nav}}^{\text{map}}(G)$ if

$$\kappa_u \leq \sum_{v \neq u} \kappa_v \quad (5.12)$$

for all $u \in V$.

Proof. Fix a processor $p_0 \in P$ and let

$$F' := \left\{ (x, b) \in P_{\text{nav}}^{\text{map}}(G) : \sum_{u \in V} \kappa_u x_{u,p_0} - b = 0 \right\} \subseteq \{(x, b) \in P_{\text{nav}}^{\text{map}}(G) : \lambda x - \mu b = \beta\} =: F,$$

where F is the facet containing F' . Since $(X_{p_0}, \kappa_V) \in F'$ and $(X_{p_0} + \hat{x}_{u,k}, \kappa_V) \in F'$ for $k \neq p_0$ and $u \in V$ we have $\lambda_{u,k} = 0$ unless $k = p_0$.

By (5.12) we have that also $(X_{p_0} - \hat{x}_{v,p_0} + \hat{x}_{v,k}, \kappa_V - \kappa_v) \in F'$ for $v \in V$ and $k \neq p_0$. Together with $(X_{p_0}, \kappa_V) \in F'$ this yields $\lambda_{v,p_0} = -\kappa_v \mu$ and the claim is proved.

Notice that if (5.12) is not satisfied, then there is exactly one block $u_0 \in V$ that satisfies $\kappa_{u_0} > \sum_{v \neq u_0} \kappa_v$. In this case an optimal solution to (NAVMAP) is easily constructed: Map u_0 to an arbitrary processor p_0 and map all other blocks to arbitrary blocks different from p_0 . \square

Further Valid Inequalities

We conclude polyhedral analysis by presenting several valid inequalities that are useful for actually solving instances of (NAVMAP), but for which we have not been able to prove or disprove whether they are facet-defining. Although we prove validity for $P_{\text{nav}}^{\text{map},=}(G)$ only in most cases, many of the inequalities are also valid for $P_{\text{nav}}^{\text{map}}(G)$.

In some cases it is possible to determine an upper limit N on the number of blocks that may be mapped to a processor in an optimal solution. If we have such a limit, then

$$\sum_{u \in V} x_{u,p} \leq N \quad p \in P \quad (5.13)$$

is obviously valid for $P_{\text{nav}}^{\text{map},=}(G)$. We will see in Section 7.3 how upper bounds N can be determined in practice.

As opposed to (5.13) the remaining valid inequalities in this section all involve variable b as well. Let us begin with an inequality that is valid if we have one block that is much bigger than all other blocks.

Theorem 14. *Let $\kappa_{\max} := \max_{u \in V} \kappa_u$ and $d := \kappa^* - \kappa_{\max}$. If $\kappa^* \leq 2\kappa_{\max}$, then inequality*

$$\sum_{u \in V} (\kappa_u - d) \cdot x_{u,p} + 2d \leq b \quad p \in P \quad (5.14)$$

is valid for $P_{\text{nav}}^{\text{map},=}(G)$.

Proof. For a feasible solution (x', b') and a fixed processor p_0 let $U := \{u \in V : x'_{u,p_0} = 1\}$ be the set of nodes that are mapped to p_0 . If $|U| \geq 2$, then the left-hand side of (5.14) is no bigger than $\sum_{u \in U} \kappa_u$ and the inequality is satisfied.

Assume $|U| = 0$, i. e., no nodes are mapped to processor p_0 . The inequality then reads $2d \leq b$. We have $2d = 2\kappa^* - 2\kappa_{\max} \leq \kappa^*$ and (5.14) is satisfied.

If now $|U| = 1$, then $U = \{u_0\}$ for some $u_0 \in U$ and (5.14) becomes $\kappa_{u_0} + d \leq b'$. As $\kappa_{u_0} \leq \kappa_{\max}$ we have

$$b' \geq \kappa^* = \kappa_{\max} + d \geq \kappa_{u_0} + d$$

and (5.14) is satisfied. \square

For the next few inequalities we use subsets U of blocks and P' of processors such that the instance of (NAVMAP) that is defined by U and P' yields an objective function value $\kappa^*(U, |P'|)$ that exceeds κ^* .

Theorem 15. *Let $U \subseteq V$ and $P' \subset P$ such that $d := \kappa^*(U, |P'|) - \kappa^* > 0$. Then*

$$\kappa^*(U, |P'|) \leq b + d \sum_{u \in U} \sum_{p \in P \setminus P'} x_{u,p} \quad (5.15)$$

is valid for $P_{\text{nav}}^{\text{map},=}(G)$.

Proof. If no block from U is mapped to a processor in $P \setminus P'$, then all blocks in U are mapped to P' and we have $b \geq \kappa^*(U, |P'|)$. If at least one block $u_0 \in U$ is mapped to $P \setminus P'$, inequality (5.15) turns into

$$\kappa^* \leq b + d \sum_{\substack{u \in U \\ u \neq u_0}} \sum_{p \in P \setminus P'} x_{u,p}$$

which is obviously valid for $P_{\text{nav}}^{\text{map},=}(G)$. \square

Inequality (5.15) states that at least one block from U must be mapped to a processor not in P' , otherwise the most heavily loaded processor will bear at least $\kappa^*(U, |P'|)$ control volumes. This statement can sometimes be strengthened. Assume we have sets $U \subseteq V$ and $P' \subset P$ such that $\kappa^*(U, |P'|) > \kappa^*$. If there is $k \in \mathbb{N}$ with $k > 0$ such that $\kappa^*(U \setminus U_k, |P'|) \geq \kappa(U, |P'|)$ for all k -element subsets $U_k \subseteq U$, then obviously

$$\kappa^*(U, |P'|) \leq b + \frac{d}{k} \sum_{u \in U} \sum_{p \in P \setminus P'} x_{u,p} \quad (5.16)$$

is valid for $P_{\text{nav}}^{\text{map},=}(G)$. Essentially this strengthened inequality specifies that at least k blocks from U must be mapped to a processor not in P' before the value of b may drop below $\kappa^*(U, |P'|)$.

In the following classes of inequalities we consider ordered subsets of blocks that are ordered by either increasing or decreasing block size and derive valid inequalities from them.

Theorem 16. *Let $U \subseteq V$ and $\mathfrak{o} : \{0, \dots, |U| - 1\} \rightarrow U$ be a decreasing block-ordering. Moreover, define*

$$\begin{aligned} \kappa(N) &:= \sum_{j=0}^N \kappa_{\mathfrak{o}(j)} \quad \text{and} \\ k &:= \min_{i=0, \dots, |U|-1} \frac{\max\{\kappa(i), \kappa^*\} + i\kappa^* - \kappa(i)}{i+1}. \end{aligned}$$

Then

$$\sum_{u \in U} (\kappa_u + k - \kappa^*) x_{u,p} + k \leq b \quad (5.17)$$

is valid for $P_{nav}^{map,=}(G)$ for all $p \in P$.

Proof. For a feasible solution (x', b') let $U' := \{u \in U : x'_{u,p} = 1\}$ denote the set of blocks that are mapped to processor p . We obtain

$$\begin{aligned} \sum_{u \in U'} (\kappa_u + k - \kappa^*) x'_{u,p} + k &\leq \sum_{u \in U'} \kappa_u + (|U'| + 1)k - |U'| \kappa^* \\ &\leq \sum_{u \in U'} \kappa_u + \max\{\kappa(|U'|), \kappa^*\} + |U'| \kappa^* - \kappa(|U'|) - |U'| \kappa^* \\ &\leq \sum_{u \in U'} \kappa_u + \max\{\kappa(|U'|), \kappa^*\} - \kappa(|U'|). \end{aligned} \quad (5.18)$$

If $\max\{\kappa(|U'|), \kappa^*\} = \kappa(|U'|)$ then (5.18) is equal to $\sum_{u \in U'} \kappa_u$ and inequality (5.17) is satisfied. Otherwise we have $\max\{\kappa(|U'|), \kappa^*\} = \kappa^*$. Since $\sum_{u \in U'} \kappa_u \leq \kappa(|U'|)$ this implies that (5.18) is at most κ^* and inequality (5.17) is satisfied. \square

Different inequalities arise if we consider sets of blocks that are ordered by increasing rather than decreasing block size.

Theorem 17. Let $\sigma : \{0, \dots, n-1\} \rightarrow V$ be an increasing block-ordering and set $k := \min\{l \in \mathbb{N} : \sum_{i=0}^l \kappa_{\sigma(i)} > \kappa^*\}$, $U := \{\sigma(l) : l = 0, \dots, k\}$ and $d := \kappa(U) - \kappa^*$. Then inequality

$$\kappa^* - kd + \sum_{u \in V} dx_{u,p} \leq b \quad p \in P \quad (5.19)$$

is valid for $P_{nav}^{map,=}(G)$.

Proof. For the proof assume that (x', b') is a feasible solution and observe that $d \leq \kappa_{\sigma(j)}$ for $j \geq k$. Let $U' := \{u \in V : x'_{u,p} = 1\}$ denote the set of blocks that are mapped processor p . To show validity of (5.19) we distinguish two different cases:

- $|U'| < |U|$. In this case, the left-hand side of (5.19) is at most κ^* and the inequality is satisfied.
- $|U'| \geq |U|$. Let U_1 denote the $k+1$ smallest blocks in U' and set $U_2 = U' \setminus U_1$.

Observing that $U_2 \cap U' = \emptyset$ we obtain for the left-hand side of (5.19)

$$\begin{aligned} \kappa^* - kd + \sum_{u \in V} dx_{u,p}^* &= \kappa^* - kd + \sum_{u \in U_1} d + \sum_{u \in U_2} d \\ &= \kappa^* + d + \sum_{u \in U_2} d \\ &\leq \kappa(U) + \sum_{u \in U_2} \kappa_u. \end{aligned}$$

As b' must be at least as big as this term inequality (5.19) is satisfied. □

In some cases, we can further strengthen inequality (5.19). Set $k = \min\{l \in \{0, \dots, n-1\} : \sum_{i=0}^l \kappa_{\mathfrak{o}(i)} > \kappa^*\}$, $U := \{\mathfrak{o}(l) : l = 0, \dots, k\}$ and $d := \kappa(U) - \kappa^*$ as in Theorem 17. Moreover, define $d' := \max\{d, \kappa_u / \kappa_{\mathfrak{o}(k)}\}$. It is then obvious, that

$$\kappa^* - kd + \sum_{u \in V} d' x_{u,p} \leq b \tag{5.20}$$

is valid for $P_{\text{nav}}^{\text{map},=}(G)$ (observe that the term kd still uses the old value d). If there are nodes $u \in V$ with $\kappa_u > \kappa_{\mathfrak{s}(k)}$, then the coefficients for the respective $x_{u,p}$ -variables in (5.20) are larger than the ones in (5.19), thus yielding a slightly stronger inequality.

The last valid inequality in this section is a more general version of (5.19) and a direct adaption of the facet-defining inequality (30) in [121].

Theorem 18. *Let $U \subseteq V$ such that $\sum_{u \in U} \kappa_u = \kappa^* + \lambda$ for some $\lambda > 0$ and $\sum_{u \in U-v} \kappa_u < \kappa^*$. Moreover set $\tilde{U} := \{u \in U : \kappa_u > \lambda\}$ (observe $v \in \tilde{U}$). Then inequality*

$$\sum_{u \in \tilde{U}} \lambda x_{u,p} + \sum_{u \in U \setminus \tilde{U}} \kappa_u x_{u,p} + \kappa^* \leq (|\tilde{U}| - 1)\lambda + \sum_{u \in U \setminus \tilde{U}} \kappa_u + b \quad p \in P \tag{5.21}$$

is valid for $P_{\text{nav}}^{\text{map},=}(G)$.

Proof. For the proof we rewrite (5.21) as

$$\kappa^* + \sum_{u \in \tilde{U}} \lambda x_{u,p} - (|\tilde{U}| - 1)\lambda + \sum_{u \in U \setminus \tilde{U}} \kappa_u x_{u,p} - \sum_{u \in U \setminus \tilde{U}} \kappa_u \leq b. \tag{5.22}$$

For a feasible solution (x', b') define $k := |\{u \in \tilde{U} : x'_{u,p}\}|$. Unless $k = |\tilde{U}|$ the left-hand side of (5.22) does not exceed κ^* and the inequality is therefore valid.

Assume $k = |\tilde{U}|$ and $x'_{u,p} = 1$ for $u \in U \setminus \tilde{U}$. If we can prove validity of (5.22) in this case then validity is proved in general. We have

$$\begin{aligned} \kappa^* + \sum_{u \in \tilde{U}} \lambda x'_{u,p} - (|\tilde{U}| - 1)\lambda + \sum_{u \in U \setminus \tilde{U}} \kappa_u x'_{u,p} - \sum_{u \in U \setminus \tilde{U}} \kappa_u &= \kappa^* + \lambda \\ &= \sum_{u \in U} \kappa_u \\ &\leq b' \end{aligned}$$

where the last inequality holds since all nodes from U are mapped to the same processor p . \square

5.1.2 Representative Block-Mapping

One drawback of (NAVMAP) is its inherent symmetry: given a feasible solution to (NAVMAP), any permutation on the processor set yields another solution with the same objective function value. So for each feasible solution, there are at least $\mathcal{O}(|P|!)$ equivalent solutions. This unnecessarily blows up the Branch-and-Cut tree and decreases performance of Branch-and-Cut algorithms. We will now describe a model that avoids symmetry inherent to the formulation and will discuss symmetry issues in more detail in Section 6.7.

The block-mapping formulation presented in this section is new and has – to the best of our knowledge – not yet been discussed in the literature. It is based on ideas from [22] and shows that the idea of using representatives in order to eliminate model-intrinsic symmetries is not limited to colouring problems, but can also be applied to problems like Multiple Knapsack or General Assignment.

The idea of representative models as in [22] is to represent certain sets by a single element contained in the sets. In our case we will represent the set of blocks that are mapped to the same processor by *one single* block from this set. In other words, if V_p is the set of blocks mapped to processor p , then each node $u \in V_p$ either is the representative of V_p or is represented by another node $v \in V_p$. For a formulation as an integer program we introduce binary decision variables

$$x_{r,u} = \begin{cases} 1 & \text{if } r \text{ represents } u \text{ and} \\ 0 & \text{otherwise} \end{cases} \quad \text{for all } (r, u) \in V \times V.$$

If a block $u \in V$ represents itself (i. e., $x_{u,u} = 1$), then we say that u is the representative of all nodes mapped to the same processor as u . These variables are now independent of the actual numbering of processors in P , so all symmetry issues arising from permutations of the processor set are remedied. However, we have introduced another kind of symmetry: If

V_p denotes the set of blocks mapped to p then we may choose an *arbitrary* block $u \in V_p$ as representative for V_p . So given a feasible solution we can generate an equivalent solution by simply exchanging the representative for V_p . In order to handle this issue, we may require that a block $u \in V$ can never represent a block $v \in V$ that has a smaller index. More explicitly, we set²

$$x_{r,u} = 0 \quad r \in V, u < r. \quad (5.23)$$

This allows *exactly one* representative for each of the sets V_p and therefore solve the symmetry problems discussed above.

To obtain an integer programming formulation for (OGPC) we establish the following constraints:

1. Each block $u \in V$ either represents itself or is represented by another block, i. e.,

$$\sum_{r \in V} x_{r,u} \geq 1$$

for all $u \in V$.

2. A block $u \in V$ can be represented by block $v \in V$ only if v is the representative of a set of blocks mapped to the same processor. This implies

$$x_{r,u} \leq x_{r,r}$$

for all $r \in V$ and $u \neq r$.

3. According to our interpretation, each block that represents itself represents the set of blocks mapped to a certain processor. So the number of processors in the mapping is given by $\sum_{r \in V} x_{r,r}$ and we require

$$\sum_{r \in V} x_{r,r} \leq |P|$$

(the objective function will imply equality here).

4. If $r \in V$ represents a set of blocks mapped to a common processor, then $\sum_{u \in V} \kappa_u x_{r,u}$ yields the number of control volumes that are mapped to this processor. To make b at least as big as the number of control volumes on the most heavily loaded processor we therefore require

$$\sum_{u \in V} \kappa_u x_{r,u} \leq b$$

for all $r \in V$.

²Though simple, this idea was interestingly not described in [22].

Given a grid graph $G = (V, E)$ the above ideas yield

(REPMAP)	minimise b	(5.24a)
	$\sum_{r \in V} x_{r,u} \geq 1 \quad u \in V$	(5.24b)
	$x_{r,u} \leq x_{r,r} \quad (r, u) \in V \times V, r \neq u$	(5.24c)
	$\sum_{u \in V} \kappa_u \cdot x_{r,u} \leq b \quad r \in V$	(5.24d)
	$\sum_{u \in V} x_{u,u} \leq P $	(5.24e)
	x binary.	(5.24f)

As for (NAVMAP), model (REPMAP) does not depend on the edge set of G . Let

$$P_{\text{rep}}^{\text{map}}(G) := \text{conv} \left\{ (x, b) \in \{0, 1\}^{V \times V} \times \mathbb{N} : (x, b) \text{ satisfies (5.24b) through (5.24e)} \right\}$$

denote the polyhedron defined by the convex hull of feasible solutions to (REPMAP) (notice that we do not impose (5.23) here).

Dimension and Facets of $P_{\text{rep}}^{\text{map}}(G)$

Before we start proving several facts about $P_{\text{rep}}^{\text{map}}(G)$ let us define some vectors that come in handy in the proofs:

$$\begin{aligned} X_r &:= \sum_{u \in V} \hat{x}_{r,u} & r \in V \\ X_r^u &:= \sum_{u \in V} \hat{x}_{r,u} + \hat{x}_{u,u} & r \in V, u \neq r \\ X_r^{u,v} &:= \sum_{u \in V} \hat{x}_{r,u} + \hat{x}_{u,u} + \hat{x}_{u,v} & r \in V, u \neq r, v \neq u. \end{aligned}$$

In X_r each node $u \in V$ is represented by r , i. e., all nodes are mapped to the same processor. The same happens in X_r^u but here node u additionally represents itself, so two processors are used. In $X_r^{u,v}$ each node is represented by r and u and v are additionally represented by u . Notice that all three vectors define feasible points of $P_{\text{rep}}^{\text{map}}(G)$ as long as $|V| \geq 2$ and $|P| \geq 2$. In the following we will assume $|V| \geq 3$ and $|P| \geq 2$, otherwise the block-mapping problem would be trivial.

Theorem 19. $P_{\text{rep}}^{\text{map}}(G)$ is full-dimensional, i. e., $\dim(P_{\text{rep}}^{\text{map}}(G)) = n^2 + 1$.

Proof. Assume $\dim(P_{\text{rep}}^{\text{map}}(G)) < n^2 + 1$. Then there exists an equality

$$\lambda x + \mu b = \beta \tag{5.25}$$

that is satisfied by all points $(x, b) \in P_{\text{rep}}^{\text{map}}(G)$.

For $r \in V$ and $u \neq r$ the vectors (X_r, κ_V) and (X_r^u, κ_V) are both contained in $P_{\text{rep}}^{\text{map}}(G)$. Plugging these two points into (5.25) yields $\lambda_{u,u} = 0$. Moreover, the points $(X_r^{u,v}, \kappa_V)$ are also contained in $P_{\text{rep}}^{\text{map}}(G)$. Together with (X_r^u, κ_V) being in $P_{\text{rep}}^{\text{map}}(G)$ this yields $\lambda_{u,v} = 0$ for $u, v \in U$ with $u \neq v$. Finally the points (X_r, κ_V) and $(X_r, \kappa_V + 1)$ are contained in $P_{\text{rep}}^{\text{map}}(G)$ for all $r \in V$, hence $\mu = 0$.

So $\lambda = \mu = 0$ and the only equality satisfied by all points in $P_{\text{rep}}^{\text{map}}(G)$ simultaneously is the trivial one: $0^T(x, b) = 0$.

Since the points in $P_{\text{rep}}^{\text{map}}(G)$ do not satisfy a non-trivial equation simultaneously the polytope $P_{\text{rep}}^{\text{map}}(G)$ is full-dimensional. \square

We proceed by showing that nearly all inequalities defined in (REPMAP) give facets of $P_{\text{rep}}^{\text{map}}(G)$. Let us start with the bounding inequalities for x .

Theorem 20. *The bounding inequalities*

$$x_{u,u} \leq 1 \quad u \in V \tag{5.26}$$

$$x_{r,u} \geq 0 \quad r \neq u, r, u \in V \tag{5.27}$$

define facets of $P_{\text{rep}}^{\text{map}}(G)$.

Proof. We begin with (5.26) and fix $u_0 \in V$. Assume that

$$F' := \{(x, b) \in P_{\text{rep}}^{\text{map}}(G) : x_{u_0, u_0} = 1\} \subseteq \{(x, b) \in P_{\text{rep}}^{\text{map}}(G) : \lambda x + \mu b = \beta\} =: F,$$

where F is the facet containing F' . We will show that $\lambda_{u,v} = 0$ unless $(u, v) = (u_0, u_0)$ and $\mu = 0$, thus proving that $F' = F$. The vectors (X_{u_0}, κ_V) and $(X_{u_0}^u, \kappa_V)$ are in F' if $u \neq u_0$. Thus $\lambda_{u,u} = 0$ unless $u = u_0$. Moreover, since $(X_{u_0}^u, \kappa_V)$ and $(X_{u_0}^{u,v}, \kappa_V)$ are in F' for $u, v \in V$ with $u \neq u_0$ and $v \neq u$ we get $\lambda_{u,v} = 0$ for $u \neq u_0$ and $v \neq u$. For $\lambda_{u_0, u}$ with $u \neq u_0$ observe that $(X_u^{u_0}, \kappa_V) \in F'$ and $(X_u^{u_0, u}, \kappa_V) \in F'$ and thus $\lambda_{u_0, u} = 0$ if $u \neq u_0$. Finally we have $(X_{u_0}, \kappa_V) \in F'$ and $(X_{u_0}, \kappa_V + 1) \in F'$, thus $\mu = 0$ which concludes the proof.

For the proof of (5.27) let $r_0, u_0 \in V$ with $r_0 \neq u_0$. Moreover, let $r \in V \setminus \{r_0, u_0\}$ and assume

$$F' := \{(x, b) \in P_{\text{rep}}^{\text{map}}(G) : x_{r_0, u_0} = 0\} \subseteq \{(x, b) \in P_{\text{rep}}^{\text{map}}(G) : \lambda x + \mu b = \beta\} =: F,$$

where again F is the facet containing F' . Since $(X_r, \kappa_V) \in F'$ and $(X_r, \kappa_V + 1) \in F'$ we conclude that $\mu = 0$. For $v \in V$ with $v \neq r, r_0$ we get that $(X_r, \kappa_V) \in F'$ as well as $(X_r^v, \kappa_V) \in F'$, showing that $\lambda_{v,v} = 0$. Also $(X_v, \kappa_V) \in F'$ and $(X_v^r, \kappa_V) \in F'$ as well as $(X_v^{r_0}, \kappa_V) \in F'$ and we get $\lambda_{r,r} = \lambda_{r_0,r_0} = 0$. For $(u, v) \neq (r_0, u_0)$ and $u \neq r, v$ we have that $(X_r^u, \kappa_V) \in F'$ and $(X_r^{u,v}, \kappa_V) \in F'$, proving that $\lambda_{u,v} = 0$ if $(u, v) \neq (r_0, u_0)$ and $u \neq r$. To see that $\lambda_{r,u} = 0$ for $u \neq r$ observe that $(X_{r_0}^{r,u_0} - \hat{x}_{r_0,u_0}, \kappa_V) \in F'$ and $(X_{r_0}^{r,u_0} - \hat{x}_{r_0,u_0} + \hat{x}_{r,u}, \kappa_V) \in F'$ for all $u \neq r$. This leaves λ_{r_0,u_0} as only non-zero coefficient of (λ, μ) , thus proving the theorem. \square

Not only the bounding inequalities but also the requirement that each node be mapped to (at least) one processor defines a facet:

Theorem 21. *The mapping inequalities*

$$\sum_{r \in V} x_{r,u} \geq 1 \quad u \in V \quad (5.28)$$

define facets of $P_{\text{rep}}^{\text{map}}(G)$.

Proof. Let $u_0 \in V$ and

$$F' := \left\{ (x, b) \in P_{\text{rep}}^{\text{map}}(G) : \sum_{r \in V} x_{r,u_0} = 1 \right\} \subseteq \{ (x, b) \in P_{\text{rep}}^{\text{map}}(G) : \lambda x + \mu b = \beta \} =: F$$

where F is the facet containing F' . Since $(X_r, \kappa_V) \in F'$ and $(X_r, \kappa_V + 1) \in F'$ for any $r \in V$ we conclude $\mu = 0$. For $r_1 \neq r_2$ with $r_1, r_2 \neq u_0$ (remember that we assume $|V| \geq 3$) we have that $(X_{r_1}, \kappa_V) \in F'$ and $(X_{r_1}^{r_2}, \kappa_V) \in F'$, showing that $\lambda_{r,r} = 0$ for all $r \neq u_0$. To see that $\lambda_{r,u} = 0$ for $u \neq u_0$ and r arbitrary, observe that $(X_{r_1}^{r_2}, \kappa_V) \in F'$ and $(X_{r_1}^{r_2,u}, \kappa_V) \in F'$ for $u \neq u_0$ and $r_1 \neq r_2$ with $r_1, r_2 \neq u$. Finally, for $r \neq u_0$ we have $(X_r, \kappa_V) \in F'$ and $(X_r^{u_0} - \hat{x}_{r,u_0}, \kappa_V) \in F'$, implying that $\lambda_{r,u_0} = \lambda_{u_0,u_0}$. This proves our claim. \square

Theorem 22. *The representative inequalities*

$$x_{r,u} \leq x_{r,r} \quad r \neq u, r, u \in V \quad (5.29)$$

define facets of $P_{\text{rep}}^{\text{map}}(G)$.

Proof. Let $r_0, u_0 \in V$ with $r_0 \neq u_0$ and

$$F' := \{ (x, b) \in P_{\text{rep}}^{\text{map}}(G) : x_{r_0,u_0} - x_{r_0,r_0} = 0 \} \subseteq \{ (x, b) \in P_{\text{rep}}^{\text{map}}(G) : \lambda x + \mu b = \beta \} =: F$$

where F is the facet containing F' . Since $(X_{r_0}, \kappa_V) \in F'$ and $(X_{r_0}^u, \kappa_V) \in F'$ for $u \neq r_0$ we get that $\lambda_{u,u} = 0$ if $u \neq r_0$. This, together with the fact that $(X_{r_0}, \kappa_V) \in F'$ and $(X_{r_0}^{u,v}, \kappa_V) \in F'$

if $u \neq r_0$ and $v \neq u$ yields $\lambda_{u,v} = 0$ if $u \neq r_0$ and $v \neq u$. We also have $(X_{r_0}, \kappa_V) \in F'$ and $(X_{r_0}, \kappa_V + 1) \in F'$ and thus $\mu = 0$.

We have shown that $\lambda_{u,v} = 0$ if $(u, v) \neq (r_0, u_0), (r_0, r_0)$ and must still show that $\lambda_{r_0, u_0} = \lambda_{r_0, r_0}$ which would imply that $F = F'$. To this end observe that for $r \neq r_0$ the vectors (X_r, κ_V) and $(X_r^{r_0, u_0}, \kappa_V)$ are contained in F' . This immediately yields $\lambda_{r_0, u_0} - \lambda_{u_0, u_0} = 0$ which finishes the proof. \square

Theorem 23. *If $|P| < |V|$ then the processor inequality*

$$\sum_{u \in V} x_{u,u} \leq |P| \quad (5.30)$$

defines a facet of $P_{\text{rep}}^{\text{map}}(G)$.

Proof. For $R = \{r_1, \dots, r_p\} \subset V$ with $|R| = |P|$ set

$$X_R := \sum_{u \in V} \hat{x}_{r_1, u} + \sum_{r=2}^{|R|} x_{r, r}.$$

In the feasible solution X_R all nodes are represented by r_1 and each node $r_2, \dots, r_{|P|}$ additionally represents itself. We assume

$$F' := \left\{ (x, b) \in P_{\text{rep}}^{\text{map}}(G) : \sum_{r \in V} x_{r, r} = |P| \right\} \subseteq \{(x, b) \in P_{\text{rep}}^{\text{map}}(G) : \lambda x + \mu b = \beta\} =: F$$

where F is the facet containing F' . Then $(X_R, \kappa_V) \in F'$ and $(X_R, \kappa_V + 1) \in F'$ for any set $R \subseteq V$ with $|R| = |P|$, proving that $\mu = 0$. Moreover, for $u \in V$ there is at least one set $R^0 = \{r_1^0, \dots, r_p^0\} \subset V$ with $|R^0| = |P|$ such that $r_1^0 \neq u$ and $u \in R^0$. For $v \neq u$ We then have $(X_{R^0}, \kappa_V) \in F'$ and $(X_{R^0} + \hat{x}_{u, v}, \kappa_V) \in F'$. Thus $\lambda_{u, v} = 0$ for all $u, v \in V$ with $u \neq v$.

Now let $u, v \in V$ with $u \neq v$. Since $|V| > |P|$ there exist sets $R^u, R^v \subset V$ $|R^u| = |R^v| = |P|$, $r_1^u \neq u$ and $r_1^v \neq v$ for which $R^u \setminus R^v = \{u\}$ and $R^v \setminus R^u = \{v\}$. In other words, the set R^u and R^v differ only in u and v . As $(X_{R^u}, \kappa_V) \in F'$ and $(X_{R^v}, \kappa_V) \in F'$ we get $\lambda_{u, u} = \lambda_{v, v}$ for all $u, v \in V$ with $u \neq v$ and the claim is proved. \square

Further Valid Inequalities

We now presented several inequalities that are valid for (REPMAP) and may be used to strengthen the problem formulation. Some of the inequalities presented are just adaptations of inequalities for (NAVMAP), others are new and specific for (REPMAP).

As before an upper limit N on the number of blocks that may be mapped to a processor directly yields a valid inequality

$$\sum_{u \neq r} x_{r,u} \leq (N-1)x_{r,r} \quad r \in V. \quad (5.31)$$

Observe that this inequality is different from (5.13): since no block can be represented by r unless $x_{r,r} = 1$ we obtain here a slightly stronger formulation than just $\sum_{u \in V} x_{r,u} \leq N$.

We show next, how inequalities (5.15), and (5.21) that are valid for (NAVMAP) can be adopted to (REPMAP). As (5.31) indicated, we can use the fact that no block can be represented by $r \in V$ unless $x_{r,r} = 1$. Exploiting this fact sometimes leads to stronger formulation as the straightforward adoptions.

Theorem 24. *Let $U \subseteq V$ and $R \subseteq V$ such that $d := \kappa^*(U, |R|) - \kappa^* > 0$. Then inequality*

$$\kappa^*(U, |R|) \leq b + d \sum_{u \in U} \sum_{r \in V \setminus R} x_{r,u} \quad (5.32)$$

is valid for $P_{rep}^{map}(G)$.

Proof. Inequality (5.32) is obviously valid unless all blocks in U are represented by blocks in R . In this case however, U is mapped to at most $|R|$ processors and $b \geq \kappa^*(U, |R|)$ must hold. \square

Theorem 25. *Let $U \subseteq V$ and $v \in U$ such that $\sum_{u \in U} \kappa_u = \kappa^* + \lambda$ for $\lambda > 0$ and $\sum_{u \in U - v} \kappa_u < \kappa^*$. Moreover, set $\tilde{U} := \{u \in U : \kappa_u > \lambda\}$. Then for $r \in V \setminus U$ inequality*

$$\sum_{u \in \tilde{U}} \lambda x_{r,u} + \sum_{u \in U \setminus \tilde{U}} \kappa_u x_{r,u} + \kappa^* \leq ((|\tilde{U}| - 1)\lambda + \sum_{u \in U \setminus \tilde{U}} \kappa_u) x_{r,r} + b \quad (5.33)$$

is valid for (REPMAP).

Proof. If $x_{r,r} = 0$ then no block can be represented by r and (5.33) reads $\kappa^* \leq b$ which is obviously satisfied by any feasible solution. For $x_{r,r} = 1$ the proof is analogous to the proof of valid inequality (5.21), see page 63. \square

Let us finally consider a class of valid inequalities that is not simply an adaption of inequalities for (NAVMAP). To this end, call a node $u \in V$ *active* in a feasible solution x^* if $x_{u,u}^* = 1$, i. e., if this node is currently used as representative. We also assume that $|P| \geq 2$, otherwise the solution to (REPMAP) is trivial.

Theorem 26. Let $\sigma : V \rightarrow V$ be a permutation such that $\sigma(u) \neq u$ for all $u \in V$. Then

$$\sum_{u \in V} x_{u,u} + \sum_{u \in V} x_{\sigma(u),u} \geq 2 \quad (5.34)$$

is valid for $P_{rep}^{map}(G)$.

Proof. To show validity observe that (5.34) is valid as soon as there are at least two active nodes. Since we always have at least one active node we must consider only this case. Assume that $v_0 \in V$ is the only active node. Then $x_{v_0,v_0} = 1$ and there is a node $w \in V$ with $\sigma^{-1}(w) = v_0$. Thus $x_{v_0,w} = 1$ for some $w \in V$ and (5.34) is satisfied. \square

5.1.3 Mapping with Slots

For the last block-mapping model, we no longer consider the mapping of blocks to processors explicitly. Instead we investigate the optimisation problem on a “per edge” basis. In other words, instead of nodes we map the edges to processors or pairs of processors.

To this end, we fix an arbitrary orientation on $G = (V, E)$. For simplicity we call the directed graph resulting from this orientation $G = (V, E)$ again, but the edge set E now consists of ordered pairs (u, v) rather than unordered ones. Each edge $e = (u, v)$ may be mapped to a single processor p or might run between two different processors p and k . In the first case, both endpoints u and v of e are mapped to p . In the latter case we distinguish the case in which e starts in p and ends in k (i. e., u is mapped to p and v to k) and the case in which e starts in k and ends in p (u is mapped to k and v to p). To formalise this, we provide for each edge so-called *slots* $(p, k) \in P \times P$ and require that each edge must be assigned to exactly one slot. If an edge e is assigned to slot (p, k) then it starts in p and ends in k (see Figure 5.1). If $p = k$ then e lies completely on p and is intra-processor, in all other cases it is inter-processor. To allow compact notation we define

$$x_{e,\{p,k\}} := x_{e,p,k} + x_{e,k,p} \quad \text{for } e \in E \text{ and } p \neq k.$$

For the integer programming model we introduce binary variables $x_{e,p,k}$ for all $e \in E$ and $(p, k) \in P \times P$ that are 1 if and only if edge e is mapped to slot (p, k) and 0 otherwise. These variables allow us to formulate (OGPC):

1. Each edge $e \in E$ must be mapped to exactly one slot, hence

$$\sum_{p,k \in P} x_{e,p,k} = 1$$

for all $e \in E$.

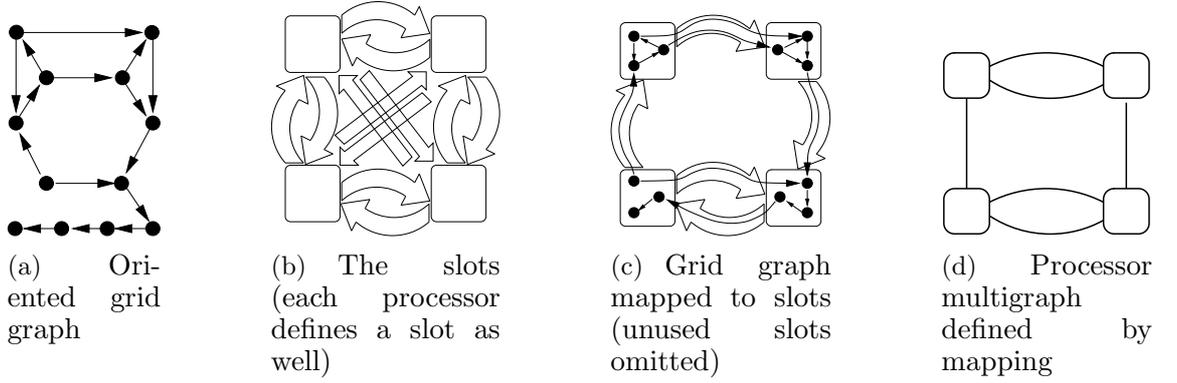


Figure 5.1: Mapping a grid graph in model (SLOTMAP).

- Next we must observe that mapping edge uv to slot (p, k) restricts the slots to which edges in $\delta(uv)$ can be mapped: Edges in $\delta^+(u)$ must then start in p and edges in $\delta^-(u)$ must end there. Likewise, edges in $\delta^+(v)$ must start in k and edges in $\delta^-(v)$ must end there. This gives rise to inequality

$$\sum_{k \in P} x_{uv,p,k} = \sum_{k \in P} x_{e,p,k}$$

for all $p \in P$ and $e \in \delta^+(u)$. Similar inequalities must be added for uv and $\delta^-(u)$ as well as for uv and $\delta^+(v)$ and $\delta^-(v)$. Notice that uv is itself contained in $\delta^+(u)$ and $\delta^-(v)$, but we do not need an inequality in this case.

- In order to count the number of control volumes that are mapped to processor p , we first observe that for a node $u \in V$ the sum $\sum_{k \in P} (\sum_{e \in \delta^+(u)} x_{e,p,k} + \sum_{e \in \delta^-(u)} x_{e,k,p})$ is equal to $\deg(u)$ if and only if u is mapped to processor p . In all other cases this sum is zero. Consequently, the sum of control volumes mapped to processor p is given by

$$\sum_{uv \in E} \sum_{k \in P} \left(\frac{\kappa_u}{\deg(u)} x_{u,p,k} + \frac{\kappa_v}{\deg(u)} x_{v,k,p} \right)$$

For sake of brevity we define $\kappa'_u = \kappa_u / \deg(u)$ and demand that

$$\sum_{uv \in E} \sum_{k \in P} (\kappa'_u x_{u,p,k} + \kappa'_v x_{v,k,p}) \leq b$$

for all $p \in P$.

Summarising 1 through 3 we obtain for a grid graph $G = (V, E)$

(SLOTMAP)	minimise b	(5.35a)
	$\sum_{(p,k) \in P \times P} x_{e,p,k} = 1$	$e \in E$ (5.35b)
	$\sum_{uv \in E} \sum_{k \in P} (\kappa'_u x_{uv,p,k} + \kappa'_v x_{uv,k,p}) \leq b$	$p \in P$ (5.35c)
	$\sum_{k \in P} x_{uv,p,k} - \sum_{k \in P} x_{f,p,k} = 0$	$uv \in E, p \in P, f \in \delta^+(u)$ (5.35d)
	$\sum_{k \in P} x_{uv,p,k} - \sum_{k \in P} x_{f,k,p} = 0$	$uv \in E, p \in P, f \in \delta^-(u)$ (5.35e)
	$\sum_{k \in P} x_{uv,k,p} - \sum_{k \in P} x_{f,p,k} = 0$	$uv \in E, p \in P, f \in \delta^+(v)$ (5.35f)
	$\sum_{k \in P} x_{uv,k,p} - \sum_{k \in P} x_{f,k,p} = 0$	$uv \in E, p \in P, f \in \delta^-(v)$ (5.35g)
	x binary	(5.35h)

Observe that model (SLOTMAP) is the first model to depend not only on the nodes in G , but also on the edges E . Though this is more complicated at first glance, we will see later that it enables us to make certain statements about the processor multigraph that cannot be made easily using the previous models.

We denote by

$$P_{\text{slot}}^{\text{map}}(G) := \text{conv} \left\{ (x, b) \in \{0, 1\}^{E \times P \times P} \times \mathbb{N} : (x, b) \text{ feasible for (SLOTMAP)} \right\}$$

the polyhedron defined by the feasible solutions to (SLOTMAP) for a grid graph $G = (V, E)$. Recall that as opposed to (NAVMAP) and (REPMAP) the model in (SLOTMAP) *does* depend on the edge set E in the grid graph! Moreover, we do not require that each block is mapped to *at least* one processor. Instead we demand that each edge is mapped to *exactly one* of the slots (see (5.35b)), which implies that each block is mapped to *exactly one* processor.

For subsequent proofs we set

$$S(u, p, k) := \sum_{e \in \delta^+(u)} \hat{x}_{e,p,k} + \sum_{e \in \delta^-(u)} \hat{x}_{e,k,p},$$

$$S(u, p, k, uv) := \sum_{e \in \delta^+(u)-uv} \hat{x}_{e,p,k} + \sum_{e \in \delta^-(u)-uv} \hat{x}_{e,k,p}$$

and define the following vectors that are feasible solutions to (SLOTMAP):

$$\begin{aligned}
X_p &:= \sum_{e \in E} \hat{x}_{e,p,p} + \kappa_V \hat{b} \\
X'_p &:= \sum_{e \in E} \hat{x}_{e,p,p} + (\kappa_V + 1) \hat{b} \\
X_p^{u,k} &:= \sum_{e \in E[V-u]} \hat{x}_{e,p,p} + S(u, k, p) + \kappa_V \hat{b} \\
X_p^{uv,k} &:= \sum_{e \in E[V-u-v]} \hat{x}_{e,p,p} + S(u, k, p, uv) + S(v, k, p, uv) + \hat{x}_{uv,k,k} + \kappa_V \hat{b} \\
X_p^{uv,k_1,k_2} &:= \sum_{e \in E[V-u-v]} \hat{x}_{e,p,p} + S(u, k_1, p, uv) + S(v, k_2, p, uv) + \hat{x}_{uv,k_1,k_2} + \kappa_V \hat{b}
\end{aligned}$$

In X_p all nodes are mapped to processor p . The same is true for X'_p , but in this case we use a suboptimal value for variable b . If $p \neq k$ are two distinct processors, then $X_p^{u,k}$ defines a solution in which all nodes but u are mapped to p and u is mapped to k . Similarly is $X_p^{uv,k}$, where all nodes but u and v are mapped to p and u and v are mapped to k . X_p^{uv,k_1,k_2} is a solution in which all blocks but u and v are mapped to processor p , u is mapped to k_1 and v to k_2 .

Dimension and Facets of $P_{\text{slot}}^{\text{map}}(G)$

As before we start polyhedral analysis of $P_{\text{slot}}^{\text{map}}(G)$ by exhibiting its dimension:

Theorem 27. *If $G = (V, E)$ is a grid graph on n nodes and m edges, then the dimension of $P_{\text{slot}}^{\text{map}}(G)$ is equal to $m|P|^2 + 1 - (m + (2m - n)(|P| - 1))$.*

Proof. Before proving the theorem, we observe that

$$\begin{aligned}
m|P|^2 + 1 - (m + (2m - n)(|P| - 1)) &= m|P|^2 + 1 - \left(m + (|P| - 1) \sum_{u \in V} (\deg_G(u) - 1) \right) \\
&= 1 + mp(p - 2) + m + n(p - 1).
\end{aligned}$$

We will therefore find $m + (|P| - 1) \sum_{u \in V} (\deg_G(u) - 1)$ linear independent equations that are satisfied by all points $P_{\text{slot}}^{\text{map}}(G)$ to show $\dim(P_{\text{slot}}^{\text{map}}) \leq m|P|^2 + 1 - (m + (2m - n)(|P| - 1))$ and exhibit $2 + mp(p - 2) + m + n(p - 1)$ affinely independent points in $P_{\text{slot}}^{\text{map}}(G)$ to show $\dim(P_{\text{slot}}^{\text{map}}) \geq m|P|^2 + 1 - (m + (2m - n)(|P| - 1))$.

Consider the following set of $2 + mp(p - 2) + m + n(p - 1)$ feasible points in $P_{\text{slot}}^{\text{map}}(G)$:

$$X_0 \tag{5.36}$$

$$X'_0 \tag{5.37}$$

$$X_0^{u,k} \quad u \in V, k \neq 0 \tag{5.38}$$

$$X_0^{e,k} \quad e \in E, k \neq 0 \tag{5.39}$$

$$X_0^{e,k_1,k_2} \quad e \in E, k_1, k_2 \in P \setminus \{0\}, k_1 \neq k_2. \tag{5.40}$$

Observe that the set of points specified by (5.40) is empty if $|P| = 2$. Assume that we have real scalars $a_0, a'_0, a_0^{u,k}, a_0^e, a_p^{e,k_1,k_1}$ such that

$$\begin{aligned} a_0 X_0 + a'_0 X'_0 + \sum_{u \in V} \sum_{k \neq 0} a_0^{u,k} X_0^{u,k} + \sum_{e \in E} \sum_{k \neq 0} a_0^{e,k} X_0^{e,k} + \sum_{e \in E} \sum_{\substack{k_1, k_2 \in P \setminus \{0\} \\ k_1 \neq k_2}} a_0^{e,k_1,k_2} X_0^{e,k_1,k_2} &= 0 \\ a_0 + a'_0 + \sum_{u \in V} \sum_{k \neq 0} a_0^{u,k} + \sum_{e \in E} \sum_{k \neq 0} a_0^{e,k} + \sum_{e \in E} \sum_{\substack{k_1, k_2 \in P \setminus \{0\} \\ k_1 \neq k_2}} a_0^{e,k_1,k_2} &= 0. \end{aligned}$$

Then for $k \neq 0$ and $e \in E$ the entries for $x_{e,k,k}$ immediately yield $a_0^{e,k} = 0$. Furthermore, for $uv \in E, k_1, k_2 \in P \setminus \{0\}$ and $k_1 \neq k_2$ the entries for x_{uv,k_1,k_2} imply $a_0^{uv,k_1,k_2} = 0$. This being proved, it is obvious that for $u \in V, uv \in E$ and $k \neq 0$, the entries for $x_{uv,k,0}$ (or $x_{vu,0,k}$) imply $a_0^{u,k} = 0$. Finally, the entries for b — together with the requirement that all scalars sum to zero — yield $a'_0 = 0$ and thus also $a_0 = 0$. Consequently, the points presented above are affinely independent and $\dim(P_{\text{slot}}^{\text{map}}(G)) \geq 1 + mp(p - 2) + m + n(p - 1)$.

To show $\dim(P_{\text{slot}}^{\text{map}}(G)) \leq m|P|^2 + 1 - (m + (2m - n)(|P| - 1))$ we find $(|P| - 1)(\deg_G(u) - 1)$ linearly independent equations for each node $u \in V$ with $\deg_G(u) > 1$ and an additional set of m other equations. To this end set $e_u := \min\{e \in N(u)\}$, i. e., e_u is the edge with smallest index among all edges incident at u . Moreover, for $u \in V, f \in \delta(u) \setminus \{e_u\}$ and $p \in P$ define

$$\begin{aligned} g(u, f, p) &:= \begin{cases} \sum_{k \in P} x_{f,p,k} - \sum_{k \in P} x_{e_u,p,k} & \text{if } e_u \in \delta^+(u), f \in \delta^+(u) \\ \sum_{k \in P} x_{f,p,k} - \sum_{k \in P} x_{e_u,k,p} & \text{if } e_u \in \delta^-(u), f \in \delta^+(u) \\ \sum_{k \in P} x_{f,k,p} - \sum_{k \in P} x_{e_u,p,k} & \text{if } e_u \in \delta^+(u), f \in \delta^-(u) \\ \sum_{k \in P} x_{f,k,p} - \sum_{k \in P} x_{e_u,k,p} & \text{if } e_u \in \delta^-(u), f \in \delta^-(u) \end{cases} \\ h(e) &:= \sum_{k,l \in P} x_{e,k,l} \quad e \in E. \end{aligned}$$

Then each feasible point in $P_{\text{slot}}^{\text{map}}(G)$ satisfies $g(u, f, p) = 0$ and $h(e) = 1$ (these are constraints (5.35d) through (5.35g) and (5.35b)). Furthermore, the $m + (\sum_{u \in V} (\deg_G(u) - 1))(|P| - 1)$ equations

$$g(u, f, p) = 0 \quad u \in V, \deg_G(u) > 1, f \in \delta(u) \setminus \{e_u\}, p \neq 0 \tag{5.41}$$

$$h(e) = 1 \quad e \in E \tag{5.42}$$

can be shown to be linearly independent. To see that, assume they are not. Then there is a vector $\lambda \neq 0$ such that $\lambda^T A = 0$, where A is the matrix defined by the left-hand side of (5.41) and (5.42). Let $\lambda_{u,f,p}$ denote the coefficient associated with $g(u, f, p) = 0$ and let λ_e denote the coefficient associated with $h(e) = 1$. As variable $x_{e,0,0}$ has a non-zero coefficient only in $h(e) = 1$, we immediately conclude $\lambda_e = 0$ for all $e \in E$. Consider now the edge $e' = u'v'$ that has the highest index. If $\deg_G(v') > 1$, then for $k \neq 0$ variable $x_{e',k,0}$ has a non-zero coefficient in exactly one equation, namely $g(v', e', k) = 0$. This implies $\lambda_{v',e',k} = 0$. If $\deg_G(u') > 1$ variable $x_{u',k,0}$ for $k \in P$ has non-zero coefficient in exactly one equation: $g(u', e', k)$. We therefore have $\lambda_{u',e',k} = 0$. Observing that $x_{e',p,k}$ and $x_{e',k,p}$ for $p \neq 0$ have non-zero coefficients only in $g(u', e', p) = 0$ and $g(v', e', p) = 0$ we see that all entries of λ that are associated with equations that have non-zero coefficients for $x_{e',p,k}$ (p and k arbitrary) are zero. We may therefore recursively apply our argument to the edge with second-biggest index in E and obtain $\lambda = 0$. So the linear equation specified in (5.41) and (5.42) are linearly independent and the claim is proved. \square

The set of affinely independent points used in the proof of Theorem 27 allows us to show

Theorem 28. *For a graph $G = (V, E)$, the lower-bound inequalities*

$$x_{e,p,p} \geq 0 \quad e \in E, p \in P \quad (5.43)$$

$$x_{e,p,k} \geq 0 \quad e \in E, p, k \in P, p \neq k \quad (5.44)$$

define facets of $P_{\text{slot}}^{\text{map}}(G)$ if $|P| > 2$.

Proof. To show that $x_{e_0,p_0,p_0} \geq 0$ is facet-defining for $e_0 \in E$ and $p_0 \in P$ fix a processor $p \in P$ with $p \neq p_0$ and consider points

$$\begin{aligned} & X_p \\ & X'_p \\ & X_p^{u,k} \quad u \in V, k \neq p \\ & X_p^{e,k} \quad e \in E, k \neq p, (e, k) \neq (e_0, k_0) \\ & X_p^{e,k_1,k_2} \quad e \in E, k_1, k_2 \in P \setminus \{p\}, k_1 \neq k_2. \end{aligned}$$

If we choose $p = 0$ this is exactly the set of points that was used in the proof of Theorem 27, with the single exception that point $X_p^{e_0,k_0}$ is not contained in this set. All these $\dim(P_{\text{slot}}^{\text{map}}(G))$ points are affinely independent (by the proof of Theorem 27) and contained in $\{(x, b) \in P_{\text{slot}}^{\text{map}} : x_{e_0,p_0,p_0} = 0\}$. Thus $x_{e_0,p_0,p_0} \geq 0$ is facet-defining for $P_{\text{slot}}^{\text{map}}(G)$.

If $|P| > 2$ it is easy to see that $x_{e_0,p_0,k_0} \geq 1$ defines a facet for $P_{\text{slot}}^{\text{map}}(G)$ for $e_0 \in E$ and

$p_0, k_0 \in P$ with $p_0 \neq k_0$: Pick $p \in P$ with $p \neq p_0, k_0$ and consider points

$$\begin{aligned} X_p & \\ X'_p & \\ X_p^{u,k} & \quad u \in V, k \neq p \\ X_p^{e,k} & \quad e \in E, k \neq p, \\ X_p^{e,k_1,k_2} & \quad e \in E, k_1, k_2 \in P \setminus \{p\}, k_1 \neq k_2, (e, k_1, k_2) \neq (e_0, p_0, k_0). \end{aligned}$$

These points are obviously feasible for $\{(x, b) \in P_{\text{slot}}^{\text{map}}(G) : x_{e_0, p_0, k_0} = 0\}$. As before, these are the same points as used in the proof of Theorem 27, where only point $X_p^{e_0, p_0, k_0}$ is missing. The $\dim(P_{\text{slot}}^{\text{map}}(G))$ points are therefore affinely independent and the claim is proved. \square

For the upper bound inequalities $x_{e,p,k} \leq 1$ observe that they are implied by (5.35b) and $x_{e,p,k} \geq 0$. Thus these inequalities are redundant and never define facets of $P_{\text{slot}}^{\text{map}}(G)$.

We can also prove that (5.35c) defines a facet of $P_{\text{slot}}^{\text{map}}(G)$ by slightly adjusting the set of feasible points used in the two previous proofs:

Theorem 29. *Inequality (5.35c) defines a facet of $P_{\text{slot}}^{\text{map}}(G)$.*

Proof. Consider the points

$$\begin{aligned} Y_0 & := X_0 \\ Y_0^{u,k} & := X_0^{u,k} - (\kappa_V - \max\{\kappa_V - \kappa_u, \kappa_u\})\hat{b} \quad u \in V, k \neq 0 \\ Y_0^{uv,k} & := X_0^{uv,k} - (\kappa_V - \max\{\kappa_V - (\kappa_u + \kappa_v), \kappa_u + \kappa_v\})\hat{b} \quad uv \in E, k \neq 0 \\ Y_0^{uv,k_1,k_2} & := X_0^{uv,k_1,k_2} - (\kappa_V - \max\{\kappa_V - \max\{\kappa_u, \kappa_v\}, \max\{\kappa_u, \kappa_v\}\})\hat{b} \\ & \quad uv \in E, k_1, k_2 \in P \setminus \{0\}, k_1 \neq k_2. \end{aligned}$$

All these points are feasible for $P_{\text{slot}}^{\text{map}}(G)$ and clearly satisfy (5.35c) at equality. Moreover, by the same arguments as in the proof for Theorem 27 we easily see that they are independent. As the number of points is equal to

$$\begin{aligned} & 1 + |V|(|P| - 1) + |E|(|P| - 1) + |E|(|P| - 1)(|P| - 2) \\ & = 1 + n(|P| - 1) + mp(|P| - 2) + m \\ & = \dim(P_{\text{slot}}^{\text{map}}(G)) \end{aligned}$$

the claim is proved. \square

Valid Inequalities

Let $e, f, g \in E$ such that they form a triangle. It is clear, that if e is inter-processor, than not both, f and g can be intra-processor. This gives rise to the following valid inequality:

Theorem 30. *If $e, f, g \in E$ such that these edges form a triangle, then*

$$\sum_{p \in P} (x_{f,p,p} + x_{g,p,p}) \leq 1 + \sum_{p \in P} x_{e,p,p} \quad (5.45)$$

is valid for $P_{slot}^{map}(G)$.

Proof. Obviously, the left-hand side of (5.45) never exceeds 2, so we must only show that the inequality is valid if $\sum_{p \in P} (x_{f,p,p} + x_{g,p,p}) = 2$. In this case, both f and g are intra-processor. Since $\{e, f, g\}$ is a triangle, we know that e must also be intra-processor, thus $\sum_{p \in P} x_{e,p,p} = 1$ and the inequality is satisfied. \square

Observe that (5.45) is a strengthened aggregation of the per-processor version of this inequality, given by

$$x_{f,p,p} + x_{g,p,p} \leq 1 + x_{e,p,p} \quad p \in P.$$

Moreover, inequality (5.45) can be generalised in two different ways: First of all, we are not limited to triangles. Instead every cycle of length at least 3 gives rise to an inequality similar to (5.45):

Theorem 31. *If $\{e, f_1, \dots, f_k\} \subseteq E$ is a cycle in G , then*

$$\sum_{p \in P} \sum_{i=1}^k x_{f_i,p,p} \leq (k-1) + \sum_{p \in P} x_{e,p,p} \quad (5.46)$$

is valid for $P_{slot}^{map}(G)$.

Proof. The proof is analogous to the proof of Theorem 30: Since the left-hand side of (5.46) never exceeds k we must show validity only for the case in which $\sum_{p \in P} \sum_{i=1}^k x_{f_i,p,p} = k$. In this case however, all f_i are intra-processor. Moreover, since $\{e, f_1, \dots, f_k\}$ is a cycle, all f_i must be mapped to the *same* processor. As e is incident to two edges f_j that are mapped to the same processor e is also intra-processor on this processor. Thus $\sum_{p \in P} x_{e,p,p} = 1$ and (5.46) is satisfied. \square

If the cycle C is even then there is also an alternate formulation of this inequality:

Theorem 32. *Let $C = \{e_1, \dots, e_k\}$ be a cycle in G comprising of an even number of edges and let $M \subset C$ be a perfect matching in C . Defining $C' = C \setminus M$ inequality*

$$|M| \cdot \sum_{e \in C'} x_{e,p,p} \leq (|C'| - 1) \cdot |M| + \sum_{e \in M} x_{e,p,p}, \quad (5.47)$$

is valid for $P_{slot}^{map}(G)$ for all $p \in P$.

Proof. We prove validity of (5.47) for an arbitrary but fixed processor $p_0 \in P$. Unless all edges in C' are mapped to p_0 the left-hand side of (5.47) is no bigger than $|C'| \cdot |M|$ and (5.47) is satisfied.

If all edges in C' are mapped to p_0 , inequality (5.47) reduces to $|M| \leq \sum_{e \in M} x_{e,p_0,p_0}$. On the other hand, since M is a matching and all edges in $C \setminus M$ are mapped to p_0 , each edge $e \in M$ is incident to two edges $f \in C'$ that are mapped to p_0 . Thus each edge $e \in M$ must be mapped to p_0 and $x_{e,p_0,p_0} = 1$ for all $e \in M$. Consequently (5.47) is satisfied. \square

Notice that as opposed to (5.45) and (5.46), inequality (5.47) does not sum over all processors but involves only one processor instead.

The other generalisation of (5.45) uses the fact, that we need not necessarily sum over all processors in this inequality:

Theorem 33. *Let $\{e, f, g\} \subseteq E$ be a triangle in G and $K \subseteq P$. Then the inequality*

$$\sum_{p \in K} (x_{f,p,p} + x_{g,p,p}) + \sum_{k,l \notin K} x_{e,k,l} \leq 1 + \sum_{p \in K} x_{e,p,p} \quad (5.48)$$

is valid for $P_{slot}^{map}(G)$.

Proof. If $K = \emptyset$ then (5.48) is obviously satisfied for any feasible solution. So let $K \neq \emptyset$ and distinguish the cases $\sum_{k,l \notin K} x_{e,k,l} = 0$ and $\sum_{k,l \notin K} x_{e,k,l} = 1$.

In the latter case e is mapped to a slot that has no endpoint on a processor in K . Since $\{e, f, g\}$ is a triangle, neither f nor g can be mapped to a slot that has both endpoints in K . Thus $\sum_{p \in K} (x_{f,p,p} + x_{g,p,p}) = 0$ and (5.48) is satisfied.

If now $\sum_{k,l \notin K} x_{e,k,l} = 0$, then we have two sub-cases: If $\sum_{p \in K} (x_{f,p,p} + x_{g,p,p}) \leq 1$ then (5.48) is trivially satisfied. So assume $\sum_{p \in K} (x_{f,p,p} + x_{g,p,p}) = 2$. In this case both, f and g , are mapped to a slot that has both endpoints on the same processor. Since $\{e, f, g\}$ is a triangle, both edges must be mapped to the same processor, p' say. Again, since $\{e, f, g\}$ is a triangle, edge e must also be mapped to p' . We get $\sum_{p \in K} x_{e,p,p} = x_{e,p',p'} = 1$ and the claim is proved. \square

Notice that for $K = P$ inequalities (5.45) and (5.48) are the same. Just like (5.45), inequality (5.48) is not restricted to triangles but can be extended and defined on any cycle C in G that has length at least 3.

Let us finally present a valid inequality that not only involves x -variables but also the capacity counting variable b .

Theorem 34. Let $uv \in E$ with $\kappa_u + \kappa_v \leq \kappa^*$. Set $U = N(u) \cup N(v) \cup \{u, v\}$, $F^+ = (\delta^+(u) \cup \delta^+(v)) \setminus \{uv, vu\}$, and $F^- = (\delta^-(u) \cup \delta^-(v)) \setminus \{uv, vu\}$. Moreover, for $f \in F^+ \cup F^-$ let $w(f)$ denote the endpoint of f that is different from u and v . Then inequality

$$\kappa_u + \kappa_v + (\kappa(U) - \kappa_u - \kappa_v)x_{uv,p,p} \leq b + \sum_{f \in F^+} \kappa_{w(f)} \sum_{k \neq p} x_{f,p,k} + \sum_{f \in F^-} \kappa_{w(f)} \sum_{k \neq p} x_{f,k,p} \quad p \in P \quad (5.49)$$

is valid for (SLOTMAP).

Proof. As $\kappa_u + \kappa_v \leq \kappa^*$ by assumption, inequality (5.49) is valid if $x_{uv,p,p} = 0$. If otherwise $x_{uv,p,p} = 1$, then uv is intra-processor and u and v are both mapped to the same processor p . If $\sum_{k \neq p} x_{f,p,k} = 0$ for $f \in F^+$ then f is intra-processor and $w(f)$ is mapped to p as well. Applying similar arguments $f \in F^-$ proves validity of (5.49). \square

5.2 Edge-Colouring Models

We will now describe three different integer programming models for the edge-colouring problem on multigraphs. For each model we assume that we are given a multigraph $G = (V, E)$ and want to find a minimal edge-colouring for this graph.

5.2.1 Naive Edge-Colouring

The edge-colouring model to be described has – to the best of our knowledge – not been analysed in the literature yet. On the other hand, a formulation in the same spirit for node-colouring problems has received great attention [38, 27, 39]. Mutatis mutandis, some results of these papers can be easily transferred to the edge-colouring model we describe below.

For a multigraph $G = (V, E)$ and the set $C = \{0, \dots, |E| - 1\}$ of potential colours we introduce binary decision variables

$$y_{e,c} = \begin{cases} 1 & \text{if edge } e \text{ receives colour } c \text{ in } M_P \text{ and} \\ 0 & \text{otherwise} \end{cases} \quad (e, c) \in E \times C, \quad (5.50)$$

$$z_c = \begin{cases} 1 & \text{if colour } c \text{ is used in the edge-colouring of } M_P \text{ and} \\ 0 & \text{otherwise} \end{cases} \quad c \in C. \quad (5.51)$$

Although $|C| = |E|$ is not very meaningful in practice, we use this oversized set of potential colours as it simplifies polyhedral analysis.

To formulate the edge-colouring problem as an integer program we impose the following constraints:

1. Each edge $e \in E$ must receive a colour, so

$$\sum_{c \in E} y_{e,c} \geq 1$$

for all edges $e \in E$.

2. Moreover, an edge e may receive colour c only if this colour is indeed used in the edge-colouring of G . Thus we require

$$y_{e,c} \leq z_c$$

for all colours $c \in C$.

3. In order to obtain a feasible edge-colouring, incident edges must not have the same colour. This can be expressed by requiring

$$\sum_{v \in N(u)} y_{uv,c} \leq 1$$

for all $u \in V$ and $c \in E$.

4. The number of colours in the edge-colouring is given by

$$\sum_{c \in C} z_c,$$

so our objective is to minimise this sum.

Together with the binary decision variables described above, constraints 1 through 4 yield integer program

(NAVCOL)	$\text{minimise } \sum_{c \in C} z_c \tag{5.52a}$
	$\sum_{c \in C} y_{e,c} \geq 1 \quad e \in E \tag{5.52b}$
	$\sum_{v \in N(u)} y_{uv,c} \leq z_c \quad u \in V, c \in C \tag{5.52c}$
	$y, z \text{ binary.} \tag{5.52d}$

Observe that in this model constraints 2 and 3 have been combined into (5.52c). In the following we consider the polyhedron

$$P_{\text{nav}}^{\text{col}}(G) := \text{conv} \left\{ (y, z) \in \{0, 1\}^{E \times C \times C} : (y, z) \text{ satisfies (5.52b) and (5.52c)} \right\}$$

for a multigraph $G = (V, E)$. As stated earlier, this polyhedron has – to the best of our knowledge – not been investigated in the literature before. It is however similar to node-colouring polyhedra described and analysed in [38, 27, 39]. Consequently, many of the proofs in this section are simply adaptations of proofs in these papers.

Dimension and Facets of $P_{\text{nav}}^{\text{col}}(G)$

By Theorem 5 we assume without loss of generality that $\bar{N}(e)$ is not a matching for all $e \in E$ and that G does not contain universal edges. This immediately implies that G is not a star, on which the edge-colouring problem is easy anyway. Recall also that we assume $|C| = |E|$, i. e., we have as many colours as we have edges in the grid graph.

In order to simplify the proofs below, we define the following vectors that are incidence vectors of feasible solutions to (NAVCOL):

- (Y_c, Z_c) incidence vector of an edge-colouring that does not use c .
- $(Y_{c_1}^{c_2}, Z_{c_1}^{c_2})$ incidence vector of an edge-colouring that does not use c_1 but uses c_2 .
- $(Y_{c_1}^{F, c_2}, Z_{c_1}^{F, c_2})$ incidence vector of an edge-colouring that does not use c_1 and in which exactly one edge in F is coloured c_2 .

Here $F \subseteq E$ is an arbitrary non-empty set of edges. We assume that in all these vectors each edge receives exactly one colour and that $z_c = 1$ if and only if there is an edge that receives colour c .

Notice that since G does not contain universal edges, there is always a feasible edge-colouring that does not use colour c for any given colour c .

Theorem 35. *If $|E| > 2$ then $P_{\text{nav}}^{\text{col}}(G)$ is full-dimensional, i. e., $\dim(P_{\text{nav}}^{\text{col}}(G)) = m^2 + m$.*

Proof. Assume that $P_{\text{nav}}^{\text{col}}(G)$ is not full-dimensional. Then there is an equality $\lambda y + \mu z = \beta$ with $(\lambda, \mu) \neq (0, 0)$ that is satisfied by all points in $P_{\text{nav}}^{\text{col}}(G)$.

Since $(Y_c, Z_c) \in P_{\text{nav}}^{\text{col}}(G)$ and $(Y_c, Z_c + \hat{z}_c) \in P_{\text{nav}}^{\text{col}}(G)$ for all $c \in C$ we conclude that $\mu = 0$. Furthermore, $(Y_c, \mathbb{1}) \in P_{\text{nav}}^{\text{col}}(G)$ and $(Y_c + \hat{y}_{e,c}, \mathbb{1}) \in P_{\text{nav}}^{\text{col}}(G)$ for all $e \in E$ and $c \in C$. Thus also $\lambda = 0$ and the derived contradiction proves the claim. \square

For the facial structure of $P_{\text{nav}}^{\text{col}}(G)$ we again start with the inequalities that occur in formulation (NAVCOL).

Theorem 36. *The bounding inequalities*

$$z_c \leq 1 \quad c \in C \quad (5.53)$$

$$y_{e,c} \geq 0 \quad e \in E, c \in C \quad (5.54)$$

define facets of $P_{\text{nav}}^{\text{col}}(G)$.

Proof. To prove that (5.53) is facet-defining let $c_0 \in C$ and

$$F' := \{(y, z) \in P_{\text{nav}}^{\text{col}}(G) : z_{c_0} = 1\} \subseteq \{(y, z) \in P_{\text{nav}}^{\text{col}}(G) : \lambda y + \mu z = \beta\} =: F,$$

where F is the facet containing F' . Then $(Y_c^{c_0}, Z_c^{c_0}) \in F'$ and $(Y_c^{c_0}, Z_c^{c_0} + \hat{z}_c) \in F'$ for $c \neq c_0$, showing that $\mu_c = 0$ for $c \neq c_0$. Moreover, the point $(Y_c^{c_0} + \hat{y}_{e,c}, Z_c^{c_0} + \hat{z}_c)$ is also contained in F' for all $e \in E$ and $c \neq c_0$, yielding $\lambda_{e,c} = 0$ for all $e \in E$ and $c \neq c_0$.

To show that $\lambda_{e,c_0} = 0$ for all $e \in E$ observe that $(Y_{c_0}, Z_{c_0} + \hat{z}_{c_0}) \in F'$ and $(Y_{c_0} + \hat{y}_{e,c_0}, Z_{c_0} + \hat{z}_{c_0}) \in F'$ for all $e \in E$. Thus $\lambda_{e,c_0} = 0$ for all $e \in E$ and the claim is proved.

Now we show that (5.54) defines a facet of $P_{\text{nav}}^{\text{col}}(G)$. To this end let $e_0 \in E$ and $c_0 \in C$ and

$$F' := \{(y, z) \in P_{\text{nav}}^{\text{col}}(G) : y_{e_0,c_0} = 0\} \subseteq \{(y, z) \in P_{\text{nav}}^{\text{col}}(G) : \lambda y + \mu z = \beta\} =: F,$$

where F is the facet containing F' . Assume that $c \in C$ and let (Y^c, Z^c) denote the incidence vector of a feasible edge-colouring in which colour c is not used and edge e_0 is not coloured by c_0 (notice that $c = c_0$ is allowed here). It is clear, that such a colouring always exists, since G contains no universal edges and colours can be permuted arbitrarily. Then $(Y^c, Z^c) \in F'$ and $(Y^c, Z^c + \hat{z}_c) \in F'$, showing that $\mu_c = 0$. Since c was arbitrary, we conclude $\mu = 0$.

The points (Y^c, Z^c) and $(Y^c + \hat{y}_{e,c}, Z^c + \hat{z}_c)$ are both contained in F' unless $(e, c) = (e_0, c_0)$. Thus $\lambda_{e,c} = 0$ if $(e, c) \neq (e_0, c_0)$ and the claim is proved. \square

The requirement that each edge must receive at least one colour defines a facet of $P_{\text{nav}}^{\text{col}}(G)$ as well:

Theorem 37. *For $e \in E$ the colouring inequality*

$$\sum_{c \in C} y_{e,c} \geq 1 \quad (5.55)$$

defines a facet of $P_{\text{nav}}^{\text{col}}(G)$.

Proof. Fix an edge $e_0 \in E$ and let

$$F' := \left\{ (y, z) \in P_{\text{nav}}^{\text{col}}(G) : \sum_{c \in C} y_{e_0, c} = 1 \right\} \subseteq \{(y, z) \in P_{\text{nav}}^{\text{col}}(G) : \lambda y + \mu z = \beta\} =: F,$$

where F is the facet containing F' . Since $(Y_c, Z_c) \in F'$ and $(Y_c, Z_c + \hat{z}_c) \in F'$ for all $c \in C$ we get $\mu = 0$. Furthermore, $(Y_c, \mathbb{1}) \in F'$ and $(Y_c + \hat{y}_{e, c}, \mathbb{1}) \in F'$ for $e \neq e_0$, thus $\lambda_{e, c} = 0$ for $e \neq e_0$ and $c \in C$.

Finally, $(Y_{c_1}^{\{e_0\}, c_2}, \mathbb{1}) \in F'$ and $(Y_{c_1}^{\{e_0\}, c_2} - \hat{y}_{e_0, c_2} + \hat{y}_{e_0, c_1}, \mathbb{1}) \in F'$ for $c_1 \neq c_2 \in C$, showing that $\lambda_{e_0, c_1} = \lambda_{e_0, c_2}$ for all $c_1, c_2 \in C$. \square

Let us now show in which cases (5.52c) defines a facet of $P_{\text{nav}}^{\text{col}}(G)$.

Theorem 38 (Star Inequality). *For $u \in V$ let $S_u = \{e \in E : e \cap \{u\} = \{u\}\}$ be the star around u . Then the star inequality*

$$\sum_{e \in S_u} y_{e, c} \leq z_c \tag{5.56}$$

defines a facet of $P_{\text{nav}}^{\text{col}}(G)$ if $|E| > |S_u| \geq 2$ and there is no edge $e \in E \setminus S_u$ that is incident to all edges in S_u .

Proof. In order to proof that (5.56) is facet-defining fix a colour $c_0 \in C$ and a node $u \in V$. Assume that

$$F' := \left\{ (y, z) \in P_{\text{nav}}^{\text{col}}(G) : \sum_{e \in S_u} y_{e, c_0} - z_{c_0} = 0 \right\} \subseteq \{(y, z) \in P_{\text{nav}}^{\text{col}}(G) : \lambda y - \mu z = \beta\} =: F$$

where F is the facet containing F' . For $c \neq c_0$ the vectors $(Y_c^{S_u, c_0}, Z_c^{S_u, c_0})$ and $(Y_c^{S_u, c_0}, Z_c^{S_u, c_0} + \hat{z}_c)$ are both contained in F' , showing that $\mu_c = 0$ unless $c = c_0$. Moreover, we have $(Y_c^{S_u, c_0}, Z_c^{S_u, c_0} + \hat{z}_c) \in F'$ and $(Y_c^{S_u, c_0} + \hat{y}_{e, c}, Z_c^{S_u, c_0} + \hat{z}_c) \in F'$ for $e \in E$ and $c \neq c_0$. Thus $\lambda_{e, c} = 0$ for $e \in E$ and $c \neq c_0$.

Also $(Y_{c_0}, Z_{c_0}) \in F'$ and $(Y_{c_0} + \hat{y}_{e, c_0}, Z_{c_0} + \hat{z}_{c_0}) \in F'$ for all $e \in S_u$, showing that $0 = \lambda_{e, c_0} - \mu_{c_0}$ for all $e \in S_u$. Hence $\lambda_{e, c_0} = \mu_{c_0}$ for $e \in S_u$.

Now pick any edge $f \in E \setminus S_u$. Since f is not incident to all edges in S_u by assumption, there is an edge $e \in S_u$ such that $e \cap f = \emptyset$. Since $(Y_{c_0} + \hat{y}_{e, c_0}, Z_{c_0} + \hat{z}_{c_0}) \in F'$ and $(Y_{c_0} + \hat{y}_{e, c_0} + \hat{y}_{f, c_0}, Z_{c_0} + \hat{z}_{c_0}) \in F'$ we conclude that $\lambda_{f, c_0} = 0$ for all $f \in E \setminus S_u$.

Summarising these considerations we get that $\lambda_{e, c_0} = \mu_{c_0}$ for all $e \in S_u$ and that all other entries of λ and μ are zero. This proves the claim. \square

The condition in Theorem 38 is satisfied if S_u covers at least 4 nodes. If instead S_u covers exactly 3 nodes and an edge $e \in E \setminus S_u$ is incident to all edges in S_u then $S_u \cup \{e\}$ forms a triangle. For such triangles we have

Theorem 39 (Triangle Inequality). *If $u, v, w \in V$ are three distinct and mutually adjacent nodes and $T_{u,v,w} = \{e \in E : |e \cap \{u, v, w\}| = 2\}$ then the triangle inequality*

$$\sum_{e \in T_{u,v,w}} y_{e,c} \leq z_c \quad c \in C \quad (5.57)$$

is valid and facet-defining for $P_{\text{nav}}^{\text{col}}(G)$.

Proof. We set $T := T_{u,v,w}$. Since each edge $e \in T$ is incident to all other edges $f \in T$ at most one edge in T may receive colour c . If one of these edges is coloured c then $z_c = 1$ by (5.52c).

To show that (5.57) is facet-defining fix a colour $c_0 \in C$ and assume that

$$F' := \left\{ (y, z) \in P_{\text{nav}}^{\text{col}}(G) : \sum_{e \in T} y_{e,c_0} - z_{c_0} = 0 \right\} \subseteq \{(y, z) \in P_{\text{nav}}^{\text{col}}(G) : \lambda y - \mu z = \beta\} =: F$$

where F is the facet containing F' . Since $(Y_c^{T,c_0}, Z_c^{T,c_0}) \in F'$ and $(Y_c^{T,c_0}, Z_c^{T,c_0} + \hat{z}_c) \in F'$ for all $c \neq c_0$ we immediately get $\mu_c = 0$ unless $c = c_0$. Moreover, $(Y_c^{T,c_0}, Z_c^{T,c_0} + \hat{z}_c) \in F'$ and $(Y_c^{T,c_0} + \hat{y}_{e,c}, Z_c^{T,c_0} + \hat{z}_{c_0}) \in F'$ for all $e \in E$ and $c \neq c_0$, showing that $\lambda_{e,c} = 0$ for all $e \in E$ and $c \neq c_0$. For $f \in E \setminus T$ there exists an edge $e_f \in T$ such that $f \cap e_f = \emptyset$. For these edges we get that $(Y_{c_0} + \hat{y}_{e_f,c_0}, Z_{c_0} + \hat{z}_{c_0}) \in F'$ and $(Y_{c_0} + \hat{y}_{e_f,c_0} + \hat{y}_{f,c_0}, Z_{c_0} + \hat{z}_{c_0}) \in F'$, showing that $\lambda_{f,c_0} = 0$ for $f \notin T$.

To show that $\lambda_{e,c_0} = \mu_{c_0}$ for all $e \in T$ observe that $(Y_{c_0}, Z_{c_0}) \in F'$ and $(Y_{c_0} + \hat{y}_{e,c_0}, Z_{c_0} + \hat{z}_{c_0}) \in F'$ for all $e \in T$. This implies $0 = \lambda_{e,c_0} - \mu_{c_0}$ and thus $\lambda_{e,c_0} = \mu_{c_0}$ for all $e \in T$.

We have shown that the only non-zero entries in λ and μ are λ_{e,c_0} for $e \in T$ and μ_{c_0} and that all these entries are equal. Hence the theorem is proved. \square

Further Valid Inequalities

In order to improve the integer programming formulation in (NAVCOL) we now present several valid inequalities. Notice that apart from the inequalities presented here, other classes of valid inequalities can be obtained by ‘‘converting’’ inequalities that are valid for the node-colouring problem. A node-colouring model similar to (NAVCOL) and sets of additional valid inequalities can be found in [38, 27, 39].

We start with an inequality that considers the chromatic index of subgraphs of G .

Theorem 40. Let $H = (U, F)$ be a subgraph of G and $C' \subseteq C$ such that $|C'| > |C| - \chi'(H)$. Then inequality

$$\sum_{e \in F} \sum_{c \in C'} y_{e,c} \geq \chi'(H) - |C \setminus C'| \quad (5.58)$$

is valid for $P_{nav}^{col}(G)$.

Proof. We know that $\chi'(G) \geq \chi'(H)$. So a feasible edge-colouring on H uses at least $\chi'(H)$ colours, at most $|C \setminus C'|$ of which are not contained in C' . Consequently at least $\chi'(H) - |C \setminus C'|$ of these colours must be contained in C' and the inequality is valid. \square

A variant of (5.58) is

Theorem 41. Let $H = (U, F)$ be a subgraph of G and $C' \subseteq C$ such that $|C'| > |C| - \chi'(H)$. Moreover, let $C_1, C_2 \subseteq C'$ such that $C_1 \cap C_2 = \emptyset$ and $C_1 \cup C_2 = C'$. Then inequality

$$\sum_{e \in F} \sum_{c \in C_1} y_{e,c} + \sum_{c \in C_2} z_c \geq \chi'(H) - |C \setminus C'| \quad (5.59)$$

is valid for $P_{nav}^{col}(G)$.

Proof. As in the proof of Theorem 40 we know that $k := \chi'(H) - |C \setminus C'|$ colours from C' must be used in a feasible edge-colouring of H (and therefore of G). This implies $\sum_{c \in C'} z'_c \geq k$ for any feasible solution (y', z') . By (5.52c) we have

$$\sum_{e \in F} \sum_{c \in C_1} y'_{e,c} + \sum_{c \in C_2} z'_c \geq \sum_{c \in C'} z'_c$$

and inequality (5.59) is satisfied.

Notice that the prerequisites in the theorem (and the proof) allow any one of the sets C_1 and C_2 to be empty. \square

Another way to exploit properties of subgraphs of G is

Theorem 42. Let $H = (U, F)$ be a subgraph of G . Then

$$\sum_{e \in F} y_{e,c} \leq \nu(H) z_c \quad c \in C \quad (5.60)$$

is valid for $P_{nav}^{col}(G)$.

Proof. Obviously, a colour class in an edge-colouring for G cannot contain more than $\nu(H)$ edges from H . Since an edge can receive colour $c \in C$ only if $z_c = 1$ inequality (5.60) is valid. \square

Observe that we obtain the star-inequality (5.56) from (5.60) if $H = (U, F)$ is a star in G . Just like stars also cycles have a known matching number and we obtain

Corollary 43. *If $H = (U, F)$ is a cycle with k nodes in G (potentially with multiple edges), then*

$$\sum_{e \in F} y_{e,c} \leq \left\lfloor \frac{k}{2} \right\rfloor z_c \quad c \in C \quad (5.61)$$

is valid for $P_{nav}^{col}(G)$.

Notice that we obtain the triangle inequality (5.57) if the cycle has exactly three vertices.

Another possibility to derive a valid inequality from a (simple) cycle in G is (see also [38])

Theorem 44. *Let $H = (u_1u_2, \dots, u_ku_1)$ be a (simple) cycle on $k > 2$ nodes in G and assume that c_1, \dots, c_k are mutually different colours. Moreover, denote by $(uv)^\parallel$ the set of edges parallel to uv (including uv). Then*

$$\sum_{e \in (u_1u_2)^\parallel} y_{e,c_1} + \sum_{i=2}^{k-1} \sum_{e \in (u_iu_{i+1})^\parallel} (y_{e,c_{i-1}} + y_{e,c_i}) + \sum_{e \in (u_{k-1}u_k)^\parallel} y_{e,c_{k-1}} + \sum_{e \in (u_ku_1)^\parallel} y_{e,c_1} \leq k-2 + z_{c_1} \quad (5.62)$$

is valid for $P_{nav}^{col}(G)$.

Proof. Obviously $\sum_{i=2}^{k-1} \sum_{e \in (u_iu_{i+1})^\parallel} (y_{e,c_{i-1}} + y_{e,c_i}) \leq k-2$ for any feasible point in $P_{nav}^{col}(G)$. Moreover, at most two edges in $(u_ku_1)^\parallel \cup (u_1u_2)^\parallel \cup (u_2u_3)^\parallel$ may receive colour c_1 . If at most one edge receives this colour, the proof is finished. Assume that exactly two edges in this set are coloured c_1 . This must be one edge $e_0 \in (u_ku_1)^\parallel$ and one edge $e_1 \in (u_2u_3)^\parallel$. Thus $y_{e_0,c_1} = y_{e_1,c_1} = 1$, $z_{c_1} = 1$ and $y_{e_3,c_1} = 0$ for $e_3 \in (u_1u_2, c_1)^\parallel$, which proves the claim. \square

5.2.2 Representative Edge-Colouring

Similar to the block-mapping problem, also the the edge-colouring problem has a formulation that employs representatives and avoids model intrinsic symmetries. Our representative formulation of the edge-colouring problem is based on an idea of Campelo, Correa and Frota [22] who described such a formulation for the node-colouring problem on graphs. Similar to model (REPMAP) the idea is to (uniquely) represent each colour class by one edge in this class. If E_c is the set of edges that are coloured c , then each edge $e \in E_c$ either is the representative for E_c or is represented by some other edge $f \in E_c$. Similar to Section 5.1.2 this yields binary decision variables

$$y_{r,e} = \begin{cases} 1 & \text{if } r \text{ represents } e \text{ and} \\ 0 & \text{otherwise.} \end{cases} \quad \text{for all } (r, e) \in E \times E.$$

As before, these variables are independent of the actual numbering of the colours in C and therefore remedy the symmetry issues raised by permutations of the colour set C (see also Section 5.1.2). Since a representative $r \in E_c$ is again freely selectable (see Section 5.1.2), we introduced another symmetry issue. This is because changing the representative for E_c in a feasible solution yields another feasible solution with the same objective function value. However, we can fix this problem by requiring that an edge $e \in E$ may never represent an edge $f \in E$ that has a smaller index³. We may do so by setting

$$y_{r,e} = 0 \quad r \in E, e < r. \quad (5.63)$$

As in Section 5.1.2 this enforces a *unique* representative for each of the colour classes E_c and thus removes the symmetry problems described above.

In order to formulate the edge-colouring problem for a multigraph $G = (V, E)$ we impose

1. Each edge $e \in E$ must belong to a colour class and therefore either represent itself or be represented by another edge. This is expressed by requiring

$$\sum_{r \in E} y_{r,e} \geq 1$$

for all edges $e \in E$.

2. An edge $r \in E$ may represent other edges $e \in E$ only if it represents itself:

$$y_{r,e} \leq y_{r,r}$$

for all $e \in E$.

3. Incident edges must not receive the same colour and are therefore required to be in different colour classes. This can be expressed by

$$\sum_{v \in N(u)} y_{r,uv} \leq y_{r,r}$$

for all $u \in V$ and $r \in E$.

4. According to our interpretation, each edge that represents itself gives rise to a new colour class. So the number of colours used is given by

$$\sum_{r \in E} y_{r,r}.$$

For a multigraph $G = (V, E)$ we obtain (see also [22]):

³Again, this idea cannot be found in [22].

(REPCOL)	$\text{minimise } \sum_{e \in E} y_{e,e} \tag{5.64a}$
	$\sum_{r \in E} y_{r,e} \geq 1 \quad e \in E \tag{5.64b}$
	$y_{r,e} \leq y_{r,r} \quad (r, e) \in E \times E, r \neq e \tag{5.64c}$
	$\sum_{v \in N(u)} y_{r,uv} \leq y_{r,r} \quad u \in V, r \in E \tag{5.64d}$
	$y \text{ binary.} \tag{5.64e}$

Notice that (5.64d) implies (5.64c) so we may drop the latter. For ease of exposition we do not impose the simplifying constraints given by (5.63).

The integer programming problem defined in (REPCOL) is very similar to the one in [22] (although the latter one is for node-colouring) and so are the polyhedra. Thus many of the proofs in this section are simply adaptations of proofs in [22].

Before plunging into details of polyhedral analysis, let us show how the complexity of model (REPCOL) can be reduced. Observe that $y_{e,f} = 0$ if $e \neq f$ are incident. Hence we delete from (REPCOL) all variables $y_{e,f}$ where the edges $e \neq f$ are incident. If we denote by $\bar{N}(e)$ the set of edges *not* incident to e , then we are left with $\sum_{e \in E} (\bar{N}(e) + 1)$ variables (the term “+1” stems from the fact that an edge may represent itself). For ease of exposition, we assume that $\bar{N} \neq \emptyset$ for edges $e \in E$. In fact, if there is an edge $e \in E$ with $\bar{N}(e) = \emptyset$, then this edge is incident to *all* other edges; it is universal and can be handled by appropriate preprocessing techniques (see Theorem 5). Also by Theorem 5 we can assume that $\bar{N}(e)$ is non-singleton for all edges $e \in E$.

Let

$$P_{\text{rep}}^{\text{col}}(G) := \text{conv} \left\{ y \in \bigcup_{e \in E} \{0, 1\}^{\{e\} \times (\bar{N}(e) \cup \{e\})} : y \text{ satisfies (5.64b) through (5.64d)} \right\}$$

denote the convex hull of feasible solutions to (REPCOL) for the multigraph $G = (V, E)$.

Before we start to work on this polyhedron we define some vectors that will be useful in the following.

$$\begin{aligned} Y &:= \sum_{e \in E} \hat{y}_{e,e} \\ Y^{\bar{e}} &:= Y - \hat{y}_{e,e} + \hat{y}_{r,e} \quad \text{for } e \in E \text{ and a representative } r \neq e \\ Y^{ef} &:= Y + \hat{y}_{e,f} \quad \text{for } e \in E \text{ and } f \in \bar{N}(e). \end{aligned}$$

The vector Y represents a solution in which each edge represents itself. One of the $|E|$ vectors $Y^{\bar{e}}$ represents a solution in which each edge but e represents itself and e is represented by some other edge. Notice that for each edge $e \in E$ there is at least one possible vector $Y^{\bar{e}}$ since we assumed that there is no edge that is incident to all other edges. One of the $\sum_{e \in E} \bar{N}(e)$ vectors Y^{ef} represents a solution in which each edge represents itself and edge f is *additionally* represented by e .

Dimension and Facets of $P_{\text{rep}}^{\text{col}}(G)$

Let us first establish the dimension of $P_{\text{rep}}^{\text{col}}(G)$, the polytope that is defined by (REPCOL) for a graph G .

Theorem 45. *For any multigraph $G = (V, E)$ with $|\bar{N}(e)| \geq 2$ for all $e \in E$ the polytope $P_{\text{rep}}^{\text{col}}(G)$ is full-dimensional, i. e., $\dim(P_{\text{rep}}^{\text{col}}(G)) = \sum_{e \in E} (\bar{N}(e) + 1)$.*

Proof. Consider the following $1 + \sum_{e \in E} (\bar{N}(e) + 1)$ distinct vectors in $P_{\text{rep}}^{\text{col}}(G)$: Y , $Y^{\bar{e}}$ for $e \in E$ and Y^{ef} for $e \in E$ and $f \in \bar{N}(e)$. To show that they are affinely independent assume we have a_0 , a_e for $e \in E$ and a_{ef} for $e \in E$ and $f \in \bar{N}(e)$, all of them in \mathbb{R} , such that

$$a_0 + \sum_{e \in E} a_e + \sum_{e \in E, f \in \bar{N}(e)} a_{ef} = 0 \quad (5.65)$$

and

$$a_0 Y + \sum_{e \in E} a_e Y^{\bar{e}} + \sum_{e \in E, f \in \bar{N}(e)} a_{ef} Y^{ef} = 0. \quad (5.66)$$

For $g \in E$ looking at the y_{gg} entries quickly yields

$$a_0 + \sum_{e \in E, e \neq g} a_e + \sum_{e \in E, f \in \bar{N}(e)} a_{ef} = 0$$

and therefore $a_g = 0$. Inspecting the y_{gh} entries for $g \in E$ and $h \in \bar{N}(g)$ now easily shows $a_{gh} = 0$ and thus also $a_0 = 0$. So we proved that any scalars a_0 , a_e and a_{ef} satisfying (5.65) and (5.66) are identically zero. Thus our vectors Y , $Y^{\bar{e}}$ and Y^{ef} are affinely independent and the theorem is proved. \square

Theorem 46. *The bounding inequalities*

$$y_{e,e} \leq 1 \quad e \in E \quad \text{and} \quad (5.67)$$

$$y_{r,e} \geq 0 \quad e \in E, r \in \bar{N}(e) \quad (5.68)$$

are facets of $P_{\text{rep}}^{\text{col}}(G)$.

Proof. We start with $y_{e,e} \leq 1$. Observe that $y_{e,e} = 1$ holds for all the vectors Y , $Y^{\bar{g}}$ and Y^{gh} except for $Y^{\bar{e}}$. By the proof of Theorem 45 these vectors are affinely independent. We have thus found $\dim(P_{\text{rep}}^{\text{col}}(G))$ affinely independent vectors that satisfy $y_{e,e} \leq 1$ at equality which proves that this inequality defines a facet of $P_{\text{rep}}^{\text{col}}(G)$.

Furthermore, the inequality $y_{r,e} \geq 0$ is satisfied at equality by all vectors Y , $Y^{\bar{g}}$ and Y^{gh} but Y^{re} if we choose in the definition of $Y^{\bar{e}}$ a representative different from r . This is always possible since we assumed that $\bar{N}(e)$ is never singleton and therefore each edge $e \in E$ has at least two different potential representatives apart from itself. Again we have found a set of $\dim(P_{\text{rep}}^{\text{col}}(G))$ affinely independent feasible points that prove our claim. \square

The next facet is given by inequality (5.64b):

Theorem 47. *The colouring inequality*

$$\sum_{r \in \bar{N}(e) \cup \{e\}} y_{r,e} \geq 1$$

defines a facet of $P_{\text{rep}}^{\text{col}}(G)$ for all $e \in E$.

Proof. Fix $e_0 \in E$ and assume

$$F' := \left\{ y \in P_{\text{rep}}^{\text{col}}(G) : \sum_{r \in \bar{N}(e_0) \cup \{e_0\}} y_{r,e_0} = 1 \right\} \subseteq \{y \in P_{\text{rep}}^{\text{col}}(G) : \lambda y = \beta\} =: F$$

where F is the facet containing F' . Then $Y \in F'$ and for $g \in E$ and $h \in \bar{N}(g)$ we also have $Y - \hat{y}_{g,g} + \hat{y}_{h,g} \in F'$. This leads to $\lambda Y = \lambda(Y - \hat{y}_{g,g} + \hat{y}_{h,g})$ which implies $\lambda_{g,g} = \lambda_{h,g}$. Moreover, if we choose $g \neq e_0$ then $Y + \hat{y}_{h,g} \in F'$ and since Y is also in F' we get $\lambda_{h,g} = 0$. As we have already shown that $\lambda_{h,e_0} = \lambda_{e_0,e_0}$ for $h \in \bar{N}(e_0)$, the claim is proved. \square

Theorem 48. *Let $e \in E$, $H \subseteq \bar{N}(e)$ and denote by $\nu(G[H])$ the maximum size of a matching in $G[H]$. Then*

$$\sum_{f \in H} y_{e,f} \leq \nu(G[H]) y_{e,e} \tag{5.69}$$

is valid for $P_{\text{rep}}^{\text{col}}(G)$. Moreover, (5.69) defines a facet of $P_{\text{rep}}^{\text{col}}(G)$ if for any two edges $f, g \in H$ there are matchings M^f and M^g with $|M^f| = |M^g| = \nu(G[H])$ such that $M^f \setminus M^g = \{f\}$ and $M^g \setminus M^f = \{g\}$. In other words, M^f covers f , M^g covers g and both matchings differ only in f and g .

Proof. To see validity observe that in H no more than $\nu(G[H])$ edges can be represented by the same edge. Since an edge $f \in H$ can be represented by e only if $y_{e,e} = 1$, inequality 5.69 is valid.

In order to prove that (5.69) is facet-defining under the assumptions in the theorem, fix an edge $e_0 \in E$, and let

$$F' := \left\{ y \in P_{\text{rep}}^{\text{col}}(G) : \sum_{f \in H} y_{e_0, f} - \nu(G[H])y_{e_0, e_0} = 0 \right\} \subseteq \{y \in P_{\text{rep}}^{\text{col}}(G) : \lambda y = \beta\} =: F$$

where F is the facet containing F' . Moreover, let M be a maximum matching in $G[H]$ and set $Y^M = \sum_{f \in E} \hat{y}_{f, f} + \sum_{f \in M} \hat{y}_{e_0, f}$.

For an edge $g \neq e_0$ and $h \in \bar{N}(g)$ we have $Y^M \in F'$ and $Y^M + \hat{y}_{g, h} \in F'$. Thus $\lambda_{g, h} = 0$ for $g \neq e_0$ and $h \in \bar{N}(g)$. If $h \in \bar{N}(e_0) \setminus H$ then $Y^M + \hat{y}_{e_0, h} \in F'$. Together with $Y^M \in F'$ this yields $\lambda_{e_0, h} = 0$ for $h \in \bar{N}(e_0) \setminus H$.

Let us now proof that $\lambda_{h, h} = 0$ for $h \neq e_0$. To this end assume $h \neq e_0$ and pick $g \in \bar{N}(h)$ with $g \neq e_0$ (this is always possible since we assumed $|\bar{N}(h)| \geq 2$). Then $Y^M + \hat{y}_{g, h} \in F'$ and $Y^M + \hat{y}_{g, h} - \hat{y}_{h, h} \in F'$, leading to $\lambda_{h, h} = 0$ for $h \neq e_0$.

Assume now $g, h \in H$ with $g \neq h$. By the prerequisite to the theorem there are maximal matchings M^g and M^h in $G[H]$ for which $M^g \setminus M^h = \{g\}$ and $M^h \setminus M^g = \{h\}$. Since both, Y^{M^g} and Y^{M^h} are in F we get $\lambda_{e_0, g} = \lambda_{e_0, h}$ for all $g, h \in H$.

To conclude the proof, fix an arbitrary edge $h \in H$. Then $Y^M + \hat{y}_{h, e_0} \in F'$ and $Y^{h e_0} - \hat{y}_{e_0, e_0} \in F'$, yielding $\sum_{f \in M} \lambda_{e_0, f} - \lambda_{e_0, e_0} = 0$. Since $M \subseteq H$ and $\lambda_{e_0, f} = \lambda_{e_0, g}$ for all $f, g \in H$, we get $\nu(G[H])\lambda_{e_0, f} = \lambda_{e_0, e_0}$ for all $f \in H$ and the claim is proved. \square

A consequence of Theorem 48 is the star inequality for $P_{\text{rep}}^{\text{col}}(G)$:

Corollary 49. *If $e \in E$ and $S \subseteq \bar{N}(e)$ is such that $G[S]$ is a star containing at least two edges, then the star inequality*

$$\sum_{f \in S} y_{e, f} \leq y_{e, e} \tag{5.70}$$

defines a facet of $P_{\text{rep}}^{\text{col}}(G)$.

Proof. Since S is a star, we have $\nu(G[S]) = 1$. Moreover, for two edges $f, g \in S$, the matchings M^f and M^g required by Theorem 48 are easily constructed as $M^f = \{f\}$ and $M^g = \{g\}$. \square

Further Valid Inequalities

Many inequalities that are valid for (NAVCOL) have similar formulations for model (REPCOL). Adapting inequalities from Section 5.2.1 that were based on the matching number $\nu(H)$ of

certain subgraphs H of G yields instances of the facet-defining inequality (5.69). Hence we do not show the adaptations here.

Two inequalities that are not covered by (5.69) are (5.58) and (5.59). Adjusting these inequalities to (REPCOL) yields the following two valid inequalities.

Theorem 50. *Let $H = (U, F)$ be a subgraph of G and $E' \subseteq E$ such that $|E'| > |E| - \chi'(H)$. Then inequality*

$$\sum_{e \in F} \sum_{r \in E'} y_{r,e} \geq \chi'(H) - |E \setminus E'| \quad (5.71)$$

is valid for $P_{rep}^{col}(G)$.

Proof. We know that $\chi'(G) \geq \chi'(H)$. So a feasible edge-colouring on H uses at least $\chi'(H)$ colours, at most $|E \setminus E'|$ of which are not contained in E' . Consequently at least $\chi'(H) - |E \setminus E'|$ of these colours must be contained in E' and the inequality is valid. \square

Theorem 51. *Let $H = (U, F)$ be a subgraph of G and $E' \subseteq E$ such that $|E'| > |E| - \chi'(H)$. Moreover, let $E_1, E_2 \subseteq E'$ such that $E_1 \cap E_2 = \emptyset$ and $E_1 \cup E_2 = E'$. Then inequality*

$$\sum_{e \in F} \sum_{r \in E_1} y_{r,e} + \sum_{r \in E_2} y_{r,r} \geq \chi'(H) - |E \setminus E'| \quad (5.72)$$

is valid for $P_{rep}^{col}(G)$.

As the proof of (5.72) is simple and analogous the proof of (5.71) and (5.59) we omit it here.

5.2.3 Edge-Colouring with Matchings

We already outlined in Chapter 2 how the edge-colouring problem can be phrased as a set-covering problem where the covering sets are maximum cardinality matchings and the sets to be covered are the edges. Formulations of the edge-colouring problem that are based on matchings have been investigated in [152, 131]. However, in [152] the covering sets are any set of non-incident edges (in our terminology this is $\mathcal{M}(G)$ and not $[\mathcal{M}(G)]$), while in [131] the edge-colouring problem is restricted to simple graphs. All the authors do not require that covering matchings are of maximum cardinality. Though their approaches have several advantages (see Section 2.2), the drawback of them is the large size of $\mathcal{M}(G)$. This set grows rapidly with the number of *edges* (and not nodes) in the multigraph. Consequently it is infeasible to enumerate all matchings in $\mathcal{M}(M_P)$ and put them into the model at once. Instead one must use column-generation to dynamically generate them [131].

We use in our edge-colouring model the matching set $[\mathcal{M}^*(n)]$ where n is the number of nodes in the multigraph and stick to maximum cardinality matchings. This has the advantage that

for small number of nodes the family of covering sets is relatively small and can easily be enumerated.

Let $G = (V, E)$ be some multigraph. Introducing variables $z_{[M]} \in \mathbb{N}$ for each $[M] \in [\mathcal{M}^*(n)]$ we have only one single constraint to satisfy: each edge $e \in E$ must be covered by at least $\mu(e)$ matchings. This yields

(MATCOL)	$\text{minimise } \sum_{[M] \in [\mathcal{M}^*(n)]} z_{[M]} \tag{5.73a}$
	$\sum_{[M] \in [\mathcal{M}^*(n, uv)]} z_{[M]} \geq \mu(uv) \quad uv \in E \tag{5.73b}$
	$z \text{ integral.} \tag{5.73c}$

Observe that in a set $F \subseteq E$ of parallel edges, constraint (5.73b) must be established for only one edge $e \in F$. Thus this constraint gives rise to at most $|V| \cdot (|V| - 1)/2$ concrete inequalities.

If the graph G to be edge-coloured is simple, then $\mu(e) = 1$ for all $e \in E$ and (MATCOL) reads (see also [131])

$$\text{minimise } \mathbf{1}^T z \tag{5.74}$$

$$Az \geq \mathbf{1} \tag{5.75}$$

$$z \text{ binary} \tag{5.76}$$

where A is the edge-matching-incidence matrix. The same problem structure arises if G is a multigraph and we define the problem using the bigger sets $\mathcal{M}(G)$ or $\mathcal{M}^*(G)$ (see 2.2.2). Problems of this type are known as *Set Covering Problems* [7, 26] and have been extensively studied in the literature. We refer the interested reader to the annotated bibliography in [26] and to [23] for a survey of polyhedral properties and recent solution algorithms.

In our case however, $G = (V, E)$ is usually not simple, i. e., we have a problem of the form

$$\text{minimise } \mathbf{1}^T z \tag{5.77}$$

$$Az \geq c \tag{5.78}$$

$$z \text{ integral} \tag{5.79}$$

where $c \in \mathbb{N}^E$. Here A is again the edge-matching-incidence matrix, but the right-hand side of the inequality system now contains arbitrary positive integers.

For a multigraph $G = (V, E)$ let

$$P_{\text{mat}}^{\text{col}}(G) := \text{conv} \{z \in \mathbb{N}^{[\mathcal{M}^*(|V|)]} : z \text{ feasible for (MATCOL)}\}$$

denote the polyhedron defined by the feasible solutions to (MATCOL).

Theorem 52. *For any multigraph $G = (V, E)$ the polyhedron $P_{\text{mat}}^{\text{col}}(G)$ is full-dimensional.*

Proof. Assume that Z is a feasible point in $P_{\text{mat}}^{\text{col}}(G)$. Then $Z + \hat{z}_{[M]}$ is also feasible for $P_{\text{mat}}^{\text{col}}(G)$ for all $[M] \in [\mathcal{M}^*(|V|)]$. By Lemma 7 the $|\mathcal{M}^*(|V|)| + 1$ points $Z, Z + \hat{z}_{[M]}$ are affinely independent and the claim is proved. \square

Theorem 53. *The bounding inequality $z_{[M]} \geq 0$ defines a facet of $P_{\text{mat}}^{\text{col}}(G)$ for each $[M] \in [\mathcal{M}^*(|V|)]$ if $|V| > 4$.*

Proof. Fix a matching $[M] \in [\mathcal{M}^*(|P|)]|V|$ and let F be the facet that contains $\{z \in P_{\text{mat}}^{\text{col}}(G) : z_{[M]} = 0\}$. Since $|V| > 4$ we know that each edge $e \in E$ is covered by at least two different matchings. Hence we can construct a feasible solution in which matching M is not used. Let Z be such a solution. Then $Z \in F$ and for all $[M'] \in [\mathcal{M}^*(|V|)]$ with $[M'] \neq [M]$ also $Z + \hat{z}_{[M']} \in F$. This yields $|\mathcal{M}^*(|V|)|$ points that are contained in F and by Lemma 7 affinely independent and finishes the proof. \square

In order to show in which cases (5.73b) is facet-defining for $P_{\text{mat}}^{\text{col}}(G)$ we first need a simple Lemma.

Lemma 54. *Let K_n be a complete graph on at least 5 nodes (i. e., $n \geq 5$) and let e and f be two edges in K_n with $e \neq f$. Then there exists at least one matching $M_f \in \mathcal{M}^*(K_n)$ that covers f but not e .*

Proof. If e and f are incident in K_n then any matching $M \in \mathcal{M}^*(K_n)$ that covers f does the job. So assume that e and f are non-incident and pick any edge g that is incident to e . Since any two non-incident edges in K_n can be extended to a maximum matching, there is a matching $M_f \in \mathcal{M}^*(K_n)$ with $f, g \in M_f$. Since $g \in M_f$ this matching does not cover e and the claim is proved. \square

With the above Lemma we are now able to show when (5.73b) is facet-defining.

Theorem 55. *For $e \in E$ the multiplicity inequality*

$$\sum_{[M] \in [\mathcal{M}^*(|V|)]} z_{[M]} \geq \mu(e) \tag{5.80}$$

defines a facet of $P_{\text{mat}}^{\text{col}}(G)$ if one of the following conditions is satisfied:

(i) $|V| \leq 4$ and $\mu(e) = \max_{f \in [M_e]} \mu(f)$, where $[M_e]$ is the unique matching in $[\mathcal{M}^*(|V|)]$ that covers e .

(ii) The multigraph contains at least 5 nodes.

Proof. We first prove case (i). Recall that in this case each edge is covered by exactly one matching. Let

$$F' := \{z \in P_{\text{mat}}^{\text{col}}(G) : z_{[M_e]} = \mu(e)\} \subseteq \{z \in P_{\text{mat}}^{\text{col}}(G) : \lambda z = \beta\} =: F$$

where F is the facet containing F' . Since $\mu(e) = \max_{f \in [M_e]} \mu(f)$ there is a feasible solution Z with $z_{[M_e]} = \mu(e)$. Obviously, we then have $Z \in F'$. Moreover, $Z + z_{[M]}$ is contained in F' for all $[M] \neq [M_e]$, proving that $\lambda_{[M]} = 0$ if $[M] \neq [M_e]$. Since the only non-zero entry in λ is the entry for $[M_e]$ the claim is proved.

For case (ii) assume that $|V| \geq 5$ and recall that for each edge $f \in E$ there are at least two matchings that cover f . Let $C_f := [\mathcal{M}^*(|V|, f)]$ denote the set of all matchings covering f and assume

$$F' := \left\{ z \in P_{\text{mat}}^{\text{col}}(G) : \sum_{[M] \in C_e} z_{[M]} = \mu(e) \right\} \subseteq \{z \in P_{\text{mat}}^{\text{col}}(G) : \lambda z = \beta\} =: F.$$

where F is the facet containing F' . Let Z be a feasible solution to the edge-colouring problem that is contained in F' . Such a solution can always be constructed as follows:

1. Pick any matching $[M^0] \in C_e$ and set $z_{[M^0]} = \mu(e)$.
2. For each edge $f \in E$ that is left uncovered after the first step pick a matching $[M'] \in [\mathcal{M}^*(|V|)] \setminus C_e$ and cover f by $[M']$. At least one such matching always exists by Lemma 54.

For all $[M] \notin C_e$ the point $Z + \hat{z}_{[M]}$ is also contained in F' , showing that $\lambda_{[M]} = 0$ for all these matchings. Now pick $[M'] \in C_e$ with $[M'] \neq [M^0]$. Moreover, pick $[M_1], \dots, [M_k]$ not in C_e such that $M' \setminus \{e\} \subseteq \bigcup_{i=1}^k M_i$ (by Lemma 54 there always exist matchings M_i satisfying these prerequisites).

Then $Z - \hat{z}_{[M^0]} + \hat{z}_{[M']} + \sum_{i=1}^k \hat{z}_{[M_i]}$ is feasible and contained in F' . Since $Z \in F'$ and $\lambda_{[M_i]} = 0$ for $i = 1, \dots, k$ we have that $\lambda_{[M^0]} = \lambda_{[M']}$ for all $[M'] \in C_e$ with $[M'] \neq [M^0]$. This proves that $F' = F$ and thus (5.80) is facet-defining. \square

It is clear that further valid inequalities for $P_{\text{mat}}^{\text{col}}(G)$ will depend on the actual structure of G (at least on the multiplicities of its edges). We intend to use model (MATCOL) to describe

the edge-colouring problem on the processor multigraphs of block-mappings, i. e., on graphs that have an unknown and dynamically changing structure. As not even the multiplicities of processor multigraphs can be predicted, we refrain here from discussion of further valid inequalities.

Chapter 6

Joined Models

No one can guarantee success in [mixed integer programming],
but only deserve it.
— *Winston Churchill*

In the previous chapter we presented several models to formulate the block-mapping or edge-colouring problem as stand-alone problems. Our aim is however to derive a model for (OGPC). In this problem we must find a block-mapping with small maximum processor size that at the same time allows an edge-colouring of the respective processor multigraph with a small number of colours. We will now combine several of the individual models so as to obtain a model for (OGPC). Basically, any pair of block-mapping and edge-colouring model can be used to describe (OGPC). However, to keep discussion more easy to follow, we combine only related models like (NAVMAP) and (NAVCOL) or (REPMAP) and (REPCOL). We also combine (SLOTMAP) and (MATCOL), as the former allows us to directly express multiplicities of edges required for the latter. In order to keep resulting problem instances asymptotically small we also combine (SLOTMAP) and (NAVCOL). We skip all other combinations for two reasons: combining (NAVMAP) or (REPMAP) and (MATCOL) would require introduction of new variables that count multiplicities of edges in processor multigraphs and the other combinations left simply seem not very promising to us.

For all models presented here we assume that the grid is given as graph $G = (V, E)$ where V contains the blocks and the edges in E represent the communication requirements between blocks. This graph is assumed to be connected since otherwise we could decompose the simulation. For each block $u \in V$ we have its size $\kappa_u \in \mathbb{N}$ in control volumes. Moreover, we have a set P of processors and a capacity limit K for each processor $p \in P$ (i. e., we assume that all processors have the same capacity).

All models require an integral variable b that models the number of control volumes on the processor with the highest load. Remember that this processor determines the time required

for executing the arithmetic part of the simulation. In practice, this variable can be chosen to be continuous: since all block sizes κ_u are integral and b is a sum of block sizes, it will be implicitly integral in each optimal solution.

Some of the programs also need a set C of colours available for edge-colouring the processor multigraph M_P . The minimal size of the set C is unknown beforehand¹, but since $|C|$ directly influences the sizes of the problem instances it is important to choose C as small as possible. One choice that is always feasible is $|C| = |E|$. In Section 7.1 we will see that we can do much better than this.

Unfortunately the polyhedra defined by the models to be presented are very complex and thus hard to analyse. Among other reasons one reason for this is the large number of variables involved as well as the fact that we must analyse the edge-colouring problem on a multigraph that is defined only dynamically by the values of certain variables. We thus refrain in all but one cases from theoretical investigations of the polyhedra defined by integer programming formulations. Instead we will present valid inequalities as before and will additionally discuss the optimal solution of the respective LP relaxation. The optimal objective function value of this relaxation is the first lower bound produced by the Branch-and-Cut Algorithm (Algorithm 4) and thus gives a first indication on the strength of the problem formulation at hand.

6.1 Capacitated or Uncapacitated Processors?

Basically, there are two main scenarios in which the optimal graph-partitioning problem arises: Either we have a simulation that could be run on a single machine, but we want to execute in parallel in order to save time, or the simulation in question cannot be run on a single machine and thus requires us to use multiple processors. In either case we are given a set P of available processors but it need not be meaningful to use all of them. This is because using a lot of processors also may incur a high performance penalty caused by communication overhead. Thus in an optimal solution to our problem, some processors $p \in P$ might be idle. This indicates that “activating” an additional processor from the set of idle ones will increase the communication overhead such that even the best possible grid mapping leads to non-optimal execution times.

Since we assume homogeneous processors throughout this work, we clearly need no processor capacities in the first scenario described above: A solution will activate as many processors as required to minimise the simulation time. Depending on the structure of the grid graph and the actual values of $t_a > 0$ and $t_c > 0$ a solution may even use only one processor,

¹This was “a priori” in the first draft, but Markus, the philosopher, claimed that this term (as well as “a posteriori”) may not be used in ordinary theses like this one, but only in highly philosophical reasoning. So beware if you ever encounter a reviewer that happens to be a philosopher.

indicating that parallelising the given simulation on that hardware does not save any time.

In the second scenario it is not feasible to use only one processor (and even two or three might be too few). The sheer size of the simulation exceeds the capabilities of a single machine and we must use multiple ones. In order to make sure that a processor does not get assigned more control volumes than it can actually handle we must definitely use a capacity limit $K < \infty$ for each processor.

Thus, sometimes we need capacity limits to correctly represent the current instance of our optimisation problem and sometimes we do not. Notice however, that even in the second scenario processor capacities are not mandatory: The ratio t_a/t_c between arithmetic and communication time has a big influence on the performance that is gained by (massively) parallelising the application. If this ratio is such that an optimal solution requires multiple active processors then an explicit capacity limit might become superfluous since each optimal solution will implicitly satisfy it.

Yet another way to interpret a finite upper bound on K is as follows: The *load-balancing efficiency* of a block-to-processor mapping is commonly defined as²

$$L_{\text{eff}} := \frac{\kappa_V}{|P| \cdot \max_{p \in P} \kappa(p)} \quad (6.1)$$

where $\kappa(p)$ is sum of control volumes mapped to processor p . If $L_{\text{eff}} = 1$ then the load is perfectly balanced and each processor bears the same number of control volumes. If $L_{\text{eff}} < 1$ then the load is unbalanced and L_{eff} describes the *amount of imbalance*.

A finite capacity limit $K < \infty$ implies $\kappa(p) < K$ for all $p \in P$ and thus

$$L_{\text{eff}} \geq \frac{\kappa_V}{|P| \cdot K}. \quad (6.2)$$

So $K < \infty$ immediately yields a lower bound on the load-balancing efficiency of all feasible mappings. From (6.2) we can also conclude that a minimal load-balancing efficiency of L_0 can be asserted by requiring $K \leq \kappa_V / (L_0 \cdot |P|)$. So a finite capacity limit gives us direct control over the minimal load-balancing efficiency that we are willing to accept.

In the following we assume that t_c , t_a and the size of $|P|$ are such that no explicit capacity limits must be given, because every optimal solution will implicitly satisfy them. We nevertheless impose an upper bound K on b because this will help us to derive (stronger) valid inequalities for the models. Notice that this upper bound K is not necessarily the

²Although implemented in some software packages, I consider this definition a little confusing: An optimal load-balancing has at least $\lceil \kappa_V / |P| \rceil$ control volumes on its maximum processor. If κ_V is not divisible by $|P|$ we can thus never reach a load-balancing efficiency of 1, which is counterintuitive in my eyes. Even worse, it is not even true that a load-balancing with no more than $\lceil \kappa_V / |P| \rceil$ control volumes on the maximum processor always exists (otherwise big parts of this thesis would be void). So using L_{eff} to report efficiency somewhat disguises the fact that the quality of an optimal load-balancing is unknown.

processors' capacity but may also represent an upper bound on the maximum number of control volumes on a processor in an optimal solution. Such a bound can for example be found as follows: Determine

$$\mathbf{m}^* = \operatorname{argmin}\{b(\mathbf{m}) : \mathbf{m} \text{ is a mapping}\}$$

and let $C^{\max} \in \mathbb{N}$ with $\chi'(M_P(\mathbf{m}^*)) \leq C^{\max}$. If we have more than two processors and intend to use each of them, then it is clear that in any solution to (OGPC) there is at least one processor in $M_P(\mathbf{m}^*)$ that has degree greater or equal to two. If we want to improve \mathbf{m}^* with respect to communication overhead, we can thus save no more than $C^{\max} - 2$ communication rounds. In other words, the best we can hope for is a mapping \mathbf{m}' with $b(\mathbf{m}') = b(\mathbf{m}^*)$ and $\chi'(M_P(\mathbf{m}')) = 2$. This means that the maximum time we can save by changing \mathbf{m}^* is $t_c \cdot (C^{\max} - 2)$. This implies that $(K - b(\mathbf{m}^*)) \cdot t_a \leq (C^{\max} - 2) \cdot t_c$ and therefore

$$K \leq (C^{\max} - 2) \cdot \frac{t_c}{t_a} + b(\mathbf{m}^*). \quad (6.3)$$

Depending on the concrete values of t_a and t_c , this yields a tight bound on the maximum number of control volumes per processor.

Other ways to obtain an upper bound on the maximum processor size in an optimal solution are the heuristic algorithms that we describe in Chapter 9 below. These algorithms yield a mapping \mathbf{m}^h together with an edge-colouring of $M_P(\mathbf{m}^h)$ that uses $C^h \in \mathbb{N}$ colours. The objective function value for this mapping is $z^h = t_a \cdot b(\mathbf{m}^h) + t_c \cdot C^h$. Since the optimal objective function cannot exceed z^h and since we need at least two communication rounds (see above), we get

$$t_a \cdot K + 2 \cdot t_c \leq z^h \iff K < \frac{z^h - 2 \cdot t_c}{t_a}.$$

Depending on the quality of the heuristic solution, this can yield tight bounds for the maximum processor size in optimal solutions. Tighter upper bounds on the number of control volumes per processor can be derived if we have a lower bound l_c on the chromatic index of optimal processor multigraphs with $l_c > 2$. We will see in Chapter 7 how such bounds can be obtained.

6.2 Naive Model

Our first integer programming model for (OGPC) is a combination of the two naive models (NAVMAP) and (NAVCOL). The model does not introduce new variables and is thus

built on the binary variables

$$x_{u,p} = \begin{cases} 1 & \text{if block } u \text{ is mapped to processor } p \text{ and} \\ 0 & \text{otherwise} \end{cases} \quad \text{for } (u,p) \in V \times P, \quad (6.4)$$

$$y_{e,c} = \begin{cases} 1 & \text{if edge } e \text{ receives colour } c \text{ in } M_P \text{ and} \\ 0 & \text{otherwise} \end{cases} \quad \text{for } (e,c) \in E \times C, \quad (6.5)$$

$$z_c = \begin{cases} 1 & \text{if colour } c \text{ is used in the edge-colouring of } M_P \text{ and} \\ 0 & \text{otherwise} \end{cases} \quad \text{for } c \in C. \quad (6.6)$$

Again $C = \{0, \dots, c_{\max}\}$ is a set of colours that contains at least as many colours as are required for the optimal solution of the problem. Of course, this number of colours is unknown beforehand, so $|C|$ must be estimated. In Chapter 7 we will explain how good bounds for $|C|$ can be determined. Observe that in contrast to (NAVCOL) we do not have a variable $y_{e,c}$ for each colour $c \in C$ and each edge in the (processor) multigraph. Since the processor multigraph is unknown beforehand, we must formulate the edge-colouring problem on the edges of the grid-graph. This requires a variable $y_{e,c}$ for each colour $c \in C$ and each edge $e \in E$ (even for those edges that will have both endpoints on the same processor). On the other hand, not all edges in E will be inter-processor and thus — as opposed to (NAVCOL) — the sums $\sum_{c \in C} y_{e,c}$ are not necessarily required to be equal to one.

In addition to the binary variables described above we use variable b that was mentioned before and represents the number of control volumes on the processor with the highest load. With these variables at hand we formulate (OGPC) as follows:

1. Each block $u \in V$ must be mapped to exactly one processor, so we require

$$\sum_{p \in P} x_{u,p} = 1$$

for all $u \in V$.

2. The number of control volumes on a processor $p \in P$ is given by $\sum_{u \in V} \kappa_u x_{u,p}$. As we want b to be at least as big as all the processor loads, we add constraints

$$\sum_{u \in V} \kappa_u x_{u,p} \leq b$$

for all $p \in P$. Minimising b then yields that b is equal to the load on the largest processor in any optimal solution.

3. An edge $uv \in E$ is inter-processor if u and v are mapped to different processors, i. e., if $|x_{u,p} - x_{v,p}| = 1$ for any $p \in P$. Inter-processor edges must receive a colour, so we

require

$$\sum_{c \in C} y_{uv,c} \geq x_{u,p} - x_{v,p} \quad \text{and}$$

$$\sum_{c \in C} y_{uv,c} \geq x_{v,p} - x_{u,p}$$

for all edges, all colours and all processors. Notice that we cannot model the function $|x_{u,p} - x_{v,p}|$ using linear constraints only, thus we must generate one inequality for each of the two cases in which $|x_{u,p} - x_{v,p}| = 1$.

4. An edge $e \in E$ may receive colour $c \in C$ only if this colour is used in the edge-colouring of the processor multigraph. So we have

$$y_{e,c} \leq z_c$$

for all edges and all colours.

5. Two different edges $uv, st \in E$ may not receive the same colour (if any), if the two nodes u and s are mapped to the same processor. This is because in this case the edges are incident in the processor multigraph. Both nodes are mapped to the same processor p if and only if $x_{u,p} + x_{s,p} = 2$, so constraint

$$x_{u,p} + x_{s,p} + y_{uv,c} + y_{st,c} \leq 3$$

asserts that not both edges are coloured c in this case. Analogous constraints are required for the cases in which u and t , v and s or v and t are mapped to the same processor.

6. As b is as least as big as the biggest processor and $\sum_{c \in C} z_c$ counts the number of colours used, our aim is to minimise

$$t_a \cdot b + t_c \cdot \sum_{c \in C} z_c.$$

Collecting 1 through 6 into an integer programming model yields

(NAIVE)

$$\text{minimise } t_a \cdot b + t_c \cdot \sum_{c \in C} z_c \quad (6.7a)$$

$$\sum_{p \in P} x_{u,p} = 1 \quad u \in V \quad (6.7b)$$

$$\sum_{u \in V} \kappa_u x_{u,p} \leq b \quad p \in P \quad (6.7c)$$

$$\sum_{c \in C} y_{e,c} \geq \begin{cases} x_{u,p} - x_{v,p} \\ x_{v,p} - x_{u,p} \end{cases} \quad uv \in E, p \in P \quad (6.7d)$$

$$y_{e,c} \leq z_c \quad e \in E, c \in C \quad (6.7e)$$

$$y_{uv,c} + y_{st,c} + x_{u,p} + x_{s,p} \leq 3 \quad uv \neq st \in E, p \in P, c \in C \quad (6.7f)$$

$$y_{uv,c} + y_{st,c} + x_{u,p} + x_{t,p} \leq 3 \quad uv \neq st \in E, p \in P, c \in C \quad (6.7g)$$

$$y_{uv,c} + y_{st,c} + x_{v,p} + x_{s,p} \leq 3 \quad uv \neq st \in E, p \in P, c \in C \quad (6.7h)$$

$$y_{uv,c} + y_{st,c} + x_{v,p} + x_{t,p} \leq 3 \quad uv \neq st \in E, p \in P, c \in C \quad (6.7i)$$

$$b \leq K \quad (6.7j)$$

$$x, y, z \text{ binary} \quad (6.7k)$$

The integer programming model just described is a very naive one: Each constraint of the informal problem description was directly translated into a linear constraint. Some simplifications and compactifications of this model are immediate:

- The constraints (6.7e) and (6.7f) can be easily combined into one single constraint:

$$y_{uv,c} + y_{st,c} + x_{u,p} + x_{s,p} \leq 2 + z_c \quad uv \neq st \in E, p \in P, c \in C \quad (6.8)$$

(and likewise for (6.7e) and (6.7g), (6.7h) or (6.7i)). This is equivalent to imposing (6.7e) and (6.7f) since no edge can be coloured by colour c unless c is used and thus $z_c = 1$.

- The number of inequalities described by (6.7f) through (6.7i) is very large, namely $\mathcal{O}(|E|^2 \times P \times C|)$. The complexity can be reduced by using the following inequality instead

$$\sum_{uw \in E} y_{uw,c} + \sum_{vw \in E} y_{vw,c} + x_{u,p} + x_{v,p} \leq 2 + z_c \quad u \neq v \in V, p \in P, c \in C. \quad (6.9)$$

This inequality represents the following reformulation of the incident constraint for edge-colouring: If two different nodes $u \in V$ and $v \in V$ are mapped to the same processor p , then $x_u^p + x_v^p = 2$ and only one of the edges incident to either of the two nodes in G may receive colour c (and this can happen only if colour c is used in the edge-colouring of the processor multigraph). The new formulation of the incidence constraint reduces the number of inequalities required to $\mathcal{O}(|V|^2 \times P \times C)$.

Observe that if u and v are adjacent in G , then the variable $y_{uv,c}$ occurs twice on the left-hand side of (6.9). This seems strange at first glance but is no problem since uv is not coloured at all if u and v are mapped to the same processor, thus $y_{uv,c} = 0$ in this case.

6.2.1 LP Relaxation

Consider (NAIVE) where all variables have been relaxed to be continuous. Denote the new problem by (ELP).

Theorem 56. *For any optimal solution (x^*, y^*, z^*, b^*) to (ELP) we have $b^* \geq \kappa_V/|P|$.*

Proof. Assume that (x^*, y^*, z^*, b^*) is a feasible solution to (ELP) for which $b^* = K' < \kappa_V/|P|$. Then by (6.7c) we have $\sum_{u \in V} \kappa_u x_{u,p}^* \leq K'$ for all $p \in P$ and thus

$$\sum_{p \in P} \sum_{u \in V} \kappa_u x_{u,p}^* \leq |P| \cdot K' < \kappa_V.$$

On the other hand, since $\sum_{p \in P} x_{u,p}^* = 1$ for all $u \in V$ by (6.7b) we also have

$$\kappa_V = \sum_{u \in V} \left(\kappa_u \cdot \sum_{p \in P} x_{u,p}^* \right) = \sum_{p \in P} \sum_{u \in V} \kappa_u x_{u,p}^* < \kappa_V.$$

This contradiction proves the claim. □

The next theorem shows that there is indeed an optimal solution to (ELP) with objective function value $\kappa_V/|P|$, thus the optimal value of the LP-relaxation is always equal to the arithmetic time required for a mapping with load-balancing efficiency 1.

Theorem 57. *The vector $(x^0, y^0, z^0, b^0) = ((1/|P|) \cdot \mathbf{1}, 0, 0, \kappa_V/|P|)$ is the unique optimal solution of (ELP).*

Proof. Feasibility and optimality is obvious. To see uniqueness, assume that there is another optimal solution (x', y', z', b') with objective function value $\kappa_V/|P|$. Since $b' \geq \kappa_V/|P|$ by

Theorem 56 we immediately get $z' = 0$ and $y' = 0$. Inequality (6.7d) then implies $x'_{u,p} = x'_{v,p}$ for all $uv \in E$. Fix an arbitrary node $u_0 \in V$. As G is connected, it contains a spanning tree rooted in u_0 and we obtain $x'_{u,p} = x'_{u_0,p}$ for all $u \in V$. By (6.7c) we have

$$b' = \frac{\kappa_V}{|P|} \geq \sum_{u \in V} \kappa_u x'_{u,p} = \sum_{u \in V} \kappa_u x'_{u_0,p} = \kappa_V x'_{u_0,p}.$$

This implies $x'_{u_0,p} \leq 1/|P|$ for all $p \in P$ and since $\sum_{p \in P} x'_{u_0,p} = 1$ by (6.7b) we get $x_{u_0,p} = 1/|P|$. Consequently, the two solutions (x^0, y^0, z^0, b^0) and (x', y', z', b') are identical and the claim is proved. \square

6.2.2 Valid Inequalities

It is clear that all inequalities that are valid for (NAVMAP) are also valid for (NAIVE) and we will not repeat them here. Nevertheless, we point out that some of these inequalities cut off the optimal solution $(x^0, y^0, z^0, b^0) = ((1/|P|) \cdot \mathbf{1}, 0, 0, \kappa_V/|P|)$ of the LP-relaxation of (NAIVE):

- If $|V| < 2|P|$ we get that (5.14) cuts off the optimal LP solution (x^0, y^0, z^0, b^0) for in this case we have

$$\frac{\sum_{u \in V} (\kappa_u - k)}{|P|} + 2k = \frac{1}{|P|} (\kappa_V - |V|k + 2|P|k) = \frac{1}{|P|} (\kappa_V + \underbrace{(2|P| - |V|)k}_{>0}) > \frac{\kappa_V}{|P|}.$$

- Inequality (5.15) is another inequality that has the potential to cut off the optimal LP solution (x^0, y^0, z^0, b^0) . To see this, we rewrite (5.15) as

$$K^*(U, |P| - 1) - K_U^* \cdot \sum_{u \in U} x_{u,p} \leq b.$$

If $|U| < |P|$, then substituting x^0 and b^0 yields

$$\begin{aligned} & K^*(U, |P| - 1) - (K^*(U, |P| - 1) - K^*) \cdot \sum_{u \in U} x_{u,p}^0 \\ &= K^*(U, |P| - 1) - \frac{|U|}{|P|} \cdot (K^*(U, |P| - 1) - K^*) \\ &\geq K^*(U, |P| - 1) - \frac{|P| - 1}{|P|} \cdot (K^*(U, |P| - 1) - K^*) \\ &= \frac{1}{|P|} K^*(U, |P| - 1) + \frac{|P| - 1}{|P|} K^* \\ &> \frac{K^*}{|P|} \leq \frac{\kappa_V}{|P|}. \end{aligned}$$

Valid inequalities for (NAVCOL) are however not that easily transferred to (NAIVE). The reason for this is that in (NAVCOL) we assumed the multigraph to be known and fixed. In (NAIVE) on the other hand, the processor multigraph M_P depends on the current mapping and is thus not known beforehand.

In the following we will present several valid inequalities for (NAIVE) that cannot be formulated for either of the individual models.

Consider a node $u \in V$ in the grid graph $G = (V, E)$ and the star $S = \{e \in E : e \cap \{u\} = \{u\}\}$ around this node. If two edges $e_1, e_2 \in S$ are inter-processor edges, they must be incident in M_P . Thus no two edges in S may receive the same colour and inequality

$$\sum_{e \in S} y_{e,c} \leq 1 \quad (6.10)$$

is valid for all $c \in C$. Notice that we may replace the right-hand side of (6.10) by z_c and the inequality is still valid but might be slightly stronger. Inequality (6.10) is similar to constraint (5.52c) in (NAVCOL). The context of both inequalities is however a little different because in (6.10) not every edge on the left-hand side must receive a colour: only inter-processor edges are required to be coloured. In (5.52c) on the other hand each of the edges involved were required to be coloured in a feasible solution.

Assume that block $u_0 \in V$ is mapped to processor $p_0 \in P$. Then each neighbour $v \in N(u)$ that is not mapped to p_0 as well introduces an edge that is incident to p_0 in the processor multigraph. As all these edges are incident to the same node p_0 in M_P they must receive different colours and we obtain that

$$\deg_G(u)x_{u,p} - \sum_{v \in N(u)} x_{v,p} \leq \sum_{c \in C} z_c \quad p \in P \quad (6.11)$$

is valid for (NAIVE). The good thing about (6.11) is that it *directly* relates x - and z -variables without using intermediate y -variables.

Inequality (6.11) is obviously valid not only for a single block, but also for a set $U \subseteq V$ of blocks:

$$\sum_{u \in U} \left(\deg_G(u)x_{u,p} - \sum_{v \in N(u)} x_{v,p} \right) \leq \sum_{c \in C} z_c \quad p \in P. \quad (6.12)$$

Notice that this inequality is not just a simple aggregation of (6.11). It sums up the left-hand sides of several instance of (6.11) while keeping the right-hand side unchanged. Inequality (6.12) has the biggest impact if all blocks in U are mutually non-adjacent and are indeed mapped to the same processor in an optimal solution.

Theorem 58. *If $u \in V$ and $C' \subseteq C$ such that $|C'| = |C| - \deg(u) + 1$, then*

$$\sum_{v \in N(u)} x_{v,p} - x_{u,p} \leq \deg(u) - 1 + \sum_{e \in \delta(u)} \sum_{c \in C'} y_{e,c} \quad (6.13)$$

is valid for (NAIVE).

Proof. Inequality (6.13) is obviously satisfied by each feasible solution (x', y', z', b') unless $\sum_{v \in N(u)} x'_{v,p_0} = \deg(u)$ for some $p_0 \in P$. If u is also mapped to p_0 , we have $x'_{u,p_0} = 1$ and (6.13) is satisfied. Otherwise (u is not mapped to p_0) all edges in $\delta(u)$ must be coloured. Since $|C| - |C'| < \deg(u)$, at least one edge $e_0 \in \delta(u)$ must receive a colour $c_0 \in C'$. Thus $y'_{e_0,c_0} = 1$ and inequality (6.13) is satisfied. \square

Obviously, inequality (6.13) can also be defined on a subset $N' \subset N(u)$ of the neighbourhood of u and is still valid, provided that we choose C' such that $|C'| = |C| - |N'| + 1$. This restricted formulation may be helpful in cases in which some of the nodes in $N(u)$ are (always) mapped to the same processor as u .

Similar to (5.49) for (SLOTMAP) is

Theorem 59. *Let $u_0 \in V$ and set $U = \{u_0\} \cup N(u_0)$. Then inequality*

$$\kappa_{u_0} + (\kappa(U) - \kappa_{u_0})x_{u_0,p} \leq b + \sum_{uv \in \delta(u_0)} \kappa_v \sum_{c \in C} y_{uv,c} \quad p \in P \quad (6.14)$$

is valid for (NAIVE).

Proof. Inequality (6.14) is obviously valid if $x_{u_0,p} = 0$. So assume $x_{u_0,p} = 1$. Observing that $\sum_{c \in C} y_{uv,c} = 0$ for $v \in N(u_0)$ implies $x_{v,p} = 1$, validity is also obvious in this case. \square

Observe that inequality (6.14) is based on the same ideas as (5.49) for (SLOTMAP), while the latter could be formulated as individual inequality.

In some cases, inequality (6.14) cuts off the LP relaxation (x^0, y^0, z^0, b^0) described in Section 6.2.1. To see this, we put all terms in (6.14) to the left-hand side and obtain

$$\begin{aligned} \kappa_{u_0} + (\kappa(U) - \kappa_{u_0})x_{u_0,p}^0 - \sum_{uv \in \delta(u_0)} \kappa_v \sum_{c \in C} y_{e,c}^0 - b^0 &= \kappa_{u_0} + (\kappa(U) - \kappa_{u_0})\frac{1}{|P|} - 0 - \frac{\kappa(V)}{|P|} \\ &= \frac{|P| - 1}{|P|}\kappa_{u_0} - \frac{\kappa(V \setminus U)}{|P|} \\ &\stackrel{!}{\leq} 0. \end{aligned}$$

If $(|P| - 1)\kappa_{u_0} > \kappa(V \setminus U)$ then the last line is not satisfied and (x^0, y^0, z^0, b^0) violates (6.14).

6.3 Representative Model

We already mentioned several times that the naive formulation of (OGPC) and its subproblems is full of intrinsic symmetries. If we have a feasible solution to (NAIVE), then any permutation of P or C gives rise to a new feasible solution with the same objective function value. Thus each feasible solution to (NAIVE) has $|P|! \cdot |C|!$ equivalent feasible solutions. This leads to excessively redundant enumeration in Branch-and-Cut algorithms. In order to resolve these symmetry issues we proposed models (REPMAP) and (REPCOL) that were based on ideas from [22] and ruled out these symmetry issues. Recall that both models were based on representatives. In (REPMAP) the set of blocks mapped to the same processor was represented by a distinguished block (the block with smallest index), while in (REPCOL) the edges in a colour class were represented by the edge with smallest index in that class. Combining both models into an integer programming problem introduces two classes of binary decision variables:

$$x_{r_V, u} = \begin{cases} 1 & \text{if } r \text{ represents } u \text{ and} \\ 0 & \text{otherwise} \end{cases} \quad \text{for all } (r_V, u) \in V \times V,$$

$$y_{r_E, e} = \begin{cases} 1 & \text{if } r \text{ represents } e \text{ and} \\ 0 & \text{otherwise.} \end{cases} \quad \text{for all } (r_E, e) \in E \times E.$$

We use these variables and b for formulate (OGPC):

1. As we want exactly one representative for each block $u \in V$, we require

$$\sum_{r_V \in V} x_{r_V, u} = 1$$

for all $u \in V$. Moreover, blocks can represent other blocks only if they represent themselves, i. e., $x_{r_V, u} \leq x_{r_V, r_V}$ for all $r_V, u \in V$.

2. Each block that represents itself corresponds to a distinct processor. Since we have only the processors from P available, we impose

$$\sum_{r_V \in V} x_{r_V, r_V} \leq |P|,$$

i. e., the number of blocks representing themselves must not exceed the number of processors.

3. The number of control volumes on the processor represented by block $r_V \in V$ is given by $\sum_{u \in V} \kappa_u x_{r_V, u}$. Observe that this sum is zero if r_V does not represent itself, i. e., if r_V is not used as representative. b must be as least as big as this sum, so we require

$$\sum_{u \in V} \kappa_u x_{r_V, u} \leq b$$

for all $r_V \in V$. Minimising b will then yield that b equals the number of control volumes on the biggest processor.

4. An edge $uv \in E$ is inter-processor if its endpoints u and v are mapped to different processors, i. e., have different representatives. This is the case if $|x_{r_V,u} - x_{r_V,v}| = 1$ for any $r_V \in V$. An edge that is inter-processor must be coloured and therefore belong to a colour class. Consequently, an inter-processor edge must have a representative and we require

$$\begin{aligned} \sum_{r_E \in E} y_{r_E,uv} &\geq x_{r_V,u} - x_{r_V,v} && \text{and} \\ \sum_{r_E \in E} y_{r_E,uv} &\geq x_{r_V,u} - x_{r_V,v}. \end{aligned}$$

Observe that again we cannot express $|x_{r_V,u} - x_{r_V,v}| = 1$ using only linear constraints and must therefore generate an individual inequality for the both cases in which the absolute value may be equal to 1.

5. An edge $r_E \in E$ can be the representative of a colour class only if it represents itself. So we have

$$y_{r_E,e} \leq y_{r_E,r_E}$$

for all $r_E, e \in E$.

6. Two edges $uv, st \in E$ are incident in the processor multigraph if u and s are mapped to the same processor, which in turn is the case if they both have the same representative. Consequently, if $x_{r_V,u} + x_{r_V,s} = 2$ then uv and st may not receive the same colour. Receiving not the same colour is the same as being in two different colour classes (if any) which in turn is equivalent to having different representatives (if any). This incidence condition is expressed by

$$x_{r_V,u} + x_{r_V,s} + y_{r_E,uv} + y_{r_E,st} \leq 3.$$

Analogous constraints are required for the cases in which u and t , v and s or v and t are mapped to the same processor.

7. Variable b yields the number of control volumes on the biggest processor and $\sum_{r_E \in E} y_{r_E,r_E}$ evaluates to the number of colour classes used in the edge-colouring of the processor multigraph. So our objective is to minimise

$$t_a \cdot b + t_c \sum_{r_E \in E} y_{r_E,r_E}.$$

We summarise 1 through 7 in the following integer programming model.

(REP)		$\text{minimise } t_a \cdot b + t_c \cdot \sum_{r_E \in E} y_{r_E, e} \quad (6.15a)$
	$\sum_{r_V \in V} x_{r_V, u} = 1 \quad u \in V \quad (6.15b)$	$x_{r_V, u} - x_{r_V, r_V} \leq 0 \quad (r_V, u) \in V \times V, r_V \neq u \quad (6.15c)$
	$\sum_{u \in V} \kappa_u \cdot x_{r_V, u} \leq b \quad r_V \in V \quad (6.15d)$	$\sum_{r_V \in V} x_{r_V, r_V} = P \quad (6.15e)$
	$(x_{r_V, u} - x_{r_V, v}) \leq \sum_{r_E \in E} y_{r_E, uv} \quad uv \in E, r_V \in V \quad (6.15f)$	$(x_{r_V, v} - x_{r_V, u}) \leq \sum_{r_E \in E} y_{r_E, uv} \quad uv \in E, r_V \in V \quad (6.15g)$
	$y_{r_E, e} - y_{r_E, r_E} \leq 0 \quad (r_E, e) \in E \times E, r_E \neq e \quad (6.15h)$	
	$y_{r_E, uv} + y_{r_E, st} + x_{r_V, u} + x_{r_V, s} \leq 3 \quad (uv, st, r_E) \in E^3, r_V \in V \quad (6.15i)$	$y_{r_E, uv} + y_{r_E, st} + x_{r_V, v} + x_{r_V, s} \leq 3 \quad (uv, st, r_E) \in E^3, r_V \in V \quad (6.15j)$
	$y_{r_E, uv} + y_{r_E, st} + x_{r_V, u} + x_{r_V, t} \leq 3 \quad (uv, st, r_E) \in E^3, r_V \in V \quad (6.15k)$	$y_{r_E, uv} + y_{r_E, st} + x_{r_V, v} + x_{r_V, t} \leq 3 \quad (uv, st, r_E) \in E^3, r_V \in V \quad (6.15l)$
	$b \leq K \quad (6.15m)$	$x, y \text{ binary} \quad (6.15n)$

As we already mentioned in Sections 5.1.2 and 5.2.2 the representative formulation contains itself intrinsic symmetries: exchanging the representative for a set yields another solution with the same objective function value. However, these symmetry issues are easily handled by requiring (see also (5.23) and (5.63))

$$x_{r_V, u} = 0 \quad \text{for } (r_V, u) \in V \times V \text{ with } r_V > u, \quad (6.16)$$

$$y_{r_E, e} = 0 \quad \text{for } (r_E, e) \in E \times E \text{ with } r_E > e. \quad (6.17)$$

This not only removes about half of the variables from the problem but also requires a *unique* representative for each of the sets in question.

Compared with the model (NAIVE), model (REP) offers the following advantages:

- The problem intrinsic symmetry created by permutations of P and/or C is not an issue for this model. For each feasible processor mapping and each feasible edge-colouring of the processor multigraph there is exactly one assignment of the variables x and y that represent this mapping and colouring.
- There is no need for an explicit set of colours C . Instead the number of required colours is given by the number of edges that represent themselves. Consequently we do not need the set C of available colours and need not to worry about overestimating its size and thereby unnecessarily blowing up the size of the instance.

Apart from the above advantages, (REP) also has several drawbacks when compared to (NAIVE):

- The number of binary variables in this formulation is much larger than the number of variables in (NAIVE): Instead of $|V \times P|$ variables we now have $\mathcal{O}(|V \times V|)$ variables that describe block-mapping. In order to describe the edge-colouring problem on the processor multigraph we need $|E \times E|$ variables, while we needed only $|E \times C| + |C|$ variables in (NAIVE). Both numbers are asymptotically $\mathcal{O}(|E^2|)$ but as we will see in Section 7 we usually have $|C| \ll |E|$.
- (REP) carries much more inequalities than (NAIVE): The incidence constraint is modelled by $\mathcal{O}(|E^3 \times V|)$ inequalities while the (NAIVE) model requires only $\mathcal{O}(|V^2 \times C \times P|)$ inequalities for this constraint. We may remedy this problem by using the same technique as in Section 6.2 and replace (6.15i) through (6.15j) by

$$\sum_{wu \in N(u)} y_{r_E, wu} + \sum_{wv \in N(v)} y_{r_E, wv} + x_{r_V, u} + x_{r_V, v} \leq 3 \quad u \neq v \in V, r_E \in E, r_V \in V \quad (6.18)$$

but this still leaves us with $\mathcal{O}(|V^3 \times E|)$ inequalities to represent the incidence constraint.

- It turns out that the objective function value of the LP-relaxation of instances of (REP) is much smaller than the objective function value of the same instance of (NAIVE). This indicates that (REP) is a weaker formulation than (NAIVE).

6.3.1 LP Relaxation

Again we start analysis of the model by investigating its LP-relaxation. The theorems and proofs in this section are very similar to Section 6.2.1. For sake of completeness we nevertheless spell out all details.

Consider (REP) where all variables have been relaxed to be continuous. Denote the new problem by (RLP).

Theorem 60. *For any optimal solution (x^*, y^*, b^*) to (RLP) we have $b^* \geq \kappa_V/|V|$.*

Proof. The proof for this theorem is similar to the proof of Theorem 56. This is due to the fact that in either case an optimal (fractional) solution is achieved if each block is distributed evenly over all available processors.

Assume that (x^*, y^*, b^*) is a feasible solution to (RLP) for which $b^* = K' < \kappa_V/|V|$. Then by (6.15d) we have $\sum_{u \in V} \kappa_u x_{r,u}^* \leq K'$ for all $r \in V$ and thus

$$\sum_{r \in V} \sum_{u \in V} \kappa_u x_{r,u}^* \leq |V| \cdot K' < \kappa_V.$$

On the other hand, since $\sum_{r \in V} x_{r,u}^* = 1$ for all $u \in V$ by (6.15b) we also have

$$\kappa_V = \sum_{u \in V} \kappa_u \cdot \sum_{r \in V} x_{r,u}^* = \sum_{r \in V} \sum_{u \in V} \kappa_u x_{r,u}^* < \kappa_V.$$

This contradiction proves the claim. □

Again there is an optimal solution for (RLP) that has objective function value $\kappa_V/|P|$:

Theorem 61. *The vector $(x^0, y^0, b^0) = ((1/|V|) \cdot \mathbf{1}, 0, \kappa_V/|V|)$ is the unique optimal solution to (RLP).*

Proof. Feasibility and optimality of (x^0, y^0, b^0) are obvious. To see uniqueness, assume there is another solution (x', y', b') with objective function value $\kappa_V/|V|$. From Theorem 60 we immediately conclude $b' = \kappa_V/|V|$ and $y' = 0$. The latter implies $x'_{r_V,u} = x'_{r_V,v}$ for all edges $uv \in E$ (see (6.15f) and (6.15g)). We fix a node $u_0 \in V$ and get — by the existence of a spanning tree rooted in u_0 — $x'_{r_V,u} = x'_{r_V,u_0}$ for all $u \in V$. Capacity restriction (6.15d) then implies

$$b' = \kappa_V/|V| \geq \sum_{u \in V} \kappa_u x'_{r_V,u} = \sum_{u \in V} \kappa_u x'_{r_V,u_0} = \kappa_V x'_{r_V,u_0}.$$

This yields $x'_{r_V,u_0} \leq 1/|V|$ and since $\sum_{r_V \in V} x'_{r_V,u_0} = 1$ by (6.15b) we get $x_{r_V,u_0} = 1/|V|$. So the two optimal solutions coincide and the claim is proved. □

Observe that we usually have $|P| \ll |V|$. Thus the lower bound provided by the LP relaxation of (REP) is in most cases considerably worse than the one provided by the LP relaxation of (NAIVE).

6.3.2 Valid Inequalities

Again we supplement our integer programming formulation by several valid inequalities. In order to make derivation of valid inequalities more easy, We assume here that $\sum_{r_E \in E} y_{e,uv} + x_{r_V,u} + x_{r_V,v} \leq 2$ for all $r_V \in V$ and $uv \in E$, i. e., an edge uv is coloured if and only if it is inter-processor. Notice that optimal solution to (REP) can be easily modified to conform to this assumption by uncolouring all intra-processor edges.

Some inequalities that were formulated for (NAIVE) in Section 6.2.2 are also easily formulated for (REP). Among these inequalities are (6.10), (6.11) and (6.12). For (REP) these inequalities are

$$\sum_{v \in N(u)} y_{r_E,uv} \leq y_{r_E,r_E} \quad u \in V, r_E \in E \quad (6.19)$$

$$\deg_G(u)x_{r_V,u} - \sum_{v \in N(u)} x_{r_V,v} \leq \sum_{r_E \in E} y_{r_E,r_E} \quad u \in V, r_V \in V \quad (6.20)$$

$$\sum_{u \in U} \left(\deg_G(u)x_{r_V,u} - \sum_{v \in N(u)} x_{r_V,v} \right) \leq \sum_{r_E \in E} y_{r_E,r_E} \quad U \subseteq V, r_V \in V. \quad (6.21)$$

Inequality (6.19) requires that edges that are incident to the same node in the grid graph G must not be in the same colour class. To understand (6.20) recall that if $u \in U$ is mapped to the processor represented by r_V , then each neighbour of u that is not mapped to the same processor introduces an inter-processor edge incident to the processor represented by r_V . All these inter-processor edges are incident to the same processor, thus the number of colours used must be at least as big as the number of them. Inequality (6.21) is based on the same fact as (6.20), but this time we formulate it for multiple nodes.

Theorem 62. *If $uv \in E$ and $r_0 \in V$, then inequalities*

$$\begin{aligned} \sum_{r_V \neq r_0} (x_{r_V,u} - x_{r_V,v}) &\leq \sum_{r_E \in E} y_{r_E,uv} \quad \text{and} \\ \sum_{r_V \neq r_0} (x_{r_V,v} - x_{r_V,u}) &\leq \sum_{r_E \in E} y_{r_E,uv} \end{aligned} \quad (6.22)$$

are valid for (REP).

Proof. It is obviously sufficient to prove validity of (6.22). To this end, observe that $\sum_{r_V \neq r_0} x_{r_V,u} \leq 1$ and $\sum_{r_V \neq r_0} x_{r_V,v} \leq 1$ by (6.15b). Moreover, if $\sum_{r_V \neq r_0} x_{r_V,v} = 1$ or if the left-hand side of (6.22) is zero, then the inequality is clearly satisfied. Assume $\sum_{r_V \neq r_0} x_{r_V,u} = 1$ and $\sum_{r_V \neq r_0} x_{r_V,v} = 0$. Then the left-hand side of (6.22) is one and block v is represented by another block than u (v is represented by r_0 and u by a node different from r_0). In other words edge uv must be inter-processor. Thus $\sum_{r_E \in E} y_{r_E,uv} \geq 1$ and the claim is proved. \square

Observe that $\sum_{r_V \neq r_0} x_{r_V, u} = 1 - x_{v_0, u}$ and thus inequalities (6.22) and (6.22) are just reformulations of (6.15f) and (6.15g).

The next valid inequality to be presented is not restricted to variables x and y . Instead it also involves the continuous variable b as well as the concrete block sizes.

Theorem 63. *Let $U = \{u_0, \dots, u_l\} \subseteq V$ be a set of $l \geq 2$ nodes such that $u_{l-1}u_l \in E$. Then*

$$\sum_{i=0}^{l-2} \kappa_{u_i} x_{u_i, u_l} + \kappa(U) \leq b + \sum_{i=0}^{l-2} \kappa_{u_i} \sum_{\substack{r_V \in V \\ r_V \neq u_l}} x_{r_V, u_i} + \kappa_{u_{l-1}} \sum_{r_E \in E} y_{r_E, u_{l-1}u_l} \quad (6.23)$$

is valid for (REP).

Proof. If $x_{r_0, u_l} = 1$ for $r_0 \in \{u_i : i = 0, \dots, l-2\}$ then nodes r_0 and u_l are mapped to the same processor. In this case, obviously $x_{u_i, u_l} = 0$ and thus $\sum_{\substack{r_V \in V \\ r_V \neq u_l}} x_{r_V, u_i} = 1$ for $i = 0, \dots, l-2$. Thus (6.23) reduces to

$$\kappa_{r_0} + \kappa_{u_{l-1}} + \kappa_{u_l} \leq b + \kappa_{u_{l-1}} \sum_{r_E \in E} y_{r_E, u_{l-1}u_l}.$$

If $u_{l-1}u_l$ is intra-processor (the right-hand side sum is zero), then all three nodes r_0 , u_{l-1} and u_l are mapped to the same processor and the inequality is satisfied. If otherwise $u_{l-1}u_l$ is inter-processor, we are left with $\kappa_{r_0} + \kappa_{u_l} \leq b$ which is obviously satisfied.

Assume now $x_{r, u_l} = 0$ for $r \in \{u_i : i = 0, \dots, l-2\}$. If additionally $u_{l-1}u_l$ is intra-processor, inequality (6.23) reduces to

$$\kappa(U) \leq b + \sum_{i=0}^{l-2} \kappa_{u_i} \sum_{\substack{r_V \in V \\ r_V \neq u_l}} x_{r_V, u_i}. \quad (6.24)$$

Observing that $\sum_{\substack{r_V \in V \\ r_V \neq u_l}} x_{r_V, u_i} = 0$ if and only if u_i is represented by u_l and equal to 1 otherwise, validity is obvious. If otherwise $u_{l-1}u_l$ is inter-processor, the left-hand side of (6.24) reduces to $\kappa(U) - \kappa_{u_{l-1}}$ and the inequality is satisfied for the same reasons as before. \square

If we choose $U = \{u_0, \dots, u_l\} = V$, assuming that u_{l-1} and u_l are adjacent in G , then this inequality (6.23) cuts off the optimal solution to the LP-relaxation (x^0, y^0, b^0) described in Theorem 61 above. To see this, we rewrite the inequality as

$$\sum_{r_V = u_0}^{u_{l-2}} \kappa_{r_V} x_{r_V, u_l} + \kappa(U) - \sum_{u = u_0}^{u_{l-2}} \kappa_u \sum_{r_V \neq u_l} x_{r_V, u} - \kappa_{u_{l-1}} \sum_{r_E \in E} y_{r_E, u_{l-1}u_l} \leq b \quad (6.25)$$

and obtain

$$\begin{aligned}
& \sum_{r_V=u_0}^{u_{l-2}} \kappa_{r_V} x_{r_V, u_l}^0 + \kappa(V) - \sum_{u=u_0}^{u_{l-2}} \kappa_u \sum_{r_V \neq u_l} x_{r_V, u}^0 - \kappa_{u_{l-1}} \sum_{r_E \in E} y_{r_E, u_{l-1} u_l}^0 \\
&= 0 + \kappa_V - \sum_{u=u_0}^{u_{l-2}} \frac{(|V|-1)\kappa_u}{|V|} - 0 \\
&= \frac{1}{|V|} \kappa_V + \frac{|V|-1}{|V|} (\kappa_{u_{l-1}} + \kappa_{u_l}) \\
&> \frac{1}{|V|} \kappa_V.
\end{aligned}$$

Since $b^0 = \kappa_V/|V|$ the optimal solution to the LP relaxation violates (6.25).

For model (REP) we can define a valid inequality similar to (6.14) which was valid for (NAIVE) (see page 109).

Theorem 64. *Let $u_0 \in V$ and set $U = \{u_0\} \cup N(u_0)$. Then inequality*

$$\kappa_{u_0} + (\kappa(U) - \kappa_{u_0})x_{u_0, u_0} \leq b + \sum_{uv \in \delta(u_0)} \kappa_v \sum_{e \in E} y_{e, uv} \tag{6.26}$$

is valid for (REP).

Proof. Validity is obvious if $x_{u_0, u_0} = 0$. So assume $x_{u_0, u_0} = 1$. Observing that $\sum_{e \in E} y_{e, uv} = 0$ for $uv \in \delta(u_0)$ implies that v is mapped to the same processor as u_0 , we easily see that (6.26) is satisfied by any feasible solution to (REP). \square

If $(|V|-1)\kappa_{u_0} > \kappa(V \setminus U)$, then the optimal solution (x^0, y^0, b^0) to the LP relaxation described in Section 6.3.1 violates (6.26), for in this case we have

$$\begin{aligned}
\kappa_{u_0} + (\kappa(U) - \kappa_{u_0})x_{u_0, u_0}^0 - b^0 - \sum_{uv \in \delta(u_0)} \kappa_v \sum_{e \in E} y_{e, uv}^0 &= \kappa_{u_0} + \frac{\kappa(U)}{|V|} - \frac{\kappa_{u_0}}{|V|} - \frac{\kappa(V)}{|V|} - 0 \\
&= \frac{|V|-1}{|V|} \kappa_{u_0} - \frac{\kappa(V \setminus U)}{|V|} \\
&> 0.
\end{aligned}$$

However, we can do better than simply adapting (6.14) from (NAIVE) to (REP) and obtain (6.26). To this end, observe that $x_{u, v} = 1$ for $u, v \in V$ with $u \neq v$ encodes the fact that blocks u and v are mapped to same processor. This leads to

Theorem 65. Let $u_0, v_0 \in V$ with $u_0 \neq v_0$ and $\kappa_{u_0} + \kappa_{v_0} \leq \kappa^*$. Set $U = N(u_0) \cup N(v_0) \cup \{u_0, v_0\}$ and $F = (\delta(u_0) \cup \delta(v_0)) \setminus \{u_0 v_0\}$. Moreover, for $f \in F$ let $w(f)$ denote the endpoint of f that is different from u_0 and v_0 . Then inequality

$$\kappa_{u_0} + \kappa_{v_0} + (\kappa(U) - \kappa_{u_0} - \kappa_{v_0})(x_{u_0, v_0} + x_{v_0, u_0}) \leq b + \sum_{f \in F} \kappa_{w(f)} \sum_{e \in E} y_{e, f} \quad (6.27)$$

is valid for (REP).

Proof. First of all observe that $x_{u_0, v_0} + x_{v_0, u_0} \in \{0, 1\}$ and that $x_{u_0, v_0} + x_{v_0, u_0} = 1$ implies that u_0 and v_0 are mapped to the same processor. By assumption $\kappa_{u_0} + \kappa_{v_0} \leq \kappa^*$ and (6.27) is valid if $x_{u_0, v_0} + x_{v_0, u_0} = 0$.

If $x_{u_0, v_0} + x_{v_0, u_0} = 1$, we observe as before that $\sum_{e \in E} y_{e, f} = 0$ for $f \in F$ implies that f is intra-processor and thus both endpoints of f are mapped to the same processor as u_0 and v_0 . Hence (6.27) is satisfied. \square

Plugging (x^0, y^0, b^0) into (6.27) yields

$$\begin{aligned} & \kappa_{u_0} + \kappa_{v_0} + (\kappa(U) - \kappa_{u_0} - \kappa_{v_0})(x_{u_0, v_0}^0 + x_{v_0, u_0}^0) - b - \sum_{f \in F} \kappa_{w(f)} \sum_{e \in E} y_{e, f}^0 \\ &= \kappa_{u_0} + \kappa_{v_0} + (\kappa(U) - \kappa_{u_0} - \kappa_{v_0}) \frac{2}{|V|} - \frac{\kappa(V)}{|V|} - 0 \\ &= \frac{|V| - 2}{|V|} (\kappa_{u_0} + \kappa_{v_0}) - \frac{\kappa(V) - 2\kappa(U)}{|V|}. \end{aligned}$$

Thus (6.27) cuts off the optimal solution to the LP relaxation of (REP) if $(|V| - 2)(\kappa_{u_0} + \kappa_{v_0}) > \kappa(V) - 2\kappa(U)$.

6.4 Slot Model

A third formulation of (OGPC) as mixed integer program arises by combination of models (SLOTMAP) and (MATCOL). This is easily done since in (SLOTMAP) the multiplicity of an edge $pk \in M_P$ is given by

$$\mu_{M_P}(pk) = \sum_{e \in E} x_{e, \{p, k\}}. \quad (6.28)$$

To obtain an integer programming model we again fix an arbitrary orientation on $G = (V, E)$ and call the new graph $G = (V, E)$ as well. For the edge-colouring part of (OGPC) we use $[\mathcal{M}^*(|P|)]$, the set of equivalence classes of all maximum cardinality matchings that are possible in multigraphs on $|P|$ nodes. If $\mu_{M_P}(pk)$ is the multiplicity of edge $pk \in M_P$ (we

set $\mu_{M_P}(pk) = 0$ if p and k are not adjacent in M_P), the edge-colouring problem requires us to select for each matching $[M] \in [\mathcal{M}^*(|P|)]$ a multiplicity $\lambda_{[M]} \in \mathbb{N}$ such that

$$\sum_{[M] \in [\mathcal{M}^*(|P|, pk)]} \lambda_{[M]} \geq \mu_{M_P}(pk) \quad pk \in E(K_{|P|}). \quad (6.29)$$

This immediately gives rise to integral variables $z_{[M]}$ that count how often the matchings represented by $[M]$ are used in the edge-colouring of M_P . We also need binary variables $x_{e,p,k}$ for all $e \in E$ and $(p, k) \in P \times P$ that are 1 if and only if edge e is mapped to slot (p, k) and 0 otherwise. Together with b these variables allow us to formulate (OGPC):

1. Each edge $e \in E$ must be mapped to exactly one slot and we require

$$\sum_{p,k \in P} x_{e,p,k} = 1$$

for all $e \in E$.

2. Next we must observe, that mapping edge uv to slot pk restricts the slots to which edges in $\delta(uv)$ can be mapped: Edges in $\delta^+(u)$ must then start in p and edges in $\delta^-(u)$ must end there. Likewise, edges in $\delta^+(v)$ must start in k and edges in $\delta^-(v)$ must end there. This gives rise to inequality

$$\sum_{k \in P} x_{uv,p,k} = \sum_{k \in P} x_{e,p,k} \quad \text{for } p \in P, e \in \delta^+(u).$$

Similar inequalities must be added for uv and $\delta^-(u)$ as well as for uv and $\delta^+(v)$ and $\delta^-(v)$. Notice that uv is itself contained in $\delta^+(u)$ and $\delta^-(v)$, but we do not need an inequality in this case.

3. As in model (SLOTMAP) the sum of control volumes mapped to processor $p \in P$ is given by $\sum_{k \in P} (\sum_{e \in \delta^+(u)} \kappa'_u x_{e,p,k} + \sum_{e \in \delta^-(u)} \kappa'_v x_{e,k,p})$ and we therefore require (see also Section 5.1.3)

$$\sum_{uv \in E} \sum_{k \in P} (\kappa'_u x_{u,p,k} + \kappa'_v x_{v,k,p}) \leq b$$

for all $p \in P$.

4. The number of edges running between processors p and k in the processor multigraph is given by $\sum_{e \in E} (x_{e,p,k} + x_{e,k,p})$. As indicated in above each edge in the processor multigraph must be covered by a matching, so we demand that

$$\sum_{[M] \in [\mathcal{M}^*(|P|, pk)]} z_{[M]} \geq \sum_{e \in E} x_{e,\{p,k\}}$$

for $pk \in E(K_{|P|})$.

5. As before, the number of control volumes on the maximum processor is given by b . The sum of colours used in the edge-colouring of the processor multigraph is this time determined by $\sum_{[M] \in [\mathcal{M}^*(|P|)]} z_{[M]}$ and we therefore aim at minimising

$$t_a \cdot b + t_c \cdot \sum_{[M] \in [\mathcal{M}^*(|P|)]} z_{[M]}.$$

Summarising 1 through 5, we get:

(Slot)	$\text{minimise } t_a \cdot b + t_c \cdot \sum_{[M] \in [\mathcal{M}^*(P)]} z_{[M]} \quad (6.30a)$
	$\sum_{(p,k) \in P \times P} x_{e,p,k} = 1 \quad e \in E \quad (6.30b)$
	$\sum_{uv \in E} \sum_{k \in P} (\kappa'_u x_{uv,p,k} + \kappa'_v x_{uv,k,p}) \leq b \quad p \in P \quad (6.30c)$
	$\sum_{k \in P} x_{uv,p,k} - \sum_{k \in P} x_{f,p,k} = 0 \quad uv \in E, p \in P, f \in \delta^+(u) \quad (6.30d)$
	$\sum_{k \in P} x_{uv,p,k} - \sum_{k \in P} x_{f,k,p} = 0 \quad uv \in E, p \in P, f \in \delta^-(u) \quad (6.30e)$
	$\sum_{k \in P} x_{uv,k,p} - \sum_{k \in P} x_{f,p,k} = 0 \quad uv \in E, p \in P, f \in \delta^+(v) \quad (6.30f)$
	$\sum_{k \in P} x_{uv,k,p} - \sum_{k \in P} x_{f,k,p} = 0 \quad uv \in E, p \in P, f \in \delta^-(v) \quad (6.30g)$
	$\sum_{e \in E} (x_{e,\{p,k\}} \leq \sum_{[M] \in [\mathcal{M}^*(P ,pk)]} z_{[M]} \quad (p,k) \in P^2 \quad (6.30h)$
	$b \leq K \quad (6.30i)$
	$x, z \text{ binary} \quad (6.30j)$

The above model has several desirable properties that the previous models do not have:

- We explicitly have the multiplicity of an edge $pk \in M_P$ available: It is given by the sum (see also (6.30h))

$$\mu_{M_P}(pk) = \sum_{e \in E} x_{e,\{p,k\}} \quad (6.31)$$

where $x_{e,\{p,k\}} = x_{e,p,k} + x_{e,k,p}$. This allows us to easily formulate several aspects that are based on the multiplicity of edges in M_P (such as formulating the edge-colouring problem as set-covering problem).

- Similar to the multiplicities, the degree of a processor p in M_P is given by

$$\deg_{M_P}(p) = \sum_{e \in E} \sum_{k \neq p} x_{e,\{p,k\}}. \quad (6.32)$$

The ability to express the degree of nodes in M_P explicitly will be useful in the derivation and application of lower bounds (see Chapter 7).

- As we will see in Section 6.8, (SLOT) (and especially its variant (SLOT*) discussed below) allows us to formulate (OGPC) in a reasonably compact fashion.

Moreover, it will turn out that formulation (SLOT) of (OGPC) is easily extended to other hardware architectures. We will discuss this in Section 6.9 below.

Yet again, all these advantages come at some price:

- Instead of $|V \times P|$ or $\mathcal{O}(|V^2|)$ we now have $|E \times P^2|$ binary decision variables to model the mapping part of the problem.
- The size of the set $[\mathcal{M}^*(|P|)]$ grows very rapidly with the number of processors (see Theorem 2). Thus the number of z -variables will be extremely large for problems with a large number of processors. However, we will see that this problem can be remedied by using another formulation of the edge-colouring problem in multigraphs.

Model (SLOT) allows a relaxation that is not possible with the other models. To see this recall from Section 2.2 that

$$\chi^*(G_m) = \min \left\{ \sum_{M \in \mathcal{M}(G_m)} \lambda_M : \lambda \in \mathbb{R}_+^{\mathcal{M}(G_m)}, \sum_{M \in \mathcal{M}(G_m), e \in M} \lambda_M \chi^M = \mathbb{1} \right\}$$

for any multigraph G_m on n nodes and that also

$$\chi^*(G_m) \leq \min \left\{ \sum_{[M] \in [\mathcal{M}^*(n)]} \lambda_{[M]} : \lambda \in \mathbb{R}_+^{[\mathcal{M}^*(n)]}, \sum_{[M] \in [\mathcal{M}^*(n,e)]} \lambda_{[M]} \geq \mu(e) \text{ for all } e \in E \right\}.$$

Relaxing the integrality constraints on the z -variables in (SLOT) we get $\sum_{[M] \in [\mathcal{M}^*(|P|)]} z'_{[M]} \geq \chi^*(M_P)$ for any feasible solution (x', y', z', b') . Now remember the Goldberg-Seymour conjecture that claims $\chi'(G_m) = \max\{\Gamma(G_m), \Delta(G_m) + 1\}$ for any multigraph G_m . Since $\Gamma(G_m) = \chi^*(G_m)$ this would imply that $\sum_{[M] \in [\mathcal{M}^*(|P|)]} z'_{[M]}$ differs from the chromatic index of M_P by at most one. Provided we assume the Goldberg-Seymour conjecture to be true, we may simplify our problem formulation by relaxing the integrality constraints on the z -variables.

6.4.1 LP Relaxation

Again, theorems and proofs in this section are nearly identical to those in Sections 6.2.1 and 6.3.1 and we provide details only for sake of completeness.

Consider (SLOT) where all variables have been relaxed to be continuous. Denote the new problem by (SLP).

Theorem 66. *For any optimal solution (x^*, z^*, b^*) to (SLP) we have $b^* \geq \kappa_V/|P|$.*

Proof. Again the proof is similar to the proof of Theorem 56 and again this is due to the fact that an optimal fractional solution is to distribute each block evenly over the available processors.

Assume that (x^*, z^*, b^*) is a feasible solution to (SLP) for which $b^* = K' < \kappa_V/|P|$. Then by (6.30c) we have $\sum_{uv \in E} \sum_{k \in P} (\kappa'_u x_{uv,p,k}^* + \kappa'_v x_{uv,k,p}^*) \leq K'$ for all $p \in P$ and thus

$$\sum_{p \in P} \sum_{uv \in E} \sum_{k \in P} (\kappa'_u x_{uv,p,k}^* + \kappa'_v x_{uv,k,p}^*) \leq |P| \cdot K' < \kappa_V.$$

On the other hand, since $\sum_{p \in P} \sum_{k \in P} x_{uv,p,k}^* = 1$ for all $uv \in E$ by (6.30b) we also have

$$\begin{aligned} \kappa_V &= \sum_{uv \in E} \left(\kappa'_u \cdot \sum_{p \in P} \sum_{k \in P} x_{uv,p,k}^* + \kappa'_v \cdot \sum_{p \in P} \sum_{k \in P} x_{uv,p,k}^* \right) \\ &= \sum_{p \in P} \sum_{uv \in E} \sum_{k \in P} (\kappa'_u x_{uv,p,k}^* + \kappa'_v x_{uv,k,p}^*) \\ &< \kappa_V. \end{aligned}$$

The derived contradiction proves the claim. □

The next theorem shows that there is indeed an optimal solution to (SLP) with objective function value $\kappa_V/|P|$, thus the LP-relaxation of (SLOT) yields the same objective function value as the LP-relaxation of (NAIVE).

Theorem 67. *The vector (x^0, z^0, b^0) with*

$$x_{e,p,k}^0 = \begin{cases} 1/|P| & \text{if } p = k \text{ and} \\ 0 & \text{otherwise,} \end{cases}, \quad z^0 = 0, \quad b^0 = \kappa_V/|P|$$

is the unique optimal solution to (SLP).

Proof. Feasibility and optimality of (x^0, z^0, b^0) is obvious. To see uniqueness, assume that there is another solution (x', z', b') with objective function value $\kappa_V/|P|$. From Theorem 66 we conclude $b' = \kappa_V/|P|$ and thus $z' = 0$. By constraints (6.30d) through (6.30g) we obtain $x'_{e,p,p} = x'_{f,p,p}$ for any pair of incident edges $e, f \in E$. As G is connected, this implies $x'_{e,p,p} = x'_{e_0,p,p}$ for an arbitrary but fixed edge $e_0 \in E$ and all $e \in E$. By (6.30c) we have

$$b' = \kappa_V/|P| \geq \sum_{uv \in E} (\kappa'_u + \kappa'_v) x'_{uv,p,p} = \sum_{uv \in E} (\kappa'_u + \kappa'_v) x'_{e_0,p,p} = \kappa_V x'_{e_0,p,p}.$$

Thus $x_{e_0,p,p} = 1/|P|$ by (6.30b) and the claim is proved. \square

6.4.2 Polyhedral Analysis

Unlike before, some simple facts about the polyhedron defined by (SLOT) are easily established: For a grid graph $G = (V, E)$ let

$$P_s(G) := \text{conv} \left\{ (x, z, b) \in \{0, 1\}^{E \times P \times P} \times \mathbb{N}^{[\mathcal{M}^*(|P|)]} \times \mathbb{N} : (x, z, b) \text{ feasible for (SLOT)} \right\}$$

denote the polyhedron that is defined by the convex hull of feasible solutions to (SLOT). Moreover, let $Z \in \mathbb{N}^{[\mathcal{M}^*(|P|)]}$ denote the vector that has all coefficients equal to $|E|$, i. e., $Z = |E| \cdot \mathbf{1}$.

Theorem 68. *The dimension of $P_s(G)$ is $\dim(P_{\text{slot}}^{\text{map}}(G)) + |[\mathcal{M}^*(|P|)]|$.*

Proof. Obviously, the dimension of $P_s(G)$ cannot exceed $\dim(P_{\text{slot}}^{\text{map}}(G)) + |[\mathcal{M}^*(|P|)]|$. Assume that $\dim(P_{\text{slot}}^{\text{map}}(G)) = k$ and let $(x_0, b_0), \dots, (x_k, b_k)$ be $k + 1$ affinely independent points in $P_{\text{slot}}^{\text{map}}(G)$. Then (x_i, Z, b_i) is contained in $P_s(G)$. Moreover, for $[M] \in [\mathcal{M}^*(|P|)]$ point $(x_0, Z + \hat{z}_{[M]}, b_0)$ is also contained in $P_s(G)$. The $|[\mathcal{M}^*(|P|)]| + \dim(P_{\text{slot}}^{\text{map}}) + 1$ points $(x_0, Z + \hat{z}_{[M]}, b_0)$ for $[M] \in [\mathcal{M}^*(|P|)]$ and (x_i, Z, b_i) for $i = 0, \dots, k$ are easily seen to be affinely independent. Thus the claim is proved. \square

Theorem 69. *If for $G = (V, E)$ inequality $a^T x + cb \leq \alpha$ is facet-defining for $P_{\text{slot}}^{\text{map}}(G)$, then $a^T x + cb \leq \alpha$ is also facet-defining for $P_s(G)$.*

Proof. The proof of this theorem is very similar to the proof of Theorem 68. Assume that $\dim(P_{\text{slot}}^{\text{map}}(G)) = k$ and let $(x_1, b_1), \dots, (x_k, b_k)$ denote k affinely independent points in $P_{\text{slot}}^{\text{map}}(G)$ that satisfy $a^T x + cb \leq \alpha$ at equality. Obviously (x_i, Z, b_i) for $i = 1, \dots, k$ and $(x_0, Z + \hat{z}_{[M]}, b_0)$ for $[M] \in [\mathcal{M}^*(|P|)]$ are contained in $P_s(G)$ and satisfy $a^T x + cb = \alpha$. As these points are also affinely independent, the claim is proved. \square

6.4.3 Valid Inequalities

We now present valid inequalities for (SLOT). Some of these inequalities were already used in the previous models. Other inequalities are new and exploit the fact that we are able to make statements about the multiplicity of edges and the degree of processors in M_P by means of (6.31) and (6.32). Before we start describing inequalities, we first observe that

$$\gamma_{u,p} := \frac{1}{\deg_G(u)} \cdot \sum_{p \in P} \left(\sum_{e \in \delta_G^+(u)} x_{e,p,k} + \sum_{e \in \delta_G^-(u)} x_{e,k,p} \right) \quad \text{for } (u,p) \in V \times P \quad (6.33)$$

is equal to 1 if and only if node u is mapped to processor p and 0 otherwise. Observe that $\gamma_{u,p}$ here plays the same role as variable $x_{u,p}$ did in model (NAIVE). Thus, by means of $\gamma_{u,p}$, any inequality that is valid for (NAVMAP) can immediately be stated for (SLOT) as well.

Using $\gamma_{u,p}$ we can sort processors exactly as we did for (NAVMAP) in Section 5.1.1

$$\begin{aligned} \sum_{u \in V} \gamma_{u,p} &\geq \sum_{u \in V} \gamma_{u,p+1} & p = 0, \dots, |P| - 2 \text{ or} \\ \sum_{u \in V} \kappa_u \cdot \gamma_{u,p} &\geq \sum_{u \in V} \kappa_u \cdot \gamma_{u,p+1} & p = 0, \dots, |P| - 2 \text{ or} \\ \sum_{u \in V} (u+1) \cdot \gamma_{u,p} &\geq \sum_{u \in V} (u+1) \cdot \gamma_{u,p+1} & p = 0, \dots, |P| - 2. \end{aligned}$$

However, in (SLOT) we have yet another way to sort the processors, namely by their degree. In order to require that the first processor has the biggest degree, the second processor the second biggest and so on, we use (6.32) and get

$$\sum_{e \in E} \sum_{k \neq p} x_{e,\{p,k\}} \geq \sum_{e \in E} \sum_{k \neq p} x_{e,\{p+1,k\}} \quad p = 0, \dots, |P| - 2. \quad (6.34)$$

In Section 6.2 we described valid inequality (6.10) that was based on stars in the grid graph G . For model (SLOT) we have stronger variants of this inequality. These formulations are based on the following observation: for any edge $uv \in E$ and any processor $p \in P$, blocks u and v are both mapped to p if and only if $x_{uv,p,p} = 1$.

Theorem 70. *For all $uv \in E$ and $p \in P$ inequality*

$$(\deg_G(u) + \deg_G(v) - 2)x_{uv,p,p} \leq \sum_{e \in \delta(uv)} x_{e,p,p} + \sum_{[M] \in [\mathcal{M}^*(|P|,p)]} z_{[M]} \quad (6.35)$$

is valid for (SLOT).

Proof. The inequality is obviously satisfied if $x_{uv,p,p} = 0$. So assume $x_{uv,p,p} = 1$, in which case u and v are both mapped to p . Consequently each neighbour of u or v that is different from u and v and is not mapped to p as well implies an inter-processor edge that is incident to processor p . The sum of neighbours that are mapped to p is given by

$$\sum_{e \in \delta(uv)} x_{e,p,p}.$$

Moreover, it is clear that all inter-processor edges incident to p must be covered by a matching in $[\mathcal{M}^*(|P|, p)]$, hence inequality (6.35) is valid. \square

Observe that $|P|$ even implies $[\mathcal{M}^*(|P|, p)] = [\mathcal{M}^*(|P|)]$ as in this case each maximum cardinality matching on $K_{|P|}$ is perfect.

The next two valid inequalities for (SLOT) are similar to (6.35). As the proofs of validity are almost identical to the proof of Theorem 70 we leave them to the interested reader.

Theorem 71. *Inequality*

$$\begin{aligned} & (\deg_G(u) + \deg_G(v) - 2) \cdot x_{uv,p,p} & (6.36) \\ & \leq \sum_{l \neq p,k} \left(\sum_{e \in \delta^+(uv)} x_{e,p,l} + \sum_{e \in \delta^-(uv)} x_{e,l,p} \right) + \sum_{[M] \in [\mathcal{M}^*(|P|,pk)]} z_{[M]} \quad uv \in E, p, k \in P, p \neq k \end{aligned}$$

is valid for (SLOT).

Theorem 72. *For all $uv \in E$, $p, k \in P$ with $p \neq k$ inequality*

$$\begin{aligned} & (\deg_G(u) + \deg_G(v) - 1) \cdot x_{uv,p,k} & (6.37) \\ & \leq \sum_{l \neq k} \left(\sum_{e \in \delta^+(u)} x_{e,p,l} + \sum_{e \in \delta^-(u)} x_{e,l,p} \right) + \sum_{l \neq p} \left(\sum_{e \in \delta^+(v)} x_{e,k,l} + \sum_{e \in \delta^-(v)} x_{e,l,k} \right) + \sum_{[M] \in [\mathcal{M}^*(|P|,pk)]} z_{[M]} \end{aligned}$$

is valid for (SLOT).

6.5 (SLOT) on Four Processors

In case we have only four processors, we can apply a small reduction to the integer programming model (SLOT). To this end, consider the three maximal matchings in the complete graph on four nodes

$$\begin{aligned} M_1 &= \{01, 23\} \\ M_2 &= \{02, 13\} \\ M_3 &= \{03, 12\}. \end{aligned}$$

Now fix any multigraph G_0 on four nodes and set

$$e_i = \operatorname{argmax}_{e \in M_i} \mu_{G_0}(e) \quad \text{for } i = 1, 2, 3.$$

In other words, e_i is the edge in M_i that has maximal multiplicity. Then e_1 and e_3 are always incident and we can renumber the nodes in G_0 such that $e_1 = 01$ and $e_3 = 03$. For e_2 we have two different possibilities: either $e_2 = 02$ or $e_2 = 13$. We can thus assume without loss

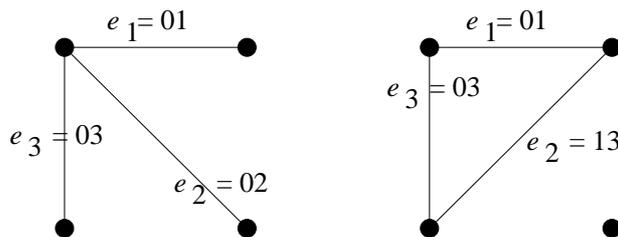


Figure 6.1: The different configurations into which the set of edges of largest multiplicity can be permuted.

of generality that the e_i are in one of the two configurations shown in Figure 6.1.

This observation leads to a special variant of (SLOT). This variant, called (SLOT4), is based on the same block-mapping formulation as (SLOT). However, the edge-colouring part of the problem can be phrased a little more compact:

(SLOT4)

$$\text{minimise } t_a \cdot b + t_c \cdot \left(z + \sum_{e \in E} (x_{e,(0,1)} + x_{e,(0,2)}) \right) \quad (6.38a)$$

$$\sum_{(p,k) \in P \times P} x_{e,p,k} = 1 \quad e \in E \quad (6.38b)$$

$$\sum_{uv \in E} \sum_{k \in P} (\kappa'_u x_{uv,p,k} + \kappa'_v x_{uv,k,p}) \leq b \quad p \in P \quad (6.38c)$$

$$\sum_{k \in P} x_{uv,p,k} - \sum_{k \in P} x_{f,p,k} = 0 \quad uv \in E, p \in P, f \in \delta^+(u) \quad (6.38d)$$

$$\sum_{k \in P} x_{uv,p,k} - \sum_{k \in P} x_{f,k,p} = 0 \quad uv \in E, p \in P, f \in \delta^-(u) \quad (6.38e)$$

$$\sum_{k \in P} x_{uv,k,p} - \sum_{k \in P} x_{f,p,k} = 0 \quad uv \in E, p \in P, f \in \delta^+(v) \quad (6.38f)$$

$$\sum_{k \in P} x_{uv,k,p} - \sum_{k \in P} x_{f,k,p} = 0 \quad uv \in E, p \in P, f \in \delta^-(v) \quad (6.38g)$$

$$\sum_{e \in E} x_{e,(2,3)} \leq \sum_{e \in E} x_{e,(0,1)} \quad (6.38h)$$

$$\sum_{e \in E} x_{e,(1,2)} \leq \sum_{e \in E} x_{e,(0,3)} \quad (6.38i)$$

$$\sum_{e \in E} x_{e,(0,2)} \leq z \quad (6.38j)$$

$$\sum_{e \in E} x_{e,(1,3)} \leq z \quad (6.38k)$$

$$b \leq K \quad (6.38l)$$

$$x, y \text{ binary} \quad (6.38m)$$

Most inequalities are the same as in (SLOT). We only changed the objective function (6.38a), dropped (6.30h) and added (6.38h) through (6.38k). Inequality (6.38h) requires that 01 is the edge of maximum multiplicity in M_1 and (6.38i) does the same for 03 and M_3 . Since we cannot make assumptions about the edge of maximum multiplicity in M_2 , we still need (6.30h) for this matching. This is the purpose of (6.38j) and (6.38k). As one can see, we only have one z -variable left (that for M_2) and consequently need to adjust the objective function accordingly.

In addition to having two variables less than (SLOT), the model (SLOT4) has another advantage: Many variables (namely all $x_{e,0,1}$, $x_{e,1,0}$, $x_{e,0,3}$ and $x_{e,3,0}$) appear directly in the

objective function and do not influence the objective function value only indirectly by way of coupling with z -variables. This can help to improve the performance of MIP solvers that are used to actually solve our problem.

Of course, this problem formulation rules out solutions in which 13 has higher multiplicity than 01 in M_P and those in which 12 has higher multiplicity than 03 in M_P . This must be kept in mind when we talk about symmetry handling in Section 6.7 below.

As (SLOT4) is only a slight modification of (SLOT) we refrain from explicitly (re-)presenting valid inequalities for this model. Instead we note that it is in all cases straightforward to adapt to (SLOT4) those inequalities that are valid for (SLOT).

6.6 (SLOT) on many Processors

One drawback of (SLOT) is the fact that we need $\lfloor |P|/2 \rfloor$ inequalities and one z -variable for each matching $[M] \in [\mathcal{M}^*(|P|)]$ (see (6.30h) in (SLOT)). Unfortunately, the number of maximal matchings in a complete graph grows very rapidly with the number of nodes in the graph (see Theorem 2). As you can read from Table 6.1 the number of disjoint matchings of

n	# matchings
4	3
6	15
8	105
10	945
12	10395
14	135135
16	2027025
18	34459425

Table 6.1: Number of pairwise disjoint maximum cardinality matchings in K_n .

maximum cardinality is already out of tractability for relatively small numbers of processors. So we must cope with large numbers of processors in a different way. The idea is to combine models (SLOTMAP) and (NAVCOL): For each edge $e \in K_{|P|}$ we introduce a variable $y_{e,c}$ that is 1 if e is coloured by c and 0 otherwise (notice that for a colour set C we only have $|P \times C|$ such variables and not $|E \times C|$ as in (NAIVE)). Furthermore, we introduce variables z_c that are 1 if c is used in the edge-colouring of M_P and 0 otherwise. The combination of (SLOTMAP) and (NAVCOL) then yields

(SLOT*)	$\text{minimise } t_a \cdot b + t_c \sum_{c \in C} z_c \quad (6.39a)$
	$\sum_{(p,k) \in P \times P} x_{e,p,k} = 1 \quad e \in E \quad (6.39b)$
	$\sum_{uv \in E} \sum_{k \in P} (\kappa'_u x_{uv,p,k} + \kappa'_v x_{uv,k,p}) \leq b \quad p \in P \quad (6.39c)$
	$\sum_{k \in P} x_{uv,p,k} - \sum_{k \in P} x_{f,p,k} = 0 \quad uv \in E, p \in P, f \in \delta^+(u) \quad (6.39d)$
	$\sum_{k \in P} x_{uv,p,k} - \sum_{k \in P} x_{f,k,p} = 0 \quad uv \in E, p \in P, f \in \delta^-(u) \quad (6.39e)$
	$\sum_{k \in P} x_{uv,k,p} - \sum_{k \in P} x_{f,p,k} = 0 \quad uv \in E, p \in P, f \in \delta^+(v) \quad (6.39f)$
	$\sum_{k \in P} x_{uv,k,p} - \sum_{k \in P} x_{f,k,p} = 0 \quad uv \in E, p \in P, f \in \delta^-(v) \quad (6.39g)$
	$\sum_{e \in E} x_{e,\{p,k\}} \leq \sum_{c \in C} y_{pk,c} \quad pk \in K_{ P } \quad (6.39h)$
	$\sum_{k \neq p} y_{pk,c} \leq z_c \quad p \in P, c \in C \quad (6.39i)$
	$b \leq K \quad (6.39j)$
	$x, y, z \text{ binary} \quad (6.39k)$

In this model constraints (6.39b) through (6.39g) are unchanged from model (SLOT). Inequality (6.30h) has been replaced by (6.39h) and (6.39i) and the objective function was adjusted to the modified z -variables. Constraint (6.39h) requires that each edge in M_P receives at least as many colours as its multiplicity demands. Constraint (6.39i) requires that incident edges do not receive the same colour in the edge-colouring of M_P and that an edge can only receive a colour that is used. This constraint is the star inequality already presented in (5.56) in Section 6.2

Notice that though similar, this formulation of the edge-colouring problem is much more compact than the one in (NAIVE). Table 6.2 below shows that we need much fewer variables and constraints to state the problem. The reason for this is that we have the multiplicity of the edges in M_P directly available by means of (6.31). This allows us to formulate the edge-colouring problem *directly* on M_P and we need not bother with the edges of the grid graph G . Especially the incidence constraint that required $\mathcal{O}(|V^2 \times P \times C|)$ inequalities in (NAIVE) can be formulated in a much more compact fashion here.

	(NAIVE)	(SLOT*)
y -variables	$\mathcal{O}(E \times C)$	$\mathcal{O}(P \times C)$
z -variables	$\mathcal{O}(C)$	$\mathcal{O}(C)$
Force y constraints	$\mathcal{O}(E \times P)$	$\mathcal{O}(P \times P)$
Incidence constraints	$\mathcal{O}(V^2 \times P \times C)$	$\mathcal{O}(P)$

Table 6.2: Number of variables and constraints that formulate the minimal edge-colouring problem for M_P in the models (NAIVE) and (SLOT*).

6.6.1 Valid Inequalities

Formulation (SLOT*) can be enhanced by several valid inequalities that we describe now. Recall from Section 2.2 that

$$\chi'(M_P) \geq \Gamma(M_P) = \max_{P' \subseteq P} \left\lfloor \frac{|E[P']|}{\lfloor \frac{|P'|}{2} \rfloor} \right\rfloor$$

where $P' \subseteq P$ is a processor subset of odd size greater than 1. This implies that the *odd set inequality*

$$\sum_{\substack{p,k \in P' \\ p \neq k}} \left(\sum_{e \in E} x_{e,\{p,k\}} \right) \leq \left\lfloor \frac{|P'|}{2} \right\rfloor \cdot \sum_{c \in C} z_c \quad (6.40)$$

is valid for (SLOT*) if $|P'|$ is odd and larger than 1.

A special case of (6.40) is where $|P'| = 3$. In this case, the edges connecting processors in P' form a triangle and we obtain the *triangle inequality*

$$\sum_{\substack{p,k \in P' \\ p \neq k}} \left(\sum_{e \in E} x_{e,\{p,k\}} \right) \leq \sum_{c \in C} z_c. \quad (6.41)$$

Derived from the fact that $\chi'(M_P) \geq \Delta(M_P)$ is the *degree inequality*

$$\sum_{k \neq p} \sum_{e \in E} x_{e,\{p,k\}} \leq \sum_{c \in C} z_c \quad p \in P \quad (6.42)$$

All of the above inequalities *directly* couple x - and z -variables (without intermediate y -variables) and are therefore quite useful in raising the lower bound in Branch-and-Cut trees.

In Chapter 8 below we will see more valid inequalities for model (SLOT) or (SLOT*). The inequalities presented there will be based on connectivity properties of the grid graph and will turn out to be quite strong in practice.

6.7 Symmetry in MIP Models

We already mentioned time and again that model-intrinsic symmetry can become a large problem when solving model instances by means of a Branch-and-Cut algorithm. Recall for example model (NAIVE). Given a feasible solution (x, y, z, b) , any permutation $\pi : P \rightarrow P$ of the processor set or any permutation $\sigma : C \rightarrow C$ of the colour set yields a new feasible solution (x', y', z', b) where

$$\begin{aligned}x'_{u,p} &= x_{u,\pi(p)} \\y'_{e,c} &= y_{e,\sigma(c)} \\z'_c &= z_{\sigma(c)}.\end{aligned}$$

Obviously the new solution has the same objective function value and is thus equivalent. This implies that a Branch-and-Cut tree for an instance of (NAIVE) contains at least $|P|! \cdot |C|!$ equivalent optimal nodes. Exploring each of these nodes explicitly leads to massively redundant enumeration and thus an unnecessary increase in the algorithm's performance.

There are usually three different strategies to handle such symmetry issues in solution procedures based on Branch-and-Cut algorithms:

Reformulation The model is reformulated such that it does not suffer from intrinsic symmetry. An example for this technique is model (MATCOL). As opposed to the edge-colouring formulation in (NAVCOL) this model does not use a colour set and is thus “immune” to permutations on the colour set. Instead it explicitly enumerates all colour classes that may be used. Reformulations for similar problems can for example be found in [128, 150, 95]. The downside of this approach is obvious: it usually involves models with an exponential number of variables that can no longer be handled explicitly, but must be taken care of by column generation or similar techniques.

Isomorphism pruning By considering the permutation group generated by the symmetrically equivalent solutions to a model instance, it is possible to define branching rules, that avoid enumeration of equivalent nodes [123, 124]. The idea here is to choose for each set S of symmetrically equivalent solutions a representative s_0 and to assert that the Branch-and-Cut algorithm only considers s_0 and ignores all other solutions $s \in S \setminus \{s_0\}$. A very recent and similar technique can also be found in [98].

Additional Constraints In many cases it is possible to fix certain variables or add new constraints that cut off (at least some of) the equivalent solutions [38, 156]. Compared to the two previous approaches this is the most “inexact” strategy, because it is usually not possible to remove all symmetry from a formulation by fixing variables or adding constraints. Moreover, there are often different, but mutually exclusive possibilities for fixing variables or adding constraints and we must therefore carry out computational experiments to find out which performs best.

With the representative models (REPMAP), (REPCOL) and (REP) as well as the edge-colouring model (MATCOL) we already presented reformulations of the block-mapping and edge-colouring problems that do not suffer from intrinsic symmetries. In the following we will now describe several possibilities to reduce symmetry by fixing variables and/or adding new constraints.

6.7.1 Variable Sorting

Assume that a feasible solution (x, y, z, b) to (NAIVE) is given and that $\kappa(p)$ denotes the number of control volumes mapped to processor p in this solution. It is obviously valid to require that $\kappa(p) \geq \kappa(k)$ if $p < k$, i. e., the processors are ordered by decreasing size. If $\kappa(p) \neq \kappa(k)$ for some $p \neq k$, then this new constraint no longer admits swapping p and k to obtain an equivalent feasible solution. So adding constraint

$$\sum_{u \in V} \kappa_u x_{u,p-1} \geq \sum_{u \in V} \kappa_u x_{u,p} \quad p = 1, \dots, |P| - 1 \quad (6.43)$$

remedies the symmetry issues described above. Notice however, that if $\kappa(p) = \kappa(k)$ then

$$\pi : P \rightarrow P, \pi(l) = \begin{cases} p & \text{if } l = k, \\ k & \text{if } l = p \text{ and} \\ l & \text{otherwise} \end{cases}$$

still is a permutation that transforms the current feasible solution to a different but equivalent one. Stated in a more abstract fashion, constraint (6.43) requires that the weighted sums of blocks mapped to processors are sorted in non-increasing order. While we used the weight function $w : V \rightarrow \mathbb{R}$, $w(u) = \kappa_u$ in (6.43), it is perfectly legal to use any other arbitrary weight function. Two obvious choices of alternate weights yield

$$\sum_{u \in V} x_{u,p-1} \geq \sum_{u \in V} x_{u,p} \quad p = 1, \dots, |P| - 1 \quad \text{and} \quad (6.44)$$

$$\sum_{u \in V} (u+1)x_{u,p-1} \geq \sum_{u \in V} (u+1)x_{u,p} \quad p = 1, \dots, |P| - 1. \quad (6.45)$$

Instead of the block size κ_u we use in (6.44) a constant weight for each block and in (6.45) the block's index as weight (notice that we increment the index by one to avoid that the first block has weight zero). In practice computational experiments must be used to identify the class of constraints that has the biggest (positive) impact on the Branch-and-Cut algorithm.

If we consider model (SLOTMAP) (or likewise (SLOT), (SLOT4) or (SLOT*)), then we find yet another way to sort processors: since the degree of processor p in the processor multigraph is given by

$$\deg_{M_P}(p) = \sum_{e \in E} \sum_{k \neq p} (x_{e,p,k} + x_{e,k,p})$$

we may sort processors by decreasing degree:

$$\sum_{e \in E} \sum_{k \neq p-1} (x_{e,p-1,k} + x_{e,k,p-1}) \geq \sum_{e \in E} \sum_{k \neq p} (x_{e,p,k} + x_{e,k,p}) \quad p = 1, \dots, |P| - 1. \quad (6.46)$$

Similar to processors, also colour classes may be sorted to reduce symmetry issues. Direct adaption of (6.44) and (6.45) yields

$$\sum_{e \in E} y_{e,c-1} \geq \sum_{e \in E} y_{e,c} \quad c = 1, \dots, |C| - 1 \quad \text{or} \quad (6.47)$$

$$\sum_{e \in E} (e+1)y_{e,c-1} \geq \sum_{e \in E} (e+1)y_{e,c} \quad c = 1, \dots, |C| - 1. \quad (6.48)$$

Yet again, any other weight function $w : E \rightarrow \mathbb{R}$ may be used and computational experiments must determine the best one.

6.7.2 Variable Fixing

Another possibility to address symmetry issues is by fixing certain variables. In the models (REPMAP) and (REPCOL) we already saw that requiring a representative to have a smaller index than the item represented eliminates all symmetry from the models. For the other block-mapping models we observe that – since processors may be ordered arbitrarily – we may without loss of generality require that block $u_0 \in V$ is mapped to processor p_0 , thereby fixing $x_{u_0,p_0} = 1$ in (NAVMAP) and $\gamma_{u_0,p_0} = 1$ in (SLOTMAP).

This being done, we may – again without loss of generality – require that block $u_1 \in V$ is mapped to processor p_0 or p_1 and so on. Additionally, for $i > 1$ we may require that block u_i is mapped to processor p_i only if one of the blocks u_0, \dots, u_{i-1} was mapped to processor p_{i-1} . Obviously, this reasoning can only be applied to $|P|$ blocks $u_0, \dots, u_{|P|-1}$ and yields for (NAIVE) (and likewise for (SLOTMAP))

$$x_{u_i,p_j} = 0 \quad i = 0, \dots, |P| - 1, j > i \quad (6.49)$$

$$x_{u_i,p_i} \leq x_{u_{i-1},p_{i-1}} \quad i = 1, \dots, |P| - 1. \quad (6.50)$$

If we manage to choose $u_0, \dots, u_{|P|-1}$ such that they are mapped to mutually different processors in each optimal solution then adding (6.49) and (6.50) to (NAIVE) makes the optimal solution unique. In other words, the Branch-and-Cut tree will contain exactly one optimal node. Non-optimal feasible solutions may however still be represented by multiple nodes.

Fixing certain blocks to certain processors also allows us to strengthen some of the valid inequalities presented in the previous sections. As an example we show how (5.19) can be strengthened if one block is fixed. Since the proof is completely analogous to the proof of Theorem 17 we omit it here.

Theorem 73. Let $\sigma : \{0, \dots, n-1\} \rightarrow V$ be an increasing block-ordering and assume that $k' \in \{0, \dots, n-1\}$ such that $k := \min\{l \in \{0, \dots, n-1\} : \kappa_{s(k')} + \sum_{i=1}^l \kappa_{s(i)} > \kappa_V\}$ satisfies $k < k'$. Similar to Theorem 17 define $U = \{s(l) : l = 0, \dots, k\}$ and $d = \kappa_{s(k')} + \kappa(U) - \kappa^*$. Then the combination of inequalities

$$x_{s(k'),p} = 1 \tag{6.51}$$

$$\kappa^* - kd + d \cdot \sum_{u \in V \setminus s(k')} x_{u,p} \leq b \tag{6.52}$$

is valid for $P_{nav}^{map}(G)$ for one single processor $p \in P$.

Two things must be observed from Theorem 73:

1. Without (6.51) inequality (6.52) is *not* valid. Both inequalities must be used in conjunction.
2. Equation (6.51) explicitly fixes block $s(k')$ to processor p . This potentially interferes with the sorting strategies for feasible solutions described above.

As before, we may apply similar arguments to colour classes in the integer programming models (NAVCOL) and (NAIVE). Instead of limiting the target processors to which blocks may be mapped, we limit the colour classes that may contain a certain edge (see also [38]). To this end we choose $|C|$ edges $e_0, \dots, e_{|E|-1}$ and require

$$y_{e_i, c_j} = 0 \quad i = 0, \dots, |E| - 1, j > i \tag{6.53}$$

$$y_{e_i, c_i} \leq y_{e_{i-1}, c_{i-1}} \quad i = 1, \dots, |E| - 1. \tag{6.54}$$

By the same arguments as before, we may render the node containing the optimal solution unique, provided that we choose $e_0, \dots, e_{|C|-1}$ such that these edges are in mutually different colour classes in each optimal solution.

While we usually have no idle processors in an optimal solution to (OGPC) or the block-mapping problem, the number $|C|$ is only an upper bound for the number of colours required to minimally edge-colour a (processor) multigraph. It is thus reasonable to require that an edge-colouring for a multigraph G may only use the first $\chi'(G)$ colours from C . For models (NAVCOL), (NAIVE) and (SLOT*) this would imply (see also [38])

$$z_{c-1} \geq z_c \quad c = 1, \dots, |C| - 1. \tag{6.55}$$

Observe that these constraints are compatible with (6.53) and (6.54), i. e., they may be used in conjunction.

Moreover, requiring (6.55) for each feasible solution to (NAVCOL), (NAIVE) and (SLOT*) immediately gives rise to further valid inequalities in these models. For example

$$\sum_{i=0}^k z_c \geq \sum_{i=1}^k (i+1)y_{e,c_i} \quad k = 0, \dots, |C| - 1 \quad (6.56)$$

is then valid for (NAVCOL). This is obvious since $y_{e,c_j} = 1$ implies that colours c_0, \dots, c_j are used and thus at least $j+1$ colours are currently applied to edges in the multigraph. Observe that (6.56) cuts off fractional solutions with $y_{e,c} = 1/|C|$. Inequality (6.56) was also formulated for the node-colouring problem in [38]. This paper also describes many other inequalities that make use of a colour sorting similar to (6.55) and are easily adapted from node-colouring to edge-colouring problems.

6.8 Comparison of Models

In Sections 6.2, 6.3 and 6.4 we have developed three different integer programming models for (OGPC). Each of the models has its own advantages and shortcomings, some of which have already been mentioned in the previous sections. In this section we compare the three models by aspects that are important when it comes to actually solving problem instances.

6.8.1 Symmetry

Both the (NAIVE)- as well as the (SLOT)-model are sensitive for symmetries in the block-mapping part of the problem: given a block-to-processor mapping, any permutation $\sigma_P : P \rightarrow P$ on the processor set P easily produces another mapping with the same objective function value. A similar problem arises for the edge-colouring part of (NAIVE) and the (SLOT*)-variant of (SLOT). Here each permutation $\sigma_C : C \rightarrow C$ on the colour set C produces a new feasible solution with the same objective function value from any feasible solution to (OGPC). This intrinsic symmetry suggests that any Branch-and-Cut tree for instances of (NAIVE) or (SLOT) will contain a lot of equivalent nodes and will thus require massively redundant enumeration. We suggested to remedy this problem by sorting processors and colour classes according to various criteria. Here we had to take care that we do not combine sorting criteria that are mutually exclusive. This care could easily be taken, but another problem arose when we performed computational experiments on the models augmented by sorting constraints. The sorting constraints rendered the LP relaxations of the problem instances much more harder. In other words, we had fewer nodes to enumerate but the computational burden to solve the LP relaxation increased at each node. This increased computational effort lead us to conclude that sorting does not improve the performance of Branch-and-Cut algorithms for our models.

With the (REP)-model that is based on representative formulations we proposed a model that avoids all the symmetry issues just described. Each block-processor mapping and each edge-colouring was represented by exactly one point in the space of feasible solutions. In these solutions a set of nodes on a processor was identified by a unique representative node and a colour class by a unique representative edge. Permutations of the processor set P or the colour set C did not affect the space of feasible solutions since these two parameters were not explicitly part of the problem formulation: C was not used at all and with respect to P it was only the size of P , i. e., the number of processor allowed, that mattered.

6.8.2 Model Size

We compare here the number of (integral) variables and constraints in the different models. All the models are potentially quite large. In general there are two different ways to cope with models that involve too many constraints to be handled efficiently:

Separation By using separation, constraints are handled in a *dynamic* fashion and are only added to the model instance if they are indeed violated during Branch-and-Cut search. We will explain this concept in more detail in Chapter 8 below.

Aggregation Multiple inequalities may be *aggregated*. This means we take multiple inequalities of the same sense (“ \leq ”, “ \geq ” or “ $=$ ”) and sum up the left- and right-hand sides up. This results in one single big inequality and each solution that satisfies the individual inequalities also satisfies the big one. However, there may be solutions that satisfy the big one, but not the individual ones. This is especially the case if we consider the LP relaxation of an integer program. Aggregation thus usually leads to weaker problem formulation and thus to inferior solver performance.

For the discussion in this section we consider neither separation nor aggregation but compare the models by the number of constraints and variables that occur if we state all inequalities explicitly.

In Table 6.3 you find the different numbers of binary and integer variables and constraints that are used in each of the models to express the different aspects of (OGPC). The table shows that the (SLOT)-model has asymptotically the most variables ($\mathcal{O}([\mathcal{M}^*(|P|)])$) and that (SLOT*) has the fewest ones if $P^2 < C$, which is usually the case. With respect to the number of constraints (REP) is the biggest ($\mathcal{O}(|V^3 \times E|)$) and (SLOT) and (SLOT*) (both have ($\mathcal{O}(|E \times P| \cdot \Delta(G))$) constraints) are the smallest. The table also indicates that the “bottleneck” in (NAIVE) and (REP) is the formulation of the incidence constraint of the edge-colouring problem. In both cases a very large number of inequalities is required to establish this constraint. In (SLOT*) this constraint gives rise to much fewer concrete inequalities since it can be formulated directly on the multigraph M_P . In (SLOT) no inequalities are

	Model	Map nodes to processors and meet capacity restrictions	Assign colours to inter-processor edges	Different colours for incident edges	Count the number of colours used	total
Variables	(NAIVE)	$\mathcal{O}(V \times P)$	$\mathcal{O}(E \times C)$	0	$\mathcal{O}(C)$	$\mathcal{O}(E \times C)$
	(REP)	$\mathcal{O}(V^2)$	$\mathcal{O}(E^2)$	0	0	$\mathcal{O}(E^2)$
	(SLOT)	$\mathcal{O}(E \times P^2)$	$\mathcal{O}(\mathcal{M}^*(P))$	0	0	$\mathcal{O}(\max\{ E \times P^2 , \mathcal{M}^*(P) \})$
	(SLOT*)	$\mathcal{O}(E \times P^2)$	$\mathcal{O}(P^2)$	0	$\mathcal{O}(C)$	$\mathcal{O}(E \times P^2)$
Constraints	(NAIVE)	$\mathcal{O}(V)$	$\mathcal{O}(E \times P)$	$\mathcal{O}(V^2 \times P \times C)$	$\mathcal{O}(E \times C)$	$\mathcal{O}(V^2 \times P \times C)$
	(REP)	$\mathcal{O}(V^2)$	$\mathcal{O}(E^2)$	$\mathcal{O}(V^3 \times E)$	0	$\mathcal{O}(V^3 \times E)$
	(SLOT)	$\mathcal{O}(E \times P \cdot \Delta(G))$	$\mathcal{O}(P^2)$	0	0	$\mathcal{O}(E \times P \cdot \Delta(G))$
	(SLOT*)	$\mathcal{O}(E \times P \cdot \Delta(G))$	$\mathcal{O}(P^2)$	$\mathcal{O}(P \times C)$	$\mathcal{O}(E \times P \cdot \Delta(G))$	$\mathcal{O}(E \times P \cdot \Delta(G))$

Table 6.3: Size of model instances in variables and inequalities to express the different aspects of (OGPC).

devoted to this constraint since it is implicit in the “set-covering by matching” formulation. However, in this model the number of variables is asymptotically extremely large.

6.8.3 LP Relaxations

We proved that the optimal objective function value for the LP relaxation of (NAIVE) and (SLOT) is of equal quality. In both cases this value is $\kappa_V/|P|$ which is equal to the number of control volumes on a processor in a perfect balancing of the computational load. On the other hand, instances of the (REP)-model have an optimal solution of their LP relaxation that has objective function value $\kappa_V/|V|$ which is usually considerably smaller than $\kappa_V/|P|$ since in most cases $|V| \gg |P|$. Moreover computational experiments showed that it takes a non-negligible amount of time in the Branch-and-Cut tree for an instance of (REP) to prove that the objective function is at least $\kappa_V/|P|$. So with respect to the lower bound provided by the LP relaxation (REP) is inferior to both (NAIVE) and (SLOT) and the latter two are equal. Observe however, that the lower bound provided by the two relaxations is no bigger than the trivial lower bound $\kappa_V/|P|$.

All optimal solutions to LP relaxations of our models shared one severe drawback: they did not imply any inter-processor edges. Instead the variables associated with block-mapping were fractional in a fashion that allowed all variables associated with inter-processor edges to be zero. This flaw of the LP relaxations already is an indicator for the fact that instances of our models are hard to solve in practice.

6.8.4 Conclusion

Due to the sheer size of the formulation of the incidence-constraint in (NAIVE) this model was ruled out quite early. Already for small numbers of processors (e. g., 4 processors) and moderate size grid graphs, explicitly enumerating these inequalities resulted in software

representations that did not fit into 2GB main memory. As also separating the incidence-constraints for (NAIVE) was of little help we dropped the model.

The same argument applies to (REP). Again the number of inequalities required to express the incidence-constraint of the edge-colouring problem is very large. Again we do not gain much by separating these constraints and therefore drop the model. Another reason why (REP) does not really suit us is the optimal objective function value of its LP relaxation. This value is worse than the one provided by the two other models. As we will see in Chapter 10 our heuristics usually find good solutions to (OGPC) and we will need Branch-and-Cut mainly as proof of quality. This means that we are interested in determining a lower bound for the objective function value of an optimal solution. When we want to find such a lower bound by means of Branch-and-Cut it is counter-productive to use models that have inferior LP relaxations.

This leaves us with the two models (SLOT) and (SLOT*) from the slot-family of models. In order to keep the problem instances at small size we use (SLOT) for small number of processors and (SLOT*) for larger numbers of processors. The number of processors at which we switch between models depends on the hardware that performs the optimisation process. Typically we use (SLOT) for up to eight processors and (SLOT*) otherwise. The models from the slot-family also have the advantage that they are easily extended to certain other hardware architectures. This extension is the topic of the next section.

6.9 Alternate Architectures

Of course there are other hardware architectures than the one assumed in this work. To some of the architectures our models can be easily adopted, other architectures cannot be formulated as (simple) extensions of our models and require new modelling devices. In this section we describe several well-known hardware setups and discuss if and how our models can be adapted to them. We describe the model extensions with respect to (SLOT) only. The other models can either be not extended or are extended similarly. For another interesting source about adaption of edge-colouring techniques to certain network topologies consider [79, 4].

6.9.1 Non-symmetric Data Exchange

In some applications data dependency is not symmetric. This means that the grid graph $G = (V, E)$ is directed rather than undirected. An edge $(u, v) \in E$ from block u to block v then models the fact that block u must send data to block v . Unlike before, such an edge does not imply communication requirement from v to u . If v is to send data to u then G

must also contain the edge (v, u) . Usually, a processor $p \in P$ may be either in sending or receiving state. This means that it cannot send data to a processor k_1 while it is receiving data from processor k_2 (even if $k_1 = k_2$). Due to this limitation, directed edges block two processors just as undirected edges do. Thus the models described here apply to directed grid graphs as well.

6.9.2 Heterogeneous Processors

It is not always the case that all processors available are identical. This especially happens if we perform our simulation on a cluster. In a heterogeneous network the arithmetic time for a block per iteration does not only depend on the size of the block but also on the processor on which the block is executed. We can account for this fact by using block sizes $\kappa_{u,p}$ for $u \in V$ and $p \in P$. These values are obtained from κ_u by multiplying it by a scalar s_p that measures the “speed” of processor p . On fast processors we would use $s_p \leq 1$ and on slow ones $s_p \geq 1$. Using this factor, blocks “appear” to be smaller on fast machines and are virtually larger on slow machines. This then models the fact that in the same amount of time a fast processor can handle more control volumes than a slow processor. Notice that by appropriately scaling the values $\kappa_{u,p}$ we can still safely assume that all processor have the same capacity K . Though we have different processor types, we still assumed that the network connections are all the same. Let us consider next, what happens if we have heterogeneous network connections.

6.9.3 Heterogeneous Network Connections

Not only the processors may be different. Also the connections between processors need not be the same. For example, several processors might be wired together directly while others are connected by Ethernet only. If this is the case, we can no longer assume that the setup time for a communication channel is the bottleneck and yields a reasonable estimate for the time required to satisfy an edge. Instead the time required to transfer data corresponding to an edge $e \in E$ may easily depend on the amount of the data that is to be transferred.

We show an extension of (SLOT) to heterogeneous network connections for the case $|P| = 4$ only and leave the other cases to the interested reader. If we have no more than four processors, then we can adopt (SLOT) by introducing new parameters:

ω_e	$e \in E$	The amount of data to be transferred on edge e
t_s^{pk}	$pk \in P^2$	Setup time between p and k
t_e^{pk}	$pk \in P^2$	Exchange time between p and k

We assume that the time required to satisfy edge $e \in E$ between processors $p \in P$ and $k \in P$

is given by $t_s^{pk} + \omega_e \cdot t_e^{pk}$ (we still assume that communication overhead for edges that have both endpoints on the same processor can be neglected). Our new objective function is then

$$\min \quad t_a \cdot b + \sum_{[M] \in [\mathcal{M}^*(|P|)]} z_{[M]} \quad (6.57)$$

and we must replace (6.30h) by

$$\sum_{[M] \in [\mathcal{M}^*(|P|, pk)]} z_{[M]} \geq \sum_{e \in E} (x_{e,p,k} \cdot (t_s^{pk} + \omega_e \cdot t_e^{pk}) + x_{e,k,p} \cdot (t_s^{kp} + \omega_e \cdot t_e^{kp})) \quad pk \in K_4. \quad (6.58)$$

Observe that t_c was dropped from the objective function (6.57) as communication overhead is instead accounted for in (6.58). In (6.58) we already incorporated the cases in which $t_s^{pk} \neq t_s^{kp}$ or which $t_e^{pk} \neq t_e^{kp}$. Observe also that z_M is no longer required to be integer. Instead it may take any non-negative real value. By (6.58) it is clear, that in a feasible solution (x', z', b') , the value z'_M yields the number of milliseconds required for the communication round given by M .

6.9.4 Non-Connected Processors

The assumption that all processors are connected directly is rather optimistic, especially when it comes to large numbers of processors. In many cases not all pairs of processors will be connected. Consider for example an architecture that is widespread in the family of parallel computers: a d -dimensional hypercube (see Figure 6.2). A d -dimensional hypercube

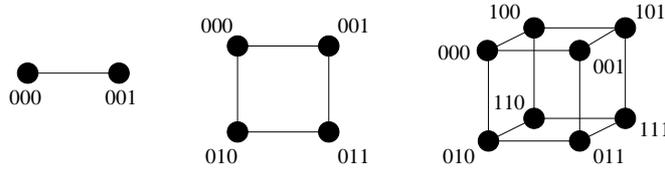


Figure 6.2: 1-, 2- and 3-dimensional hypercubes.

contains 2^d processors that are numbered 0 through $2^d - 1$. Two processors are wired together if and only if the binary representation of their processor number differs in one single bit. In such a hypercube architecture, each processor is connected to $2d$ others. Consequently, if two non-adjacent processors p and k need to exchange data during the simulation, they need not only a single connection but a whole path for the required communication channel. Moreover, if p and k start exchanging data, then all processors on this path are blocked and cannot exchange data for their own blocks since they are involved in the communication between p and k .

Let $C = (P, E_C)$ denote the (simple) graph that defines the available communication network. As depicted in Figure 6.3 we choose for any pair $p, k \in P$ with $p \neq k$ a p - k -path $P_{p,k}$ in C (if

multiple paths are available for a pair, we break ties arbitrarily but see the discussion about “Routed Communication” below). If p and k are adjacent we choose $P_{p,k} = \{pk\}$. Moreover,

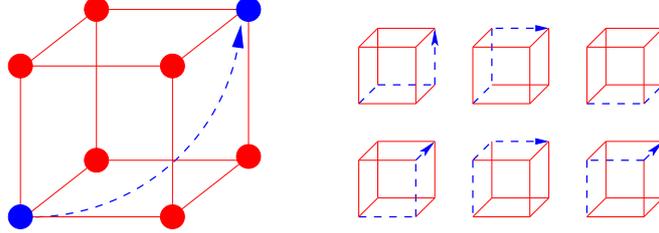


Figure 6.3: Different communication paths for non-adjacent processors in a 3-dimensional hypercube.

for $pk \in E_C$ with $p \neq k$ let P^{pk} denote the set of all paths that contain edge pk . Then the constraints described in (6.30h) turn into:

$$\sum_{P_{l,m} \in P^{pk}} \sum_{e \in E} (x_{e,l,m} + x_{e,m,l}) \leq \sum_{[M] \in [\mathcal{M}^*(|P|,pk)]} z_{[M]} \quad pk \in E_C. \quad (6.59)$$

Notice that if $P_{p,k} = \{pk\}$ for all $p, k \in P$ then this exactly yields (6.30h). Under the assumption that the communication paths are fixed for all processor pairs, we are thus able to extend (SLOT) to model this non-connected processors.

6.9.5 Routed Communication

In a more general setup, the path to use for communication between a processor pair p_1, p_2 is not predetermined. Instead there are multiple possibilities how the communication may be routed through the network and we are basically free to choose an arbitrary route. It may be even feasible to choose a *different* path for each edge of the grid graph that has one endpoint on p_1 and one on p_2 . This is the most complex assumption about the hardware since it adds one more degree of freedom to our optimisation problem, namely the route to be used to satisfy edge $e \in E$. Although this scenario is quite common in the real world, it is very complex to describe as an (integer) linear programming model and none of the models we described in this work can be easily extended to include the case of arbitrarily routed communication.

Chapter 7

Bounds

The pure and simple truth is
rarely pure and never simple.
— *Oscar Wilde*

Different classes of bounds can be used in order to speed up the solution process (especially the Branch-and-Cut algorithm):

Lower and upper bounds for the number of colours required: For models (NAIVE) and (SLOT*) described in Sections 6.2 and 6.4 we need an estimate for the number of colours that are required to optimally edge-colour the processor multigraph. This estimate is required *before* the Branch-and-Cut algorithm is started and the coarser this estimate is, the more binary variables the model instance contains, thus blowing up the Branch-and-Cut enumeration tree. If we can provide a good upper bound on $\chi'(M_P)$ we can save a considerable amount of time.

Lower bounds on $\chi'(M_P)$ are also useful because from them we can deduce lower and upper bounds on the optimal objective function value.

Lower and upper bounds for the objective function: If we have an incumbent solution in Algorithm 4 the difference between the incumbent's objective function value and a lower bound for the objective function gives us a feeling about the quality of the incumbent.

Upper bounds on the objective function on the other hand allow us to prune those nodes from the Branch-and-Cut tree that have an objective function value that is greater than the upper bound. This yields a speedup of the enumeration process.

Bounds on the processor size The number of elements that can be mapped to a processor is implicitly bounded from above by the capacity limit K . However, this capacity

limit is not necessarily tight. In other words, we are sometimes able to derive capacity limits that are smaller than K by analysing problem data or heuristic solutions. It is clear that a stricter capacity limit yields a stronger problem formulation. Moreover, we can derive bounds for the *number* of elements per processor (independent of their respective size). We already used such bounds in several simple valid inequalities. It is again obvious that stricter upper limits yield stronger problem formulations.

Before we start describing actual bounds, let us recall the trivial lower bound

$$\kappa^* \geq \kappa^{*'} = \left\lceil \frac{\kappa_V}{|P|} \right\rceil$$

from Section 3. Many of the bounds to be described here will utilise $\kappa^{*'}$. However, we can often provide stronger lower bounds than just dividing the number of control volumes by the number of processors. If we have a tighter lower bound $\kappa_2^{*'} > \kappa^{*'}$ then also all bounds that are derived from $\kappa^{*'}$ can be strengthened.

Before we start describing actual bounds, let us point out some interesting facts about bounds that are based on optimal solution of mixed integer programs (see Section 2.1). Assume we have a quantity Q that should be bounded from below and we know that $Q_l = \min\{c_Q^T x : x \in P_Q\}$ is a lower bound for Q for some cost function c_Q and polyhedron P_Q . If the minimisation problem is not a linear program the computation of Q_l is \mathcal{NP} -hard in general. In other words, finding $Q_l = \min\{c_Q^T x : x \in P_Q\}$ may consume a large amount of time in general. However, if we use the Branch-and-Cut algorithm described in Section 2.4 we can keep the computation time under control. Remember that the optimal solution q_n for LP-relaxation defined by a Branch-and-Cut node n yields a lower bound for Q_l . So we may stop the Branch-and-Cut algorithm at any time and use the smallest LP-relaxation value q^* as surrogate for Q_l . Since $q^* \leq Q_l$ and $Q_l \leq Q$, the value q^* yields a lower bound for Q as well. Using this strategy (and an analogous one for upper bounds and maximisation problems), we may spend a predefined amount of time for solving several mixed integer programs in order to obtain good bounds.

7.1 Bounds for the Number of Colours Required

The chromatic index of an arbitrary graph $G = (V, E)$ can be bounded by several parameters of the graph itself. Recall for example the two trivial bounds $\deg(G) \leq \chi'(G)$ or $\chi'(G) \leq |E|$. Since it is difficult to determine bounds for the chromatic index of the optimal processor multigraph directly, we attempt here to determine upper and lower bounds for several parameters of this graph which in turn yield bounds for the chromatic index.

We begin with upper bounds for $\chi'(M_P)$. As we will never need more colours than we have edges in G , we have the trivial upper bound $\chi'(M_P) \leq |E|$. Of course, this bound is very

coarse and we can do much better: Recall that our objective is to minimise

$$t_a \cdot b(M_P) + t_c \cdot \chi'(M_P) \tag{7.1}$$

and observe that $b(M_P) \geq \kappa^{*'}$ for any feasible processor mapping M_P . So if we have any upper bound z' on our objective function (7.1), we can easily derive an upper bound for $\chi'(M_P)$:

$$\begin{aligned} t_a \cdot b(M_P) + t_c \cdot \chi'(M_P) \leq z' &\Leftrightarrow t_c \cdot \chi'(M_P) \leq z' - t_a \cdot b(M_P) \\ &\Rightarrow t_c \cdot \chi'(M_P) \leq z' - t_a \cdot \kappa^{*'} \\ &\Leftrightarrow \chi'(M_P) \leq \frac{z' - t_a \cdot \kappa^{*'}}{t_c} \end{aligned} \tag{7.2}$$

Upper bounds for (7.1) can be efficiently computed by the heuristics we will describe in Chapter 9. As it turns out, the upper bounds for (7.1) delivered by the heuristics are rather tight and the resulting upper bounds for $\chi'(M_P)$ are pretty good. We therefore did no further investigations for upper bounds on $\chi'(M_P)$.

Let us now consider lower bounds for parameters $\Delta(M_P)$ and $E(M_P)$ of the optimal multi-graph M_P . Both bounds are easily translated into lower bounds for the chromatic index of M_P .

7.1.1 Lower Bounds on $\Delta(M_P)$

We now give several lower bounds on the maximal degree $\Delta(M_P)$ of the optimal processor multigraph. Since for any graph $\Delta(G) \leq \chi'(G)$, lower bounds on $\Delta(M_P)$ immediately yield lower bounds on $\chi'(M_P)$ and therefore for the colours required in an optimal edge-colouring.

Remember that we assume the input grid graph $G = (V, E)$ to be connected and that the time parameters t_a and t_c are such that an optimal solution to our problem leaves no processor idle. So if $|P| \geq 3$, it is clear that there must be at least one processor $p \in M_P$ for which $\deg_{M_P}(p) \geq 2$. Since $\chi'(M_P) \geq \Delta(M_P) \geq \deg_{M_P}(p)$ we know that

$$\chi'(M_P) \geq 2 \tag{7.3}$$

if we have more than two processors.

A sometimes better lower bound on $\Delta(M_P)$ can be found by considering minimal cuts in G . To this end, let $\delta(p_0)$ denote the set of inter-processor edges leaving p_0 in M_P . Then $\delta(p_0)$ is a cut that separates the nodes on p_0 from all other nodes in G . Consequently, the size $|\delta(G)|$ of a minimal cut in G is a lower bound for $\delta(p_0)$ and since $\delta(p) \leq \Delta(M_P)$ for any processor $p \in P$ we have $|\delta(G)| \leq \Delta(M_P)$.

In order to get a better lower bound than just the size of a minimal cut in G we extend the concept just mentioned to highly connected subgraphs of G : To this end, assume that $C = (U, E[U])$ is a k -connected subgraph of G for some $U \subseteq V$ with $\kappa(U) > K$. Then it is clear, that there are two nodes, u_0 and u_1 say, in U that will be mapped to different processors. Without loss of generality let p_0 and p_1 be these processors. Then, since C is k -edge-connected, there are k edge-disjoint paths from u_0 and u_1 in G and each of these paths must leave p_0 at least once (it may reenter and leave it again). Since all paths are edge-disjoint we have $\deg_{M_P}(p_0) \geq k$ and $\deg_{M_P}(p_1) \geq k$ and consequently $\Delta(M_P) \geq k$.

For the next lower bound, observe that there must be at least one processor, p_0 say, that contains $\kappa^{*'}$ or more control volumes. We will restrict (OGPC) to this single processor, that is we want to map blocks to p_0 such that the load on p_0 is no bigger than K and at least $\kappa^{*'}$, while the number of inter-processor edges incident to p_0 is minimal. We determine this number by solving the following binary program, which is a simplification of (NAIVE):

(MINDEG)	$\text{minimise } \sum_{e \in E} y_e \tag{7.4a}$
	$\sum_{v \in V} \kappa_v x_v \leq K \tag{7.4b}$
	$\sum_{v \in V} \kappa_v x_v \geq \kappa^{*'} \tag{7.4c}$
	$x_u - x_v \leq y_{uv} \quad uv \in E \tag{7.4d}$
	$x_v - x_u \leq y_{uv} \quad uv \in E \tag{7.4e}$
	$x, y \text{ binary.} \tag{7.4f}$

The aim of this model is to determine a subset $U \subseteq V$ of nodes that has weight of at least $\kappa^{*'}$ and no more than K control volumes and has a minimal number of outgoing edges, i. e., $\delta(U)$ is minimal. In the model, a binary variable x_v is set to 1 if and only if $v \in U$ and to 0 otherwise. A variable y_e is set to 1 if and only if $e \in \delta(U)$ and to 0 otherwise.

The constraints given by (7.4b) and (7.4c) require that U contains at most K and at least $\kappa^{*'}$ control volumes while the inequalities described by (7.4d) and (7.4e) assert that y_e is set to 1 if exactly one endpoint of e is in U . Together with these considerations it is clear that the objective function (7.4a) simply counts the edges in $\delta(U)$.

It turns out that optimal solutions to this binary program are easily found and we can solve

it by simply handing it over to a state-of-the-art MIP solver such as CPLEX¹. Typically, a solution is then delivered within a few seconds.

7.1.2 Lower bounds on $|E_P|$

Apart from lower bounds on $\chi'(M_P)$ implied by lower bounds on $\Delta(M_P)$ we can derive lower bounds for the number of edges in the multicut. These bounds are not directly lower bounds on $\chi'(M_P)$ but if we recall that a matching in M_P contains at most $\lfloor |P|/2 \rfloor$ edges we have

$$\chi'(M_P) \geq \left\lceil \frac{l}{\left\lfloor \frac{|P|}{2} \right\rfloor} \right\rceil \quad (7.5)$$

for any lower bound l on $|E_P|$. So lower bounds on $|E_P|$ can be useful as well.

One lower bound on $|E_P|$ is derived by a more detailed analysis of highly connected subgraphs of G . Let $C = (C_U, E_U)$ be a k -connected subgraph of G for some U_C with $\kappa(U_C) > q \cdot K$ for some integer $q \geq 1$. Then mapping U_C requires at least $q + 1$ processors (see Section 7.1.1 above) and each of these processors will have a degree of at least k . So we can conclude that

$$|E_P| \geq \left\lceil \frac{(q + 1) \cdot k}{2} \right\rceil \quad (7.6)$$

since each edge $e \in E_P$ is incident to at most two processors.

7.2 Objective Function Bounds

As we already indicated in Section 2.4, upper bounds on the optimal objective function play a crucial role in improving the performance of Branch-and-Bound and Branch-and-Cut algorithms. In our work we find such upper bounds by finding good feasible solutions to (OGPC). This in turn is achieved by the heuristics that we will describe in Chapter 9. The bounds derived by these algorithms not only improve the performance of Branch-and-Bound and Branch-and-Cut, but can in some cases also be used to strengthen the problem formulation: Assume that z' is an upper bound on the optimal objective function value. By (7.1) we get

$$\begin{aligned} t_a \cdot b(M_P) + t_c \cdot \chi'(M_P) \leq z' &\Leftrightarrow t_a \cdot b(M_P) \leq z' - t_c \cdot \chi'(M_P) \\ &\Leftrightarrow b(M_P) \leq \frac{z' - t_c \cdot \chi'(M_P)}{t_a}. \end{aligned} \quad (7.7)$$

¹See <http://www.ilog.com>.

So if $l_{\chi'}$ is a lower bound on $\chi'(M_P)$ and $(z' - t_c \cdot l_{\chi'})/t_a < K$ then we can replace K by $(z' - t_c \cdot l_{\chi'})/t_a$ in the problem formulation. This reduces the maximal feasible capacity of the processors and leads to a stronger problem formulation.

The most simple kind of lower bound for the objective function value of (OGPC) arises from adding up lower bounds. If l_a is a lower bound for the maximum number of control volumes per processor and l_c is a lower bound on the chromatic index of M_P , then no solution with objective function value smaller than $t_a \cdot l_a + t_c \cdot l_c$ exists. If only one of the bounds l_a or l_c is available we may use the trivial bounds κ^{*l} or 2 for the missing value.

Since upper bounds are produced by feasible solutions returned by heuristic algorithms, we will in the following concentrate on lower bounds for the optimal objective function value.

7.2.1 A Distinct Processor

We already mentioned that in each feasible solution there must be a processor that bears at least κ^{*l} control volumes. Without loss of generality we assume that p_0 is this processor and restrict our optimisation problem to this processor. That is we bipartition the grid graph such that one partition bears at least κ^{*l} and no more than K control volumes. This partition represents p_0 and the other partition the remaining processors. Each edge in the multicut of the bipartition found is incident to p_0 . So if we keep the nodes on p_0 fixed and mapped all nodes from the other partition to the remaining processors, then an optimal communication schedule for the mapping would still require as many communication rounds as we have in the multicut of the initial bipartition. It is thus clear that

$$t_a \cdot \kappa(p_0) + t_c \cdot \delta(p_0) \tag{7.8}$$

(where $\kappa(p_0)$ is the sum of control volumes in partition p_0 and $\delta(p_0)$ is the number of edges cut in the optimal bipartitioning) is a lower bound on the optimal objective function value of (OGPC).

In order to compute a bipartition for which (7.8) is minimal, we use the following integer program:

(LOWERBOUND)	$\text{minimise } t_a \sum_{u \in V} \kappa_u x_u + t_c \cdot \sum_{e \in E} y_e \quad (7.9a)$
	$\sum_{u \in V} \kappa_u x_u \geq \kappa^{*'} \quad (7.9b)$
	$\sum_{u \in V} \kappa_u x_u \leq K \quad (7.9c)$
	$x_u - x_v \leq y_{uv} \quad uv \in E \quad (7.9d)$
	$x_v - x_u \leq y_{uv} \quad uv \in E \quad (7.9e)$
	$x, y \text{ binary.} \quad (7.9f)$

Apart from the objective function, this integer program is the same as (MINDEG).

Again, instances of this program are sufficiently simple such that they can be solved by simple application of MIP solvers without the requirement to design a sophisticated Branch-and-Cut algorithm beforehand.

7.3 Bounding Processor Size

Various valid inequalities for integer programming formulations of (OGPC) involved the term $\kappa^*(V, |P|)$, i. e., the minimal number of control volumes that a maximal processor bears when the blocks in V are mapped to $|P|$ processors. Inequalities using this term occurred especially for models (NAVMAP) and (SLOTMAP). A trivial lower bound for the minimum size of a maximal processor in a feasible solution is $\kappa_V/|P|$. Observing that all block sizes are integral we may immediately raise this to $\lceil \kappa_V/|P| \rceil$. However, in some cases we can do even better: Let

$$g := \gcd_{u \in V} \{ \kappa_u \}$$

denote the greatest common divisor of the block sizes. Then $\kappa(p)$ — the number of control volumes mapped to processor $p \in P$ — must be divisible by g . Thus, in each feasible solution there is at least one processor that bears at least

$$\left\lceil \frac{\kappa_V/g}{|P|} \right\rceil \cdot g \quad (7.10)$$

control volumes. If $g = 1$ this bound is equal to the bound presented above. If however $g > 1$ this bound is strictly greater than the previous one. Moreover, determining (7.10) requires

only $|V|$ applications of the Euclidean algorithm to find $\gcd_{u \in V}\{\kappa_u\}$. Hence this term can be efficiently computed. Moreover, all results presented in this thesis that involved $\kappa^{*'}$ are easily checked to stay valid and yield stronger formulations if we replace $\kappa^{*'}$ by (7.10).

In Chapter 5 we saw that the number of edges that can be in the same colour class is limited by $\lfloor |P|/2 \rfloor$. An upper bound on the maximum number of blocks per processor was however not exhibited in Chapter 5 (although we used this bound to define valid inequalities). We now describe how an upper limit on the maximum number of elements per processor can be determined – provided that an upper limit u^* on the optimal objective function is known. We determine the block limit by solving an integer program that is similar to (NAIVE), but considers only one processor. To this end, we introduce binary decision variables x_u for each block $u \in V$, that are set to 1 if block u is mapped to the processor and 0 otherwise. Similar to (NAIVE), we also use binary variables y_e for each edge $e \in E$ that are 1 if edge e has exactly one endpoint on the processor and 0 otherwise. Our aim is to map as many blocks as possible to the processor, while the weighted sum of control volumes and inter-processor edges must not exceed u^* . In other words, we use an approach similar to (MINDEG) in Section 7.1: the control volumes mapped to our processor yield a lower bound on b and the inter-processor edges a lower bound on $\Delta(M_P)$. Consequently, the weighted sum of both yields a lower bound on the corresponding objective function value. We have to solve the following optimisation problem:

(MAXELEMENTS)

$$\begin{aligned} & \text{maximise } \sum_{u \in V} x_u \\ & t_a \cdot \sum_{u \in V} \kappa_u \cdot x_u + t_c \cdot \sum_{e \in E} y_e \leq u^* \\ & x_u - x_v \leq y_{uv} \quad uv \in E \\ & x_v - x_u \leq y_{uv} \quad uv \in E \\ & x, y \text{ binary.} \end{aligned}$$

Again, this problem proves sufficiently simple in practice and can be solved in a few seconds by state-of-the-art solvers.

Chapter 8

Connectivity in (SLOT) and its relatives

Furious activity is no substitute for understanding.
— *H. H. Williams*

In Chapters 5 and 6 we already presented several valid inequalities for the (SLOT)-family of models and in Chapter 7 we showed that considering certain connected structures of the grid graph $G = (V, E)$ yields lower bounds on the processor degree or the number of edges in M_P . In this chapter we will now investigate connectivity in the grid graph G more closely and will see that connected substructures of G give rise to cutting planes. While defining these cutting planes, we will exploit the fact that we can model the situation “edge e starts in p and ends in k ” by setting $x_{e,p,k} = 1$. Modelling the same situation in (NAIVE) or (REP) is not that easy and would require multiple variables to be set. This is another reason why we prefer using the family of models defined in Section 6.4 over the naive and the representative model.

Most of the cutting planes we discuss here give rise to a large (sometimes exponential) number of inequalities and can thus not be handled explicitly. In order to handle these cuts efficiently we also discuss separation algorithms for them.

Recall the similarity between the problem of optimally mapping blocks and Multiple Knapsack Problem and General Assignment Problem. Given that these problems are closely related, one may think that such prominent valid inequalities like cover- or $(1, k)$ -configuration inequalities that are valid for Multiple Knapsack Problem and General Assignment Problem can also be used for (SLOT). Unfortunately, this is not true and we will give a short explanation why these inequalities are almost certainly never violated in optimal fractional solutions to (SLOT).

8.1 Edge-Connected Covers

For certain classes of subsets $U \subseteq V$ of nodes, we can determine a lower bound on the number of edges that must be inter-processor in the subgraph $C = (U, E_C)$. One of these subsets are covers, i. e., subsets $U \subseteq V$ with $\kappa(U) > K$. These sets were already exploited in the derivation of lower bounds (see Chapter 7). In this section we will now show how we can derive valid inequalities from them. Before we start discussing inequalities, recall that for each edge pk in the processor multigraph M_P we have $\mu_{M_P}(pk) = \sum_{e \in E} x_{e, \{p,k\}}$ and for each node $p \in M_P$ we have $\deg_{M_P} p = \sum_{k \neq p} \sum_{e \in E} x_{e, \{p,k\}}$. In other words, multiplicity and degree of edges and nodes in M_P are properties that can be directly determined from the x -variables of a solution. Moreover, an edge e in the grid graph is inter-processor if $\sum_{p \neq k} x_{e, \{p,k\}} = 1$ and intra-processor otherwise (in which case $\sum_{p \in P} x_{e, p,p} = 1$).

8.1.1 Tree Cover Inequality

One of the simplest edge-connected subgraphs of connected graphs are trees. If $T = (U_T, E_T)$ is a tree and U_T is a cover, then not all nodes in U_T can be mapped to the same processor and there are two distinct nodes $u, v \in U_T$ that are mapped to different processors, p_u and p_v say. The path between u and v must therefore leave p_u at some edge and must enter p_v at some edge (the edge leaving p_u may well be the one entering p_v). Since this is true for all pairs of nodes in U_T we know that at least one edge of E_T must be inter-processor:

Theorem 74 (Tree Cover Inequality). *Let $T = (U_T, E_T)$ be a tree in G such that $\kappa(U_T) > K$. Then the Tree Cover Inequality*

$$\sum_{\substack{(p,k) \in P^2 \\ p \neq k}} \sum_{e \in E_T} x_{e,p,k} \geq 1 \quad (8.1)$$

is valid for (SLOT).

Observe that exploiting connectivity in subtrees of graphs in order to obtain valid inequalities for graph-partitioning problems is not new. In [47, 48] for example several classes of valid inequalities were derived from subtrees of the graph to be partitioned.

8.1.2 k -Connected Cover Inequality

In the derivation of the Tree Cover Inequality we made use of the fact, that a tree is 1-edge-connected, i. e., there is a path between any two nodes in the tree. This idea can be carried further if we do not restrict ourselves to trees, but consider any k -edge-connected subgraph instead:

Theorem 75 (*k*-Connected Cover Inequality). *If $C = (V_C, E_C) \subseteq G$ is a k -edge-connected subgraph of G , such that $\kappa(V_C) > K$, then the k -Connected Cover Inequality*

$$\sum_{p \neq k} \sum_{e \in E_C} x_{e,p,k} \geq k \quad (8.2)$$

is valid for (SLOT).

Proof. Since C is k -edge-connected, there are k edge-disjoint paths between any two nodes in V_C . Moreover, since C is a cover, there are again two nodes, u and v say, that are mapped to different processors p_u and p_v . Each of the k edge-disjoint paths between these two nodes must leave p_u and enter p_v (again an edge may be leaving p_u and entering p_v at the same time). Since this is true for any two nodes in V_C , we know that k of the edges in E_C are leaving p_u and k of them will be entering p_v . Consequently, at least k of the edges in E_C are inter-processor in any feasible solution to our problem. \square

8.1.3 k -Connected q -Cover Inequality

If $C = (V_C, E_C)$ is not only k -edge-connected but also a q -cover with $q > 1$, then we can push the number of inter-processor edges further:

Theorem 76 (*k*-Connected q -Cover Inequality). *Assume that $C = (V_C, E_C) \subseteq G$ is a k -edge-connected subgraph of G such that $\kappa(V_C) > q \cdot K$ for some integer $q > 1$. Then the k -Connected q -Cover Inequality given by*

$$\sum_{\substack{(p,k) \in P^2 \\ p \neq k}} \left(\sum_{e \in E_C} x_{e,p,k} \right) \geq \left\lceil \frac{(q+1) \cdot k}{2} \right\rceil \quad (8.3)$$

is valid for (SLOT).

Proof. Since $C = (V_C, E_C)$ is a q -cover, any feasible solution to our problem contains $q + 1$ nodes $u_0, \dots, u_q \in V_C$ that are mapped to pairwise different processors $p_0 \neq p_1 \neq \dots \neq p_q$. As C is also k -edge-connected we know (see the proof of (8.2)) that each of these processors has degree at least k in M_P . Consequently, there must be at least $\left\lceil \frac{(q+1) \cdot k}{2} \right\rceil$ inter-processor edges in M_P . \square

Another way to exploit k -connected q -covers is prescribing minimal degrees for processors: Let $U \subseteq V$ such that $C = (U, E[U])$ is a k -edge-connected subgraph of G with $\sum_{u \in U} \kappa_u > qK$ for some integer $q > 0$. Now consider some $u_0 \in U$ and assume that it is mapped to $p_0 \in P$. Since $q > 0$ not all nodes in U can be mapped to p_0 as well. So there is at least one node u_1

that is mapped to a processor $p_1 \in P$ different from p_0 . Since C is k -edge-connected, there are k edge-disjoint paths P_1, \dots, P_k between u_0 and u_1 and each of these paths P_i leaves p_0 with an edge e_i and the edges e_i are pairwise different. Consequently, p_0 and p_1 have minimal degree k in the processor-multigraph.

If we now have $q > 1$, then the two processors p_0 and p_1 do not have enough capacity to hold all elements from U . Hence the argument just made for u_0, u_1 and p_0, p_1 also applies to a third node $u_2 \neq u_0, u_1$ and a third processor $p_2 \neq p_0, p_1$ and we conclude that p_2 also has minimal degree k in the processor multigraph. Applying the same observation to $q > 2, \dots, |P| - 1$ leads to the following valid inequalities:

$$\sum_{k \in P} \sum_{e \in E'} (x_{e,p,k} + x_{e,k,p}) \geq k \quad p = 0, \dots, q \quad (8.4)$$

where $(V[E'], E')$ is a k -edge-connected subgraph of G with $\kappa(V[E']) > qK$.

Notice that (8.4) implicitly sorts the processors: If $q = 1$ it yields a minimal degree for p_0 , if $q = 2$ a requirement for p_0 and p_1 and so on. Consequently, this inequality may interfere with other inequalities that assume that processors can be permuted arbitrarily. Notice that the lower bound on $\Delta(M_P)$ that is implied by (8.4) for $q = 1$ was already exploited in Section 7.1.

8.1.4 Sparser Variants of Cover Inequalities

Unfortunately, the inequalities based on k -edge-connected subgraphs $C = (V_C, E_C)$ described so far are quite dense: They all are of the form

$$\sum_{pk \in E(K_{|P|})} \sum_{e \in E_C} x_{e,(p,k)} \geq l \quad (8.5)$$

for some $l \in \mathbb{N}$. Notice that all these inequalities cut off the optimal solution to the LP relaxation of (SLOT) or (SLOT*) that was described in Section 6.4.1. This is because the inequalities *explicitly* require inter-processor edges, while the optimal solution of the LP relaxation does not have inter-processor edges.

However, the left-hand side of (8.5) involves a large number of variables, namely $\mathcal{O}(|E_C| \cdot |P|^2)$ ones. There is an easy way to sparsify these inequalities: Recall from (6.30b) that $\sum_{(p,k) \in P^2} x_{e,p,k} = 1$ for all $e \in E$. So (8.5) is equivalent to

$$\sum_{p \in P} \sum_{e \in E_C} x_{e,p,p} \leq |E_C| - l. \quad (8.6)$$

It is clear that (8.6) is sparser than (8.5): it contains only $\mathcal{O}(|E| \cdot |P|)$ variables on its left-hand side. This reduction of the number of coefficients can lead to a speedup in the

LP solver and therefore in the Branch-and-Cut process. And (8.6) still cuts off the optimal solution x^* to the LP relaxation of (SLOT) and (SLOT*) that was discussed in Section 6.4.1. This is easily seen as the left-hand side of (8.6) is always equal to $|E_C|$ for x^* .

As opposed to dense inequalities, sparse inequalities require fewer space in computer memory and can be handled more efficiently by LP/MIP solvers. We thus apply only the sparsified versions of the above inequalities in our code.

8.1.5 Per-Processor Version of Connected Cover Inequalities

In Sections 8.1.1 through 8.1.4 we discussed several valid inequalities that were defined over edge-connected covers of the grid graph G and required a certain number of inter-processor edges. We now use the same substructures of G (i. e., edge-connected covers) to derive inequalities that bound from below the number of edges that must leave or enter certain processors.

In order to further reduce the number of non-zero coefficients in inequality (8.6) we may restrict this inequality to a single processor.

Theorem 77. *If $C = (V_C, E_C)$ is a k -edge-connected subgraph of $G = (V, E)$ with $\kappa(V_C) > K$, then inequality*

$$\sum_{e \in E_C} x_{e,p,p} \leq |E_C| - k \quad p \in P \quad (8.7)$$

is valid for (SLOT).

Proof. Consider an arbitrary but fixed processor $p_0 \in P$ and a feasible solution x' . If no node from V_C is mapped to p_0 , then (8.7) is satisfied. So assume that at least one node $u_0 \in V_C$ is mapped to p_0 . Since V_C is a cover, there is a node $u_1 \in V_C$ that is not mapped to p_0 . Since C is k -edge-connected at least k of the edges in E_C must be inter-processor edges incident to p_0 . Consequently, the number of edges from E_C that are inter-processor edges on p_0 is bounded from above by $|E_C| - k$ and inequality (8.7) is satisfied. \square

Unfortunately inequality (8.7) no longer has the property that it cuts off the optimal LP solution discussed in Section 6.4.1. It only cuts off this solution if $k > |E_C| \cdot (1 - 1/|P|)$.

Another way to use k -edge-connected subgraphs of G to define valid inequalities that involve processors more explicitly is as follows.

Assume that $T = (V_T, E_T)$ is a tree in G with $\kappa(V_T) > K$. Further let $u \in V_T$. For ease of exposition we assume that – in the orientation we initially fixed on G – all edges in E_T are oriented away from u in (see Figure 8.1). Notice however that this is not a requirement, the

inequality presented below can be adjusted in an obvious way to match any orientation of G . Now if node u is mapped to processor p , then at least one of the edges in E_T must leave processor p , since V_T is a cover. Recalling the definition of $\gamma_{u,p}$ from page 118 we get that

$$\sum_{k \neq p} \sum_{e \in E_T} x_{e,p,k} \geq \gamma_{u,p} \quad p \in P \quad (8.8)$$

is valid for (SLOT). Moreover, if we recall the optimal solution (x^*, z^*) for the LP-relaxation of (SLOT) we exhibited in Section 6.4.1, then (8.8) cuts off this solution. This is because $\gamma_{u,p}^* = 1/p$ for all $u \in V_T$ and all $p \in P$, while the left-hand side of (8.8) is zero in all these cases. In order to obtain a tree $T = (V_T, E_T)$ for (8.8) we have implemented three different strategies (we assume that $\kappa_V > K$):

1. For a node $u \in V$, compute T as the spanning tree rooted in u .
2. For a node $u \in V$ start with $T = (\{u\}, \emptyset)$. In each step, find the edge $vw = \operatorname{argmax}\{\kappa_w : v \in V_T, w \notin V_T, vw \in E\}$ and add vw to T . This step is repeated until $\kappa(V_T) > K$. A tree obtained by this strategy usually contains fewer edges than a spanning tree for G and thus yields stronger inequalities.
3. Separate (8.8) (see below).

Notice that the fewer edges the tree $T = (V_T, E_T)$ contains the stronger inequality (8.8) is. This is because if we add (8.8) for multiple trees T_1, \dots, T_l for which $\bigcap_{i=1}^l T_i \neq \emptyset$, then we can satisfy all inequalities of type (8.8) defined over the T_i by making a *single* edge in $\bigcap_{i=1}^l T_i \neq \emptyset$ inter-processor. If however all T_i were disjoint, then each tree would require its own inter-processor edge to satisfy the corresponding instance of (8.8).

As before, we can not only apply the above ideas to 1-edge-connected covers (trees) but can extend them to k -edge-connected covers as well. To this end, let $C = (V_C, E_C)$ be a k -edge-connected subgraph with $\kappa(V_C) > K$. Fix a node $u \in V_C$ and for each $v \in V_C \setminus \{u\}$ fix k edge-disjoint u - v -paths $P_{u,v,1}, \dots, P_{u,v,k}$ in C . Let $P_{u,v,i}^+$ denote the edges that occur in $P_{u,v,i}$ in forward direction (with respect to the initial orientation fixed on G) and $P_{u,v,i}^-$ those that occur in backward direction. If $P^+ = \bigcup_{i=1}^k \bigcup_{v \in V_C \setminus \{u\}} P_{u,v,i}^+$ and $P^- = \bigcup_{i=1}^k \bigcup_{v \in V_C \setminus \{u\}} P_{u,v,i}^-$

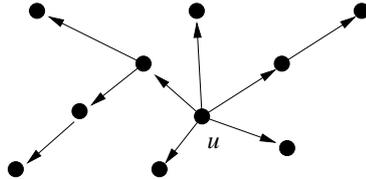


Figure 8.1: A tree in which each edge is oriented away from u .

(notice that an edge $e \in E_C$ might be in both of these sets) then the following inequality is valid for (SLOT):

$$\sum_{l \neq p} \sum_{v \in V_C \setminus \{u\}} \left(\sum_{e \in P^+} x_{e,p,l} + \sum_{e \in P^-} x_{e,l,p} \right) \geq k \cdot \gamma_{u,p} \quad p \in P. \quad (8.9)$$

To see validity observe that the right-hand side of (8.9) is k if node u is mapped to processor p and 0 otherwise. If u is mapped to p , then there is at least one node $v \in V_C \setminus \{u\}$ that is not mapped to p , since C is a cover. For this node the k edge-disjoint paths $P_{u,v,i}$ must leave processor p at some edge. Since all the variables corresponding to these edges are contained in the left-hand side of (8.9), the inequality is satisfied. As for trees it is desirable to have the k -edge-connected subgraph for which we introduce inequality (8.9) as small as possible since this yields the strongest formulations.

Once we know that p_0 has minimal degree k (refer to Chapter 7 to learn how things like this can be known) the following fact is easily established: Among the other processors in $P \setminus \{p_0\}$ there must be a processor, p_1 say, that has minimal degree $\lceil \frac{k}{|P|-1} \rceil$ and there are at least this many edges running between p_0 and p_1 . This is true because all the k inter-processor edges incident to p_0 must be incident to some processor from $P \setminus \{p_0\}$ (since otherwise the edges would not be inter-processor). Consequently, k inter-processor edges are incident to processors in $P \setminus \{p_0\}$, hence there is one processor p_1 that is incident to at least $\lceil \frac{k}{|P|-1} \rceil$ of them. Moreover, all these edges start in p_0 and end in p_1 , which leads to the following valid inequality:

$$\sum_{e \in E} x_{e,\{p_0,p_1\}} \geq \left\lceil \frac{k}{|P|-1} \right\rceil \quad (8.10)$$

Notice that we did not state an inequality that required processor p_0 to have a minimal degree of k . In order to enforce such a property, we can easily derive an appropriate inequality from (6.32).

8.2 Separating Valid Inequalities

As the careful reader has probably noticed by now, the actual number of inequalities generated by a valid inequality may be exponential in the input size in most cases. For example, if $G = (V, E) = K_n$ then any subset $U \subseteq V$ yields a $(|U| - 1)$ -edge-connected subgraph $(U, E[U])$ of G and any of these subgraphs that contains more than one node might be a cover. Thus we cannot add these inequalities to our optimisation problem *explicitly*. Instead we handle them *implicitly* by separation. Separation is a powerful tool in integer programming that was independently introduced by Gomory [69, 71] and Danzig, Fulkerson, and Johnson [31].

The idea of separation is similar to the cutting plane algorithm described in Chapter 2: We do not add the inequalities in question explicitly. Instead we solve the LP relaxation of the problem without considering any of these inequalities. Afterwards, we check all the inequalities whether they are violated by the optimal solution of the LP relaxation. If so, we add the violated inequalities to the LP relaxation and reoptimise. The advantage of this approach is that we add only those inequalities that are in fact violated and do not bother with those that are implicitly satisfied. This usually keeps the LP problems to solve at a moderate size.

Testing *all* instances of a valid inequality for violation would amount to enumerating all of them. As there are exponentially many of them, this is not a feasible approach. Instead one (and so did we) develops efficient heuristics that have good chances to find a violated cut (if there is one) but have no guarantee to find one. Thus, if the heuristic does not find a cut we do not know whether there is none or if the heuristic simply did not find it.

In order to separate an inequality we assume that we are given a fractional (optimal) solution (x^*, z^*) to the continuous relaxation at the current Branch-and-Cut node. For ease of exposition we define

$$x_e^* = \sum_{\substack{(p,k) \in P^2 \\ p \neq k}} x_{e,p,k}^*$$

for all $e \in E$. If $x_e^* = 1$ then e is inter-processor, if $x_e^* = 0$ then e is intra-processor. If $x_e^* \in]0, 1[$ it somehow measures the amount to which e is inter-/intra-processor.

The first separation algorithm is for the Tree Cover Inequality (8.1).

8.2.1 Separating Tree Covers

Our separation scheme for Tree Cover Inequalities (8.1) roughly follows the algorithm described in [48]. Given the fractional solution x^* we want to find a tree $T = (V_T, E_T)$ with $V_T \subseteq V$ and $E_T \subseteq E$ such that $\sum_{u \in V_T} \kappa_u > K$ (the nodes in T form a cover) and $\sum_{e \in E_T} x_e^* < 1$ (the tree cover inequality is violated). This calls for a tree with many nodes but few edges. In order to cope with these two contradicting objectives we create an auxiliary directed graph $G' = (V, E')$ by replacing each edge in E by a pair of antiparallel directed edges. On the edges of G' we define

$$w : E' \rightarrow \mathbb{R}, (u, v) \mapsto \frac{x_{uv}^*}{\kappa_v}. \quad (8.11)$$

This weight function w assigns small weights to edges with small values in x^* and to edges that are incident to “big” nodes. In order to separate a violated tree cover inequality we start with a singleton set $V_T = \{u\}$ for some node $u \in V$ and set $E_T = \emptyset$. We then search

for the edge

$$(u, v) = \operatorname{argmin}\{w(u, v) : (u, v) \in G', u \in V_T, v \notin V_T\}$$

and add v to V_T and (u, v) to E_T . This step is repeated until either $\kappa(V_T) > K$, $\sum_{e \in E_T} x_e^* \geq 1$ or no further edge is found. In the first case, we have found a violated tree and in the latter two cases the procedure fails.

More formally, we use the following algorithm to find a violated tree for the Tree Cover Inequality:

Algorithm 6 *Separation of violated Tree Cover Inequalities*

Input: Grid graph $G = (V, E)$, fractional solution x^* , start node $u \in V$, threshold value f (see below)

Output: A violated tree $T = (V_T, E_T)$ or \emptyset

```

1  Set  $V_T = \{u\}$ ,  $E_T = \emptyset$ ,  $s = \kappa_u$ .
   // Variable  $y$  keeps track of  $\sum_{e \in E_T} x_e^*$ 
2  Set  $y = 0$ .
3  While  $s \leq K$  Do
4    Set  $(u, v) = \operatorname{argmin}\{w(u, v) : (u, v) \in G', u \in V_T, v \notin V_T, x_{uv}^* < f\}$ .
5    If no such edge exists Then set  $(u, v) = \operatorname{argmin}\{w(u, v) : (u, v) \in G', u \in V_T, v \notin V_T\}$ .
6    If no such edge exists Then Return  $\emptyset$ .
7    Set  $y = y + x_{uv}^*$ .
8    If  $y > 1$  Then Return  $\emptyset$ .
9    Set  $s = s + \kappa_v$ .
10   Set  $V_T = V_T \cup \{v\}$  and  $E_T = E_T \cup uv$ .
11 End While
12 Return  $(V_T, E_T)$ 

```

Notice the slight difference to the textual description above (and the algorithm in [48]) that occurs in step 4: Instead of finding $uv = \operatorname{argmin}\{w(uv) : u \in V_T, v \notin V_T\}$ in one shot we first find $u_1v_1 = \operatorname{argmin}\{w(uv) : u \in V_T, v \notin V_T, x_{uv}^* \leq f\}$. If such an edge is found it is used to augment T . Otherwise we find (and use) $u_2v_2 = \operatorname{argmin}\{w(uv) : u \in V_T, v \notin V_T, x_{uv}^* > f\}$. The rationale behind this is as follows: Due to the definition of our weight function w , an edge $e \in E$ that has high value x_e^* but is incident to very big nodes might look too attractive to the algorithm. Thus we define a threshold $f \in [0, 1]$ and prefer augmentation with edges for which the corresponding value of x^* is below this threshold.

8.2.2 Separating Biconnected Covers

Unfortunately, finding a violated 2-edge-connected cover inequality is much harder than finding a violated tree-cover inequality. Instead of starting with an empty edge set and finding a violated inequality from scratch, we start with a violated tree-cover inequality and try to augment the tree to a biconnected subset of edges. In this situation the following fact is useful (see also Figure 8.2):

Theorem 78 (Biconnectivity augmentation for trees, [77]). *If $T = (V, E)$ is an undirected and unweighted tree, then the following algorithm augments T to a 2-edge-connected structure using a minimal number of edges:*

1. *DFS-traverse T and label the nodes when they are first visited. Let $L = \{l_1, \dots, l_k\}$ be the leaves of T , sorted by increasing DFS-number.*
2. *For $i = 1, \dots, \lfloor k/2 \rfloor$ connect leaves l_i and $l_{i+\lceil k/2 \rceil}$ by adding edge $\{l_i, l_{i+\lceil k/2 \rceil}\}$ to T .*
3. *If k is odd then pick some leaf $l \in L$ with $l \neq l_{\lceil k/2 \rceil}$ and add edge $\{l, l_{\lceil k/2 \rceil}\}$ to T , i. e., connect the middle leaf to one of the other leaves.*

The algorithm in Theorem 78 is a simplified version of the biconnectivity algorithm presented in [77]. The original algorithm takes as input an *arbitrary* graph, computes its *cactus representation* [107], extends DFS traversal to cactus graphs and applies the above algorithm to the cactus representation of the graph. Using this algorithm, the edge-connectivity of any graph can be increased by 1. Besides the algorithm presented in Theorem 78 there are many other algorithms for edge-connectivity augmentation, see [57].

Clearly, the algorithm in Theorem 78 runs in time $\mathcal{O}(n)$, where n is the number of nodes in T . However, the algorithm makes use of two strong assumptions that are not satisfied in our setting:

1. All potential edges to be added to T have the same weight.
2. In order to make T biconnected we are allowed to pick *any* two nodes and connect them by an edge, i. e., there is no prescribed edge set from which the augmenting edges must originate.

Both assumptions are clearly violated: As in the separation algorithm for the Tree-Cover inequality we would like to use the weight function w defined by (8.11) on the edges and find a minimum weight biconnected subgraph. Thus edges in G have different weights and, of course, we may only pick edges from G to augment T . If we introduce weights on the edges of a graph and aim at finding a connectivity augmentation of minimum weight, then the edge-connectivity augmentation problem becomes \mathcal{NP} -hard [46, 58] – even for trees.

However, we can easily adjust steps 2 and 3 of the algorithm given in Theorem 78 to meet our requirements: For a tree $T \subseteq G = (V, E)$ to be augmented and a current fractional solution x^* we define a weight function $w' : E \rightarrow \mathbb{R} \cup \{\infty\}$ by

$$w(e) = \begin{cases} \infty & \text{if } e \in T \text{ and} \\ x_e^* & \text{otherwise (see above for the definition of } x_e^* \text{).} \end{cases} \quad (8.12)$$

Under this weight function, a path has infinite length, if it uses an edge that is contained in the tree and finite length otherwise. For two leaves $l_1, l_2 \in L$ we do not simply add the edge $\{l_1, l_2\}$. Instead we find a path between l_1 and l_2 that is shortest with respect to the weight function w . If this path has infinite length our augmentation strategy failed and we stop. Otherwise we add the path to T , thus connecting leaves l_1 and l_2 (see Figure 8.2). Similarly if the number k of leaves is odd: We do not connect the middle leaf to an arbitrary other edge. Instead, we compute a shortest-path tree with respect to w that is rooted in $l_{\lceil k/2 \rceil}$. We then connect the middle leaf $l_{\lceil k/2 \rceil}$ to the leaf with the smallest distance. Again, if this distance is infinite then our algorithm fails and we stop. Notice that our modified algorithm not only adds edges to T but may also add nodes.

It is obvious, that if our algorithm succeeds, then the resulting subgraph of G will be biconnected. Moreover, if P_{l_1, l_2} and P_{l_3, l_4} are two paths chosen during the algorithms to connect leafs l_1 and l_2 as well as leafs l_3 and l_4 , then it is clear that these paths are not required to be edge-disjoint. As we want a biconnected subgraph with as few edges as possible we encourage non-edge-disjoint paths between leafs as follows: If a path P_{l_1, l_2} is added to connect leafs l_1 and l_2 , then the weights of all edges on P_{l_1, l_2} are set to 0, thus making them attractive for the next path to be added.

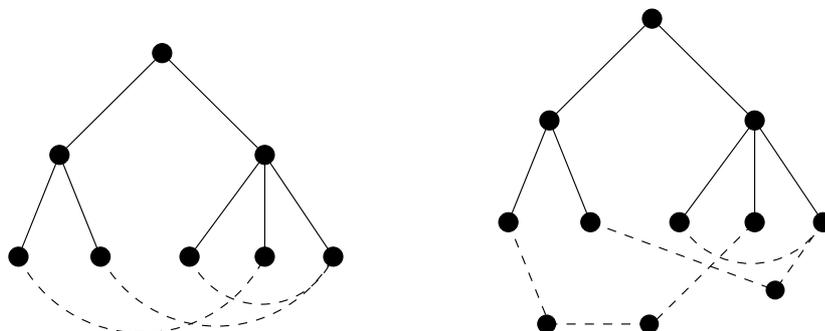


Figure 8.2: Tree augmented to a 2-edge connected subgraph. Using only edges (left picture) and using paths (right picture).

Formalizing these considerations we obtain the following algorithm to augment a tree to a 2-edge-connected subgraph:

Algorithm 7 *Augment a tree to biconnectivity*

Input: A tree $T = (V_T, E_T)$ with $\kappa(V_T) > K$.

Output: A biconnected subgraph $B = (V_B, E_B)$ with $\kappa(V_B) > K$ and $\sum_{e \in E_T} x_e^* < 2$ or \emptyset .

- 1 Set $V_B = V_T$ and $E_B = E_T$.
- 2 Set $y = \sum_{e \in E_B} x_e^*$.
- 3 Initialise an array \mathbf{w} of weights according to (8.12).
- 4 Set $L = \emptyset$.
- 5 DFS-traverse T and append each leaf to L as soon as it is encountered.
- 6 **For** $l = 1, \dots, \lfloor |L|/2 \rfloor$ **Do**
- 7 Find shortest \mathbf{w} -path P between $L(l)$ and $L(\lfloor |L|/2 \rfloor + l)$.
- 8 **If** this path has infinite length **Then Return** \emptyset .
- 9 Let $|P|$, P_V and P_E denote the length, the nodes and the edges on P .
- 10 Set $y = y + |P|$.
- 11 **If** $y \geq 2$ **Then Return** \emptyset .
- 12 Set $V_B = V_B \cup P_V$ and $E_B = E_B \cup P_E$.
- 13 **For** $e \in P_E$ **Do** $\mathbf{w}(e) = 0$.
- 14 **End For**
- 15 **If** $|L|$ is odd **Then**
- 16 Find a shortest path tree with respect to \mathbf{w} for $l = L(\lfloor |L|/2 \rfloor)$ and let $d(v)$ denote the length of the l - v -path in this tree.
- 17 Set $u = \operatorname{argmin}\{d(v) : v \neq l, v \in L\}$.
- 18 **If** $y + d(u) \geq 2$ **Then Return** \emptyset .
- 19 Add the l - u -path to V_B and E_B .
- 20 **End If**
- 21 **Return** (V_B, E_B) .

Notice that in this algorithm we assume that the first element in list L has index 1.

Assume we have successfully augmented our violated tree-cover T to a biconnected subgraph $B = (V_B, E_B)$ such that $\sum_{e \in E_B} x_e^* < 2$. Since we applied only a heuristic to find B , the set E_B of edges is not necessarily minimal. In order to reduce the number of edges in B (so as to further strengthen and sparsify the derived violated inequality) we may compute a 2-edge-connected spanning subgraph of B that contains only a minimum number of edges. Unfortunately, the problem of finding a minimum k -edge-connected spanning subgraph in an l -edge-connected graph for $k \leq l$ is \mathcal{NP} -complete for $k > 1$ [113]. For $k = 2$ this can be proved by reduction to the Hamiltonian cycle problem: A 2-edge-connected subgraph

contains at least $2|V|$ edges, since each node must have degree at least two in this subgraph. Consequently, a graph $G = (V, E)$ has a Hamiltonian cycle if and only if it has a minimum 2-edge-connected spanning subgraph with $2|V|$ edges.

As usual the weighted case is even more complicated than the unweighted [113]. For our purpose however, the unweighted case is sufficient since we already have a biconnected (sub)graph B with the desired property that the sum of edge-weights does not exceed 2 and we only aim at shrinking this graph. For the unweighted minimum biconnected spanning subgraph problem several approximation algorithms with performance guarantee smaller than 2 and polynomial (or even linear) asymptotic running time exist [113, 110, 112, 96] and can be used for our purpose.

In summary, we perform the following steps to find a violated 2-edge-connected cover inequality:

1. Find a violated tree-cover T .
2. Using Algorithm 7, augment T to get a 2-edge-connected subgraph $B = (V_B, E_B)$.
3. Reduce B by computing a minimum spanning 2-edge-connected subgraph (V_B, E'_B) of B .

8.2.3 Separation of k -Connected Cover Inequalities

In order to separate k -edge-connected cover inequalities for $k > 2$ we could basically start with a violated tree- or biconnected-cover inequality and attempt to repeatedly augment the edge-connectivity until the current structure has an edge-connectivity of at least k . Algorithms to solve problems of this kind are given in [57, 163, 77]. There are also algorithms to directly augment edge-connectivity from 1 or 2 to k (or more general, from $r < k$ to k) [57, 77]. Unfortunately, these algorithms and the ones that augment edge-connectivity by 1 only are quite involved and make use of sophisticated data structures such as the cactus representation [107] of a graph. From our point of view these algorithms are too complex to be run as subroutine in an efficient separation procedure.

Moreover, the input graphs on which we tested our optimisation algorithms hardly ever had an edge-connectivity larger than 2. So given a violated tree- or biconnected-cover inequality it is not clear whether the connectivity of the corresponding edge set can be augmented beyond 2 at all. Due to these considerations we never tried to find violated k -edge-connected cover inequalities for $k > 2$. However, an algorithm similar to Algorithm 7 could be found easily by extending the algorithm from [77] just as we extended the algorithm in Theorem 78.

8.2.4 Handling k -Connected q -Cover Inequalities

In [47, 48] several ideas are described to use separation algorithms for simple cover inequalities in order to separate q -cover inequalities. The considerations there apply to standard graph-partitioning problems but could easily be extended to our case.

However, for the k -connected q -cover inequality we used a completely different approach. Our approach is not a separation algorithm and is motivated by the fact that already separation for biconnected-cover inequalities is rather difficult. Our idea is to not find violated k -connected q -covers dynamically but to compute a reasonable set of such covers beforehand and statically add the inequalities defined by them.

To this end we determine for $k = 1, \dots, \Delta(G)$ all k -connected components of the grid graph $G = (V, E)$. Using an algorithm of Nagamochi and Watanabe [130] this can be done in time $\mathcal{O}(|V| \cdot |E| \cdot \min\{k, |V|, \sqrt{|E|}\})$. Notice that it is usually no problem that this bound is quite large ($|V|^2 \cdot |E|$ in the worst case), since we run the algorithm only once. Assume that C is a k -connected component of G that was found by this algorithm. We then find the maximal integer q such that $\kappa(C) > qK$. If $q > 1$ we have found a q -cover and C gives rise to a k -connected q -cover inequality which is added to the model.

This static approach requires some computational overhead for determining the k -connected components of G . However, these components can also be used to find lower bounds for the chromatic index of the optimal processor multigraph (see Chapter 7).

8.3 Knapsack Cover Inequalities

The experienced reader will have noticed, that mapping blocks to processors is a special instance of the Multiple Knapsack or General Assignment Problem. In both problems, a set of items $i \in I$ must be mapped to a set of containers $j \in J$ so as to maximise profit or minimise cost. Usually we have weights $\kappa_{i,j}$ and cost $\omega_{i,j}$ for all $(i, j) \in I \times J$, container capacities K_j for each $j \in J$ and binary variables $x_{i,j}$. We set $x_{i,j} = 1$ if and only if item i is mapped to knapsack j . A general binary programming formulation for this problem is then

$$\text{minimise or maximise } \sum_{i \in I} \sum_{j \in J} \omega_{i,j} \cdot x_{i,j} \quad (8.13)$$

$$\sum_{j \in J} x_{i,j} = 1 \quad i \in I \quad (8.14)$$

$$\sum_{i \in I} \kappa_{i,j} \cdot x_{i,j} \leq K_j \quad j \in J \quad (8.15)$$

$$x \text{ binary.} \quad (8.16)$$

In the Multiple Knapsack Problem we usually want to maximise (8.13) while in the General Assignment Problem we want to minimise it. Moreover, in the Multiple Knapsack Problem we use to have “less than or equal” in (8.14) and $\kappa_{i,j} = \kappa_{i,j'}$ for $j, j' \in J$. Observe that (8.13), (8.14) and (8.15) are directly related to (6.7a), (6.7b) and (6.7c), respectively.

Both, the polyhedral structure of the Multiple Knapsack polytope [50, 51, 161] as well as the polyhedral structure of the General Assignment polytope [73, 72, 35] that are implied by the integer programming formulation above have been examined by a lot of authors. Although complete descriptions of the corresponding polytopes are not available, several facet-defining inequalities are known that usually dramatically improve the speed of solving these problem by Branch-and-Cut algorithms [49, 51, 146, 37, 25]. Among these facets are the prominent *cover*-, *1-k-configuration*- and *multiple-cover* inequalities [51, 72]. All these inequalities are based on the concept of a violated cover, i. e., on a set of items $I' \subseteq I$ and a container $j' \in J$ (sometimes multiple containers are involved) such that

$$\sum_{i \in I'} \kappa_{i,j'} > K_{j'} \quad (8.17)$$

and

$$\sum_{i \in I'} x_{i,j'}^* > |I'| - 1 \quad (8.18)$$

for a current LP solution x^* at some node in the Branch-and-Cut tree. Since $\sum_{i \in I'} \kappa_{i,j'} > K_{j'}$ we know that not all items from I' can be mapped to container j' simultaneously. Consequently, inequality $\sum_{i \in I'} x_{i,j'} \leq |I'| - 1$ is valid and violated by the current solution x^* . By extending the concept of single covers, many other classes of cutting planes can be derived [49, 73, 72].

For a given (fractional) solution to the Multiple Knapsack or General Assignment Problem x^* such violated covers (I', j') are usually easily found and many of them can be separated efficiently [51]. In our case however, we do not have the strict capacity limits K_j but want to minimise the maximum load of the processors. i. e., we have something like the following integer program:

(LB)	minimise b	(8.19a)
	$\sum_{j \in J} x_{i,j} = 1 \quad i \in I$	(8.19b)
	$\sum_{i \in I} \kappa_i \cdot x_{i,j} \leq b \quad j \in J$	(8.19c)
	x binary.	(8.19d)

where $b \leq K$ for some uniform capacity limit K (Notice that we also have $\kappa_{i,j_1} = \kappa_{i,j_2}$ for $j_1, j_2 \in J$). If we take the approach that $K = \infty$ (see Section 6.1), we obviously cannot have a violated cover inequality at all. So suppose that $K < \infty$ and set $x_{i,j}^* = 1/|J|$ for all $(i, j) \in I \times J$. Then $(x^*, \sum_{i \in I} \kappa_i/|J|)$ is an optimal solution to the LP relaxation of (LB) (see Section 6.2). This solution is independent of the actual weights κ_i ! Moreover, for any set $I' \subseteq I$ and any container $j' \in J$ we have $\sum_{i \in I'} x_{i,j'}^* = |I'|/|J| < |I'|$. Consequently, the optimal solution of the LP-relaxation of (LB) *never* violates a cover inequality. The same argument suggests that it is very unlikely that *any* LP solution in a Branch-and-Cut tree for (LB) will violate a cover inequality. In fact computational experiments showed that usually none of the cuts described above were violated in Branch-and-Cut trees for instances of our problems.

Chapter 9

Heuristics

It is better to solve the right problem the wrong way
than the wrong problem the right way.
— *Dick Hamming*

Heuristics play a prominent role in the realm of \mathcal{NP} -complete or \mathcal{NP} -hard problems. The running time of exact algorithms for problems in these classes increases (super-)exponentially with the size of the problem instance. It is thus not feasible to apply these exact algorithms to large instances. This calls for heuristic algorithms that have an asymptotic running time that increases only *polynomially* with the size of the problem instances, but nevertheless yield solutions of satisfactory quality. Naturally, these heuristics cannot guarantee to find the optimal solution (for otherwise we had $\mathcal{P} = \mathcal{NP}$ and one of the big questions in information theory would be settled [152]). The goal of heuristics is instead to find a good solution in a reasonable amount of time. Obtaining a good feasible solution (i. e., one with small objective function value) is of interest in our context for two reasons: If the instance of (OGPC) is too big to be solved by Branch-and-Cut, then a heuristic can be used to obtain a solution at all. The other reason is that every feasible solution to a problem instance implicitly yields an upper bound u on the optimal objective function value of the respective instance. These upper bounds can then be used in Branch-and-Cut algorithms (see Algorithm 4) to prune branches from the Branch-and-Cut tree and thus speed up the enumerative part of the algorithm.

In this chapter we will present four different classes of heuristics. *Heuristics for initial assignments* are used to determine an initial assignment of blocks to processors. These heuristics are usually quite simple and fast and their only purpose is to provide more sophisticated heuristics with a feasible starting solution.

Local improvement heuristics start with a feasible solution (e. g., one that was produced by an initial assignment heuristic) and attempt to improve this solution by *locally changing* it. Here

locally changing means that we modify only parts of the solution and do not allow reverting the whole solution. Considering only parts of the solution for potential improvement keeps the algorithms simple and fast.

The third class of heuristics are *rounding heuristics*. These heuristics are invoked while the Branch-and-Cut tree of a problem instance is explored (see Algorithm 4) and work with the optimal fractional solution at a Branch-and-Cut node. The heuristic attempts to exploit the information that is provided by the current fractional solution so as to construct a feasible solution with a small objective function value.

Finally, we will present *heuristic simplifications* of (OGPC). These simplifications are based on assumptions that are not true in general, but are believed to nevertheless allow near-optimal solutions. Considering these assumptions as being satisfied enables us to reduce either size or complexity of problem instances by fixing variables or adding constraints.

9.1 Heuristics for Initial Assignment

The heuristics described in this section start with the grid graph $G = (V, E)$ and the processor set $P = \{p_0, \dots, p_{|P|-1}\}$. The aim is to assign each block $v \in V$ to a processor $p \in P$ such that the objective function value of the mapping obtained is small. Once a block has been assigned to a processor, this block is fixed and never moved to another processor again. Heuristics that move blocks around are described in Section 9.2 below. When computing an initial assignment we take into account the objective function at different levels of accuracy. We first suggest some schemes that only consider the maximum load b and attempt to minimise it. After this we introduce other heuristics that not only attempt to balance the computational load but also aim at a small chromatic index in the resulting processor multigraph.

Notice that none of the initial assignment strategies described below computes an edge-colouring of the processor multigraph. In order to determine the objective function value of an assignment returned by one of these algorithms we must compute such a colouring. We have used different algorithms (both from the literature and from our mind) to do so and will describe them in Section 9.4 below.

Before we start describing algorithms, let us mention a drawback that is common to all these algorithms: Each of the algorithms described below might produce a mapping that does not meet the capacity requirement K . If an algorithm returns such a mapping \mathbf{m} with $b(\mathbf{m}) > K$ it has failed to determine a feasible mapping. As all algorithms usually fill a processor until it bears at least κ^{*} control volumes, such an infeasible mapping is most likely to occur when κ^{*} is close to K or when the blocks have very different sizes. If an algorithm ends up with an infeasible mapping, we either switch to another algorithm or we restart the algorithm

with the value of κ^{*} slightly decreased.

9.1.1 Algorithms Aiming at Balanced Processor

As stated above, the first few algorithms to be described ignore the communication overhead and aim only at balanced processors. Such algorithms are reasonable if the communication overhead t_c is very small when compared to the arithmetic time t_a .

Filling Up Processors

The most simple algorithm for assigning blocks to processors while attempting to keep the maximal load small is to iteratively fill up the processors. That is we start with processor p_0 and assign blocks to this processor until its size is at least $\kappa_V/|P|$. Then we advance to the next processor and repeat. The limit $\kappa_V/|P|$ here stems from the fact that in a perfect balancing there is at least one processor that bears this many control volumes. A formal description of the iteration yields

Algorithm 8 *Iteratively filling up processors*

Input: A list L of blocks and the processor set P .

Output: A mapping \mathbf{m} .

```

1  For  $i = 0, \dots, |P| - 1$  Do
2    Set  $s = 0$ .
3    While  $s < \kappa_V/|P|$  Do
4      Remove the first block  $u$  from  $L$ .
5      Set  $\mathbf{m}(u) = p_i$  and  $s = s + \kappa_u$ .
6    End While
7  End For
8  Return  $\mathbf{m}$ .
```

This algorithm takes only time $\mathcal{O}(|V|)$ to assign all blocks from the grid graph $G = (V, E)$. Notice that Algorithm 8 is sensitive to the order in which the blocks are stored in L . In other words the performance of the algorithm may increase or decrease depending on the order of the blocks. We will discuss different reasonable schemes for sorting the blocks in Section 10.5.

Notice that Algorithm 8 always maps all blocks. This is achieved by the condition $s < \kappa_V/|P|$ in step 3. Due to this condition, we only switch to the next processor if the current processor

bears at least $\kappa_V/|P|$ control volumes. Since this is true for all processors, we either run out of blocks before all processors reach this limit or each processor bears at least $\kappa_V/|P|$ control volumes. In the latter case, a total of at least κ_V control volumes are mapped, i. e., all blocks must be assigned to a processor.

In rare cases (especially if $\kappa_V/|P|$ is close to K) it may however happen, that the solution returned by Algorithm 8 violates the capacity constraints. In other words, the solution may contain a processor that bears more than K control volumes. In this case Algorithm 8 failed and we must use another one to obtain an initial assignment (sometimes sorting the blocks in L in a different manner may even be enough).

Assigning Blocks to the Leastly Loaded Processor

In order to improve the balance of the initial assignment, we not simply fill up the processors one after the other. Instead we keep a list L of unmapped nodes and always assign the next block from L to the processor that currently bears the smallest number of control volumes. By doing so, we always map the current block in a fashion that yields the minimal increase of the maximum processor size. A formal description of this strategy is

Algorithm 9 *Assign blocks to leastly loaded processor*

Input: A list L of blocks and the processor set P .

Output: A mapping \mathbf{m} .

```

1  Set  $s_p = 0$  for all  $p \in P$ .
2  While  $L \neq \emptyset$  Do
3    Set  $p^* = \operatorname{argmin}\{s_p : p \in P\}$  (breaking ties arbitrarily).
4    Remove the first element  $u$  from  $L$ .
5    Set  $\mathbf{m}(u) = p^*$  and  $s_{p^*} = s_{p^*} + \kappa_u$ .
6  End While
7  Return  $\mathbf{m}$ .
```

One drawback of this algorithm is step 3. Here we must find the processor of minimal size which takes time $\mathcal{O}(|P|)$ in general and the whole algorithm runs in time $\mathcal{O}(|V \times P|)$. Like Algorithm 8, Algorithm 9 is sensitive to the order in which the blocks are stored in L . Again we defer the discussion of reasonable block sorting strategies to Section 10.5.

Circularly Assigning Blocks

In order to obtain a heuristic algorithm that aims at balanced load but runs in linear time (i. e., $\mathcal{O}(|V|)$) we proceed as follows: Instead of putting the current block onto the processor that bears the smallest number of control volumes, we iterate over the processors in a circular fashion and assign the current block to the current processor. The iteration over processors is bidirectional. In other words, the first $|P|$ nodes from L are assigned to processors $p_0, \dots, p_{|P|-1}$. After this blocks $|P| + 1, \dots, 2|P|$ from L are assigned to processors $p_{|P|-1}, \dots, p_0$ respectively:

Algorithm 10 *Assign blocks in a circular fashion*

Input: A list L that stores the blocks and the processor set P .

Output: A mapping \mathbf{m} .

```

1  Set  $f = \text{true}$  and  $i = 0$ .
2  While  $L \neq \emptyset$  Do
3    Remove the first element  $u$  from  $L$ .
4    Set  $\mathbf{m}(u) = p_i$ .
5    If  $f$  is true Then
6      Set  $i = i + 1$ .
7      If  $i = |P|$  then set  $f = \text{false}$  and  $i = |P| - 1$ .
8    Else
9      Set  $i = i - 1$ .
10     If  $i < 0$  then set  $f = \text{true}$  and  $i = 0$ .
11   End If
12 End While
13 Return  $\mathbf{m}$ .
```

Iterating over the processors in forward and backward direction is especially useful if the nodes in L are sorted by decreasing size. To see this assume that we have n nodes with weights $n, \dots, 1$ and that n is divisible by $|P|$. Then after the first forward and backward iteration we have the following weights assigned to the processors:

$$\begin{array}{c|c|c|c}
 p_0 & p_2 & \dots & p_{|P|-1} \\
 \hline
 n & n - 1 & \dots & n - |P| + 1 \\
 n - 2|P| + 1 & n - 2|P| + 2 & \dots & n - |P|
 \end{array}$$

In other words, each processor bears $n - 2|P| + 1$ control volumes and the load is perfectly balanced. After the next forward and backward iteration, the load is again perfectly balanced and so on. If instead we iterated in only one direction over the processors the load would be heavily unbalanced in the mapping returned by the algorithm.

9.1.2 Assignment Algorithms with Communication Support

The initial assignment strategies described so far only aimed at balancing the load between the processors and did not account for edges in the multicut.

We now describe several algorithms that find initial assignments and attempt to anticipate the induced communication overhead. All of the algorithms to be described are refinements of the “fill up” strategy Algorithm 8. They only differ in the ways in which step 4 of this algorithm is implemented. In general, the algorithms do not simply select the first unmapped block and assign it to the current processor. Instead, they choose an unmapped block u such that assigning u to the current processor is likely to yield a processor multigraph with a small chromatic index. Since obtaining the chromatic index of an arbitrary multigraph is an \mathcal{NP} -complete problem, we cannot find the block that implies the smallest chromatic index efficiently. Instead, we use different surrogates for the chromatic index that work well in practice.

Processors of Small Degree

The first algorithm is inspired by the two upper bounds $\chi'(G) \leq \lfloor 3\Delta(G)/2 \rfloor$ and $\chi'(G) \leq \Delta(G) + \mu(G)$, that are valid for any multigraph G (see [155] and [160]). These bounds suggest that a multigraph in which all nodes have small degree is likely to have a small chromatic index. When we assign block u to the current processor p , then each neighbour v of u that is not mapped to p increases the current degree of p by one. Neighbours of u that are already mapped to another processor $k \neq p$ also increase the current degree of p but we do not consider them here to keep the algorithm simple. So if we want p to have a small degree then it is reasonable to always pick the block that has the smallest number of unmapped neighbours. To this end we use an array `deg` that stores for each unmapped block u the number of unmapped neighbours, i. e., the amount by which the degree of p would increase if u was mapped to it. In order to break ties when picking the block with the minimal number of unmapped neighbours we introduce the excess of a block. For a current processor size s and an unmapped block u the *excess* of u (with respect to s) is $\max\{s + \kappa_u - \kappa_V/|P|, 0\}$. In other words, the excess of u yields the number of control volumes by which the load on p would exceed the perfect load $\kappa_V/|P|$ if u was mapped to p . Hence, if multiple blocks result in the same increase of the processor’s degree, then we pick the block with the smallest excess (and break remaining ties arbitrarily).

Algorithm 11 *Small Degree Assignment*

Input: A list L of blocks and the processor set P .

Output: A mapping \mathbf{m} .

```

1  For  $i = 0, \dots, |P| - 1$  Do
2    For  $u \in L$  Do  $\text{deg}(u) = \text{deg}_G(u)$ .
3    Set  $s = 0$ .
4    While  $s < \kappa_V/|P|$  Do
5      Find  $u^* = \text{argmin}\{\text{deg}(u) : u \in L\}$ . If there are multiple blocks that
      attain this minimum pick the one with smallest excess.
6      Remove  $u^*$  from  $L$ .
7      Set  $\mathbf{m}(u^*) = p_i$  and  $s = s + \kappa_{u^*}$ .
8      For  $v \in N(u^*) \cap L$  Do  $\text{deg}(v) = \text{deg}(v) - 1$ .
9    End While
10 End For
11 Return  $\mathbf{m}$ .

```

Although finding the block with minimal value in deg can be implemented in $\mathcal{O}(\Delta(G))$ by a bucket sort approach, the algorithm still requires time $\mathcal{O}(|V \times V|)$. This is because we use the excess of blocks as tie-breaking rule. Finding the block of minimum excess in step 5 takes time $\mathcal{O}(|V|)$. Since the steps in the innermost loop are executed at most $\mathcal{O}(|V|)$ times, the resulting asymptotic running time of Algorithm 11 is $\mathcal{O}(|V \times V|)$.

Small Multicut

Another estimate for the chromatic index of a multigraph $G = (V, E)$ is $|E|$. In our situation, the number of edges in the processor multigraph M_P is given by the number of edges in the multicut. If we slightly modify Algorithm 11, we obtain an algorithm that aims at a small size of the multicut instead of keeping the degree of processors small. Assume that we are currently filling up processor $p_i \in P$ with $i > 0$ (i. e., p_i is not the first processor to be filled). For an unmapped block $u \in V$, each neighbour $v \in N(u)$ that is mapped to a processor p' with $p' < p_i$ the edge uv will definitely be contained in the final multicut. As we do not want to change the target processors of mapped blocks (in this algorithm) we pick as next block u the block with the smallest number of unmapped neighbours.

Algorithm 12 *Small Multicut Assignment*Input: A list L of blocks and the processor set P .Output: A mapping \mathbf{m} .

```

1  For  $p = i, \dots, |P| - 1$  Do
2    For  $u \in L$  Do  $\text{deg}(u) = |N(u) \cap L|$ .
3    Set  $s = 0$ .
4    While  $s < \kappa_V / |P|$  Do
5      Find  $u^* = \text{argmin}\{\text{deg}(u) : u \in L\}$ . If there are multiple blocks that
      attain this minimum pick the one with smallest excess.
6      Remove  $u^*$  from  $L$ .
7      Set  $\mathbf{m}(u^*) = p_i$  and  $s = s + \kappa_{u^*}$ .
8      For  $v \in N(u^*) \cap L$  Do  $\text{deg}(v) = \text{deg}(v) - 1$ .
9    End While
10 End For
11 Return  $\mathbf{m}$ .

```

Observe that in step 2 we initialise $\text{deg}(u)$ with the number of unmapped neighbours of u . In step 8 we decrement $\text{deg}(v)$ for each unmapped neighbour v of u^* . This is because edge uv disappears from the multicut if we map v to the same processor as u^* . The runtime analysis of Algorithm 12 is analogous to the analysis of Algorithm 11 and yields that the asymptotic running time is $\mathcal{O}(|V \times V|)$.

Connected Subgraphs

The next algorithm tries to assign blocks to processors such that the blocks assigned to the same processor induce a connected subgraph of the grid graph G . The hope is that this strategy will render most of the edges of the grid graph intra-processor and will thus yield a small multicut.

When assigning blocks to processor p , we always pick the block among the unmapped ones, that has the most neighbours on p (breaking ties by excess). We pick this block because it is “most heavily” connected to the subgraph currently mapped to p . In order to keep track of the number of neighbours a block u has on processor p we use an array called \mathbf{N} . This array stores at position $\mathbf{N}(u)$ the number of neighbours block u has on the current processor p .

Algorithm 13 *Connected Subgraphs*Input: A list L of blocks and the processor set P .Output: A feasible mapping \mathbf{m} .

```

1  Set  $i = 0$ .
2  While  $L \neq \emptyset$  and  $i < |P|$  Do
3    Set  $\mathbf{N}(u) = 0$  for all  $u \in L$ .
4    Set  $s = 0$ .
5    While  $s < \kappa_V/|P|$  and  $L \neq \emptyset$  Do
6      Find  $u^* = \operatorname{argmax}\{\mathbf{N}(u) : u \in L\}$ , break ties by the blocks' excess.
7      Remove  $u^*$  from  $L$  and set  $s = s + \kappa_{u^*}$ ,  $\mathbf{m}(u^*) = p_i$ .
8      For each  $v \in L$  that is adjacent to  $u^*$  set  $\mathbf{N}(v) = \mathbf{N}(v) + 1$ .
9    End While
10   Set  $i = i + 1$ .
11 End While
12 Return  $\mathbf{m}$ .

```

Finding the maximal element in step 6 takes us time $\mathcal{O}(|V|)$, while the update in step 8 requires only $\mathcal{O}(\Delta(G))$. Since the loop is executed exactly $|V|$ times, we get that Algorithm 13 runs in time $\mathcal{O}(|V^2|)$.

In step 6 we pick the block with the most neighbours on p_i . This is reasonable because this is the block “most heavily connected” to the subgraph currently mapped to p_i . On the other hand, when p_i becomes more and more loaded and we are more and more likely to put the last block onto p_i , we should consider another criterion: The number of edges that are added to the multicut by assigning a block to p_i . So towards the end of the innermost loop we should rather pick

$$u^* = \operatorname{argmin}\{\deg_G(u) - \mathbf{N}(u) : u \in L, \mathbf{N}(u) > 0\} \quad (9.1)$$

in step 6. This yields a block that is connected to the subgraph currently mapped to p_i and has a minimum number of neighbours not on p_i . Since each neighbour not on p_i threatens to add an edge to the multicut our hope is that mapping u^* to p does not increase the size of the multicut too much. The question is how to recognise when processor p_i is almost full. To this end, we introduce a factor $f \in [0, 1]$. As long as $s \leq f \cdot \kappa^*$ we pick blocks as described in Algorithm 13 and as soon as $s > f \cdot \kappa^*$ we switch to the strategy defined by (9.1).

There is one situation that needs special attention here: It may happen that $\max\{\mathbf{N}(u) : u \in L\} = 0$ and $L \neq \emptyset$ in step 6. This means that no unmapped block is adjacent to the subgraph currently mapped to p_i . In this case, we simply pick a block of maximum degree among the unmapped blocks. Likewise, it may happen that $\{u \in L : \mathbf{N}(u) > 0\} = \emptyset$ when we attempt to apply the strategy defined by (9.1). If this happens we instead pick a block of minimum degree among the unmapped blocks.

Apart from dynamically adjusting the block selection strategy, there is another way to potentially improve Algorithm 13. To see this recall that array \mathbf{N} counts the number of neighbours a block has on the current processor. Picking the block u with largest value $\mathbf{N}(u)$ slightly favours blocks with high degree. To see this, assume we have two unmapped blocks u and v with $\deg(u) = 6$, $\deg(v) = 2$, $\mathbf{N}(u) = 3$ and $\mathbf{N}(v) = 2$. The selection strategy described above would prefer u to v because u has more neighbours on the current processor. Since u has three neighbours that are not mapped to p_i , mapping u to p_i removes three potential multicut edges, but also introduces three new ones. On the other hand, mapping v to p would remove two potential multicut edges without adding any new. It is thus more promising to map v to p . To implement this new strategy, we choose in step 6 the block that has the highest *relative* connectivity to the subgraph currently mapped to p , i. e.,

$$u = \operatorname{argmax}\{\mathbf{N}(u)/\deg(u) : u \in L\}. \quad (9.2)$$

This selection strategy not only takes into account the number of neighbours on the current processor but also relates this number to the degree of the block under consideration.

9.2 Local Search Heuristics

A *local search* algorithm is a meta-heuristic that tries to find an optimal solution by moving from solution to solution in the space of feasible solutions. In order to render the algorithm efficient, it is in general not allowed to move from one feasible solution to *any* other feasible solution. Instead one defines a *neighbourhood* that defines for each feasible solution a set of *adjacent* feasible solutions to which the algorithm may move. This gives rise to the *neighbourhood graph* $G_N = (V_N, E_N)$ of the local search algorithm in which each node $u \in V_N$ corresponds to a feasible solution and two nodes u and v are connected by a directed edge $(u, v) \in E_N$ if the algorithm is allowed to move from u to v . The whole algorithm can then be considered and implemented as moving around in the neighbourhood graph. For performance reasons this (directed) graph is usually not explicitly constructed. Instead one defines allowed *moves* – that means possible changes – that may be applied to a feasible solution to reach another feasible solution. For (OGPC) such a move would for example be to move a node from one processor to another. The set of allowed moves then implicitly defines a neighbourhood for each feasible solution. Starting from an (arbitrary) initial solution s^0 the local search algorithm looks for a neighbour with a better objective function value. If such a neighbour is found the algorithm moves to this neighbour and iterates. If no improving neighbour exists the algorithm stops and returns the current solution. In other words, we have the following general local search algorithm:

Algorithm 14 *General Local Search Algorithm*

Input: An objective function z that is to be minimised, a start solution s and the neighbourhood graph $G = (S, E)$.

Output: A solution s that has no improving neighbours.

```
1  Set  $s^* = \operatorname{argmin}\{z(s') : s' \in N(s)\}$ .
2  If  $z(s^*) < z(s)$  Then
3    Set  $s = s^*$ .
4    Goto step 1.
5  End If
6  Return  $s$ .
```

In [53] Fiduccia and Mattheyses presented a heuristic for a problem similar to (OGPC), namely the problem of minimising the edge-cut while the capacity of processors is restricted. This heuristic was originally designed for bipartition (i. e., for exactly two processors) but has been extended to multiple processors [48]. Thanks to a sophisticated bucket sort approach this heuristic runs in linear time. The key idea is to define the *gain* of a move as the number of edges by which the multicut would increase or decrease if the move was applied. This gain depends only on the current processor of the node to be moved and its neighbours. Using bucket sort it is then quite easy to find the best move available in short time. Fiduccia and Mattheyses [53] showed how this gain can be updated efficiently after a move and were thus able to design a very efficient algorithm. Unfortunately, we cannot hope for an algorithm as efficient as the one by Fiduccia and Mattheyses. The reason is, that our objective function is highly *non-local*. That is, after a local change the objective function value cannot be updated locally. In fact, after performing any change to a mapping \mathbf{m} , if we want to determine the objective function value of the changed mapping, we must compute the maximum load among *all* processors and must consider the *whole* multigraph $M_P(\mathbf{m})$ to find its chromatic index. Although they do not have a running time estimate as impressive as the algorithm by Fiduccia and Mattheyses, the local search heuristics we implemented perform reasonably well in practice.

Our local improvement heuristics start with an initial feasible mapping \mathbf{m}^0 and attempt to improve this mapping by locally modifying it. Locally means that only a few nodes are moved around and no complex modifications are performed. Restricting the algorithms to local improvements keeps them simple and fast. If a local improvement (an improving *move*) is found, the initial mapping is replaced by the improved one and the whole process is iterated. This is done until no local improvement is found in which case the algorithm stops and the current mapping is returned. In our algorithms we consider the following three classes of moves (see also Figure 9.1):

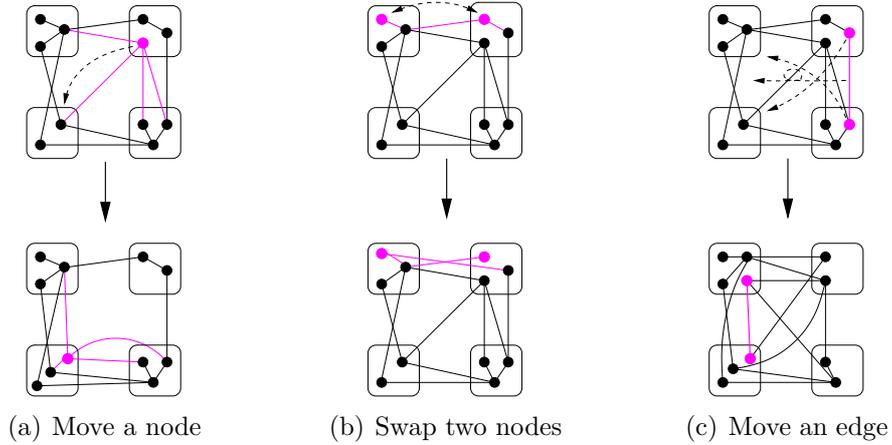


Figure 9.1: The three different classes of moves that we consider.

Move node We pick a node u that is currently mapped to processor p and move it to a processor $k \neq p$.

Swap nodes We pick two nodes $u, v \in V$ that are mapped to different processor p and k . Then we move u to k and v to p .

Move edge We pick an edge $uv \in E$, where u is currently mapped to p_u and v is currently mapped to p_v ($p_u = p_v$ is allowed valid here). We then pick processors $p'_u \neq p_u$ and $p'_v \neq p_v$ and move u to p'_u and v to p'_v . Moving the two nodes is the same as moving the edge uv around.

Allowing these kinds of moves Step 1 of Algorithm 14 is implemented by means of the following loop:

Algorithm 15 *Local Search Loop*

Input: The current mapping \mathbf{m} and its objective function value z .

Output: A potentially improved mapping and its objective function value.

- 1 Store the current objective function value z .
- 2 Set $m^* = \text{Nil}$, $z^* = \infty$.
- 3 **For** each feasible move m **Do**
- 4 Apply m to \mathbf{m} and compute the new objective function value z' .
- 5 **If** $z' < z^*$ **Then** set $z^* = z'$ and $m^* = m$.
- 6 Undo move m in \mathbf{m} .
- 7 **If** $z^* < z$ apply m^* to \mathbf{m} and set $z = z^*$.
- 8 **End For**
- 9 **Return** (\mathbf{m}, z) .

If $z^* < z$ in Step 7 we have found an improving move m^* , otherwise we must stop the local search algorithm. In order for the loop in Algorithm 15 to be executed efficiently, we must implement the following operations such that they can be performed quickly:

- Move a node in a mapping. Notice that all allowed moves can be viewed as concatenation of node moves and it is thus sufficient to be able to move one node efficiently.
- Determine the objective function value of a mapping.

Moving around nodes in a mapping is quite easy. In order to move node u from its current processor to processor k we simply set $\mathbf{m}(u) = k$. Evaluating the objective function value for a given mapping \mathbf{m} is however much more difficult. First we must find the size of the maximum processor. Moreover, evaluating the objective function value of \mathbf{m} implies finding a minimal edge-colouring for the processor multigraph defined by \mathbf{m} . As we already stated in Chapter 2 this is an \mathcal{NP} -complete problem and cannot be solved efficiently in general. However, if we have at most four processors, then M_P has no more than four nodes and $\chi'(M_P)$ can be easily determined (see Section 6.4), provided that we can determine the multiplicity of edges efficiently. If we have more than four processors we are out of luck and will resort to approximations of $\chi'(M_P)$ that can be determined in polynomial time.

Data Structures

We now describe several arrays which allow us to efficiently evaluate the objective function value of a mapping \mathbf{m} after an arbitrary set of moves have been applied to it. These arrays are

Name	Index	Content
size	P	Element size (p) yields the current size of processor p .
mul	$P \times P$	For $p, k \in P$ element mul (p, k) yields the number of edges with one endpoint on p and the other endpoint on k .
deg	P	deg (p) is the current degree of processor $p \in M_P$.
nei	$V \times P$	For $u \in V$ and $p \in P$ element nei (u, p) yields the number of neighbours of u that are currently mapped to processor p .

Observe that if $p \neq k$, then **mul**(p, k) and **mul**(k, p) both store the multiplicity $\mu(pk)$ of edge $pk \in M_P$. Although not every array is used in each algorithm, we provide initialisation and update algorithms that operate on all arrays. It is straightforward to identify and remove operations that are related to unused arrays.

Given an initial mapping \mathbf{m} the above arrays are easily initialised by the following code fragment.

Algorithm 16 *Initialise Arrays*

Input: An initial mapping \mathbf{m} .

Output: Correctly initialised arrays **size**, **mul**, **deg** and **nei**.

```

1  Set all elements in all arrays to zero.
   // Initialise size
2  For  $u \in V$  Do  $\mathbf{size}(\mathbf{m}(u)) = \mathbf{size}(\mathbf{m}(u)) + \kappa_u$ .
   // Initialise mul
3  For  $uv \in E$  Do
4     Set  $\mathbf{mul}(\mathbf{m}(u), \mathbf{m}(v)) = \mathbf{mul}(\mathbf{m}(u), \mathbf{m}(v)) + 1$ .
5     Set  $\mathbf{mul}(\mathbf{m}(v), \mathbf{m}(u)) = \mathbf{mul}(\mathbf{m}(v), \mathbf{m}(u)) + 1$ .
6  End For
   // Initialise deg
7  For  $uv \in E$  Do
8     If  $\mathbf{m}(u) \neq \mathbf{m}(v)$  Then
9         Set  $\mathbf{deg}(\mathbf{m}(u)) = \mathbf{deg}(\mathbf{m}(u)) + 1$ .
10        Set  $\mathbf{deg}(\mathbf{m}(v)) = \mathbf{deg}(\mathbf{m}(v)) + 1$ .
11    End If
12 End For
   // Initialise nei
13 For  $u \in V$  Do
14    For all neighbours  $v$  of  $u$  Do
15        Set  $\mathbf{nei}(u, \mathbf{m}(v)) = \mathbf{nei}(u, \mathbf{m}(v)) + 1$ .
16    End For
17 End For
18 Return the arrays.

```

From Algorithm 16 we conclude that initialising the arrays **size**, **mul**, **deg** and **nei** can be done in time $\mathcal{O}(|V|)$, $\mathcal{O}(|E|)$, $\mathcal{O}(|E|)$ and $\mathcal{O}(\max\{|V \times P|, |E|\})$ respectively.

During the local search algorithm, when a node $u \in V$ is moved from its current processor p_u to another processor p'_u , then all the arrays described above must be updated. Most of these updates are straightforward. Updating the **deg** array is however a little more sophisticated and happens with the aid of the **nei** array. When we move u from p_u to p'_u then for each neighbour of u that is not mapped to p_u the degree of p_u decreases by one. The number of neighbours of u that are not mapped to p_u is given by $\mathbf{deg}_G(u) - \mathbf{nei}(u, p_u)$. On the other hand, for each neighbour of u that is mapped to p_u the degree of p_u increases by one. Thus

the degree of p_u increases by

$$\underbrace{\text{nei}(u, p_u)}_{\text{neighbours on } p_u} - \underbrace{(\text{deg}_G(u) - \text{nei}(u, p_u))}_{\text{neighbours not on } p_u} = 2 \cdot \text{nei}(u, p_u) - \text{deg}_G(u). \quad (9.3)$$

A similar argument shows that moving u from p_u to p'_u increases the degree of p'_u by

$$\underbrace{(\text{deg}_G(u) - \text{nei}(u, p'_u))}_{\text{neighbours not on } p'_u} - \underbrace{\text{nei}(u, p'_u)}_{\text{neighbours on } p'_u} = \text{deg}_G(u) - 2 \cdot \text{nei}(u, p'_u). \quad (9.4)$$

The degree of all processors different from p_u and p'_u are unchanged by such a move.

The following algorithm summarises the updates of the different arrays.

Algorithm 17 *Update Arrays*

Input: A node $u \in V$, source and target processors $p_u, p'_u \in P$ with $p_u \neq p'_u$ and the arrays **size**, **mul**, **deg** and **nei**.

Output: The updated arrays.

```

// Update size
1 Set size( $p_u$ ) = size( $p_u$ ) -  $\kappa_u$ .
2 Set size( $p'_u$ ) = size( $p'_u$ ) +  $\kappa_u$ .
// Update mul
3 For  $v \in N(u)$  Do
4   Decrement mul( $p_u, \mathbf{m}(v)$ ) and mul( $\mathbf{m}(v), p_u$ ) by 1.
5   Increment mul( $p'_u, \mathbf{m}(v)$ ) and mul( $\mathbf{m}(v), p'_u$ ) by 1.
6 End For
// Update deg (see (9.3) and (9.4) above)
7 Set deg( $p_u$ ) = deg( $p_u$ ) + 2 · nei( $u, p_u$ ) - deg $_G$ ( $u$ ).
8 Set deg( $p'_u$ ) = deg( $p'_u$ ) + deg $_G$ ( $u$ ) - 2 · nei( $u, p'_u$ ).
// Update nei
9 For  $v \in N(u)$  Do
10   Set nei( $v, p_u$ ) = nei( $v, p_u$ ) - 1.
11   Set nei( $v, p'_u$ ) = nei( $v, p'_u$ ) - 1.
12 End For
13 Return the arrays.

```

In Algorithm 17 beware of the following facts:

- The arrays must updated *before* the mapping is updated, i. e., *before* we set $\mathbf{m}(u)$ from p_u to p'_u .

- Array `deg` must be updated *before* array `nei` is updated.

So if we move a node u from its current processor p_u to another processor p'_u , then the arrays `size`, `mul`, `deg` and `nei` can be updated by Algorithm 17 in time $\mathcal{O}(1)$, $\mathcal{O}(\Delta(G))$, $\mathcal{O}(1)$ and $\mathcal{O}(\Delta(G))$ respectively. As we are able to efficiently update the arrays after a node move, we are able to efficiently keep them up to date in the whole local search loop that is given by steps 4 through 6 of Algorithm 15. Remember that all moves considered can be decomposed into a sequence of node moves and undoing a node move is easily done by moving the node back.

Objective Functions and Evaluation of them

So far we have only described how to keep certain data structures up to date efficiently. We still left open how to compute the objective function value efficiently. In order to fill in this gap we start with the most simple case and assume that we have four processors. Remember that in this case the chromatic index of $M_P(\mathbf{m})$ is given by

$$\chi'(M_P(\mathbf{m})) = \max\{\mu_{M_P}(01), \mu_{M_P}(23)\} + \max\{\mu_{M_P}(02), \mu_{M_P}(13)\} + \max\{\mu_{M_P}(03), \mu_{M_P}(12)\},$$

where $\mu_{M_P}(lk)$ is the multiplicity of the edge between p_l and p_k in the processor multigraph $M_P(\mathbf{m})$. Using the arrays `size` and `mul` we can compute the objective function value of \mathbf{m} as

$$\begin{aligned} z^*(\mathbf{m}) = & t_a \cdot \max_{p \in P} \text{size}(p) + \\ & t_c \cdot (\max\{\text{mul}(p_0, p_1), \text{mul}(p_2, p_3)\} + \\ & \max\{\text{mul}(p_0, p_2), \text{mul}(p_1, p_3)\} + \\ & \max\{\text{mul}(p_0, p_3), \text{mul}(p_1, p_2)\}) \end{aligned} \quad (9.5)$$

which takes time $\mathcal{O}(|P|)$ (i. e., constant time since $|P| = 4$). Notice that in (9.5) we do not need the arrays `deg` and `nei`. Thus one iteration of the local search loop in Algorithm 15 can be implemented in time $\mathcal{O}(\Delta(G))$ and we can therefore check one single move in time $\mathcal{O}(\Delta(G))$. Observe that there is no way to efficiently keep track of the maximum size of the processors, so we cannot avoid checking the size of each processor for the computation of the maximum size in (9.5).

The situation becomes much worse if we have more than four processors. Computing $\chi'(M_P(\mathbf{m}))$ is then an \mathcal{NP} -hard problem and we must resort to objective functions that approximate this value instead of determining it exactly. In the following we present several upper bounds that can efficiently be computed during local search. We saw in Chapter 2 that $\chi^*(G)$ is often a good approximation for $\chi'(G)$ – especially if G contains only a small number of nodes – and that $\chi^*(G)$ can be computed in polynomial time. So at first glance

it seems a good idea to use $\chi'^*(G)$ as approximation for $\chi'(G)$. However, the proof that $\chi'^*(G)$ can be determined in polynomial time for any multigraph $G = (V, E)$ contains a flaw from the algorithmic point of view: The proof is based on the fact that the separation problem over a certain polyhedron is polynomial-time solvable and that by the polynomial-time equivalence of separation and optimisation [151, 76] we can find $\chi'^*(G)$ in polynomial time using the ellipsoid method. The ellipsoid method however is a tool that is useful in theory but cannot be implemented on computers efficiently. Thus, the polynomial time algorithm to find $\chi'^*(G)$ for an arbitrary multigraph $G = (V, E)$ exists in theory but cannot be used in practice. We therefore refrain from using $\chi'^*(G)$ as approximation for $\chi'(G)$ and resort to other, more coarse approximations.

To gain more speed in our algorithms, we do not explicitly compute an edge-colouring of $M_P(\mathbf{m})$ (let alone an optimal one) in order to determine the objective function value of \mathbf{m} . We only use an upper bound. Computing an explicit edge-colouring by some heuristic (for example by Vizing's [160], Shannon's [155] or the ones described in [67, 92, 133, 24]) would yield a more accurate approximation of $\chi'(M_P)$ than the bounds we use. However, the known algorithms to compute explicit edge-colourings with an acceptable approximation guarantee all have a running time of at least $\mathcal{O}((n + \Delta)m)$. Some of them (e. g., the algorithms described in [67, 92, 133, 24]) require sophisticated and complicated data structures to be implemented efficiently. Since the asymptotic running times of these algorithms are high in comparison to the running times required for determining the upper bounds below, we opted against computing explicit edge-colourings so as to determine the objective function value of a mapping \mathbf{m} . Thus, in order to obtain a feasible solution we must compute an explicit edge-colouring for the mapping returned by the local search algorithm. This edge-colouring must not use more colours than the bound used in the algorithm suggests. Such a colouring is easy to find since for all bounds we use, polynomial time algorithms are known that find such a colouring.

A first upper bound on $\chi'(M_P)$ is the size of the multicut, i. e., the number of edges in M_P . To this end we introduce a new variable `cut` that counts the number of edges in the multicut. Initialisation and update of this variable are straightforward and we do not spell out the details here. We only notice that `cut` can be initialised in time $\mathcal{O}(|E|)$ and updated in time $\mathcal{O}(\Delta(G))$ if a node is moved. With this variable, our objective function becomes

$$z^*(\mathbf{m}) = t_a \cdot \max_{p \in P} \text{size}(p) + t_c \cdot \text{cut} \tag{9.6}$$

which is determined in time $\mathcal{O}(|P|)$. Since we only need the `size`, the effects of moving a single node can be evaluated in time $\mathcal{O}(\max\{|P|, \Delta(G)\})$.

Using the size of the multicut as upper bound for the chromatic index of the processor multigraph is a gross overestimate and we can do better. Recall from Chapter 2 the two upper bounds $\chi'(M_P) \leq \lfloor 3\Delta(M_P)/2 \rfloor$ and $\chi'(M_P) \leq \Delta(M_P) + \mu(M_P)$ by Shannon [155] and Vizing [160]. It is clear that with the help of the arrays `deg` and `mul` these bounds can be

computed in time $\mathcal{O}(|P|)$ and $\mathcal{O}(|P^2|)$ respectively:

$$z^*(\mathbf{m}) = t_a \cdot \max_{p \in P} \text{size}(p) + t_c \cdot \left\lfloor \frac{3}{2} \cdot \max_{p \in P} \text{deg}(p) \right\rfloor \quad \text{or} \quad (9.7)$$

$$z^*(\mathbf{m}) = t_a \cdot \max_{p \in P} \text{size}(p) + t_c \cdot \left(\max_{p \in P} \text{deg}(p) + \max_{p < k} \text{mul}(p, k) \right). \quad (9.8)$$

Like for the maximum processor size, there is again no way to efficiently keep track of the maximum degree and the maximum multiplicity. So in general we cannot avoid the $\mathcal{O}(|P|)$ and $\mathcal{O}(|P^2|)$ loops required to determine these numbers. However, implementing the upper bounds in (9.7) and (9.8) together with the update algorithms described above yields an algorithm in which one iteration of the local search loop takes either time $\mathcal{O}(\max\{|P|, \Delta(G)\})$ or time $\mathcal{O}(\max\{|P^2|, \Delta(G)\})$.

Let us finally describe an objective function surrogate that is based on matchings. The good thing in the case $|P| = 4$ was that each edge $e \in M_P$ was covered by *exactly one* matching in $\mathcal{M}(K_{|P|})$. Following this observation we fix a *partition into maximum matchings* of $K_{|P|}$. This is a set \mathcal{M} of maximum matchings in $K_{|P|}$ such that each edge $e \in K_{|P|}$ is covered by *exactly one* matching in \mathcal{M} . Such a cover by maximum matchings only exists if $|P|$ is even, so we will assume an even number of processors in the following. With this partition, we define our objective function surrogate as

$$z^*(\mathbf{m}) = t_a \cdot \max_{p \in P} \text{size}(p) + t_c \cdot \sum_{M \in \mathcal{M}} \max_{pk \in M} \text{mul}(p, k). \quad (9.9)$$

With the arrays described above, evaluating (9.9) takes time $\mathcal{O}(|P^2|)$ and one iteration of the local search loop in Algorithm 15 can thus be implemented in time $\mathcal{O}(\max\{|P^2|, \Delta(G)\})$ (notice that the `deg` and `nei` arrays are not used). If $|P| \gg \Delta(G)$ then we can slightly improve the asymptotic performance of this algorithm by introducing two new arrays. The two new arrays are:

Name	Index	Content
<code>cover</code>	$P \times P$	<code>cover(pk)</code> yields the matching that covers edge $pk \in K_{ P }$.
<code>mmul</code>	\mathcal{M}	<code>mmul(M)</code> is the maximum multiplicity of an edge in matching M .

The array `cover` is initialised in time $\mathcal{O}(|P^2|)$ and is then unchanged throughout the whole algorithm. Given an initial assignment \mathbf{m} , array `mmul` is easily initialised along with the array `mul` (see also Algorithm 16):

Algorithm 18 *Initialise mul and mmul*Input: Processor set P and matching cover \mathcal{M} .Output: Initialised arrays mul and mmul.

```

1  For  $(p, k) \in P \times P$  Do  $\text{mul}(p, k) = 0$ .
2  For  $M \in \mathcal{M}$  Do  $\text{mmul}(M) = 0$ .
3  For  $uv \in E$  Do
4    If  $\mathbf{m}(u) \neq \mathbf{m}(v)$  Then
5      Increase  $\text{mul}(\mathbf{m}(u), \mathbf{m}(v))$  by 1.
6      Increase  $\text{mul}(\mathbf{m}(v), \mathbf{m}(u))$  by 1.
7      If  $\text{mul}(\mathbf{m}(v), \mathbf{m}(u)) > \text{mmul}(\text{cover}(\mathbf{m}(u)\mathbf{m}(v)))$ 
8        Then set  $\text{mmul}(\text{cover}(\mathbf{m}(u)\mathbf{m}(v))) = \text{mul}(\mathbf{m}(v), \mathbf{m}(u))$ .
9    End If
10 End For
11 Return the arrays.

```

Since we have at most $|P|-1$ matchings in \mathcal{M} , this initialisation takes time $\mathcal{O}(|E|)$. Similarly, mmul can be updated along with mul:

Algorithm 19 *Update mul and mmul*Input: Node u to be moved, source p_u and target p'_u , arrays mul and mmul.Output: Update arrays mul and mmul.

```

1  For  $v \in N(u)$  Do
2    Set  $p_v = \mathbf{m}(v)$ .
3    Set  $\text{mul}(p_u, p_v) = \text{mul}(p_u, p_v) - 1$ .
4    Set  $\text{mul}(p_v, p_u) = \text{mul}(p_v, p_u) - 1$ .
5    Set  $M = \text{cover}(p_v p_u)$ .
6    If  $\text{mmul}(M) = \text{mul}(p_v p_u) + 1$  Then
7      Set  $\text{mmul}(M) = 0$ .
8      For  $e \in M$  Do
9        If  $\text{mmul}(M) < \text{mul}(e)$  Then set  $\text{mmul}(M) = \text{mul}(e)$ .
10     End For
11   End If
12   Set  $\text{mul}(p'_u, p_v) = \text{mul}(p'_u, p_v) + 1$  and  $\text{mul}(p_v, p'_u) = \text{mul}(p_v, p'_u) + 1$ .
13   Set  $M = \text{cover}(p_v p'_u)$ .
14   If  $\text{mmul}(M) < \text{mul}(p_v p'_u)$  Then  $\text{mmul}(M) = \text{mul}(p_v p'_u)$ .
15 End For
16 Return mul and mmul.

```

With the `mmul` array the objective function value is now computed as

$$z^*(\mathbf{m}) = t_a \cdot \max_{p \in P} \text{size}(p) + t_c \cdot \sum_{M \in \mathcal{M}} \text{mmul}(m). \quad (9.10)$$

Since \mathcal{M} contains exactly $|P|$ matchings, the computation of (9.10) takes time $\mathcal{O}(|P|)$. However, as you can see in steps 6 through 11 of Algorithm 19, keeping `mmul` up to date costs us an extra loop in Algorithm 19. Thus the time required for one iteration of the local search loop in Algorithm 15 does not reduce to $\mathcal{O}(|P|)$ but to $\mathcal{O}(\Delta(G) \cdot |P|)$.

In order to further improve the performance of the local search algorithm with objective function (9.9) or (9.10) we do not use only a single matching partition \mathcal{M} . Instead we use k (where k is a parameter) different partitions $(\mathcal{M}_1, \dots, \mathcal{M}_k)$ and determine the objective function as

$$z^*(\mathbf{m}) = t_a \cdot \max_{p \in P} \text{size}(p) + t_c \cdot \min_{i=1, \dots, k} \sum_{M \in \mathcal{M}_i} \max_{e \in M} \text{mul}(e). \quad (9.11)$$

We must then of course use k different arrays `cover` and `mmul`. The update of these arrays and the evaluation of (9.11) costs us time $\mathcal{O}(k \cdot \min\{\Delta, |P|\} \cdot |P|)$ in each iteration of the local search loop in Algorithm 15.

9.3 Tabu Search

One big drawback of the local search algorithm is that it is easily trapped in local minima. A point $x \in \mathbb{R}^n$ is a *local minimum* for a function $f : \mathbb{R}^n \rightarrow \mathbb{R}$ if there is $\varepsilon > 0$ such that $f(x) \leq f(x')$ for all $x' \in \mathbb{R}^n$ with $\|x - x'\| < \varepsilon$. A *global minimum* for f is a point $x \in \mathbb{R}^n$ such that $f(x) \leq f(x')$ for all $x' \in \mathbb{R}^n$. If f is convex, local and global minima coincide, otherwise they may be different and there may also be multiple local minima.

Recall the neighbourhood graph $G_N = (V_N, E_N)$ for a local search algorithm in which each node represents a feasible solution and moves are represented as edges between the nodes representing pre-image and image of the move. A node $u \in V_N$ in this graph is a local minimum for a function $f : V \rightarrow \mathbb{R}$ if $f(u) \leq f(v)$ for all $uv \in \delta_{G_N}^+(u)$. In other words, u is a local minimum if no move allowed at u improves the objective function value. As soon as a local search algorithm reaches a local minimum it stops (see Algorithm 14). Unfortunately, local minima in the neighbourhood graph can occur even if the objective function is convex. This is because we allow only local changes to a feasible solution and given a node $u \in V_N$ it may easily happen that none of the local changes allowed improves the objective function value while there might be “global” changes that do. In this case, a *sequence* of moves (i. e., local changes) would be required to reach a node that has a better objective function value but since we consider only single moves we never find this sequence and stop at the local minimum.

Many strategies have been invented and tested to defeat or remedy this local-minimum dilemma of local search algorithms. The most prominent among these strategies are Tabu Search [65], Genetic or Evolutionary algorithms [157] and Simulated Annealing [115]. However, the asymptotic running times and performance guarantees of most of these algorithms are too bad for our purpose and we only implemented the Tabu Search approach.

The idea behind Tabu Search algorithms is rather simple: we do not only apply enhancing moves. Instead we apply in each iteration of the local search loop the best possible move – even if this move yields a worse mapping. To prevent the algorithm from thrashing back and forth at local minima we also keep a list T of moves that are tabu. As soon as a move has been applied, it is put into T and cannot be reverted for a certain number of iterations. The hope is that this strategy allows the algorithm to “climb” out of local minima and to have a greater chance to find a global minimum. The Tabu Search algorithm contains no explicit stopping criterion, so it is common to stop the algorithm after a prescribed number of iterations and to return the best solution found so far. Notice that this best solution is not necessarily the current mapping in the algorithm but might have been found in an earlier iteration.

We describe the Tabu Search algorithm for the simple node move (remember Figure 9.1). We leave out details about initialising and updating arrays or computing objective function values as this can be done in a analogous fashion as before.

Algorithm 20 *Tabu Search*

Input: An initial mapping \mathbf{m}_0 , an expiry r and an iteration limit I .

Output: A potentially improved mapping \mathbf{m} .

```

1  Evaluate the objective function value  $z^0$  of  $\mathbf{m}_0$ .
2  Set  $T(u, p) = 0$  for all  $(u, p) \in V \times P$ .
3  Set  $z^* = z^0$  and  $\mathbf{m}^* = \mathbf{m}_0$ .
4  For  $i = 1, \dots, I$  Do
5      Set  $z^i = \infty$ .
6      For  $(u, p) \in V \times P$  Do
7          If  $T(u, p) > i$  Continue with the next pair.
8          Set  $\mathbf{m} = \mathbf{m}_{i-1}$  and  $\mathbf{m}(u) = p$ .
9          If  $\mathbf{m}$  is infeasible Then Continue with the next pair.
10         Determine the objective function value  $z$  of  $\mathbf{m}$ .
11         If  $z < z^i$  Then set  $z^i = z$ ,  $\mathbf{m}_i = \mathbf{m}$ ,  $u^i = u$  and  $p^i = p$ .
12     End For
13     If  $z^i < z^*$  Then set  $z^* = z^i$ ,  $\mathbf{m}^* = \mathbf{m}^i$  and  $T(u^i, \mathbf{m}_{i-1}^{-1}(u^i)) = i + r$ .
14 End For
15 Return  $\mathbf{m}^*$ .
```

As you can see in the above description we implement the tabu list by a simple two-dimensional array T . Entry $T(u, p)$ in this array yields the smallest iteration number in which node u may be moved back to processor p (see step 7).

9.4 Edge-Colouring the Processor Multigraph

As stated earlier, none of the initial assignment or local improvement strategies described above explicitly determines an edge-colouring for the processor multigraph that corresponds to the current mapping. Since we used objective function surrogates none of the algorithms requires such a colouring. However, if we want to construct a feasible solution from a mapping \mathbf{m} returned by any of these algorithms we must determine an edge-colouring for $M_P(\mathbf{m})$. If the solution to be constructed should have a small objective function value the edge-colouring should use a small (or even minimal) number of colours. Since finding a minimal edge-colouring is \mathcal{NP} -hard we must again resort to edge-colouring heuristics that are fast but are not guaranteed to find a minimal colouring. In our code, we use three different colouring algorithms: one that is applicable to the case $|P| = 4$ only, one that is based on the proof of Shannon's or Vizing's upper bound on the chromatic index of multigraphs [160] and one that is a generalisation of the preprocessing technique suggested in [24].

If $|P| = 4$ then the chromatic index $\chi'(M_P(\mathbf{m}))$ of the processor multigraph $M_P(\mathbf{m}) = (P, E(M_P(\mathbf{m})))$ is given by

$$\begin{aligned} \chi'(M_P(\mathbf{m})) = & \max\{\mu_{M_P}(01), \mu_{M_P}(23)\} + \\ & \max\{\mu_{M_P}(02), \mu_{M_P}(13)\} + \\ & \max\{\mu_{M_P}(03), \mu_{M_P}(12)\}. \end{aligned}$$

Constructing $M_P(\mathbf{m})$ and finding its chromatic index can thus be done in time $\mathcal{O}(|E|)$, provided that $|P| = 4$.

We already mentioned that Shannon [155] proved that $\chi'(G) \leq \lceil 3\Delta(G)/2 \rceil$ for any multigraph G . His proof is constructive and yields an $\mathcal{O}((n + \Delta)m)$ time algorithm to edge-colour any multigraph with no more than $\lceil 3\Delta(G)/2 \rceil$ colours. Similarly, Vizing [160] proved $\chi'(G) \leq \Delta(G) + \mu(G)$ for any multigraph G . His proof is also constructive and yields an algorithm to edge-colour any multigraph with no more than $\Delta(G) + \mu(G)$ colours in time $\mathcal{O}((n + \Delta)m)$. Both algorithms usually use as many colours as are allowed by their performance estimates. However, by using an incremental version of the algorithms', their actual performance can sometimes be improved. To this end, we only allow $\Delta(G)$ colours at the beginning of the edge-colouring algorithm. When we reach an edge uv such that u and v do not have a common missing colour, we attempt to change the current (partial) colouring of G so as to produce a common missing colour at these two nodes. If this succeeds, we use the common missing colour to edge-colour uv , otherwise we add a new colour to the set of available colours and edge-colour uv with this colour.

Algorithm 21 *Incremental Edge-Colouring*Input: A multigraph $G = (V, E)$.Output: An edge-colouring $\mathbf{c} : E \rightarrow \mathbb{N}$ for G .

```

1  Set  $C = \{1, \dots, \Delta(G)\}$ .
2  For  $uv \in E$  Do
3    If there is a colour  $c \in C$  that is missing from  $u$  and  $v$  Then
4      Set  $\mathbf{c}(uv) = c$ .
5    Else
6      Try hard to recolour  $G$  and make a colour  $c' \in C$  missing from  $u$  and  $v$ .
7      If the previous step succeeded Then
8        Set  $\mathbf{c}(uv) = c'$ .
9      Else
10       Set  $\mathbf{c}(uv) = |C| + 1$ .
11       Set  $C = C \cup \{|C| + 1\}$ .
12     End If
13   End For
14 Return  $\mathbf{c}$ .

```

“Trying hard” to make a colour missing from u and v in step 6 in this algorithm is performed either by the recolouring operations in Vizing’s algorithm or by the operations described in Shannon’s algorithm (see Section 2.2.2). Depending on which recolouring operations we use, it is clear that Algorithm 21 never uses more than $\Delta(G) + \mu(G)$ or $\lfloor 3\Delta(G)/2 \rfloor$ colours to edge-colour G .

Caprara and Rizzi [24] suggested the following two-stage approach for edge-colouring a multigraph $G = (V, E)$. Let $X = \{v \in V : \deg_G(v) = \Delta(G)\}$ be the set of nodes of maximum degree. If G contains a matching M that covers all nodes in X (a so-called X -matching), then colour all edges in M by the same colour, remove M from G and apply to $G \setminus M$ one of the edge-colouring algorithms given in [67, 92, 133]. They proved that the preprocessing step of removing an X -matching improves the performance guarantee of these algorithms.

Our edge-colouring heuristic generalises the algorithm of Caprara and Rizzi in two points:

1. We do not look only for X -matchings. Instead we define an edge weighting $w : E \rightarrow \mathbb{N}$ by

$$w(e) = \begin{cases} 1 & \text{if } e \cap X = \emptyset, \\ |e \cap X| \cdot |E| & \text{otherwise} \end{cases}$$

and find a maximum weight matching with respect to these weights. It is clear that an X -matching is a matching of maximum weight and if an X -matching exists, then

a maximum weight matching covers all nodes in X . Moreover, such a matching is not restricted to edges that cover nodes in X . It may contain additional edges that are not incident to nodes in X and is thus potentially larger (and $G \setminus M$ therefore smaller) than the pure X -matchings introduced by Caprara and Rizzi.

On the other hand, if no X -matching exists, then a maximum weight matching is a matching covering as many nodes in X as possible.

2. Instead of removing the matching found and then applying another edge-colouring algorithm, we recurse on the reduced graph. In other words, we find and remove maximum weight matchings until the remaining graph is empty.

This gives rise to the following algorithm (see also [74]):

Algorithm 22 *Edge-Colouring with Matchings*

Input: A multigraph $G = (V, E)$ with n nodes and m edges.

Output: An edge-colouring $\mathbf{c} : E \rightarrow \mathbb{N}$ for G .

```

1  Set  $G' = (V, E') = G$  and  $d(u) = \deg_G(u)$ .
2  Set  $c = 0$ .
3  While  $E' \neq \emptyset$  Do
4    Let  $G'_s = (V, E'_s)$  be the simple graph on  $n$  nodes in which two nodes  $u$  and
       $v$  are adjacent if  $uv \in E'$ .
5    For  $uv \in E'_s$  set
      
$$w(uv) = \begin{cases} 2|E'| & \text{if } d(u) = d(v) = \Delta(G'_s), \\ |E'| & \text{if } d(u) = \Delta(G'_s) \text{ or } d(v) = \Delta(G'_s) \text{ and} \\ 1 & \text{otherwise.} \end{cases}$$

6    Find a matching  $M$  in  $G'_s$  that is maximal with respect to  $w$ .
7    For  $uv \in M$  Do
8      Set  $\mathbf{c}(uv) = c$ .
9      Decrement  $d(u)$  and  $d(v)$ .
10   End For
11   Set  $c = c + 1$  and  $E' = E' \setminus M$ .
12 End While
13 Return  $\mathbf{c}$ .
```

A maximum weight matching can be found in time $\mathcal{O}(n^3)$ (see e. g., [60]) and all other operations in the outermost loop of Algorithm 22 are easily implemented in time $\mathcal{O}(n^2)$. So the algorithm has a worst case running time $\mathcal{O}(mn^3)$, as opposed to $\mathcal{O}((n+\Delta)m)$ for Vizing's

and Shannon's algorithm. It also does not offer any performance guarantee. However, empirical analysis [74, 158] showed that Algorithm 22 is fast in practice and often produces results that are better than those produced by approximation algorithms from the literature.

Another good edge-colouring algorithm was described in [67, 92, 133]. This algorithm outperforms Vizing's algorithm and also the matching-based algorithm just described from a theoretical point of view. However, this algorithm is very complex and has – to the best of our knowledge – never been implemented. So we did not attempt to implement it either.

9.5 Algorithm for a Feasible Starting Solution

So far we have described several classes of algorithms: algorithms for determining an initial feasible solution, local-search and improvement algorithms, and edge-colouring algorithms. With these ingredients we are now able to define a heuristic algorithm to determine a high-quality feasible solution to (OGPC). All the algorithms defined so far in this chapter serve as subroutines to this new algorithm to be defined. Apart from the problem instance to solve our algorithm takes the following parameters:

$\mathcal{I} = ((I_1, S_1), \dots, (I_i, S_i))$ A list of initial assignment algorithms I_i (see Section 9.1) together with their respective parameter settings S_i (if any). If the same algorithm is to be called multiple times but with different parameter settings we assume that it is contained in \mathcal{I} once for each parameter setting.

$\mathcal{A} = (A_1, \dots, A_a)$ A list of improvement strategies (see Sections 9.2 and 9.3). From the basic algorithms Algorithm 14 and Algorithm 20 we derive multiple algorithms by allowing different kinds of moves and using different objective function surrogates.

$\mathcal{E} = (E_1, \dots, E_e)$ A list of edge-colouring algorithms (see Section 9.4).

Using the (sets of) algorithms just described as subroutines we can construct a meta-heuristic (see [20] for example) to find a good feasible solution for (OGPC)

Algorithm 23 *Feasible Solution*

Input: Grid graph $G = (V, E)$, processor set P , capacity limit K , time factors t_a and t_c and algorithm lists \mathcal{I} , \mathcal{A} and \mathcal{E} as described above.

Output: A feasible mapping \mathbf{m} .

```
1  Set  $\mathbf{m}^* = \emptyset$  and  $z^* = \infty$ .
   // Find an initial solution.
2  For  $(I, S) \in \mathcal{I}$  Do
3     $\mathbf{m} = I(G, P, K, S)$ 
4    For  $E \in \mathcal{E}$  Do
5      Use  $E$  to find an edge-colouring of  $M_P(\mathbf{m})$ .
6      Let  $c$  denote the number of colours in the edge-colouring.
7      Set  $z = t_a \cdot b(\mathbf{m}) + t_c \cdot c$ .
8      If  $z < z^*$  Then set  $z^* = z$  and  $\mathbf{m}^* = \mathbf{m}$ .
9    End For
10 End For
   // Improve the initial solution.
11 For  $A \in \mathcal{A}$  Do
12    $\mathbf{m}' = A(\mathbf{m})$ .
13   For  $E \in \mathcal{E}$  Do
14     Use  $E$  to determine an edge-colouring of  $M_P(\mathbf{m}')$ .
15     Let  $c'$  denote the number of colours used.
16     Set  $z' = t_a \cdot b(\mathbf{m}') + t_c \cdot c'$ .
17     If  $z' < z^*$  Then set  $z^* = z'$  and  $\mathbf{m}^* = \mathbf{m}'$ .
18   End For
19 End For
20 Return  $\mathbf{m}^*$ .
```

You may wonder why we use multiple algorithms to find an initial solution and multiple algorithms for improvement. The reason will become clearer in Chapter 10 when we analyse the performance of the different algorithms. There we will see that there is not one single combination of algorithms that performs best. Instead, the algorithms with the best performance change from instance to instance. However, since all algorithms are pretty fast it is reasonable to just test a few so as to improve our odds that we use the best one.

9.6 Rounding Heuristics

As opposed to the heuristic algorithms described above rounding heuristics are not executed before but *during* Branch-and-Cut. These heuristics start with the optimal (fractional)

solution of the LP-relaxation at the current Branch-and-Cut node and attempt to construct a new integral feasible solution from it. The three most prominent and most successful general purpose strategies are *feasibility pump*, *local branching* and *relaxation induced neighbourhood search* (RINS).

Given the optimal fractional solution x^* to the current node, the idea behind feasibility pump [1, 18, 55] is to obtain a (not necessarily feasible) integral solution \tilde{x} by rounding all fractional entries of x^* . The node LP is then resolved, but this time with objective to minimise the distance between the solution and \tilde{x} . This yields a new fractional solution and the process can be iterated until either an integral solution is found or a time limit is reached.

If an incumbent solution \bar{x} is available, the information provided by the optimal solution x^* to the node LP can be used to improve this solution. In RINS [30] this is done by fixing all variables that have the same value in \bar{x} and x^* and re-solving the reduced MIP. This yields a new fractional solution and the process is iterated until a new integral feasible solution is found or a time limit is reached.

Another strategy to improve an incumbent solution is local branching [56]. This approach is essentially equivalent to a k -opt neighbourhood search and works as follows: Given the incumbent solution \bar{x} , add a new constraint to the original mixed integer program that requires each solution to have Hamming distance at most k to \bar{x} . If k is small, the augmented MIP is easily solved and an integral solution to the augmented problem is also feasible for the original problem. Obviously, this strategy does not exploit information provided by fractional solutions to node LPs and is usually only applied at the node at which the incumbent was discovered.

All the strategies described are based on iterated solving of mixed integer programs and are thus potentially very expensive. As we will see in Chapter 10 the solutions provided by our local search heuristics usually have a very high quality, that is, they are often close to optimal. It is thus unlikely to find feasible solutions that improve upon the incumbent solution during a Branch-and-Bound or Branch-and-Cut algorithm. We therefore do not invest much time into rounding heuristics that are as sophisticated as the ones described above. Instead we only apply two simple and fast rounding schemes which we describe now.

9.6.1 Probabilistic Rounding

One simple and widely used way of rounding is to consider the values of the LP relaxation at the current node as probability distribution. This approach has especially proven successful in the scheduling community, see e. g., [153] and references therein. For model (SLOT) we

use the term $\gamma_{u,p}$ that was defined in Section 6.4. For each (fractional) solution x' we have

$$\gamma'_{u,p} \in [0, 1] \quad \text{and} \quad \sum_{p \in P} \gamma'_{u,p} = 1$$

for any node $u \in V$. We may therefore consider the probability function that maps node u to processor p with probability $\gamma'_{u,p}$. Performing random assignments based on this distribution for all nodes $u \in V$ that have at least one fractional value $\gamma'_{u,p}$, we obtain a mapping \mathbf{m}' . If \mathbf{m}' respects the capacity constraints we may edge-colour the resulting processor multigraph $M_P(\mathbf{m}')$ by any of the algorithms suggested above, thereby obtaining a feasible solution to (OGPC). In order to improve our odds to find a good solution, we do not only generate one random assignment. Instead we create a fixed number of assignments, edge-colour the resulting processor multigraph, and compare the solution obtained with the incumbent solution. If the randomly generated solution is better than the incumbent we have found an improving solution and can replace the incumbent solution.

Determining the values $\gamma'_{u,p}$ for each $(u, p) \in V \times P$ requires time $\mathcal{O}(|E \times P^2|)$. Given \mathbf{m}' , we can setup $M_P(\mathbf{m}')$ in time $\mathcal{O}(|E|)$. Edge-colouring the multigraph $M_P(\mathbf{m}') = (P, E_{M_P})$ takes time $\mathcal{O}(|E|)$ if $|P| = 4$ and time $\mathcal{O}((|P| + \Delta(M_P(\mathbf{m}')))) \cdot |E|$ otherwise. Assuming that random selection of target processors can be done in constant time, we have that generating one random feasible solution takes time $\mathcal{O}(|E \times P^2|)$ if $|P| = 4$ and time $\mathcal{O}(\max\{|E|, |P^2|\} \cdot |E|)$ otherwise.

9.6.2 Prioritised Rounding

In order to obtain another rounding strategy, we interpret the value $x'_{e,p,k}$ in a fractional solution x' as the *desire* of edge e to be mapped to slot (p, k) . In a greedy-like algorithm we then attempt to satisfy as many desires as possible. To this end, we initialise an empty mapping \mathbf{m}' by setting $\mathbf{m}'(u) = \text{NIL}$ for all $u \in V$ and call an ordered triple $(uv, p, k) \in E \times P \times P$ *feasible*, if $\mathbf{m}'(u) \in \{\text{NIL}, p\}$ and $\mathbf{m}'(v) \in \{\text{NIL}, k\}$. We iterate over the ordered triples (e, p, k) in $E \times P \times P$ in order of non-increasing values of $x'_{e,p,k}$. If we encounter a feasible triple (uv, p, k) we set $\mathbf{m}'(u) = p$ and $\mathbf{m}'(v) = k$. Assuming that all blocks were assigned by this procedure, we have found a mapping \mathbf{m}' . If additionally this mapping satisfies the capacity constraint it gives rise to a feasible solution by edge-colouring the resulting processor multigraph $M_P(\mathbf{m}')$.

Initialising an empty mapping and putting the ordered triples in $E \times P \times P$ into the desired order can be done in time $\mathcal{O}(\log(|E \times P \times P|)|E \times P \times P|)$. Adding the time required for edge-colouring $M_P(\mathbf{m}')$ we obtain an algorithm that runs in time $\mathcal{O}(\log(|E \times P \times P|)|E \times P \times P|)$ if $|P| = 4$ and time $\mathcal{O}(\max\{\log(|E \times P \times P|) \cdot |P^2|, |E|\}|E|)$ otherwise.

9.7 Heuristic Constraints

Another way to speed up the solution process for an instance of (OGPC) are *heuristic constraints* (also called *heuristic inequalities*). Such constraints are represented by inequalities that are not valid for the associated polytope in general. However, they are either very likely to be satisfied by all optimal solutions or are supposed to allow near-optimal solutions. The aim of these inequalities is to simplify the problem by reducing the number of variables or the complexity of the LP-relaxation or the associated polyhedron. So by introducing (some of) the heuristic inequalities described below, we may speed up the solution process, but we pay with a potential loss of the optimal solution.

9.7.1 Connected Processors

The idea behind this constraint is the same as for Algorithm 13 and was described for similar problems in [93]. The aim is to make the subgraph that is induced by the nodes mapped to one processor connected. The hope is that this yields a small multicut. To this end, we create a directed graph $G' = (V, A)$ from the undirected grid graph $G = (V, E)$ by setting $A = \{(u, v) : uv \in E\}$, i. e., each edge in G gives rise to two anti-parallel edges in G' . Moreover we define a weight function $w : A \rightarrow \mathbb{N}$ as $w((u, v)) = \kappa_v$. For each node $u \in V$ we then compute the shortest path tree T_u rooted at u with respect w . Let $d_{T_u}(u, v)$ denote the distance of the shortest u - v -path in this tree. We set $\bar{N}(u) = \{v \in V \setminus \{u\} : d_{T_u}(u, v) > K - \kappa_u\}$ for all $u \in V$. So if node u is mapped to processor p and the subgraph induced by the nodes mapped to p is supposed to be connected then no node from $\bar{N}(u)$ can be mapped to processor p . This is because a u - v -path would require so many nodes on p that the capacity limit of p would be exceeded. Recalling the definition of $\gamma_{u,p}$ from Section 6.4 this gives rise to the following valid inequality in (SLOT):

$$\gamma_{v,p} + \gamma_{u,p} \leq 1 \quad i \in V, v \in \bar{N}(u). \quad (9.12)$$

We can carry this idea further: We may pick an arbitrary node $u \in V$ and require that this node be mapped to the first processor p_0 . Then all variables that imply an assignment of a node $v \in \bar{N}(u)$ to p_0 can be fixed to zero. Moreover, if $\bar{N}(u) \neq \emptyset$ then we may pick a node $u' \in \bar{N}(u)$ and require that this node be mapped to the second processor p_1 (since it cannot be mapped to p_0). Now all variables implying that a node from $\bar{N}(u')$ is mapped to processor p_1 can be fixed to zero. If $\bar{N}(u) \cap \bar{N}(u') \neq \emptyset$ we can repeat the above arguments for a node $u'' \in \bar{N}(u) \cap \bar{N}(u')$ and so forth. Depending on the grid graph $G = (V, E)$ and the capacity limit K this potentially allows to fix a lot of variables. This strategy of repeatedly fixing variables corresponding to nodes that cannot be mapped to the same processors is called *first order assignment* in [93].

9.7.2 Easily Colourable Processor Multigraphs

One aspect of (OGPC) that renders this problem difficult to solve is the embedded edge-colouring problem. Especially for large numbers of processors the formulation of this optimisation problem leads to large numbers of variables and constraints. This stems from the fact that for general multigraphs the edge-colouring problem is \mathcal{NP} -hard. However, there are classes of multigraphs for which the chromatic index is known *and* that are easily coloured (there are also classes for which the chromatic index can be determined in polynomial time but no polynomial time algorithm is known to compute a minimal edge-colouring). One of these classes are bipartite multigraphs. A *bipartite* graph (simple or multigraph) $G = (V, E)$ is a graph the nodes of which can be partitioned into two non-empty sets A and B such that $E \subset A \times B$. In other words, all edges of G have one endpoint in A and one endpoint in B . One often writes $G = (A, B, E)$ for a bipartite multigraph $G = (A \cup B, E)$. For a bipartite multigraph $G = (V, E)$ we have $\chi'(G) = \Delta(G)$ and polynomial time algorithms are known to compute an edge-colouring for G that uses $\Delta(G)$ colours [152].

If we require that the processor multigraph is bipartite we have $\chi'(M_P(\mathbf{m})) = \Delta(M_P(\mathbf{m}))$ for any feasible mapping \mathbf{m} . This allows us to drastically reduce the number of variables and constraints in the (SLOT)-model (or the (SLOT*)-model):

(BIPARTITESLOT)	$\text{minimise } t_a \cdot b + t_c \cdot D_{\max} \tag{9.13a}$
	$\sum_{k \neq p} \sum_{e \in E} (x_{e,p,k} + x_{e,k,p}) \leq D_{\max} \quad p \in P \tag{9.13b}$
	$\sum_{k \in P} x_{uv,p,k} - \sum_{k \in P} x_{f,p,k} = 0 \quad uv \in E, p \in P, f \in \delta^+(u) \tag{9.13c}$
	$\sum_{k \in P} x_{uv,p,k} - \sum_{k \in P} x_{f,k,p} = 0 \quad uv \in E, p \in P, f \in \delta^-(u) \tag{9.13d}$
	$\sum_{k \in P} x_{uv,k,p} - \sum_{k \in P} x_{f,p,k} = 0 \quad uv \in E, p \in P, f \in \delta^+(v) \tag{9.13e}$
	$\sum_{k \in P} x_{uv,k,p} - \sum_{k \in P} x_{f,k,p} = 0 \quad uv \in E, p \in P, f \in \delta^-(v) \tag{9.13f}$
	$\text{Processor multigraph is bipartite} \tag{9.13g}$
	$D_{\max} \in \mathbb{N}_+ \tag{9.13h}$
	$x \text{ binary} \tag{9.13i}$

In this model variable D_{\max} models the maximum degree in the processor multigraph. This is achieved by (9.13b) which simply counts all outgoing edges of processor p and requires that D_{\max} is at least as big as this number. Constraints (9.13c) through (9.13f) require correct mapping of edges and are the same as in (SLOT). The exact formulation of (9.13g) depends on the actual class of bipartite graph that we allow. We will give some examples for that below. Notice that (BIPARTITESLOT) does not contain any y - or z -variables and is thus much smaller than (SLOT) or (SLOT*). On the other hand, a solution to (BIPARTITESLOT) does not explicitly contain an edge-colouring of the processor multigraph. However, as we said above, it is easy to construct an optimal edge-colouring for this graph in polynomial time.

Let us give three examples of bipartite graphs that yield different specialisations of (9.13g). The first example are cycles. Cycles on an even number of nodes are always bipartite (cycles on an odd number of nodes are never). Hence, if we have an even number of processors we may require the processor multigraph to be a cycle. This means that edges are allowed only between subsequent processors and between the first and the last processor:

$$x_{e,p,k} = 0 \quad pk \in K_{|P|}, |p - k| > 1 \text{ and } \{p, k\} \neq \{0, |P| - 1\}. \quad (9.14)$$

As you can see, this fixes a large number of variables to zero, thus dramatically reducing the size of the problem instance.

If we want to be less restrictive we may require that the simple graph underlying the processor multigraph is a subgraph of the complete bipartite graph on $|P|$ nodes. In other words, we set $A = \{0, \dots, \lceil |P|/2 \rceil - 1\}$ and $B = \{\lceil |P|/2 \rceil, \dots, |P| - 1\}$ (notice that this is valid for odd $|P|$ as well) and require that all edges in the processor multigraph run between A and B :

$$x_{e,p,k} = 0 \quad pk \in K_{|P|}, |\{p, k\} \cap A| \neq 1. \quad (9.15)$$

As opposed to the cycle-restriction above, this restriction is valid for odd numbers of processors as well.

A class of bipartite communication graphs that occur in practice are hypercubes (see Section 6.9). To see that such a graph is indeed bipartite we need the well-known lemma

Lemma 79. [152] *A graph $G = (V, E)$ is bipartite if and only if it contains no cycle with an odd number of nodes.*

Assume that $H = (V, E)$ is a hypercube and $C = (\{v_0, \dots, v_k\}, \{e_0, \dots, e_k\})$ is a cycle in H . For $u \in V$ let n_u denote the node number of u in H . When we move from v_0 to v_1 in C , then – by definition of hypercubes – exactly one bit in the binary representation of n_{v_0} is flipped to obtain n_{v_1} . Consequently each edge in C represents one bit flip. If we traverse C – starting and ending in v_0 – then obviously each bit in the binary representation of node numbers is flipped an even number of times. Thus C contains an even number of edges and therefore also an even number of nodes.

In order to restrict the processor multigraph to a d -dimensional hypercube, we define $D := \{2^i : i = 0, \dots, d-1\}$. This set then contains all possible values by which the processor numbers of adjacent processors may differ. Now M_P can be easily forced to be a d -dimensional hypercube by adding the following constraint to (BIPARTITESLOT):

$$x_{e,p,k} = 0 \quad pk \in K_{|P|}, |p - k| \notin D. \quad (9.16)$$

A hypercube contains more edges than a cycle and fewer edges than the complete bipartite graph on the same number of nodes. Moreover, many computer topologies are organised in a hypercubical fashion. Restricting the processor multigraph to a hypercube is thus a good compromise between problem size and accurate modeling of reality.

Chapter 10

Computational Results

Results! Why, man, I have gotten a lot of results.
I know several thousand things that won't work.
— *Thomas Edison*

In this chapter we present some results that we obtained by our optimisation algorithms for several real-world grid-partitioning instances. The aim of this presentation is to show that (i) including a decent communication model into optimisation produces better partitionings, and (ii) good partitionings can be determined in reasonable time by the approaches we described in the previous chapters. As we already stated in Section 6.8 the (SLOT)-model and its variants are best suited to (OGPC). Thus, all analysis is based on instances of this family of models. We start by describing branching strategies that are suitable to improve the performance of Branch-and-Cut algorithms applied to such instances. Then we give a short description of the test bed instances and finally present performance statistics for our algorithms.

10.1 Branching Strategies

When no more violated cutting planes are found at the current Branch-and-Cut node we must branch (see also Algorithm 4). That is we must split the optimisation problem into a number of (simpler) subproblems. In many state-of-the-art general purpose MIP solvers the default branching strategy for binary integer programs is to pick a binary variable x_i that is fractional at the current node. Two subproblems are then created, one with $x_i = 0$ and one with $x_i = 1$. This strategy leads to a binary Branch-and-Cut tree that potentially enumerates all points in $\{0, 1\}^n$.

In order to reduce the number of points from $\{0,1\}^n$ that are explicitly enumerated by branching one may benefit from the structure of the optimisation problem at hand. For several (sub)structures of integer linear programming problems there are branching strategies that are especially suited to these classes of problems and perform considerably better than the default strategy described above. For a discussion of popular branching strategies please refer to [2]. We describe here the SOS branching rule and a branching rule we call “Branching on virtual variables” that is quite useful in solving instances of the (SLOT)-model.

10.1.1 SOS Branching

SOS branching [14] was especially designed for (integer programming) models that contain constraints like

$$\sum_{i \in I} x_i = 1 \quad x_i \in \{0, 1\} \quad (10.1)$$

where I is some index set and x_i are binary variables. The constraint (10.1) states that *exactly* one variable indexed by I must be 1 (and all other must then be zero). Constraints like (10.1) are called *Special Ordered Set* (or SOS) constraints of type one. The concept can be generalised to yield SOS constraints of type two [14]. Here the right-hand side in (10.1) is two. This implies that exactly two of the binary variables must be non-zero. Moreover, in SOS constraints of type two it is also required that the two non-zero variables are adjacent. Carrying this concept further also SOS constraints of type three [45, 127] and k [127] for $k > 3$ can be defined. If $(x_i)_{i \in I}$ is a set of SOS variables then it is not very helpful to branch on $x_i = 0$ for any $i \in I$. This is because requiring $x_i = 0$ does not restrict the problem much: all x_j for $j \in I \setminus \{i\}$ may still be 0 or 1.

It is thus common in these cases to perform SOS branching, that is to find a binary variable x_{i_0} that is fractional at the current Branch-and-Cut node and that is contained in an SOS constraint with index set I . We then pick a subset $\emptyset \neq J \subseteq I$ and create a new subproblem with $x_j = 0$ for $j \in I \setminus J$ and one with $x_j = 0$ for $j \in J$. This requires that either one of the variables indexed by J is 1 or one of the variables indexed by $I \setminus J$. If $|J| > 1$ or $|I \setminus J| > 1$ then these branching decisions have much more impact on the problem formulation than branching on a single variable.

SOS branching has successfully been applied to related problems [37, 5] and is well applicable to (OGPC) since each of the formulations in Chapter 6 contains at least one SOS constraint, see (6.7b), (6.15b) or (6.30b) for example.

10.1.2 Branching on Virtual Variables

The branching technique presented here is similar to the explicit-constraint branching technique described in [5]. Recall the (SLOT)-model from Section 6.4. In this model we described block-mapping as mapping edges to slots instead of mapping blocks to processor directly. Although this technique resulted in an increase of the number of binary variables it had several advantages for it allowed us to formulate several aspects that the other models did not. The (SLOT)-model contains SOS constraints as well, namely one per edge:

$$\sum_{p,k \in P} x_{e,p,k} = 1 \quad e \in E. \quad (10.2)$$

Since usually $|E| \gg |V|$ the SOS branching strategy would result in a potentially much bigger Branch-and-Cut tree than in the other models.

There is however a way to circumvent this problem and to use an enumeration tree that is no bigger than the tree for the (NAIVE) model for example. To this end, recall from Chapter 6.4 that in the (SLOT)-model a node u is mapped to processor p if and only if

$$\frac{1}{\deg_G(u)} \cdot \left(\sum_{k \in P} \sum_{e \in \delta^+(u)} x_{e,p,k} + \sum_{k \in P} \sum_{e \in \delta^-(u)} x_{e,k,p} \right) = 1.$$

In other words, if we define as before

$$\gamma_{u,p} := \frac{1}{\deg_G(u)} \left(\sum_{k \in P} \sum_{e \in \delta^+(u)} x_{e,p,k} + \sum_{k \in P} \sum_{e \in \delta^-(u)} x_{e,k,p} \right) \quad (10.3)$$

then $\gamma_{u,p} = 1$ if and only if u is mapped to p . Moreover, in any feasible solution to the (SLOT)-model $\gamma_{u,p}$ is either 1 or 0, depending on whether block u is mapped to processor p or not. So we may consider $\gamma_{u,p}$ itself a binary variable. We call this class of variables *virtual* because they are not explicitly part of the model but can be easily derived from model variables via (10.3). Notice that the virtual variables must satisfy the following condition:

$$\sum_{p \in P} \gamma_{u,p} = 1 \quad u \in V. \quad (10.4)$$

This again is an SOS condition for each $u \in V$ and so our branching strategy is to do SOS branching on the virtual binary variables $\gamma_{u,p}$. By (10.3) the branching decision $\gamma_{u,p} = 0$ is equivalent to

$$\sum_{k \in P} \left(\sum_{e \in \delta^+(u)} x_{e,p,k} + \sum_{e \in \delta^-(u)} x_{e,k,p} \right) = 0$$

which in turn is the same as

$$\begin{aligned} x_{e,p,k} &= 0 & e \in \delta^+(u), k \in P \\ x_{e,k,p} &= 0 & e \in \delta^-(u), k \in P. \end{aligned}$$

So at each Branch-and-Cut node we pick a fractional virtual variable γ_{u_0,p_0} . Let $P(u_0) = \{p_{u_0}^1, \dots, p_{u_0}^k\}$ denote the set of processors that is allowed for node u_0 at the current Branch-and-Cut node (initially $P(u) = P$ for all $u \in V$). We then create one branch in which we require that u_0 be mapped to a processor in $P_1 = \{p_{u_0}^i : i = 1, \dots, \lceil k/2 \rceil\}$ and one branch in which u_0 must be mapped to a processor in $P_2 = \{p_{u_0}^i : i = \lceil k/2 \rceil + 1, \dots, k\}$. In the first branch we have $\gamma_{u_0,p} = 0$ for $p \in P_2$ and in the second branch $\gamma_{u_0,p} = 0$ for $p \in P_1$.

In order to perform this strategy, we must compute the values of $\mathcal{O}(|V \times P|)$ virtual variables at each Branch-and-Cut node. According to (10.3) this can be done in time $\mathcal{O}(|E \times P \times P|)$. Notice that this also would be the time required to identify a fractional $x_{e,p,k}$ variable if we were to apply simple SOS branching on these variables. So asymptotically we do not lose performance by using the virtual variables $x'_{u,p}$ for branching instead of the real $x_{e,p,k}$ ones. Even better, since there are only $|V|$ SOS constraints (and not $|E|$ as before) for the virtual variables (see (10.4)) the Branch-and-Cut tree now is potentially much smaller than it was for SOS branching on $x_{e,p,k}$.

10.1.3 SOS Branching on Matchings

Yet another strategy to keep the potential size of the Branch-and-Cut tree small applies only to models based on (SLOTMAP) and is based on the following observation:

Theorem 80. *Assume that (x', b') is a fractional solution at any node in the Branch-and-Cut tree for a model instance that is based on (SLOTMAP). Furthermore let $e_1 = u_1u_2$, $e_2 = u_2u_3$, $e_3 = u_3u_4$ be three edges in G . If*

$$x'_{e_i,p,k} \in \{0, 1\} \quad p, k \in P \text{ and} \quad (10.5)$$

$$\sum_{p,k \in P} x'_{e_i,p,k} = 1 \quad (10.6)$$

for $i = 1, 3$, then (10.5) and (10.6) also hold for $i = 2$.

Proof. If (10.5) and (10.6) are satisfied for $i = 1, 3$, then there are (not necessarily distinct) processors $p_1, p_2, p_3, p_4 \in P$ such that $x'_{e_1,p_1,p_2} = 1$ and $x'_{e_3,p_3,p_4} = 1$. All other variables $x'_{e_1,p,k}$ and $x'_{e_3,p,k}$ are then zero. By constraints (5.35d) through (5.35g) in model (SLOTMAP) (see page 73) the fact $x'_{e_1,p_1,p_2} = 1$ implies $x'_{e_2,p,k} = 0$ for $p \neq p_2$ and $k \in P$. Likewise $x'_{e_3,p_3,p_4} = 1$ implies $x'_{e_2,p,k} = 0$ for $p \in P$ and $k \neq p_3$. Since each feasible solution to (SLOTMAP) must satisfy (10.6) (see also constraint (5.35b) in (SLOTMAP) on page 73) we get that $x'_{e_2,p_2,p_3} = 1$ and $x'_{e_2,p,k} = 0$ for $p \neq p_2$ or $k \neq p_3$. \square

In order to exploit Theorem 80 assume that M is a perfect matching in G (a perfect matching is one that is incident to all nodes). Then it is sufficient to branch on variables $x_{e,p,k}$ for $e \in M$. Once these variables all have integral values, variables $x_{f,p,k}$ will also be integral for $f \notin M$. It is thus sufficient to perform SOS branching on the variable sets $\{x_{e,p,k} : p, k \in P\}$ for $e \in M$. Moreover, by Theorem 80 all variables $x_{f,p,k}$ for $f \notin M$ can be relaxed to be continuous, thus reducing the number of binary variables in the model instance. Since a perfect matching M in G contains exactly $|V|/2$ edges, we are left with only $\mathcal{O}(|V \times P^2|)$ binary x -variables and $\mathcal{O}(|V|)$ special ordered sets (instead of $\mathcal{O}(|E \times P^2|)$ variables and $\mathcal{O}(|E|)$ sets) and the size of the Branch-and-Cut tree is thus expected to be smaller.

If G does not contain a perfect matching, let M be a maximum cardinality matching and add to M all edges $e \in E$ that have only one endpoint matched by M (there can be no edge $uv \in E$ that has both endpoints unmatched, because this would imply that M is not a maximum cardinality matching). The augmented edge set M is no longer a matching, but restricting branching to the edges in M obviously has the same effects as described above.

10.2 Test-Bed Instances

Our test-bed contains the block-structured grids of 9 different real-world simulation problems. All of these problems are related to the analysis of turbulent flows in complex geometries. Figure 10.1 and Table 10.1 give an outline of the test instances and their properties. The

Name	Blocks	Edges	Total Size (in control volumes)
Grid-1	9	19	2688
Grid-2	45	108	60
Grid-3	17	32	9056256
Grid-4	37	67	2686976
Grid-5	52	117	2753536
Grid-6	79	172	1944576
Grid-7	83	176	2138112
Grid-8	91	224	3096576
Grid-9	127	344	3362304

Table 10.1: Test-Bed instances.

grids depicted in Figure 10.1 include toy examples (Grid-1 and Grid-2) as well as grids stemming from different application areas: Grid-3 through Grid-5 were used to simulate a turbo mixer and Grid-6 through Grid-9 were used in simulation of combustion in turbines.

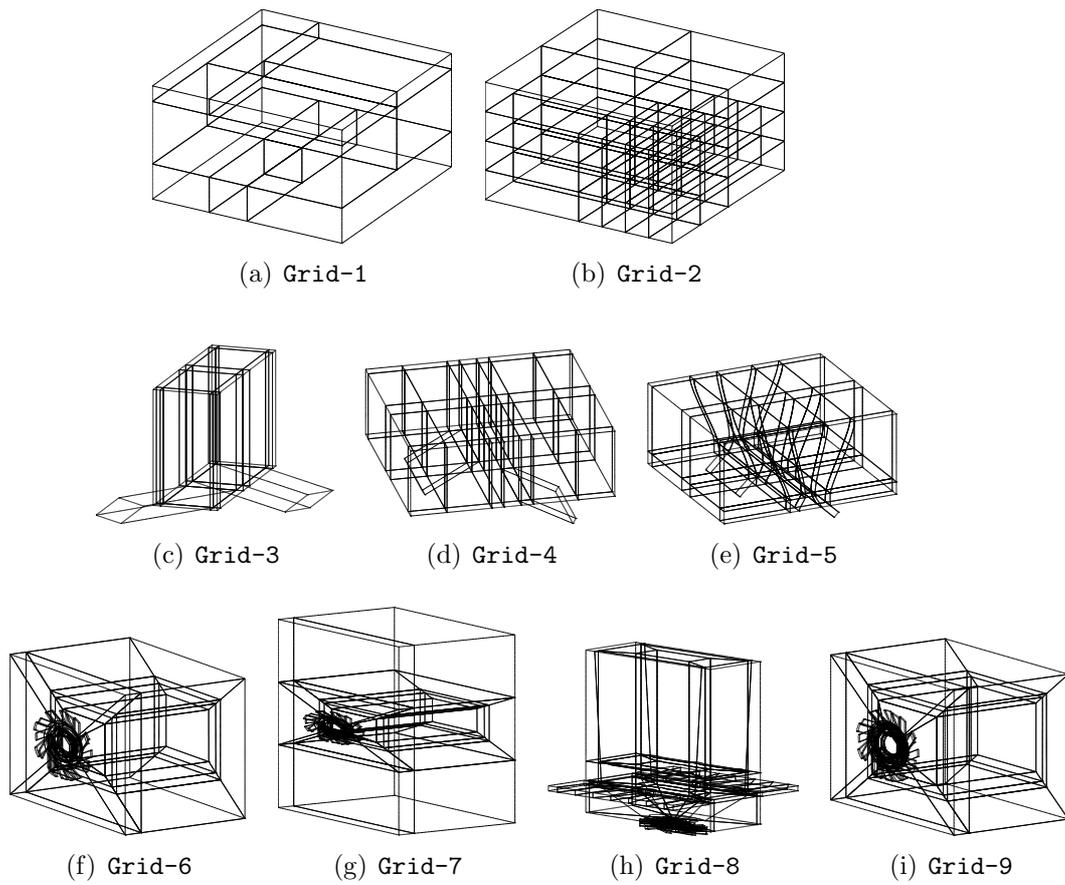


Figure 10.1: Test-Bed instances.

10.3 Preprocessing

Before we actually start finding an optimal block-mapping for a given grid, we first apply some minor modifications to the grid graph and the models described in the previous chapters. The reason for this is – once again – to help the practical solution of the respective problem instances.

We merge each node $u \in V$ with $\deg_G(u) = 1$ with its unique neighbour $v \in V$. We do this by increasing κ_u by κ_v and removing block u as well as edge uv from G . The rationale behind this approach is that for blocks of degree one it seems more efficient to handle them on the same processor as their neighbour rather than introducing communication overhead. Observe that this modification may change the set of optimal solutions and is thus a heuristic preprocessing technique. We nevertheless apply this modification to all our test graphs and Table 10.1 already showed statistics for the simplified graphs.

Apart from reducing the number of nodes and edges in the grid graph, merging nodes of degree one with their respective neighbours has another advantage: the new graph is potentially biconnected. This is easily seen because the graph is still connected and each node now has degree two. It is thus likely that a subset $U \subseteq V$ and edges $F \subseteq E[U]$ can be augmented to a biconnected subgraph (U', F') of G with $U \subseteq U'$ and $F \subseteq F'$. This fact is useful because it improves the probability that a tree in G can be augmented to a biconnected subgraph and thus the biconnectivity augmentation techniques described in Chapter 8 may be applied.

Another very simple preprocessing technique we apply is scaling. Consider inequality (6.30c) which is by definition equivalent to

$$\sum_{uv \in E} \sum_{k \in P} \left(\frac{\kappa_u}{\deg_G(u)} x_{uv,p,k} + \frac{\kappa_v}{\deg_G(v)} x_{uv,k,p} \right) \leq b \quad p \in P. \quad (10.7)$$

Although all block sizes are integral, the coefficients in this capacity inequality are fractional. Even worse, if $\deg_G(u) = 3$ and κ_u is not evenly divisible by three then $\kappa_u / \deg_G(u)$ is a real number that cannot be accurately expressed as IEEE floating point number in computers. We thus multiply (10.7) by the least common multiple $l(V) := \text{lcm}\{\deg_G(u) : u \in V\}$ of all degrees in G and obtained

$$\sum_{uv \in E} \sum_{k \in P} \left(\frac{l(V)\kappa_u}{\deg_G(u)} x_{uv,p,k} + \frac{l(V)\kappa_v}{\deg_G(v)} x_{uv,k,p} \right) \leq l(V) \cdot b \quad p \in P. \quad (10.8)$$

This inequality now has only integer coefficients. Moreover, computational experiments showed that $l(V)$ is usually quite small (so no integer overflow occurs) and our MIP solver performs better on models which have (10.7) replaced by (10.8). Observe that this scaling is of interest only for solving instances of (SLOT) or (SLOT*) by Branch-and-Cut algorithms. Our heuristics are completely independent of the integer programming formulations.

We also tried dividing all block sizes by $g(V) := \gcd\{\kappa_u : u \in V\}$ while multiplying t_a by $g(V)$. It is obvious that this yields an equivalent formulation but our MIP solver showed inferior performance on the modified instances. Hence we refrained from this scaling approach.

10.4 Parameter Setup

In order to analyse the performance of our heuristics and the Branch-and-Cut algorithms implied by discussions in the previous chapters, we have considered mapping our test instances to four and eight processors. In a first attempt we also considered mapping to more than eight processors but this turned out to be infeasible for two reasons. On the one hand integer programming problems for more than eight processors cannot be handled in a reasonable amount of time. On the other hand our grids are simply not designed to be mapped to more than eight processors. In other words, enforcing more than eight active processors typically yields longer simulation times than using at most eight processors.

For all the test runs we have used the following parameter setup:

$$\begin{aligned} t_a &= 1.5 \cdot 10^{-6} s \\ t_c &= 5 \cdot 10^{-2} s \\ K &= \left\lfloor \frac{\kappa_V}{0.5 \cdot |P|} \right\rfloor. \end{aligned}$$

The values for t_a and t_c were obtained by empirical analysis and the capacity limit K was chosen such that the load-balancing efficiency does not drop below 50% (see (6.1) and (6.2) in Section 6.1). All computations were performed on a Pentium 4 processor that runs at 3.2 GHz and has 2 GB of RAM. As mixed integer problem solver we applied CPLEX¹.

10.5 Local Search Heuristics

We start by analysing the performance of the heuristic algorithms we implemented. We first compare the performance of different initial assignment strategies as described in Section 9.1. Then we combine these initial strategies with the local improvement techniques given in Section 9.2.

We already mentioned several times that the cases $|P| = 4$ and $|P| > 4$ are considerably different. We therefore present first results for $|P| = 4$ and afterwards for $|P| > 4$.

¹see www.ilog.com

10.5.1 Four Processors

If we have only four processors, the edge-colouring problem on the processor multigraph is simple (solvable in polynomial time) and the number of pairwise disjoint maximum cardinality matchings in $K_{|P|}$ is small. It is thus feasible to determine the chromatic index of the processor multigraph in each iteration of local search strategies in order to find the objective function value of the current solution.

Initial Assignment Strategies

We executed each of the algorithms described in Section 9.1 for each of the test bed instances described in Figure 10.1 and Table 10.1. For algorithms for that the order in which the nodes are passed may matter, we tested five different ways of sorting nodes: sorting by increasing/decreasing size (indicated in tables by **size+** and **size-**), sorting by increasing/decreasing degree (**degree+** and **degree-**), and not sorting at all (**none**). The computational results for our test grids are depicted in Tables 10.2 through 10.5 (results for the other problem instances can be found in Appendix A.1).

Algorithm	sorting	maxsize	colours	objective	gap
fillup	size-	728	11	551.0920	45.27%
fillup	degree+	784	10	501.1760	39.82%
smallest	size-	728	11	551.0920	45.27%
circular	size-	840	10	501.2600	39.83%
small-degree		failed			
small-multicut		failed			
connected-0.00		896	7	351.3440	14.16%
connected-0.30		896	7	351.3440	14.16%
connected-0.70		896	7	351.3440	14.16%
connected-1.00		896	7	351.3440	14.16%
connected-0.00r		896	7	351.3440	14.16%
connected-0.30r		896	7	351.3440	14.16%
connected-0.70r		896	7	351.3440	14.16%
connected-1.00r		896	7	351.3440	14.16%

Table 10.2: Initial assignment algorithms on **Grid-1** and 4 processors.

The tables show in the first column the name of the algorithm. We used four different values for the swap factor f of Algorithm 13 and these values are appended to the algorithm's name. We also described a variant of Algorithm 13 in which we choose as next block to be assigned the one with highest relative instead of absolute connectivity to the current processor. Results for this variant of the algorithm are marked by an "r" at the end of the name of the algorithm. In the second column the sorting strategy applied (if any) is displayed. In the following columns the tables show the maximum processor size in the returned mapping, the number of colours used in the edge-colouring of the respective processor graph, and the objective function value of the mapping. Notice that for four processors the edge-colouring

Algorithm	sorting	maxsize	colours	objective	gap
fillup	none	15	48	2400.0225	68.75%
fillup	size+	18	38	1900.0270	60.52%
smallest	none	15	59	2950.0225	74.57%
smallest	size-	15	44	2200.0225	65.91%
circular	size-	20	48	2400.0300	68.75%
circular	degree-	20	45	2250.0300	66.67%
small-degree		15	30	1500.0225	50.00%
small-multicut		18	32	1600.0270	53.12%
connected-0.00		15	35	1750.0225	57.14%
connected-0.30		15	34	1700.0225	55.88%
connected-0.70		16	30	1500.0240	50.00%
connected-1.00		15	21	1050.0225	28.57%
connected-0.00r		15	35	1750.0225	57.14%
connected-0.30r		15	37	1850.0225	59.46%
connected-0.70r		16	33	1650.0240	54.54%
connected-1.00r		15	35	1750.0225	57.14%

Table 10.3: Initial assignment algorithms on Grid-2 and 4 processors.

Algorithm	sorting	maxsize	colours	objective	gap
fillup	none	2764800	14	4847.2002	6.19%
smallest	size-	2764800	14	4847.2002	6.19%
circular				failed	
small-degree				failed	
small-multicut				failed	
connected-0.00		2764800	10	4647.2002	2.15%
connected-0.30		2764800	10	4647.2002	2.15%
connected-0.70		2764800	10	4647.2002	2.15%
connected-1.00		2764800	10	4647.2002	2.15%
connected-0.00r		2764800	10	4647.2002	2.15%
connected-0.30r		2764800	10	4647.2002	2.15%
connected-0.70r		2764800	10	4647.2002	2.15%
connected-1.00r		2764800	14	4847.2002	6.19%

Table 10.4: Initial assignment algorithms on Grid-3 and 4 processors.

Algorithm	sorting	maxsize	colours	objective	gap
fillup	size-	671744	32	2607.6161	38.85%
fillup	size+	770048	32	2755.0721	42.13%
fillup	none	729088	17	1943.6321	17.96%
smallest	size-	679936	28	2419.9041	34.11%
smallest	degree+	729088	35	2843.6321	43.93%
smallest	size-	679936	28	2419.9041	34.11%
circular	size-	688128	32	2632.1921	39.42%
circular	degree+	802816	33	2854.2241	44.14%
circular	size+	720896	29	2531.3441	37.01%
small-degree		811008	15	1966.5121	18.92%
small-multicut		802816	14	1904.2241	16.27%
connected-0.00		696320	14	1744.4800	8.60%
connected-0.30		696320	14	1744.4800	8.60%
connected-0.70		696320	14	1744.4800	8.60%
connected-1.00		778240	14	1867.3601	14.61%
connected-0.00r		696320	14	1744.4800	8.60%
connected-0.30r		819200	14	1928.8001	17.33%
connected-0.70r		753664	22	2230.4961	28.51%
connected-1.00r		745472	19	2068.2081	22.91%

Table 10.5: Initial assignment algorithms on Grid-4 and 4 processors.

problem is easy and column “colours” indeed shows the chromatic index of the processor multigraph $M_P(\mathbf{m})$ where \mathbf{m} is the mapping returned by the respective algorithm. In the last column the tables show the gap between the algorithm’s solution and the optimal solution. If z^* is the optimal solution and z' the solution returned by the algorithm, then we define the *gap* of z' as

$$\text{gap}(z', z^*) := \frac{z' - z^*}{z'}. \quad (10.9)$$

So the gap is the relative error of the solution returned by the algorithm. If no optimal solution is known for a problem we show the gap to the best known lower bound and print it in italics. In cases in which an algorithm failed to produce a feasible solution we mark the respective line in the tables as “failed”.

In order to keep the tables compact, we did not list the results for all possible parameter settings. Instead we listed for Algorithm 8 through Algorithm 10 the results with best balancing, worst objective function and best objective function value. In some cases all parameter settings produced the same or very similar results and we show only one outcome. For Algorithm 11 through Algorithm 13 we show only the best results. Since none of the algorithms took more than one second, we do not list running times for the algorithms in the tables. In each table we marked the best solution(s) by printing them bold-faced.

Not very surprisingly, the tables show that applying heuristics that aim only at balanced load does not lead to good solutions. Nevertheless, these algorithm are able to produce an initial solution with a good balance of the computational load. Using heuristics that take into account communication overhead one way or the other immediately boosts the objective function values of the returned solutions. Among the three algorithms that care about the chromatic index of the processor multigraph, Algorithm 13 performed best for all test-bed instances. However, the value of the swap factor that yielded the best solution varies among the different instances. It is thus reasonable to always execute Algorithm 13 with different values for this factor and to pick the best solution among the different runs.

Local Improvement Algorithms

Starting from the initial solutions just computed we may now apply local improvement strategies as described in Sections 9.2 and 9.3 to enhance them. For the simple local search and the more sophisticated Tabu Search algorithms we have used the starting solution as returned by the different variants of Algorithm 13. Moreover, we have tested all eight different variants of moves. The results for these experiments are displayed in Tables 10.6 through 10.9 (results for the other problem instances can be found in Appendix A.2). The moves allowed for an algorithm are displayed as an array of + and - signs where + specifies that the move is allowed and - that it is not. The first sign refers to moving nodes, the second one to swapping nodes and the third one to moving edges. All other fields are similar to previous tables. As before we attempt to keep the tables compact by only showing the best and worst result for

each combination of moves allowed and improvement strategy applied. As we see in the

initial	enhance	moves	maxsize	colours	objective	gap
connected-0.00	local	+-	896	7	351.3440	14.16%
connected-0.00	local	+-	896	7	351.3440	14.16%
connected-0.00	local	--	896	7	351.3440	14.16%
connected-0.00	local	--	896	7	351.3440	14.16%
connected-0.00	local	++	896	7	351.3440	14.16%
connected-0.00	local	++	896	7	351.3440	14.16%
connected-0.00	local	---	896	7	351.3440	14.16%
connected-0.00	local	---	896	7	351.3440	14.16%
connected-0.00	local	+++	896	7	351.3440	14.16%
connected-0.00	local	+++	896	7	351.3440	14.16%
connected-0.00	local	--+	896	7	351.3440	14.16%
connected-0.00	local	--+	896	7	351.3440	14.16%
connected-0.00	local	+++	896	7	351.3440	14.16%
connected-0.00	local	+++	896	7	351.3440	14.16%
connected-0.00	tabu	+-	896	7	351.3440	14.16%
connected-0.00	tabu	+-	896	7	351.3440	14.16%
connected-0.00	tabu	--	896	7	351.3440	14.16%
connected-0.00	tabu	--	896	7	351.3440	14.16%
connected-0.00	tabu	++	896	7	351.3440	14.16%
connected-0.00	tabu	++	896	7	351.3440	14.16%
connected-0.00	tabu	---	896	7	351.3440	14.16%
connected-0.00	tabu	---	896	7	351.3440	14.16%
connected-0.00	tabu	+++	896	7	351.3440	14.16%
connected-0.00	tabu	+++	896	7	351.3440	14.16%
connected-0.00	tabu	--+	1064	6	301.5960	0.00%
connected-0.00	tabu	--+	1064	6	301.5960	0.00%
connected-0.00	tabu	+++	896	7	351.3440	14.16%
connected-0.00	tabu	+++	896	7	351.3440	14.16%

Table 10.6: Local improvement on Grid-1 and 4 processors.

tables both algorithms are able to determine good solutions. Looking at the initial solutions we also see that local improvement strategies usually provide significant improvements over our initial assignment strategies. Not surprisingly the more sophisticated Tabu Search algorithm outperforms the simple local search strategy. However, the set of allowed moves that yields best performance does vary from problem instance to problem instance. Also the initial assignment strategy that performs best with local improvement strategies varies between the different problem instances.

initial	enhance	moves	maxsize	colours	objective	gap
connected-1.00	local	+-	16	17	850.0240	11.76%
connected-1.00r	local	+-	18	25	1250.0270	40.00%
connected-1.00	local	--	15	20	1000.0225	25.00%
connected-0.00	local	--	24	30	1500.0360	50.00%
connected-1.00	local	++	16	17	850.0240	11.76%
connected-0.30	local	++	16	25	1250.0240	40.00%
connected-1.00	local	---	17	18	900.0255	16.66%
connected-1.00r	local	---	18	26	1300.0270	42.31%
connected-1.00	local	+++	16	17	850.0240	11.76%
connected-1.00r	local	+++	18	26	1300.0270	42.31%
connected-1.00	local	++	17	18	900.0255	16.66%
connected-1.00r	local	++	18	25	1250.0270	40.00%
connected-1.00	local	+++	16	17	850.0240	11.76%
connected-1.00r	local	+++	18	25	1250.0270	40.00%
connected-1.00	tabu	+-	16	17	850.0240	11.76%
connected-1.00r	tabu	+-	18	25	1250.0270	40.00%
connected-1.00	tabu	--	15	19	950.0225	21.05%
connected-0.00	tabu	--	24	30	1500.0360	50.00%
connected-1.00	tabu	++	16	17	850.0240	11.76%
connected-0.70	tabu	++	15	23	1150.0225	34.78%
connected-0.70	tabu	---	16	17	850.0240	11.76%
connected-1.00r	tabu	---	18	18	900.0270	16.66%
connected-0.30	tabu	+++	16	17	850.0240	11.76%
connected-0.70	tabu	+++	18	22	1100.0270	31.82%
connected-0.30	tabu	++	16	17	850.0240	11.76%
connected-0.00	tabu	++	17	22	1100.0255	31.82%
connected-1.00	tabu	+++	16	17	850.0240	11.76%
connected-0.70	tabu	+++	17	22	1100.0255	31.82%

Table 10.7: Local improvement on Grid-2 and 4 processors.

initial	enhance	moves	maxsize	colours	objective	gap
connected-0.00	local	+++	2764800	10	4647.2002	2.15%
connected-0.00	local	+++	2764800	10	4647.2002	2.15%
connected-0.00	local	++-	2764800	10	4647.2002	2.15%
connected-0.00	local	++-	2764800	10	4647.2002	2.15%
connected-0.00	local	+++	2764800	10	4647.2002	2.15%
connected-0.00	local	+++	2764800	10	4647.2002	2.15%
connected-0.00	local	---+	2764800	8	4547.2002	0.00%
connected-0.30r	local	---+	2764800	10	4647.2002	2.15%
connected-0.00	local	+++	2764800	8	4547.2002	0.00%
connected-0.30r	local	+++	2764800	10	4647.2002	2.15%
connected-0.00	local	---+	2764800	8	4547.2002	0.00%
connected-0.30r	local	---+	2764800	10	4647.2002	2.15%
connected-0.00	local	+++	2764800	8	4547.2002	0.00%
connected-0.30r	local	+++	2764800	10	4647.2002	2.15%
connected-0.00	tabu	+++	2764800	8	4547.2002	0.00%
connected-0.30r	tabu	+++	2764800	10	4647.2002	2.15%
connected-0.00	tabu	++-	2764800	10	4647.2002	2.15%
connected-0.00	tabu	++-	2764800	10	4647.2002	2.15%
connected-0.00	tabu	+++	2764800	8	4547.2002	0.00%
connected-0.00	tabu	+++	2764800	8	4547.2002	0.00%
connected-0.00	tabu	---+	2764800	8	4547.2002	0.00%
connected-1.00r	tabu	---+	2764800	10	4647.2002	2.15%
connected-0.00	tabu	+++	2764800	8	4547.2002	0.00%
connected-0.00	tabu	+++	2764800	8	4547.2002	0.00%
connected-0.00	tabu	---+	2764800	8	4547.2002	0.00%
connected-0.30r	tabu	---+	2764800	10	4647.2002	2.15%
connected-0.00	tabu	+++	2764800	8	4547.2002	0.00%
connected-0.00	tabu	+++	2764800	8	4547.2002	0.00%

Table 10.8: Local improvement on Grid-3 and 4 processors.

initial	enhance	moves	maxsize	colours	objective	gap
connected-0.00	local	+-	696320	12	1644.4800	3.04%
connected-0.70r	local	+-	688128	17	1882.1920	15.29%
connected-0.70r	local	--	729088	13	1743.6320	8.55%
connected-1.00	local	--	737280	14	1805.9200	11.71%
connected-0.00	local	++	696320	12	1644.4800	3.04%
connected-0.70r	local	++	704512	15	1806.7680	11.75%
connected-0.00	local	--	704512	12	1656.7680	3.76%
connected-0.70r	local	--	688128	18	1932.1920	17.48%
connected-0.00	local	+++	696320	12	1644.4800	3.04%
connected-0.70r	local	+++	704512	17	1906.7681	16.38%
connected-0.00	local	++	696320	12	1644.4800	3.04%
connected-0.70r	local	++	696320	17	1894.4800	15.84%
connected-0.00	local	+++	696320	12	1644.4800	3.04%
connected-0.70r	local	+++	704512	17	1906.7681	16.38%
connected-0.00	tabu	+-	696320	12	1644.4800	3.04%
connected-0.70r	tabu	+-	688128	17	1882.1920	15.29%
connected-1.00r	tabu	--	679936	12	1619.9040	1.57%
connected-1.00	tabu	--	696320	15	1794.4800	11.15%
connected-0.30r	tabu	++	696320	11	1594.4800	0.00%
connected-1.00	tabu	++	696320	13	1694.4800	5.90%
connected-0.30r	tabu	--	696320	11	1594.4800	0.00%
connected-1.00	tabu	--	720896	12	1681.3440	5.17%
connected-0.00	tabu	+++	696320	12	1644.4800	3.04%
connected-0.70r	tabu	+++	688128	15	1782.1920	10.53%
connected-1.00r	tabu	++	712704	11	1619.0560	1.52%
connected-1.00	tabu	++	696320	14	1744.4800	8.60%
connected-1.00r	tabu	+++	712704	11	1619.0560	1.52%
connected-0.70r	tabu	+++	712704	14	1769.0560	9.87%

Table 10.9: Local improvement on Grid-4 and 4 processors.

10.5.2 More than four Processors

Mapping to $|P| > 4$ processors is much more difficult than mapping to only four processors. If $|P| > 4$ the edge-colouring problem on the processor multigraph M_P suddenly becomes \mathcal{NP} -hard and the number of pairwise disjoint maximum cardinality matchings in $K_{|P|}$ starts to grow rapidly. We must thus consider objective function surrogates in local search heuristics (see Sections 9.2 and 9.3). In the case $|P| = 4$ the most successful heuristic for finding a starting solution was Algorithm 13 with its different parameter settings. Thus, we consider here only one algorithm to determine starting solutions, namely Algorithm 13. We nevertheless test this algorithm with the same parameter variants as before. Moreover, we do not consider grid **Grid-3** anymore, since this grid cannot be mapped reasonably to more than four processors.

Tables 10.10 through 10.12 show the computational results for our local search heuristics on the test instances (further tables can be found in Appendix A.3. As we saw before Algorithm 13 performs best as initial assignment strategy, so we restrict ourselves here to the two variants of this algorithm to determine a starting solution. As before the tables show the start algorithm and enhancement strategy used, the kinds of moves allowed, the objective function surrogate used, running time of algorithms as well as the usual statistics for the best solution found. For compactness the first column of each table only shows the threshold factor and an the letter 'r' to indicate the initial assignment algorithm instead of its full name. Not surprisingly all tables show that using Vizing's approximation $\chi'(G) \leq \delta(G) + \mu(G)$ yields much better results than using Shannon's approximation $\chi'(G) \leq \lceil 3\Delta(G)/2 \rceil$. However, using Shannon's objective function surrogate results in much better running times. This is because evaluation of the objective function value of a given solution can be done in time $\mathcal{O}(|P|)$ for Shannon's bound, but requires time $\mathcal{O}(|P|^2)$ for Vizing's bound (see Chapter 9).

However, it comes at a little surprise that for $|P| = 8$ the simple local search algorithm is often competitive with the more sophisticated Tabu Search. So in practice one may start with (fast) local search and use Tabu Search only if the mapping found is not satisfactory.

initial	enhance	moves	surrogate	time (s)	$ P' $	maxsize	colours	objective	gap
0.00	local	+-	vizing	0.00	6	728	6	301.0920	0.00
0.00	local	+-	shannon	0.00	5	728	7	351.0920	14.24
0.00	local	-+-	vizing	0.00	5	728	7	351.0920	14.24
0.00	local	-+-	shannon	0.00	5	728	7	351.0920	14.24
0.00	local	++-	vizing	0.00	6	728	6	301.0920	0.00
0.00	local	++-	shannon	0.00	5	728	7	351.0920	14.24
0.00	local	--+	vizing	0.00	5	728	7	351.0920	14.24
0.00	local	--+	shannon	0.00	5	728	7	351.0920	14.24
0.00	local	+++	vizing	0.00	6	728	6	301.0920	0.00
0.00	local	+++	shannon	0.00	5	728	7	351.0920	14.24
0.00	local	-++	vizing	0.00	5	728	7	351.0920	14.24
0.00	local	-++	shannon	0.00	5	728	7	351.0920	14.24
0.00	local	+++	vizing	0.00	6	728	6	301.0920	0.00
0.00	local	+++	shannon	0.00	5	728	7	351.0920	14.24
0.00	tabu	+-	vizing	0.04	5	728	7	351.0920	14.24
0.00	tabu	+-	shannon	0.02	5	728	7	351.0920	14.24
0.00	tabu	-+-	vizing	0.02	5	728	7	351.0920	14.24
0.00	tabu	-+-	shannon	0.01	5	728	7	351.0920	14.24
0.00	tabu	++-	vizing	0.07	5	728	7	351.0920	14.24
0.00	tabu	++-	shannon	0.05	5	728	7	351.0920	14.24
0.00	tabu	--+	vizing	0.28	5	728	7	351.0920	14.24
0.00	tabu	--+	shannon	0.18	5	728	7	351.0920	14.24
0.00	tabu	+++	vizing	0.35	5	728	7	351.0920	14.24
0.00	tabu	+++	shannon	0.51	5	728	7	351.0920	14.24
0.00	tabu	-++	vizing	2.19	5	728	7	351.0920	14.24
0.00	tabu	-++	shannon	0.68	5	728	7	351.0920	14.24
0.00	tabu	+++	vizing	3.77	5	728	7	351.0920	14.24
0.00	tabu	+++	shannon	0.69	5	728	7	351.0920	14.24

Table 10.10: Local improvement algorithms on Grid-1 and 8 processors.

initial	enhance	moves	surrogate	time (s)	$ P' $	maxsize	colours	objective	gap
0.00	local	+-	vizing	0.00	8	10	20	1000.0150	<i>40.00</i>
0.00	local	+-	shannon	0.00	8	10	20	1000.0150	<i>40.00</i>
0.00	local	-+-	vizing	0.02	8	10	20	1000.0150	<i>40.00</i>
0.00	local	-+-	shannon	0.00	8	10	20	1000.0150	<i>40.00</i>
0.00	local	++-	vizing	0.02	8	10	20	1000.0150	<i>40.00</i>
0.00	local	++-	shannon	0.01	8	10	20	1000.0150	<i>40.00</i>
0.70	local	---	vizing	0.19	8	10	19	950.0150	36.84
0.00	local	---	shannon	0.04	8	10	20	1000.0150	<i>40.00</i>
0.00	local	++	vizing	0.11	8	10	20	1000.0150	<i>40.00</i>
0.00	local	++	shannon	0.04	8	10	20	1000.0150	<i>40.00</i>
0.00	local	-++	vizing	0.19	8	10	20	1000.0150	<i>40.00</i>
0.00	local	-++	shannon	0.05	8	10	20	1000.0150	<i>40.00</i>
0.00	local	+++	vizing	0.12	8	10	20	1000.0150	<i>40.00</i>
0.00	local	+++	shannon	0.04	8	10	20	1000.0150	<i>40.00</i>
0.00	tabu	+-	vizing	2.63	8	10	20	1000.0150	<i>40.00</i>
0.00	tabu	+-	shannon	1.64	8	10	20	1000.0150	<i>40.00</i>
0.00	tabu	-+-	vizing	13.46	8	10	20	1000.0150	<i>40.00</i>
0.00	tabu	-+-	shannon	11.28	8	10	20	1000.0150	<i>40.00</i>
0.00	tabu	++-	vizing	16.74	8	10	20	1000.0150	<i>40.00</i>
0.00	tabu	++-	shannon	13.96	8	10	20	1000.0150	<i>40.00</i>
0.00	tabu	---	vizing	86.34	8	10	20	1000.0150	<i>40.00</i>
0.00	tabu	---	shannon	43.65	8	10	20	1000.0150	<i>40.00</i>
0.00	tabu	++	vizing	79.68	8	10	20	1000.0150	<i>40.00</i>
0.00	tabu	++	shannon	46.33	8	10	20	1000.0150	<i>40.00</i>
0.00	tabu	-++	vizing	89.48	8	10	20	1000.0150	<i>40.00</i>
0.00	tabu	-++	shannon	52.30	8	10	20	1000.0150	<i>40.00</i>
0.00	tabu	+++	vizing	91.84	8	10	20	1000.0150	<i>40.00</i>
0.00	tabu	+++	shannon	55.43	8	10	20	1000.0150	<i>40.00</i>

Table 10.11: Local improvement algorithms on Grid-2 and 8 processors.

initial	enhance	moves	surrogate	time (s)	$ P' $	maxsize	colours	objective	gap
0.30r	local	+++	vizing	0.00	8	368640	12	1152.9600	27.71
1.00	local	+++	shannon	0.00	7	450560	12	1275.8400	34.68
0.30r	local	++-	vizing	0.01	7	409600	13	1264.4000	34.09
1.00	local	++-	shannon	0.00	7	450560	12	1275.8400	34.68
0.30r	local	++-	vizing	0.03	8	360448	12	1140.6720	26.94
1.00	local	+++	shannon	0.00	7	450560	12	1275.8400	34.68
0.70	local	---	vizing	0.06	8	368640	12	1152.9600	27.71
1.00	local	---	shannon	0.01	7	450560	12	1275.8400	34.68
0.70	local	+++	vizing	0.07	8	368640	12	1152.9600	27.71
1.00	local	+++	shannon	0.01	7	450560	12	1275.8400	34.68
0.70	local	++-	vizing	0.07	8	368640	12	1152.9600	27.71
1.00	local	++-	shannon	0.01	7	450560	12	1275.8400	34.68
0.70	local	+++	vizing	0.08	8	368640	12	1152.9600	27.71
1.00	local	+++	shannon	0.01	7	450560	12	1275.8400	34.68
0.70	tabu	+++	vizing	0.98	8	368640	13	1202.9600	30.72
1.00	tabu	+++	shannon	0.63	7	450560	12	1275.8400	34.68
1.00	tabu	++-	vizing	3.89	7	450560	12	1275.8400	34.68
1.00	tabu	++-	shannon	2.59	7	450560	12	1275.8400	34.68
1.00r	tabu	+++	vizing	4.98	8	344064	14	1216.0960	31.47
1.00	tabu	+++	shannon	3.22	7	450560	12	1275.8400	34.68
0.00	tabu	---	vizing	18.19	8	344064	14	1216.0960	31.47
1.00	tabu	---	shannon	11.69	7	450560	12	1275.8400	34.68
0.70	tabu	+++	vizing	24.20	8	368640	12	1152.9600	27.71
1.00	tabu	+++	shannon	12.89	7	450560	12	1275.8400	34.68
0.30r	tabu	++-	vizing	27.05	8	344064	13	1166.0960	28.53
1.00	tabu	++-	shannon	14.79	7	450560	12	1275.8400	34.68
0.70r	tabu	+++	vizing	28.22	8	344064	14	1216.0960	31.47
1.00	tabu	+++	shannon	15.59	7	450560	12	1275.8400	34.68

Table 10.12: Local improvement algorithms on Grid-4 and 8 processors.

10.6 Branch-and-Cut Algorithms

Let us now turn to the performance analysis for the Branch-and-Cut algorithms that are implied by the branching strategies described in Section 10.1 as well as the models, valid inequalities and separation algorithms described in Chapters 5, 6 and 8. We first describe why and how we limited the solution time granted to optimisation problems and then present positive and negative results about the Branch-and-Cut algorithms

10.6.1 Computation Time

An important aspect in the computation of optimal block-mappings is the time required for finding the optimal solution. This is because each CPU second spend for optimisation could as well be spend for simulation². Consequently, optimisation time must not exceed the amount of time we actually save by using a better partitioning. Otherwise we have actually

²In practice this is not exactly true since simulation must usually performed on a high performance multi-processor while optimisation can be carried out on a single CPU workstation.

wasted time by our approach instead of saving it. The time required for one iteration in the simulation when mapping according to the currently best solution is given by the objective function value of the incumbent solution. Multiplying this time with the number of iterations required immediately yields the time required for the whole simulation. From this time and the improvement we have between subsequent incumbent solutions we may determine the number of seconds that we grant the optimisation process. Unfortunately, the number of iterations is not known beforehand and we cannot use this strategy.

Another approach would be to timeout optimisation when the lower bound provided by the Branch-and-Cut algorithm is close to the current incumbent solution and our mapping is thus near-optimal. However, the gap between lower bound and incumbent solution may not change for several hours.

Instead of the two potential strategies just described, we grant each optimisation process only a fixed amount of time. We allow a Branch-and-Cut algorithm that solves an instance on $|P|$ processor to run $|P| \cdot 450$ CPU seconds (this was chosen to have eight processor instances timeout after one hour). Notice that this limit is independent of the grid size. The rationale behind this is that the difficulty of a simulation is already reflected in the number of processors that are allocated for it. We therefore refrain from complicated formulae involving the grid size and grant time proportional to the number of processors.

10.6.2 Results

In order to actually solve instances of (OGPC) to optimality we employed models (SLOT) and (SLOT*). Models (NAIVE) and (REP) were already found to be inferior in Chapter 6 and why we do not use (SLOT4) will become clear in a moment. In Chapters 5, 6 and 8 we described many valid inequalities and separation algorithms for these two models. Theoretically, any combination of valid inequalities and separation algorithms yields another variant of Algorithm 4 (Branch-and-Cut). Of course we did not test all combinations. But we tested all combinations that seemed promising and present here the most successful approaches³.

To solve (OGPC) as instance of (SLOT) or (SLOT*) we used the following additional constraints for a grid graph $G = (V, E)$.

- We sort the blocks in G by decreasing size. For ease of exposition assume that blocks are already given in that order. We then fix block u_0 to processor p_0 and apply inequalities (6.49) and (6.50) to the virtual variables $\gamma_{u,p}$. In other words, we require that block u_1 is mapped to p_0 or p_1 and so on. Furthermore, a block u_i may be mapped only to p_i only if a previous block was mapped to processor p_{i-1} . This implies that we

³We refrain from providing computational results for other combinations of valid inequalities and separation strategies in the appendix as they all produce results inferior to the ones presented here.

are no longer able to permute processors as required by model (SLOT4) and rules out this model. Furthermore, fixing any blocks to processors immediately violates the LP relaxation presented in Section 6.4. The rationale behind this sorting strategy is the assumption that the biggest blocks are likely to be mapped to different processors in optimal solutions. This is especially true if their sum exceeds processor capacity or the lower bound κ^{*l} . As stated in Section 6.7 inequalities (6.49) and (6.50) have biggest impact when applied to block sets with exactly this property.

- We use inequalities (6.51) and (6.52) for the first processor and inequality (5.19) for all others. As these inequalities are originally defined for model (NAIVE) we must translate them using the virtual variables $\gamma_{u,p}$.
- In order to add further “noise” to the problem and violate the LP relaxation discussed in Section 6.4 we compute for each block $u \in V$ a tree cover and add the resulting tree cover inequality to (SLOT) or (SLOT*). A tree cover for u is determined by starting with the trivial tree $\{u\}$ and then recursively adding the node of maximum size among all nodes adjacent to the tree.
- Using the upper bound on the number of elements per processor (see Section 7.3) we add inequality (5.13) – again translating it using virtual variables $\gamma_{u,p}$.
- For model (SLOT*) we require $z_c \geq z_{c+1}$, thereby sorting colour classes as described in Section 6.7.

We will refer to instances of (SLOT) or (SLOT*) to which the constraints just described were added as *augmented* models.

Model (SLOT)

In order to show how much the valid inequalities added to the plain model (SLOT) as described on page 120 we show in Table 10.13 the optimal values of LP relaxations of the respective instances of (SLOT). Recall that the optimal objective function values of these relaxations are equal to $t_a \kappa_V / |P|$. To further illustrate performance of Branch-and-Cut we

	Grid-1	Grid-2	Grid-4	Grid-5	Grid-6	Grid-7	Grid-8	Grid-9
$ P = 4$	1.008	0.0225	1007.616	1032.576	729.216	801.792	1161.216	1260.864
$ P = 8$	0.504	0.0113	503.808	516.288	364.608	400.896	580.608	630.432

Table 10.13: Lower bound by LP relaxation in non-augmented models.

also separate one class of cuts, namely biconnected cover inequalities. We choose this class of inequalities as reference as we found that it performs best among all classes of inequalities to be separated. While separating biconnected cover inequalities we must observe that the

deeper we get in the Branch-and-Cut tree, the more unlikely it is to find violated inequalities of this class. This is due to the fact that the deeper we get, the more feasible our solution becomes. We therefore restrict separation of biconnected cover inequalities either to the root node or nodes of maximum depth $|P|$. This avoids wasting time in separation algorithms when the chance for finding valid inequalities is small. Moreover, to reduce the number of inequalities added and keep the LP problems at Branch-and-Cut nodes small, we only insert violated inequalities for biconnected covers C that are violated by at least one, i. e., for which

$$\sum_{p \neq k} \sum_{e \in C} x_{e,p,k}^* < 1$$

for the current fractional solution x^* .

Tables 10.14 through 10.15 show computational results of Branch-and-Cut algorithms on model (SLOT) that is augmented by valid inequalities as described above. Moreover, in some cases separation of biconnected cover inequalities was employed. In each table column “bc” shows up to which depth biconnected-cover inequalities were separated (0 means no separation at all). The column labelled “node 0” displays the Branch-and-Cut lower bound after the first node (the root relaxation) is handled. Columns “time”, “nodes”, “lb”, “ub” and “gap” show the time the algorithm took as well as the number of nodes processed, upper and lower bound at the end of the algorithm and the relative gap between them. Observe that the upper bound is the objective function value of the best integer feasible solution found in the Branch-and-Cut tree. Finally, column “cuts” lists the number of biconnected-cover cuts found and applied. We see at first glance, that the augmented model provides

	bc	node 0	final				cuts
			time	nodes	lb	ub	
Grid-1	0	301.2185	0.182	7	301.5960	0.00 %	0
	1	301.2936	0.170	12	301.5960	0.00 %	3
	4	301.2936	0.170	12	301.5960	0.00 %	3
Grid-2	0	474.5511	159.000	4223	750.0450	0.00 %	0
	1	510.0068	124.710	3544	750.0450	0.00 %	5
	4	510.0068	269.340	7098	750.0450	0.00 %	71

Table 10.14: Branch-and-Cut performance on four processors and grids **Grid-1**, **Grid-2**.

a much better LP relaxation (column “node 0”) than the pristine model does. For **Grid-1** the LP relaxation of the augmented problem is even almost equal to the optimal solution. This shows that from an algorithmic point of view the inequalities we added to the problem have a positive influence on the structure. Moreover, the tables show that with our methods we are able to solve medium sized instances up to **Grid-4** to optimality within a small amount of time. The gap of about 15 % for the final results of **Grid-6** is also acceptable. On the other hand, we see that larger problem instances such as **Grid-5**, **Grid-6** and so on cannot be solved to optimality within the amount of time we granted (1800 seconds).

	bc	node 0	final					cuts
			time	nodes	lb	ub	gap	
Grid-4	0	1259.3402	708	80026	1594.4800		0.00 %	0
	1	1231.8681	1080	102258	1594.4800		0.00 %	12
	4	1231.8681	956	91594	1594.4800		0.00 %	44
Grid-5	0	1334.7878	1800	29135	1555.4913	2176.7361	28.54 %	0
	1	1348.0373	1800	17486	1530.4658	2198.2401	30.38 %	86
	4	1348.0373	1800	12483	1531.2179	2163.6001	29.23 %	264

Table 10.15: Branch-and-Cut performance on four processors and grids
Grid-4, Grid-5.

We also see that separation and insertion of biconnected cover inequalities does further improve the relaxation at node 0. However, apart from two exceptions inserting biconnected cover inequalities leads to a decrease in the number of nodes processed, which in turn lets the algorithms terminate with inferior results. The two exceptions to this conclusion are Grid-1, where insertion of biconnected cover inequalities resulted in more nodes handled but fewer CPU seconds consumed and Grid-2 with separation only in the root node. For Grid-5, Grid-8 and Grid-9 insertion of biconnected covers does not improve performance of Branch-and-Cut but allows CPLEX to find better incumbent solutions. Observe that

	bc	node 0	final					cuts
			time	nodes	lb	ub	gap	
Grid-6	0	1005.9701	1800	18565	1530.1997	1796.4960	14.82 %	0
	1	1070.9967	1800	12350	1501.9532	1796.4960	16.40 %	60
	4	1070.9967	1800	11719	1499.3738	1796.4960	16.54 %	135
Grid-7	0	1104.6715	1800	13752	1348.3994	1822.5280	26.01 %	0
	1	1138.1153	1800	8201	1331.4697	1829.4400	27.22 %	65
	4	1138.1153	1800	7496	1340.9804	1829.4400	26.70 %	206
Grid-8	0	1423.7160	1800	8760	1782.7126	2578.6401	30.87 %	0
	1	1451.5590	1800	6435	1763.0353	2570.4320	31.41 %	2
	4	1451.5590	1800	6329	1766.1024	2622.4960	32.66 %	22
Grid-9	0	1510.0388	1800	2652	1879.1651	2443.6960	23.10 %	0
	1	1607.0981	1800	1281	1799.2397	2430.5600	25.97 %	77
	4	1607.0981	1800	1059	1808.8939	2440.9280	25.89 %	243

Table 10.16: Branch-and-Cut performance on four processors and grids
Grid-6, Grid-7, Grid-8, Grid-9.

separation of biconnected-cover inequalities itself does not consume much time. The higher computational effort with activated separation is almost only due to the fact that LPs become much more complicated if they contain biconnected-cover inequalities.

From the results just presented we conclude that in general biconnected cover inequalities cannot improve performance of Branch-and-Cut. Further computational experiments showed

that the same observations are true for other valid inequalities to be separated that we described in Chapter 8.

The tables also show that we must accept gaps between twenty and thirty percent if we allow only 1800 CPU seconds for optimisation. However, we found that if we grant more time the upper bound does usually not drop significantly, i. e., the upper bounds displayed in Tables 10.14 through 10.16 are near-optimal. Unfortunately, it usually takes many hours to prove this optimality, i. e., to raise the lower bound until it reaches the upper bound. This clearly illustrates one big flaw of our algorithms: while it is relatively easy to come up with good solutions, it is very difficult to prove their quality because Branch-and-Cut performs too bad on large problem instances.

If we map to more than four processors the gaps after termination of Branch-and-Cut become even bigger. To illustrate this we provide Table 10.17 that shows results for the case $|P| = 8$ and is organised like the tables above. As we concluded that insertion of biconnected cover inequalities does not improve performance we do not list results for variants of Branch-and-Cut that include these inequalities. On the positive side we observe that the LP relaxations

	bc	node 0	final				
			time	nodes	lb	ub	gap
Grid-1	0	301.0920	0.000	0	301.0920		0.00 %
Grid-2	0	600.0150	3600	12183	600.0150	750.0165	20.00 %
Grid-4	0	786.9731	3600	22195	833.4186	1052.9600	20.85 %
Grid-5	0	757.4228	3600	4184	804.7388	1537.6000	47.66 %
Grid-6	0	752.4141	3600	1179	824.7590	1345.8240	38.72 %
Grid-7	0	807.9581	3600	1291	812.0605	1392.3680	41.68 %
Grid-8	0	906.7760	3600	621	953.5848	1881.9840	49.33 %
Grid-9	0	960.9762	3600	80	978.3266	1882.4320	48.03 %

Table 10.17: Branch-and-Cut performance on eight processors

provided by the augmented model are much better than those of instance of the pristine models. We see that we must accept integrality gaps of up to 50 percent unless we allow more than one hour of CPU time for Branch-and-Cut. Here the situation is similar to the case $|P| = 4$: allowing (much) more time than one CPU hour does not reduce the upper bound significantly but raises the lower bound close to the upper bound eventually. This again shows that our methods are not good for proving optimality of solutions for large problem instances.

Model (SLOT*)

Let us now analyse performance of Branch-and-Cut algorithms on instances of (SLOT*). As insertion of biconnected cover inequalities was found to be ineffective in the analysis of

Branch-and-Cut algorithms before, we refrain here from using this kind of inequalities.

Instead we consider adding blossom inequalities (6.40) to the augmented model. This class of inequalities is of exponential size and should be separated in general. However, we refrain from separating this inequality and restrict ourselves to considering these inequalities on triangles (a separation algorithm can nevertheless be found in the Appendix). As triangles in the processor multigraph we consider only triplets of processors p_0 , p_1 and p_2 such that $p_1 = p_0 + 1 \pmod{|P|}$ and $p_2 = p_1 + 1 \pmod{|P|}$. This keeps the number of inequalities actually added small.

Table 10.18 shows results for mapping our grids to four processors. Column “blo” indicates whether we used blossom inequalities or not. The rest of the table layout is like before and shows consumed CPU seconds, number of nodes processed, lower and upper bounds as well as the gap at the end of the algorithms. We clearly see that adding blossom inequalities

	blo	node 0	time	nodes	final		gap
					lb	ub	
Grid-1		301.1684	0.315	51	301.5960		0.00 %
	x	301.1565	0.174	7	301.5960		0.00 %
Grid-2		501.3255	256	6259	750.0450		0.00 %
	x	450.6700	173	5138	750.0450		0.00 %
Grid-4		1258.6882	1403	99167	1594.4800		0.00 %
	x	1220.0410	1262	102580	1594.4800		0.00 %
Grid-5		1341.1645	1800	19430	1534.5810	2198.2401	30.19 %
	x	1352.2878	1800	24074	1541.6935	2193.6321	29.72 %
Grid-6		1004.9459	1800	10407	1486.3892	1796.4960	17.21 %
	x	1043.7516	1800	13051	1502.0526	1796.4960	16.39 %
Grid-7		1118.4300	1800	8763	1309.6142	1829.4400	28.41 %
	x	1107.5547	1800	10761	1324.2731	1829.4400	27.61 %
Grid-8		1423.7160	1800	5691	1759.0820	2526.5761	30.38 %
	x	1423.7160	1800	6314	1765.9081	2577.9521	31.50 %
Grid-9		1510.0388	1800	1655	1838.5577	2377.7921	22.68 %
	x	1510.0388	1800	1924	1834.7210	2443.6961	24.92 %

Table 10.18: Results for (SLOT*) on four processors.

allows us to explore more nodes in the same amount of time. The advantage of that is however not clear: for grids **Grid-1** through **Grid-7** the performance of the algorithm with blossom inequalities enabled is better than without them. However, for **Grid-8** and **Grid-9** the opposite is true. A possible explanation for this observation is as follows: Adding blossom inequalities obviously renders the problem instances easier to solve for the LP-solver. The solver can thus explore more nodes in the same amount of time. On the other hand, adding blossom inequalities also changes the structure of the problem. This may have bad effects on solver-internal heuristics that no longer work as efficient as before. For medium size problem

instances like grids **Grid-5** through **Grid-7** the positive effects of being able to enumerate up to 25 percent more nodes obviously outweighs the negative effects on the internal heuristics. On larger grid instances like **Grid-8** or **Grid-9** the number of nodes enumerated increases only by 10 and 16 percent respectively. This seems not enough to remedy the bad side-effects on internal heuristics, hence overall solver performance is worse in these cases.

Comparing Table 10.18 with the results for model (SLOT) we see that the LP relaxations of (SLOT*) are usually similar to those of (SLOT) but are inferior to the relaxations obtained with biconnected cover inequalities activated. Likewise the performance of algorithms on (SLOT*) is similar to those on (SLOT) but usually slightly inferior.

Results for Branch-and-Cut algorithms on model (SLOT*) and eight processors are depicted in Table 10.19. Unlike before adding blossom inequalities here has a clear effect: it deteriorates performance, with the single exception of **Grid-8** where the final gap is smaller under presence of blossom inequalities. We also see that the final gap delivered by Branch-and-

	blo	node 0	final				
			time	nodes	lb	ub	gap
Grid-1		301.0920	0.000	0	301.0920		0.00 %
	x	301.0920	0.000	0	301.0920		0.00 %
Grid-2		600.0150	3600	4356	600.0150	750.0165	21.20 %
	x	600.0150	3600	6452	600.0150	800.0150	25.00 %
Grid-4		775.3394	3600	7834	817.3879	1077.5360	24.14 %
	x	770.4032	3600	10871	816.9501	1078.3840	24. 24 %
Grid-5		781.7264	3600	1568	799.0586	1517.4720	47.34 %
	x	781.3562	3600	1941	803.4989	1580.6080	49.17 %
Grid-6		755.4460	3600	727	819.2479	1496.0480	45.24 %
	x	761.1436	3600	720	824.7590	1426.9280	42.20 %
Grid-7		816.5503	3600	390	834.5457	1392.3680	40.06 %
	x	820.8880	3600	736	825.4299	1392.3680	40.73 %
Grid-8		906.9532	3600	267	941.0225	2063.5520	54.40 %
	x	906.7760	3600	452	956.1894	1865.0880	48.73 %
Grid-9		960.9762	3600	47	975.5491	2018.8480	51.68 %
	x	961.1519	3600	92	975.3433	2018.8480	51.69 %

Table 10.19: Results for (SLOT*) on eight processors.

Cut algorithms on instances of (SLOT*) is not significantly better than the one returned by algorithms on (SLOT) but sometimes significantly worse (up to 5 % for **Grid-8** without blossoms). Apart from **Grid-1** and **Grid-8** also the upper bounds delivered are worse than for model (SLOT). This implies that the actual mappings found within the provided time window using (SLOT*) are usually worse than the mappings found by algorithms based on (SLOT). We therefore conclude that model (SLOT) is superior to model (SLOT*), at least for small numbers of processors.

Hypercubical Processor Multigraphs

Unfortunately, we were not able to solve all problem instances to optimality, not even in the simple case $|P| = 4$. We thus considered the heuristic simplifications described in Section 9.7 and required the processor multigraph to be a hypercube in the case $|P| = 4$. To get good performance of Branch-and-Cut algorithms we used as upper cutoff value the solution found by local search heuristics – although the associated mapping does not necessarily yield a hypercubical processor multigraph.

Computational results for this setting are displayed in Table 10.20 (this time we granted an arbitrary amount of time to the solution algorithms). We see that some instances (those marked “xxx”) are infeasible. This is because no mapping exists that yields a hypercubical processor multigraph and an objective function that is smaller than local search objective functions. For the other instances we see that we can find optimal solutions significantly faster as in the case of general processor multigraphs. Thus, in practice it might be worth

grid	nodes	time	objective
Grid-1	0	0	351.3440
Grid-2	114	6	750.0450
Grid-4	4900	38	1644.4800
Grid-5	190009	5357	2069.0561
Grid-6	xxx	xxx	xxx
Grid-7	xxx	xxx	xxx
Grid-8	44078	4010	2479.6481
Grid-9	xxx	xxx	xxx

Table 10.20: Mapping grids to a two-dimensional hypercube.

to restrict ourselves to hypercubical processor multigraphs so as to obtain a block-mapping of proven quality. On the other hand the infeasible problem instances show that some of the heuristic solutions cannot be improved by mappings that imply a simple processor multigraph. We may thus assume that the heuristic solutions are already of reasonable good quality.

10.6.3 Negative Results

Unfortunately, many things did not work out as expected. The previous section already showed that large problem instances cannot be solved to optimality by means of our Branch-and-Cut algorithms. Let us now describe explicitly what did not work as well as expected.

The first thing to mention are the branching strategies described in Sections 10.1.2 and 10.1.3. Both strategies were not able to outperform the default SOS branching strategy described

in Section 10.1.1. Even worse both alternate strategies led to an increase of running time and number of nodes explored. Some reasons for this behaviour might be as follows.

Branching on virtual variables $\gamma_{u,p}$ as described in Section 10.1.2 requires a significant computational overhead at each Branch-and-Cut node to determine the fractional values of $\gamma_{u,p}$. Moreover, Section 10.1.3 already indicated that the default branching strategy is easily modified to handle only $\mathcal{O}(|V|)$ SOS sets instead of $\mathcal{O}(|E|)$. Our guess is that CPLEX is implicitly guided by the respective fractional solutions towards a strategy that is similar to the one described in Section 10.1.3. Notice that although this is quite likely it would be pure accident as CPLEX does not have any idea about our current problem structure.

Branching on matchings only did not improve performance as well. As we just mentioned this might be because CPLEX already does something similar implicitly. Another explanation for this effect might be that using more explicit SOS constraints provides the MIP solver with more information. The solver is thus able to draw stronger conclusions from fractional variables and performs better. One observation that backs up this speculation is the fact that relaxing the integrality constraints on variables $x_{e,p,k}$ not in matchings as described in Section 10.1.3 further deteriorates performance. Here the reason is rather obvious: knowing that $x_{e,p,k}$ must be binary the MIP solver can easily “snap” fractional to integral values. If the solver has for example proved that $x_{e,p,k} > 0$ than it can conclude $x_{e,p,k} = 1$. If instead $x_{e,p,k}$ is specified to be continuous no such inferences are possible.

Another thing that did not improve performance were the rounding heuristics described in Section 9.6. However, this was more or less expected because the solutions computed by our local search heuristics are usually pretty good and close to optimal. It is thus hard to improve these solutions even by exploiting information provided by fractional solutions.

In Chapters 5 and 6 we presented much more valid inequalities than we actually used. The reason for this is that application of them does not improve solver performance. In most cases our MIP solver is even slowed down considerably by adding certain inequalities to the problem. This happens even if we handle large classes of inequalities by separation and shows that addition of these inequalities makes numerically solving the problems more difficult. One special case of these inequalities are inequalities for lower bounds. Each lower bound presented in Chapter 7 can be directly translated into a valid inequality for model (SLOT). Adding these inequalities immediately *boosts* the objective function value of the initial LP relaxation, thereby yielding a better objective function lower bound. However, problem instances with these inequalities added require much longer solution times than models without them. So in the long run adding bound inequalities does not pay.

10.7 Algorithms from the Literature

In the previous section we collected positive and negative results for our new optimisation algorithms. Apart from efficiently solving certain integer programming problems our aim was to compute block-mappings that outperform block-mappings computed by algorithms from the literature, provided the hardware model is similar to ours. Let us now show that this goal was reached.

As stated earlier the traditional approach for finding an optimal block-mapping is to require a perfectly balanced load while minimising the number of edges cut. Many heuristics [82, 83, 144, 105, 99, 11, 10, 12, 13, 53, 100] are suggested in the literature to solve this problem quickly and/or accurately. We will now compare the quality of mappings computed by these strategies to the quality of mappings our now approach produces. To this end we model the traditional approach as special instance of (SLOTMAP): We do not attempt to minimise the maximum load on processors. Instead we require that no processor bears more than κ^* control volumes and minimise the number of edges that have endpoints on different processors. This yields

(SOTA)

$$\text{minimise } \sum_{p \neq k} \sum_{e \in E} x_{e,p,k} \quad (10.10a)$$

$$\sum_{(p,k) \in P \times P} x_{e,p,k} = 1 \quad e \in E \quad (10.10b)$$

$$\sum_{uv \in E} \sum_{k \in P} (\kappa'_u x_{uv,p,k} + \kappa'_v x_{uv,k,p}) \leq \kappa^* \quad p \in P \quad (10.10c)$$

$$\sum_{k \in P} x_{uv,p,k} - \sum_{k \in P} x_{f,p,k} = 0 \quad uv \in E, p \in P, f \in \delta^+(u) \quad (10.10d)$$

$$\sum_{k \in P} x_{uv,p,k} - \sum_{k \in P} x_{f,k,p} = 0 \quad uv \in E, p \in P, f \in \delta^-(u) \quad (10.10e)$$

$$\sum_{k \in P} x_{uv,k,p} - \sum_{k \in P} x_{f,p,k} = 0 \quad uv \in E, p \in P, f \in \delta^+(v) \quad (10.10f)$$

$$\sum_{k \in P} x_{uv,k,p} - \sum_{k \in P} x_{f,k,p} = 0 \quad uv \in E, p \in P, f \in \delta^-(v) \quad (10.10g)$$

$$x \text{ binary} \quad (10.10h)$$

This model exactly describes the graph-partitioning model that is traditionally used to find optimal block-mappings: In an optimal solution to (SOTA) no processor bears more than κ^*

control volumes (see (10.10c)) and the number of edges with endpoints on different processor is minimal (see (10.10a)). In other words, in an optimal solution the computational load is optimally balanced and the edge-cut is minimal.

Since state-of-the-art heuristics aim at finding optimal solutions for the problem described by (SOTA), it is clear that a solution returned by such a heuristic cannot have a smaller edge-cut than the optimal solution to (SOTA). This implies that the optimal solution to (SOTA) provides a lower bound on the objective function value of the solution returned by any state-of-the-art heuristic. We therefore do not compare our new approach to each of the heuristics from the literature. Instead we carry out the following steps for the problem instances to be compared:

1. Determine $\kappa^*(G, |P|)$. We do this by solving an instance of (NAVMAP) to optimality.
2. Find a mapping \mathbf{m}' in which no processor bears more than $\kappa^*(G, |P|)$ control volumes and that contains a minimal number of inter-processor edges. This mapping is at least as good as any mapping that is returned by a heuristic from the literature.
3. Determine the chromatic index of the processor multigraph $M_P(\mathbf{m}')$.
4. Compute the time required for one iteration of the simulation as

$$z' := t_a \kappa^*(V, |P|) + t_c \chi'(M_P(\mathbf{m}')).$$

In other words, we first compute an optimal mapping for the traditional approach and then – fixing this mapping – we find an optimal communication schedule for the implied processor multigraph. The number $\chi'(M_P(\mathbf{m}'))$ of rounds in this schedule together with the maximum processor load κ^* yields the time z' that is required for one iteration of the corresponding simulation on our hardware. It is clear that z' is the best value we can expect from applying state-of-the-art heuristics for finding an optimal block-mapping.

We then compare z' to z_{opt} , the objective function value of an optimal solution to (SLOT). In order to make edge-colouring of $M_P(\mathbf{m}')$ easy, we restrict ourselves to the case $|P| = 4$.

Tables 10.21 through 10.23 compare the different block-mappings obtained by this approach. For grids **Grid-1**, **Grid-2**, **Grid-3**, **Grid-4**, **Grid-6** and **Grid-7**⁴ the tables show the following characteristics of the different solutions: Table 10.21 gives the number of communication rounds (equivalently the chromatic index of the processor multigraph) required in the computed block-mapping, Table 10.22 gives the number of control volumes on the most heaviest loaded processor in each mapping. Finally, Table 10.23 lists the CPU milliseconds that each mapping would require for one iteration of the respective simulation. In each table column

⁴The other grids were omitted from the presentation because computing either the values for column “old” or the values for column “exact” took too long.

Grid	old	heuristic	hypercube	exact
Grid-1	8	6	xxx	6
Grid-2	20	17	15	15
Grid-3	8	8	8	8
Grid-4	16	11	12	11
Grid-6	33	21	xxx	18
Grid-7	24	18	xxx	18

Table 10.21: State-of-the-art algorithms and new approach: Colours (communication rounds) in mappings.

“old” refers to the mappings returned by the approach described above, that attempts to emulate state-of-the-art algorithms. Column “heuristic” shows data for mappings obtained by the heuristic algorithms described in Chapter 9. Columns “hypercube” and “exact” show characteristics for exact solutions to the respective instance of (SLOT). In the first case the processor multigraph is restricted to a hypercube (which is a (multi-)cycle for $|P| = 4$), while in the second case arbitrary processor multigraphs are allowed. The last table also shows for each solution the percentage of time that it saves over the mapping described in column “old”. Cells marked “xxx” indicate that the respective optimisation problem was infeasible. We see that on the compared problem instances, the new approach is able to save up to 26 percent of the time required for a single iteration of the simulation. We also see that not only the exact solutions to (SLOT) outperform the traditional mappings, but also the heuristics described in Chapter 9 are able to save a big deal of time per iteration. It is also obvious, that restricting the processor multigraph to a hypercube yields mappings that do not require a significant amount of additional amount of time. The advantage of this approach is that the restriction on M_P renders the integer program much simpler and instances of it can thus be solved faster. An interesting case is Grid-3. For this grid, the traditional and our new approach yield mappings of the same quality. This shows that in very rare cases even the traditional approach is able to produce good mapping for our hardware (by accident).

By the results presented in tables we conclude that for the hardware architecture described in this thesis the new partitioning approach with edge-colouring outperforms the traditional

Grid	old	heuristic	hypercube	exact
Grid-1	728	1064	xxx	1064
Grid-2	15	16	30	30
Grid-3	2764800	2764800	2764800	2764800
Grid-4	671744	696320	696320	696320
Grid-6	486144	499968	xxx	564480
Grid-7	534528	580608	xxx	550656

Table 10.22: State-of-the-art algorithms and new approach: Maximum processor load in mappings.

Grid	old	heuristic		hypercube		exact	
Grid-1	401.0920	301.5960	24.81 %	xxx	xxx	301.5960	24.81 %
Grid-2	1000.0225	850.0240	15.00 %	750.0450	25.00 %	750.0450	25.00 %
Grid-3	4547.2002	4547.2002	0.00 %	4547.2002	0.00 %	4547.2002	0.00 %
Grid-4	1807.6160	1594.4800	11.80 %	1644.4800	9.02 %	1594.4800	11.80 %
Grid-6	2379.2161	1799.9520	24.35 %	xxx	xxx	1746.7200	26.58 %
Grid-7	2001.7920	1770.9120	11.53 %	xxx	xxx	1725.9840	13.78 %

Table 10.23: State-of-the-art-algorithms and new approach: Milliseconds per iteration using different mappings.

graph-partitioning approach – even when the first is solved only heuristically.

Chapter 11

Conclusions and Outlook

Predicting is difficult, especially when it involves the future.
— *Unknown*

In this thesis we presented a new formulation of the graph-partitioning problem. Our work was motivated by the desire to accurately account for communication overhead and thus find a block-mapping that yields minimal simulation time. Since state-of-the-art grid-partitioning or load-balancing algorithms model communication overhead only inadequately the need of a new model was immediate. The exact representation of the restrictions imposed by our hardware led to a model in which an optimal communication schedule is equivalent to a minimal edge-colouring of the multigraph induced by the (optimal) block-mapping. We presented three different models for each of the two individual problems into which (OGPC) naturally decomposes. Furthermore we investigated four (out of nine possible) combinations of individual models into joined models that yield integer programming formulations for (OGPC). All models were analysed with the aim to construct Branch-and-Cut algorithms that are able to solve even large scale problem instances to optimality.

Our computational results proved that using the heuristics described in Chapter 9 we are able to determine good block-mappings in a reasonable amount of time. Not surprisingly, our algorithms that are dedicated to our specific hardware model outperform general purpose state-of-the-art algorithms – at least for the case $|P| = 4$. Other algorithms that are designed for our hardware can – to the best of our knowledge – not be found in the literature, so we claim that for the hardware assumed in this thesis best results are obtained by using our algorithms. Unfortunately, our methods have been less successful for more than four processors. Already for this small number of processors the integer programming methods failed and exact solutions could only be obtained if we required the processor multigraph to be bipartite or even a hypercube. For general graphs we had to resort to heuristic methods. As we have no exact solutions for the respective problem instances, we cannot not prove the

actual quality of our heuristic solutions. However, extrapolating the results from the case $|P| = 4$ we hope that they return mappings that perform reasonably well in practice.

The conclusions drawn so far immediately imply several directions for future research and further improvement of the solution strategies. Given that it is relatively easy to solve instances of (SLOT) or (SLOT*) when the processor multigraph is restricted to hypercubes or other bipartite multigraphs, one might consider column-generation approaches [33]. In other words, we start with a subset of variables $x_{e,p,k}$ such that the resulting processor multigraph is guaranteed to be a hypercube. However, after solving the integer program to optimality, we do not stop. Instead we start adding variables $x_{e,p,k}$ that were fixed to zero when we required a hypercube as processor multigraph. After adding a (small) set of variables we reoptimise our problem. This process can be repeated until all variables (and thus all kinds of processor multigraphs) are allowed, a solution of satisfactory quality is obtained or a time limit is reached. The hope is that reoptimisation of the problem after adding a small set of variables is much faster than starting with the augmented set of variables in the first place.

Another aspect of potential improvement is further analysis of the polyhedra defined by the various integer programming models described in this thesis. We showed several times that the LP relaxation to the programs is quite weak in the sense that it does not provide good lower bounds. We even proved that for the joined models the gap between the optimal integral solution and the LP relaxation is as large as possible and the latter provides only a trivial lower bound. Although we presented several valid inequalities that are violated by the respective LP relaxations, we were not able to close the gap between optimal fractional and integral solution for large problem instances. In order to narrow this gap by means of Branch-and-Cut algorithms further valid (or even facet-defining) inequalities must be exhibited, so as to strengthen the problem formulation.

Yet another direction for future research is preprocessing of the input grid. Recall from the introduction that we partitioned control volumes into blocks so as to reduce the size of the problem instances. This partitioning is usually based on geometry information, i. e., the geometry on which the simulation is defined suggests (or demands) a certain partitioning of the control volumes. Since the blocks into which the control volumes are partitioned are induced by the geometry, it may easily happen, that the blocks are very large or greatly vary in size. This has two severe drawbacks:

1. If the blocks have very different sizes, i. e., if the difference between the smallest and biggest block is very large, then it is more difficult to balance the computational load such that each processor serves an equal number of control volumes.
2. If there is one large block of size κ_{\max} and most of the other blocks are considerably smaller, then this large blocks limits the number of processors that can be employed for the simulation in a sensible way. In fact, if we use p processors, then each processor bears at most $\lceil \kappa_V/p \rceil$ control volumes in a perfect balancing. However, since there

is at least one processor that handles κ_{\max} control volumes the maximum number of control volumes on a processor will never drop below that figure. Thus employing more than $\lceil \kappa_V / \kappa_{\max} \rceil$ processors does not reduce the simulation speed further. Even worse, the additional communication overhead may increase the time required for the whole simulation.

One way to remedy the shortcomings just mentioned would be to operate on the control volumes directly. This however leads to the problem that the sheer size of the simulation instances renders the integer programming approach infeasible. Another solution would be to carefully split the large blocks into different pieces so as to reduce the size κ_{\max} of the biggest block in the simulation. This only leads to a moderate (and controllable) increase in the problem size and still allows us to apply integer programming techniques. However, splitting a block not only introduces new nodes in the grid graph but also new edges. Moreover, the simulation software may limit the ways in which we may split blocks. So we want to split the large blocks such that the software can handle the split blocks and still a good balancing of computational load and communication cycles is possible.

First empirical experiments on the grids presented in this thesis indicate that splitting large blocks indeed leads to block-mappings that are superior to mappings based on unsplit blocks. The optimal solution of the block-splitting problem was however far beyond the scope of this thesis and defines an interesting problem to be analysed in the future.

As a final remark we recall that routed communication cannot be represented by either of our models (see Section 6.9). So one direction of future research is to develop (and solve instance of) a model that captures the case in which the communication path between processor is not predetermined but may be chosen at runtime.

Appendix A

Further Computational Results

In Chapter 10 we listed only part of the tables containing computational results. In the following we give the remaining tables.

A.1 Initial Assignment on four Processors

Algorithm	sorting	maxsize	colours	objective	gap
fillup	size-	696320	50	3544.4801	46.77%
fillup	degree-	729088	34	2793.6321	32.46%
smallest	size-	697344	45	3296.0161	42.76%
smallest	none	721920	55	3832.8801	50.77%
circular	size-	691200	49	3486.8001	45.89%
circular	degree+	858112	50	3787.1681	50.18%
circular	size+	737280	45	3355.9201	43.78%
small-degree		749568	24	2324.3521	18.82%
small-multicut		719872	26	2379.8081	20.72%
connected-0.00		741376	24	2312.0641	18.39%
connected-0.30		741376	24	2312.0641	18.39%
connected-0.70		741376	25	2362.0641	20.12%
connected-1.00		732160	22	2198.2401	14.17%
connected-0.00r		741376	24	2312.0641	18.39%
connected-0.30r		702464	31	2603.6961	27.53%
connected-0.70r		724992	28	2487.4881	24.15%
connected-1.00r		757760	36	2936.6401	35.75%

Table A.1: Initial assignment algorithms on Grid-5 and 4 processors.

Algorithm	sorting	maxsize	colours	objective	gap
fillup	size-	488448	62	3832.6721	54.43%
fillup	none	501120	41	2801.6801	37.65%
smallest	size-	488448	65	3982.6721	56.14%
smallest	size+	558720	68	4238.0801	58.79%
smallest	degree-	494208	63	3891.3121	55.11%
circular	size-	520704	69	4231.0561	58.72%
circular	degree+	594432	70	4391.6481	60.23%
circular	none	623232	62	4034.8481	56.71%
small-degree		546048	31	2369.0721	26.27%
small-multicut		513792	31	2320.6881	24.73%
connected-0.00		527616	26	2091.4240	16.48%
connected-0.30		576000	37	2714.0001	35.64%
connected-0.70		559872	25	2089.8080	16.42%
connected-1.00		562176	26	2143.2640	18.50%
connected-0.00r		527616	26	2091.4240	16.48%
connected-0.30r		559872	31	2389.8081	26.91%
connected-0.70r		555264	40	2832.8961	38.34%
connected-1.00r		638208	42	3057.3121	42.87%

Table A.2: Initial assignment algorithms on Grid-6 and 4 processors.

Algorithm	sorting	maxsize	colours	objective	gap
fillup	size-	534528	61	3851.7921	55.19%
fillup	size+	601344	44	3102.0161	44.36%
smallest	size-	536832	68	4205.2481	58.96%
smallest	size+	614016	74	4621.0241	62.65%
smallest	degree+	600192	60	3900.2881	55.75%
circular	size-	573696	73	4510.5441	61.73%
circular	none	625536	64	4138.3041	58.29%
small-degree		601344	30	2402.0161	28.14%
small-multicut		587520	25	2131.2800	19.02%
connected-0.00		569088	32	2453.6321	29.66%
connected-0.30		582912	25	2124.3680	18.75%
connected-0.70		624384	31	2486.5761	30.59%
connected-1.00		539136	28	2208.7040	21.86%
connected-0.00r		569088	32	2453.6321	29.66%
connected-0.30r		569088	28	2253.6321	23.41%
connected-0.70r		656640	30	2484.9601	30.54%
connected-1.00r		601344	36	2702.0161	36.12%

Table A.3: Initial assignment algorithms on Grid-7 and 4 processors.

Algorithm	sorting	maxsize	colours	objective	gap
fillup	size-	774144	81	5211.2161	60.22 %
fillup	degree+	794624	81	5241.9361	60.46 %
fillup	none	776192	66	4464.2881	53.57 %
smallest	size-	774144	107	6511.2161	68.16 %
smallest	degree-	870400	121	7355.6001	71.82 %
smallest	none	878592	88	5717.8881	63.75 %
circular	size-	819200	97	6078.8001	65.90 %
circular	size+	892928	100	6339.3921	67.30 %
circular	degree-	882688	92	5924.0321	65.01 %
small-degree				failed	
small-multicut		933888	53	4050.8321	48.83 %
connected-0.00		837632	36	3056.4481	32.18 %
connected-0.30		880640	37	3170.9601	34.63 %
connected-0.70		880640	37	3170.9601	34.63 %
connected-1.00		889856	33	2984.7841	30.55 %
connected-0.00r		837632	36	3056.4481	32.18 %
connected-0.30r		839680	46	3559.5201	41.76 %
connected-0.70r		815104	37	3072.6561	32.54 %
connected-1.00r		945152	40	3417.7281	39.35 %

Table A.4: Initial assignment algorithms on Grid-8 and 4 processors.

Algorithm	sorting	maxsize	colours	objective	gap
fillup	size-	840960	145	8511.4402	77.92 %
fillup	degree+	850944	153	8926.4162	78.95 %
fillup	size+	958464	48	3837.6961	51.03 %
smallest	size-	840576	139	8210.8641	77.11 %
smallest	degree+	852480	195	11028.7202	82.96 %
smallest	size-	840576	139	8210.8641	77.11 %
circular	size-	910080	136	8165.1201	76.99 %
circular	degree-	1076352	137	8464.5282	77.80 %
circular	size-	910080	136	8165.1201	76.99 %
small-degree		1081344	33	3272.0161	42.57 %
small-multicut		909312	34	3063.9681	38.67 %
connected-0.00		971520	40	3457.2801	45.65 %
connected-0.30		885504	39	3278.2561	42.68 %
connected-0.70		933888	26	2700.8321	30.42 %
connected-1.00		943104	31	2964.6561	36.61 %
connected-0.00r		971520	40	3457.2801	45.65 %
connected-0.30r		847872	35	3021.8081	37.81 %
connected-0.70r		920064	29	2830.0961	33.60 %
connected-1.00r		976896	81	5515.3441	65.93 %

Table A.5: Initial assignment algorithms on Grid-9 and 4 processors.

A.2 Local Improvement on four Processors

initial	enhance	moves	maxsize	colours	objective	gap
connected-0.00	local	+++	708608	22	2162.9121	12.77%
connected-1.00r	local	+++	715776	35	2823.6641	33.18%
connected-0.00	local	++-	708608	22	2162.9121	12.77%
connected-0.30r	local	++-	706560	28	2459.8401	23.30%
connected-0.00	local	++-	723968	22	2185.9521	13.69%
connected-1.00r	local	++-	694272	27	2391.4081	21.10%
connected-1.00	local	---+	732160	22	2198.2401	14.17%
connected-1.00r	local	---+	692224	27	2388.3361	21.00%
connected-0.00	local	+++	708608	22	2162.9121	12.77%
connected-1.00r	local	+++	734208	26	2401.3121	21.43%
connected-0.00	local	+++	708608	22	2162.9121	12.77%
connected-1.00r	local	+++	718848	28	2478.2721	23.87%
connected-0.70	local	+++	708608	22	2162.9121	12.77%
connected-1.00r	local	+++	701440	28	2452.1601	23.06%
connected-0.30r	tabu	+++	747520	19	2071.2801	8.91%
connected-0.70r	tabu	+++	724992	25	2337.4881	19.28%
connected-0.70r	tabu	++-	723968	20	2085.9521	9.55%
connected-0.30r	tabu	++-	692224	28	2438.3361	22.62%
connected-0.00	tabu	++-	703488	22	2155.2321	12.45%
connected-1.00r	tabu	++-	696320	25	2294.4801	17.77%
connected-0.00	tabu	---+	723968	18	1985.9521	4.99%
connected-0.70	tabu	---+	708608	23	2212.9121	14.74%
connected-0.30r	tabu	+++	718848	19	2028.2721	6.98%
connected-1.00r	tabu	+++	705536	26	2358.3041	19.99%
connected-1.00r	tabu	+++	707584	20	2061.3761	8.47%
connected-0.30r	tabu	+++	698368	24	2247.5521	16.05%
connected-1.00	tabu	+++	696320	22	2144.4801	12.02%
connected-1.00r	tabu	+++	692224	25	2288.3361	17.55%

Table A.6: Local improvement on Grid-5 and 4 processors.

initial	enhance	moves	maxsize	colours	objective	gap
connected-1.00	local	+--	582912	19	1824.3680	4.26%
connected-0.70r	local	+--	539136	28	2208.7040	20.92%
connected-1.00	local	-+-	516096	22	1874.1440	6.80%
connected-0.70r	local	-+-	569088	30	2353.6321	25.79%
connected-1.00	local	++-	582912	19	1824.3680	4.26%
connected-0.70r	local	++-	536832	27	2155.2480	18.96%
connected-0.70r	local	---+	534528	21	1851.7920	5.67%
connected-1.00r	local	---+	551808	25	2077.7120	15.93%
connected-1.00	local	+++	582912	19	1824.3680	4.26%
connected-0.70r	local	+++	529920	27	2144.8800	18.56%
connected-0.30r	local	-++	509184	21	1813.7760	3.70%
connected-0.70r	local	-++	529920	27	2144.8800	18.56%
connected-0.30r	local	+++	504576	21	1806.8640	3.33%
connected-0.70r	local	+++	529920	27	2144.8800	18.56%
connected-0.70r	tabu	+--	516096	21	1824.1440	4.24%
connected-1.00r	tabu	+--	561024	25	2091.5360	16.49%
connected-1.00	tabu	-+-	541440	21	1862.1600	6.20%
connected-1.00r	tabu	-+-	751104	22	2226.6561	21.55%
connected-0.70	tabu	++-	509184	21	1813.7760	3.70%
connected-0.70r	tabu	++-	496512	27	2094.7680	16.62%
connected-1.00	tabu	---+	499968	21	1799.9520	2.96%
connected-1.00r	tabu	---+	547200	25	2070.8000	15.65%
connected-0.70r	tabu	+++	546048	20	1819.0720	3.98%
connected-1.00r	tabu	+++	605952	22	2008.9280	13.05%
connected-0.30r	tabu	-++	509184	21	1813.7760	3.70%
connected-1.00r	tabu	-++	578304	24	2067.4560	15.51%
connected-0.30r	tabu	+++	504576	21	1806.8640	3.33%
connected-1.00r	tabu	+++	619776	21	1979.6640	11.77%

Table A.7: Local improvement on Grid-6 and 4 processors.

initial	enhance	moves	maxsize	colours	objective	gap
connected-0.30	local	+-	573696	22	1960.5440	11.96%
connected-0.70	local	+-	624384	26	2236.5761	22.83%
connected-1.00	local	-+-	546048	22	1919.0720	10.06%
connected-1.00r	local	-+-	578304	27	2217.4561	22.16%
connected-1.00	local	++-	569088	19	1803.6320	4.31%
connected-0.00	local	++-	552960	28	2229.4401	22.58%
connected-0.30	local	---+	582912	18	1774.3680	2.73%
connected-0.00	local	---+	578304	26	2167.4561	20.37%
connected-0.30	local	+++	582912	19	1824.3680	5.39%
connected-0.00	local	+++	564480	27	2196.7201	21.43%
connected-1.00	local	-++	569088	20	1853.6320	6.89%
connected-0.00	local	-++	564480	27	2196.7201	21.43%
connected-0.30	local	+++	582912	19	1824.3680	5.39%
connected-0.00	local	+++	564480	27	2196.7201	21.43%
connected-1.00	tabu	+-	638208	20	1957.3120	11.82%
connected-0.70	tabu	+-	624384	26	2236.5761	22.83%
connected-0.30	tabu	-+-	582912	19	1824.3680	5.39%
connected-0.00	tabu	-+-	550656	27	2175.9840	20.68%
connected-0.30	tabu	++-	582912	18	1774.3680	2.73%
connected-0.00	tabu	++-	552960	28	2229.4401	22.58%
connected-1.00	tabu	---+	580608	18	1770.9120	2.54%
connected-0.00	tabu	---+	578304	26	2167.4561	20.37%
connected-0.70r	tabu	+++	582912	18	1774.3680	2.73%
connected-0.00	tabu	+++	564480	27	2196.7201	21.43%
connected-1.00	tabu	-++	569088	20	1853.6320	6.89%
connected-0.00	tabu	-++	578304	26	2167.4561	20.37%
connected-1.00	tabu	+++	580608	18	1770.9120	2.54%
connected-0.00	tabu	+++	564480	27	2196.7201	21.43%

Table A.8: Local improvement on Grid-7 and 4 processors.

initial	enhance	moves	maxsize	colours	objective	gap
connected-0.00	local	+++	779264	28	2568.8961	19.31 %
connected-1.00	local	+++	816128	31	2774.1921	25.28 %
connected-0.30r	local	++-	839680	26	2559.5201	19.01 %
connected-0.00	local	++-	837632	29	2706.4481	23.41 %
connected-1.00	local	++-	790528	27	2535.7921	18.25 %
connected-1.00r	local	++-	868352	27	2652.5281	21.85 %
connected-0.70r	local	---+	835584	24	2453.3761	15.51 %
connected-1.00r	local	---+	868352	28	2702.5281	23.30 %
connected-1.00	local	+++	790528	27	2535.7921	18.25 %
connected-1.00r	local	+++	868352	28	2702.5281	23.30 %
connected-0.00	local	+++	803840	26	2505.7601	17.27 %
connected-0.30	local	+++	797696	29	2646.5441	21.68 %
connected-0.30r	local	+++	831488	24	2447.2321	15.30 %
connected-0.30	local	+++	802816	29	2654.2241	21.90 %
connected-0.00	tabu	+++	779264	28	2568.8961	19.31 %
connected-1.00	tabu	+++	791552	31	2737.3281	24.27 %
connected-0.70r	tabu	++-	806912	26	2510.3681	17.43 %
connected-0.30	tabu	++-	829440	29	2694.1601	23.06 %
connected-0.70r	tabu	++-	811008	24	2416.5121	14.22 %
connected-1.00r	tabu	++-	868352	27	2652.5281	21.85 %
connected-0.70r	tabu	---+	835584	24	2453.3761	15.51 %
connected-1.00r	tabu	---+	868352	27	2652.5281	21.85 %
connected-1.00	tabu	+++	790528	27	2535.7921	18.25 %
connected-1.00r	tabu	+++	868352	27	2652.5281	21.85 %
connected-0.30r	tabu	+++	831488	24	2447.2321	15.30 %
connected-0.30	tabu	+++	779264	29	2618.8961	20.85 %
connected-0.70r	tabu	+++	811008	24	2416.5121	14.22 %
connected-0.30	tabu	+++	790528	29	2635.7921	21.36 %

Table A.9: Local improvement on Grid-8 and 4 processors.

initial	enhance	moves	maxsize	colours	objective	gap
connected-1.00	local	+++	864768	23	2447.1521	23.21 %
connected-0.30r	local	+++	847872	35	3021.8081	37.81 %
connected-0.70	local	++-	884736	22	2427.1041	22.58 %
connected-1.00r	local	++-	972288	34	3158.4321	40.50 %
connected-1.00	local	++-	864768	22	2397.1521	21.61 %
connected-0.30r	local	++-	847872	34	2971.8081	36.77 %
connected-0.70r	local	---+	866304	24	2499.4561	24.82 %
connected-0.30	local	---+	869376	34	3004.0641	37.45 %
connected-0.70r	local	+++	872448	22	2408.6721	21.98 %
connected-0.30	local	+++	869376	34	3004.0641	37.45 %
connected-1.00r	local	---+	886272	22	2429.4081	22.65 %
connected-0.30	local	---+	851712	34	2977.5681	36.89 %
connected-0.70r	local	+++	906240	22	2459.3601	23.59 %
connected-0.30	local	+++	851712	34	2977.5681	36.89 %
connected-1.00	tabu	+++	864768	23	2447.1521	23.21 %
connected-0.30	tabu	+++	897024	32	2945.5361	36.20 %
connected-0.70r	tabu	++-	884736	21	2377.1041	20.95 %
connected-0.30	tabu	++-	848640	34	2972.9601	36.79 %
connected-0.70r	tabu	++-	847872	22	2371.8081	20.77 %
connected-0.30	tabu	++-	872448	32	2908.6721	35.39 %
connected-0.70	tabu	---+	858624	22	2387.9361	21.31 %
connected-0.30r	tabu	---+	854016	27	2631.0241	28.58 %
connected-0.70r	tabu	+++	847872	22	2371.8081	20.77 %
connected-0.30r	tabu	+++	847872	30	2771.8081	32.20 %
connected-0.00	tabu	---+	847872	22	2371.8081	20.77 %
connected-0.30r	tabu	---+	844800	34	2967.2001	36.67 %
connected-0.00	tabu	+++	860928	22	2391.3921	21.42 %
connected-0.30r	tabu	+++	866304	28	2699.4561	30.39 %

Table A.10: Local improvement on Grid-9 and 4 processors.

A.3 Local Improvement on eight Processors

initial	enhance	moves	surrogate	time (s)	$ P' $	maxsize	colours	objective	gap
1.00	local	---	vizing	0.01	8	387072	20	1580.6080	49.09
1.00	local	---	shannon	0.00	8	387072	20	1580.6080	49.09
1.00	local	--+	vizing	0.04	8	387072	20	1580.6080	49.09
1.00	local	--+	shannon	0.00	8	387072	20	1580.6080	49.09
1.00	local	++-	vizing	0.07	8	387072	20	1580.6080	49.09
1.00	local	++-	shannon	0.01	8	387072	20	1580.6080	49.09
1.00	local	---+	vizing	0.22	8	387072	20	1580.6080	49.09
1.00	local	---+	shannon	0.06	8	387072	20	1580.6080	49.09
1.00	local	+++	vizing	0.18	8	387072	20	1580.6080	49.09
1.00	local	+++	shannon	0.06	8	387072	20	1580.6080	49.09
1.00	local	---+	vizing	0.31	8	387072	20	1580.6080	49.09
1.00	local	---+	shannon	0.03	8	387072	20	1580.6080	49.09
1.00	local	+++	vizing	0.43	8	387072	20	1580.6080	49.09
1.00	local	+++	shannon	0.07	8	387072	20	1580.6080	49.09
1.00	tabu	---	vizing	2.81	8	387072	20	1580.6080	49.09
1.00	tabu	---	shannon	1.73	8	387072	20	1580.6080	49.09
1.00	tabu	--+	vizing	16.06	8	387072	20	1580.6080	49.09
1.00	tabu	--+	shannon	10.62	8	387072	20	1580.6080	49.09
1.00	tabu	++-	vizing	19.00	8	387072	20	1580.6080	49.09
1.00	tabu	++-	shannon	12.28	8	387072	20	1580.6080	49.09
1.00	tabu	---+	vizing	71.01	8	387072	20	1580.6080	49.09
1.00	tabu	---+	shannon	45.89	8	387072	20	1580.6080	49.09
1.00	tabu	+++	vizing	83.51	8	387072	20	1580.6080	49.09
1.00	tabu	+++	shannon	49.15	8	387072	20	1580.6080	49.09
1.00	tabu	---+	vizing	97.57	8	387072	20	1580.6080	49.09
1.00	tabu	---+	shannon	56.82	8	387072	20	1580.6080	49.09
1.00	tabu	+++	vizing	100.35	8	387072	20	1580.6080	49.09
1.00	tabu	+++	shannon	61.56	8	387072	20	1580.6080	49.09

Table A.11: Local improvement algorithms on Grid-5 and 8 processors.

initial	enhance	moves	surrogate	time (s)	$ P' $	maxsize	colours	objective	gap
0.70r	local	+++	vizing	0.02	8	262656	23	1543.9840	46.58
0.00	local	+++	shannon	0.00	7	334080	26	1801.1200	54.21
0.30	local	++-	vizing	0.09	7	324864	24	1687.2960	51.12
0.00	local	++-	shannon	0.01	7	334080	26	1801.1200	54.21
0.00	local	+++	vizing	0.09	8	255744	24	1583.6160	47.92
0.00	local	+++	shannon	0.01	7	334080	26	1801.1200	54.21
0.30	local	---	vizing	0.32	8	248832	25	1623.2480	49.19
0.00	local	+++	shannon	0.04	7	334080	26	1801.1200	54.21
0.30	local	+++	vizing	0.39	8	248832	23	1523.2480	45.86
0.00	local	+++	shannon	0.04	7	334080	26	1801.1200	54.21
0.30	local	++-	vizing	0.38	8	248832	23	1523.2480	45.86
0.00	local	++-	shannon	0.05	7	334080	26	1801.1200	54.21
0.30	local	+++	vizing	0.47	8	248832	23	1523.2480	45.86
0.00	local	+++	shannon	0.05	7	334080	26	1801.1200	54.21
0.70r	tabu	+++	vizing	5.11	8	260352	24	1590.5280	48.15
0.00	tabu	+++	shannon	3.21	7	334080	26	1801.1200	54.21
0.70r	tabu	+-	vizing	32.48	7	313344	26	1770.0160	53.40
0.00	tabu	+-	shannon	22.43	7	334080	26	1801.1200	54.21
1.00r	tabu	+++	vizing	38.52	8	258048	23	1537.0720	46.34
0.00	tabu	+++	shannon	26.21	7	334080	26	1801.1200	54.21
0.30	tabu	---	vizing	161.29	8	244224	27	1716.3360	51.95
0.00	tabu	+++	shannon	102.48	7	334080	26	1801.1200	54.21
0.30	tabu	+++	vizing	182.70	8	248832	23	1523.2480	45.86
0.00	tabu	+++	shannon	108.51	7	334080	26	1801.1200	54.21
0.30	tabu	++-	vizing	210.55	8	244224	26	1666.3360	50.50
0.00	tabu	++-	shannon	127.68	7	334080	26	1801.1200	54.21
0.30	tabu	+++	vizing	218.07	8	248832	23	1523.2480	45.86
0.00	tabu	+++	shannon	132.67	7	334080	26	1801.1200	54.21

Table A.12: Local improvement algorithms on Grid-6 and 8 processors.

initial	enhance	moves	surrogate	time (s)	$ P' $	maxsize	colours	objective	gap
0.30	local	+++	vizing	0.01	8	320256	19	1430.3840	41.66
0.30	local	+++	shannon	0.00	8	294912	19	1392.3680	40.06
0.30	local	+-	vizing	0.02	8	294912	19	1392.3680	40.06
0.30	local	+-	shannon	0.01	8	320256	19	1430.3840	41.66
0.30	local	+++	vizing	0.04	8	320256	19	1430.3840	41.66
0.30	local	+++	shannon	0.02	8	294912	19	1392.3680	40.06
0.30	local	---	vizing	0.33	8	320256	19	1430.3840	41.66
0.30	local	+++	shannon	0.04	8	320256	19	1430.3840	41.66
0.30	local	+++	vizing	0.34	8	320256	19	1430.3840	41.66
0.30	local	+++	shannon	0.09	8	294912	19	1392.3680	40.06
0.30	local	+-	vizing	0.24	8	320256	19	1430.3840	41.66
0.30	local	+++	shannon	0.05	8	320256	19	1430.3840	41.66
0.30	local	+++	vizing	0.24	8	320256	19	1430.3840	41.66
0.30	local	+++	shannon	0.10	8	294912	19	1392.3680	40.06
0.30	tabu	+++	vizing	5.50	8	320256	19	1430.3840	41.66
0.30	tabu	+-	shannon	3.50	8	294912	19	1392.3680	40.06
0.30	tabu	+-	vizing	36.87	8	320256	19	1430.3840	41.66
0.30	tabu	+-	shannon	24.47	8	320256	19	1430.3840	41.66
0.30	tabu	+++	vizing	43.12	8	320256	19	1430.3840	41.66
0.30	tabu	+++	shannon	28.08	8	294912	19	1392.3680	40.06
0.30	tabu	---	vizing	168.21	8	320256	19	1430.3840	41.66
0.30	tabu	+++	shannon	109.27	8	320256	19	1430.3840	41.66
0.30	tabu	+++	vizing	190.40	8	320256	19	1430.3840	41.66
0.30	tabu	+++	shannon	112.97	8	294912	19	1392.3680	40.06
0.30	tabu	+-	vizing	221.76	8	320256	19	1430.3840	41.66
0.30	tabu	+++	shannon	136.78	8	320256	19	1430.3840	41.66
0.30	tabu	+++	vizing	226.76	8	320256	19	1430.3840	41.66
0.30	tabu	+++	shannon	138.64	8	294912	19	1392.3680	40.06

Table A.13: Local improvement algorithms on Grid-7 and 8 processors.

initial	enhance	moves	surrogate	time (s)	$ P' $	maxsize	colours	objective	gap
0.30r	local	---	vizing	0.00	8	442368	28	2063.5520	53.66
0.30r	local	---	shannon	0.00	8	447488	31	2221.2320	56.95
0.30r	local	--+	vizing	0.03	8	447488	31	2221.2320	56.95
0.30r	local	--+	shannon	0.01	8	447488	31	2221.2320	56.95
0.30r	local	+++	vizing	0.03	8	447488	31	2221.2320	56.95
0.30r	local	+++	shannon	0.01	8	447488	31	2221.2320	56.95
0.30r	local	---	vizing	0.54	8	447488	31	2221.2320	56.95
0.30r	local	---	shannon	0.03	8	447488	31	2221.2320	56.95
0.30r	local	+++	vizing	0.11	8	442368	28	2063.5520	53.66
0.30r	local	+++	shannon	0.03	8	447488	31	2221.2320	56.95
0.30r	local	--+	vizing	0.76	8	447488	31	2221.2320	56.95
0.30r	local	--+	shannon	0.04	8	447488	31	2221.2320	56.95
0.30r	local	+++	vizing	0.43	8	447488	31	2221.2320	56.95
0.30r	local	+++	shannon	0.04	8	447488	31	2221.2320	56.95
0.30r	tabu	---	vizing	10.72	8	447488	31	2221.2320	56.95
0.30r	tabu	---	shannon	6.51	8	447488	31	2221.2320	56.95
0.30r	tabu	--+	vizing	108.62	8	447488	31	2221.2320	56.95
0.30r	tabu	--+	shannon	4.80	8	447488	31	2221.2320	56.95
0.30r	tabu	+++	vizing	124.07	8	447488	31	2221.2320	56.95
0.30r	tabu	+++	shannon	43.98	8	447488	31	2221.2320	56.95
0.30r	tabu	---	vizing	364.88	8	447488	31	2221.2320	56.95
0.30r	tabu	---	shannon	235.11	8	447488	31	2221.2320	56.95
0.30r	tabu	+++	vizing	397.78	8	447488	31	2221.2320	56.95
0.30r	tabu	+++	shannon	242.38	8	447488	31	2221.2320	56.95
0.00	tabu	--+	vizing	499.68	8	393216	32	2189.8240	56.33
0.30r	tabu	--+	shannon	263.44	8	447488	31	2221.2320	56.95
0.30r	tabu	+++	vizing	508.89	8	447488	31	2221.2320	56.95
0.30r	tabu	+++	shannon	320.04	8	447488	31	2221.2320	56.95

Table A.14: Local improvement algorithms on Grid-8 and 8 processors.

initial	enhance	moves	surrogate	time (s)	$ P' $	maxsize	colours	objective	gap
0.70r	local	---	vizing	0.02	8	454656	29	2131.9840	54.11
0.70r	local	---	shannon	0.01	8	503808	26	2055.7120	52.41
0.70r	local	--+	vizing	0.11	8	552960	28	2229.4401	56.12
0.70r	local	--+	shannon	0.05	8	552960	28	2229.4401	56.12
0.70r	local	+++	vizing	0.24	8	442368	29	2113.5520	53.71
0.70r	local	+++	shannon	0.05	8	503808	26	2055.7120	52.41
0.70r	local	---	vizing	0.65	8	552960	28	2229.4401	56.12
0.70r	local	---	shannon	0.10	8	552960	28	2229.4401	56.12
0.70r	local	+++	vizing	1.31	8	430080	28	2045.1200	52.16
0.70r	local	+++	shannon	0.21	8	503808	26	2055.7120	52.41
0.70r	local	--+	vizing	1.39	8	552960	28	2229.4401	56.12
0.70r	local	--+	shannon	0.25	8	552960	28	2229.4401	56.12
1.00	local	+++	vizing	1.97	8	479232	26	2018.8480	51.54
0.70r	local	+++	shannon	0.26	8	503808	26	2055.7120	52.41
0.70r	tabu	---	vizing	19.22	8	442368	30	2163.5520	54.78
0.70r	tabu	---	shannon	11.82	8	503808	26	2055.7120	52.41
0.70r	tabu	--+	vizing	205.81	8	552960	28	2229.4401	56.12
0.70r	tabu	--+	shannon	10.05	8	552960	28	2229.4401	56.12
0.70r	tabu	+++	vizing	224.76	8	430080	26	1945.1200	49.70
0.70r	tabu	+++	shannon	144.98	8	503808	26	2055.7120	52.41
0.70r	tabu	---	vizing	859.21	8	422016	30	2133.0240	54.13
0.70r	tabu	---	shannon	561.55	8	552960	28	2229.4401	56.12
0.70r	tabu	+++	vizing	909.64	8	430080	28	2045.1200	52.16
0.70r	tabu	+++	shannon	539.27	8	503808	26	2055.7120	52.41
0.70r	tabu	--+	vizing	1093.64	8	421632	31	2182.4480	55.17
0.70r	tabu	--+	shannon	685.11	8	552960	28	2229.4401	56.12
1.00	tabu	+++	vizing	1105.60	8	422400	31	2183.6000	55.20
0.70r	tabu	+++	shannon	698.28	8	503808	26	2055.7120	52.41

Table A.15: Local improvement algorithms on Grid-9 and 8 processors.

Appendix B

Separation of Blossom-Inequalities

Recall valid inequality (6.40) for model (SLOT*)

$$\sum_{\substack{p,k \in P' \\ p \neq k}} \left(\sum_{e \in E} x_{e, \{p,k\}} \right) \leq \left\lfloor \frac{|P'|}{2} \right\rfloor \cdot \sum_{c \in C} z_c \quad (\text{B.1})$$

where $P' \subseteq P$ was a set of processors of odd cardinality¹. Variants of this inequality also occurred in models (NAIVE) and (REP). In general (B.1) gives rise to $\sum_{i=1}^{\lfloor |P|/2 \rfloor} \binom{|P|}{2i+1}$ concrete inequalities. We will show now, how these inequalities can be separated.

Before we explain the actual separation strategy, we first need a famous result from matching theory. For a graph $G = (V, E)$ and a subset $F \subseteq E$ of edges let χ^F denote the incidence vector of F in \mathbb{R}^E . Moreover, let

$$P_{\text{mat}}(G) := \text{conv}\{\chi^M : M \text{ is matching in } G\}$$

denote the *matching polytope*, i. e., the convex hull (in \mathbb{R}^E) of all incidence vectors of matchings in G .

Theorem 81 (Matching Polytope [44, 152]). *For a graph $G = (V, E)$ a point $x \in \mathbb{R}^E$ belongs to $P_{\text{mat}}(G)$ if and only if it satisfies the following three inequalities²:*

$$x_e \geq 0 \quad e \in E \quad (\text{B.2})$$

$$\sum_{e \in \delta(v)} x_e \leq 1 \quad v \in V \quad (\text{B.3})$$

$$\sum_{e \in E[U]} x_e \leq \left\lfloor \frac{1}{2}|U| \right\rfloor \quad U \subseteq V, |U| \text{ odd}, |U| > 1. \quad (\text{B.4})$$

¹In fact the set P' was required to have cardinality at least three, but (B.1) is obviously also valid if $|P'| = 1$

²Inequality (B.4) is nearly identical to (B.1) and is well known as *blossom inequality*, hence the title of this chapter.

Consider now the problem to decide for a given point $x' \in \mathbb{R}^E$ whether $x \in P_{\text{mat}}(G)$ or to find an inequality that is valid for $P_{\text{mat}}(G)$ but not satisfied by x' . This is also called the *separation problem* over $P_{\text{mat}}(G)$ and is equivalent to the optimisation problem over $P_{\text{mat}}(G)$ (see e. g., [76, 151]). The latter problem is precisely the problem of finding a maximum (weight) matching in G and can be solved in polynomial time [60]. Thus the separation problem over $P_{\text{mat}}(G)$ is polynomial time solvable for any graph G . We will use this result to separate (B.1).

Assume we are given a point (x^*, z^*, b^*) for (SLOT*) and want to test whether this point violates (B.1). We define $Z := \sum_{c \in C} z_c^*$ and must find a set P' that satisfies $|P'|$ odd and

$$\sum_{\substack{p,k \in P' \\ p \neq k}} \left(\sum_{e \in E} x_{e,\{p,k\}} \right) > Z \left\lfloor \frac{|P'|}{2} \right\rfloor.$$

If $Z = 0$ we simply check $x_{e,p,k}^*$ for $e \in E$ and $p \neq k$. If one of these variables is non-zero, inequality (B.1) is violated for $P' = \{p, k, l\}$, where $l \in P$ is an arbitrary processor different from p and k .

If $Z > 0$ we define for an edge $pk \in K_{|P|}$

$$y_{pk} := \frac{1}{Z} \sum_{e \in E} (x_{e,p,k}^* + x_{e,k,p}^*).$$

We can now decide in polynomial time whether $y \in P_{\text{mat}}(K_{|P|})$. If it is, we have that

$$\sum_{pk \in E[P']} y_{pk} = \frac{1}{Z} \sum_{pk \in E[P']} \sum_{e \in E} (x_{e,p,k}^* + x_{e,k,p}^*) \leq \left\lfloor \frac{1}{2} |P'| \right\rfloor$$

for any odd set $P' \subseteq P$. Hence (x^*, z^*, b^*) does not violate (B.1).

Otherwise we either find a processor $p \in P$ such that

$$1 < \sum_{pk \in \delta_{K_{|P|}}(p)} y_e = \frac{1}{Z} \sum_{e \in E} \sum_{k \neq p} (x_{e,p,k}^* + x_{e,k,p}^*)$$

or an odd set P' such that

$$\left\lfloor \frac{1}{2} |P'| \right\rfloor < \sum_{pk \in E_{K_{|P|}}(P')} y_e = \frac{1}{Z} \sum_{e \in E} \sum_{\substack{p,k \in P' \\ p \neq k}} (x_{e,p,k}^* + x_{e,k,p}^*).$$

In both cases we easily construct the instance of (B.1) that is violated by (x^*, z^*, b^*) .

Schrijver [152] describes a strongly polynomial algorithm to solve the separation problem over the perfect matching polytope, which is slightly different to the general matching polytope. Polynomial time solvability of the weighted matching problem implies that a similar algorithm exists for the general matching polytope and this algorithm can be used to separate inequality (B.1) in polynomial time.

Bibliography

- [1] T. Achterberg and T. Berthold. Improving the feasibility pump. Technical report, Konrad-Zuse-Zentrum für Informationstechnik, 2005.
- [2] T. Achterberg, T. Koch, and A. Martin. Branching rules revisited. *Operations Research Letters*, 33:42–54, 2005.
- [3] L. D. Andersen. On edge-colorings of graphs. *Mathematica Scandinavica*, 40:161–175, 1977.
- [4] Anderson, Hall, Hartline, Hobbs, Karlin, Saia, Swaminathan, and Wilkes. An experimental study of data migration algorithms. In *WAE: International Workshop on Algorithm Engineering*. LNCS, 2001.
- [5] J. A. Appleget and R. K. Wood. Explicit-constraint branching for solving mixed-integer programs. In M. Laguna and J. González-Velarde, editors, *Computing Tools for Modeling, Optimization and Simulation*. Kluwer Academic Publishers, Boston, 2000.
- [6] C. Avanthay, A. Hertz, and N. Zufferey. A variable neighborhood search for graph coloring. *European Journal of Operations Research*, 151:379–388, 2003.
- [7] E. Balas. A class of location, distribution and scheduling problems: Modeling and solution methods. In P. Gray and L. Yuanzhang, editors, *Proceedings of the Chinese-U.S. Symposium on Systems Analysis*. John Wiley and Sons, 1983.
- [8] A. Bar-Noy, R. Motwani, and J. Naor. A lower bound for on-line edge coloring.
- [9] A. Bar-Noy, R. Motwani, and J. Naor. The greedy algorithm is optimal for on-line edge coloring. *Information Processing Letters*, pages 251–253, 1992.
- [10] P. Bastian. *Parallele adaptive Mehrgitterverfahren*. PhD thesis, Universität Heidelberg, 1994. ICA-Bericht 94-1.
- [11] P. Bastian. Dynamic load balancing for parallel adaptive multigrid methods on unstructured meshes. In S. Wagner, editor, *Computational Fluid Dynamics on Parallel Systems*, volume 50 of *Notes on Numerical Fluid Mechanics*, Braunschweig, 1995. Vieweg.

- [12] P. Bastian. Load balancing for adaptive multigrid methods. *SIAM J. Sci. Stat. Comput.*, 19(4):1303–1321, 1998.
- [13] P. Bastian and G. Wittum. Adaptive multigrid methods: The UG concept. In *Proceedings of the 9th GAMM Seminar Kiel*, Notes on Numerical Fluid Mechanics. Vieweg, 1994.
- [14] E. M. L. Beale and J. A. Tomlin. Special facilities in a general mathematical programming system for non-convex problems using ordered sets of variables. In J. R. Lawrence, editor, *Proceedings of the 5th International Operations Research Conference*, pages 447–454. Tavistock Publications Limited, London, 1970.
- [15] B. Beauquier, J.-C. Bermond, L. Gargano, P. Hell, S. Pérennes, and U. Vaccaro. Graph problems arising from wavelength-routing in all-optical networks. Technical Report RR-3165, The French National Institute for Research in Computer Science and Control, 1997.
- [16] C. Berge. *Theorie des graphes et ses applications*. Dunod, Paris, 1958.
- [17] C. Berge. *Graphes and Hypergraphs*, volume 6. North-Holland Mathematical Library, 1973.
- [18] L. Bertacco, M. Fischetti, and A. Lodi. A feasibility pump heuristic for general mixed-integer problems. Technical Report OR-05-5, University of Padova, 2005.
- [19] J. Blazek. *Computational Fluid Dynamics: Principles and Applications*. Elsevier Science, 2001.
- [20] C. Blum and A. Roli. Metaheuristics in combinatorial optimization: Overview and conceptual comparison. *ACM Computing Surveys*, 35(3):268–308, 2003.
- [21] P. Brooker and N. Christofides. The optimal partitioning of graphs. *SIAM Journal on Applied Mathematics*, 30:55–69, 1976.
- [22] M. Campelo, R. Correa, and Y. Frota. Cliques, holes and the vertex coloring polytope. *Information Processing Letters*, 89:159–164, 2004.
- [23] A. Caprara, M. Fischetti, and P. Toth. Algorithms for the set covering problem. *Annals of Operations Research*, 98:353–371, 2000.
- [24] A. Caprara and R. Rizzi. Improving a family of approximation algorithms to edge color multigraphs. *Information Processing Letters*, 68(1):11–15, 1998.
- [25] D. G. Cattrysse and L. N. van Wassenhove. A survey of algorithms for the generalized assignment problem. *European Journal of Operational Research*, 72:167–174, 1992.

- [26] S. Ceria, P. Nobili, and A. Sassano. Set covering problem. In M. Dell’Amico, F. Maffioli, and S. Martello, editors, *Annotated Bibliographies in Combinatorial Optimization*, pages 415–428. John Wiley and Sons, 1997.
- [27] P. Coll, J. Marenco, I. M. Diaz, and P. Zabala. Facets of the graph coloring polytope. *Annals of Operations Research*, 116:79–90, 2002.
- [28] F. Comellas and J. Ozon. Graph coloring algorithms for assignment problems in radio networks. *Applications of Neural Networks to Telecommunications*, 2:49–56, 1995.
- [29] W. J. Cook, W. H. Cunningham, W. R. Pulleyblank, and A. Schrijver. *Combinatorial Optimization*. Wiley-Interscience, 1998.
- [30] E. Danna, C. L. Pape, and E. Rothberg. Exploring relaxation induced neighborhoods to improve mip solutions. Technical Report ILOG Technical Report 03-004, ILOG, June 2003.
- [31] G. Dantzig, R. Fulkerson, and S. Johnson. Solution of a large-scale traveling-salesman problem. *Journal of the Operations Research Society of America*, 1954.
- [32] G. B. Dantzig. Maximization of a linear function of variables subject to linear inequalities. In *Activity Analysis of Production and Allocation — Proceedings of a Conference*, pages 359–373. Wiley, New York, 1951.
- [33] G. B. Dantzig and P. Wolfe. Decomposition principle for linear programs. *Operations Research*, 8:101–111, 1960.
- [34] M. Dawande and J. Kalagnanam. The multiple knapsack problem with color constraints. IBM research report, IBM Research Division, T.J. Watson Research Center, Yorktown Heights, New York 10598, March 1998.
- [35] I. de Farias and G. Nemhauser. A family of inequalities for the generalized assignment polytope. *Operations Research Letters*, 29:49–51, 2001.
- [36] I. de Farias and G. Nemhauser. A polyhedral study of the cardinality constrained knapsack problem. *Lecture Notes In Computer Science: Proceedings of the 9th International IPCO Conference on Integer Programming and Combinatorial Optimization*, pages 291–303, 2002.
- [37] I. R. de Farias, E. L. Johnson, and G. L. Nemhauser. A generalized assignment problem with special ordered sets: A polyhedral approach. *Mathematical Programming*, 89:187–203, 2000.
- [38] I. M. Díaz and P. Zabala. A polyhedral approach for graph coloring. In J. Szwarcfiter and S. Song, editors, *Electronic Notes in Discrete Mathematics*, volume 7. Elsevier, 2001.

- [39] I. M. Diaz and P. Zabala. A branch-and-cut algorithm for graph coloring. Technical Report TR 02-001, Departamento de Computaci' on Universidad de Buenos Aires, November 2002.
- [40] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.
- [41] D. Durand, R. Jain, and D. Tseytlin. Distributed scheduling algorithms to improve the performance of parallel data transfer. Technical report, DIMACS, 1994.
- [42] D. Durand, R. Jain, and D. Tseytlin. Applying randomized edge coloring algorithms to distributed communication. In *Proceedings of the seventh annual ACM symposium on Parallel algorithms and architectures*, pages 262–274. ACM Press New York, NY, USA, 1995.
- [43] D. Durand, R. Jain, and D. Tseytlin. Parallel I/O scheduling using randomized, distributed edge coloring algorithms. *Journal of Parallel and Distributed Computing*, 63(6):611–618, 2003.
- [44] J. Edmonds. Maximum matching and a polyhedron with 0,1 vertices. *Journal of Research National Bureau of Standards Section B69*, pages 125–130, 1965.
- [45] L. F. Escudero. S3 sets, an extension of the beale-tomlin special ordered sets. *Mathematical Programming, Series B*, 42:113–123, 1988.
- [46] K. P. Eswaran and R. E. Tarjan. Augmentation problems. *SIAM Journal on Computing*, 5(4):653–665, December 1976.
- [47] C. Ferreira, A. Martin, C. de Souza, R. Weismantel, and L. A. Wolsey. Formulations and valid inequalities for the the node capacitated graph partitioning problem. *Mathematical Programming*, 74:247–267, 1996.
- [48] C. Ferreira, A. Martin, C. de Souza, R. Weismantel, and L. A. Wolsey. The node capacitated graph partitioning problem: A computational study. *Mathematical Programming, series B*, 81:229–256, 1998.
- [49] C. E. Ferreira, A. Martin, and R. Weismantel. A cutting plane based algorithm for the multiple knapsack problem, 1993.
- [50] C. E. Ferreira, A. Martin, and R. Weismantel. Facets for the multiple knapsack problem, 1993.
- [51] C. E. Ferreira, A. Martin, and R. Weismantel. Solving multiple knapsack problems by cutting planes. *SIAM J. Optimization*, 6:858–877, 1996.
- [52] J. H. Ferziger and M. Peric. *Computational Methods for Fluid Dynamics*. Springer, Berlin, 3 edition, November 2001.

- [53] C. M. Fiduccia and R. M. Mattheyses. A linear-time heuristic for improving network partitions. *Annual ACM IEEE Design Automation Conference, Proceedings of the 19th conference on Design automation*, pages 175–181, 1982.
- [54] S. Fiorini and R. J. Wilson. *Edge-colourings of graphs*. Pitman, 1977.
- [55] M. Fischetti, F. Glover, and A. Lodi. The feasibility pump. *Mathematical Programming*, 104:91–104, 2005.
- [56] M. Fischetti and A. Lodi. Local branching. *Mathematical Programming*, 98:23–47, 2003.
- [57] A. Frank. Connectivity augmentation problems in network design. In J. R. Birge and K. G. Murty, editors, *Mathematical Programming: State of the Art 1994*, pages 34–63. The University of Michigan, 1994.
- [58] G. N. Fredrickson and J. Jájá. Approximation algorithms for several graph augmentation problems. *SIAM Journal on Computing*, 10(2):270–283, May 1981.
- [59] A. Fügenschuh and A. Martin. Computational integer programming and cutting planes. In K. Aardal, G. Nemhauser, and R. Weismantel, editors, *Discrete Optimization*, volume 12 of *Handbooks in Operations Research and Management Science*. Elsevier, December 2005.
- [60] H. Gabow. An efficient implementation of edmond’s algorithm for maximum matching on graphs. *Journal of the ACM*, 23:221–234, 1976.
- [61] Z. Galil and D. Leven. NP completeness of finding the chromatic index of regular graphs. *Journal of Algorithms*, 4:35–44, 1983.
- [62] P. Galinier and J.-K. Hao. Hybrid evolutionary algorithms for graph coloring. *Journal of Combinatorial Optimization*, 3:379–397, 1999.
- [63] M. Garey, D. Johnson, and L. Stockmeyer. Some simplified NP-complete problems. *Theoretical Computer Science*, 1:237–267, 1976.
- [64] M. R. Garey and D. S. Johnson. *Computers and Intractability—A Guide to the Theory of NP-Completeness*. W. H. Freeman And Company, New York, 1979.
- [65] F. Glover. Tabu Search: A Tutorial. *Interfaces (Special Issue on the Practice of Mathematical Programming)*, 20(1):74 – 94, July 1990.
- [66] M. K. Goldberg. On multigraphs with almost maximal chromatic class. *Diskret. Analiz.*, 23:3–7, 1973.
- [67] M. K. Goldberg. Edge-coloring of multigraphs: Recoloring technique. *Journal of Graph Theory*, 8:123–137, 1984.

- [68] L. Golubchik, S. Khuller, Y.-A. Kim, S. Shargorodskaya, and Y.-C. J. Wan. Data migration on parallel disks. In *ESA*, pages 689–701, 2004.
- [69] R. E. Gomory. Outline of an algorithm for integer solutions to linear programs. *Bulletin of the American Mathematical Society*, 64:275–278, 1958.
- [70] R. E. Gomory. Solving linear programming problems in integers. In R. Bellman and J. M. Hall, editors, *Combinatorial Analysis*, pages 211–215. American Mathematical Society, Providence, Rhode Island, 1960.
- [71] R. E. Gomory. An algorithm for integer solutions to linear programs. In R. L. Graves and P. Wolfe, editors, *Recent Advances in Mathematical Programming*, pages 269–302. McGraw-Hill, Inc, 1963.
- [72] E. S. Gottlieb and M. R. Rao. $(1, k)$ -configuration facets for the generalized assignment problem. *Mathematical Programming*, 46:53–60, 1990.
- [73] E. S. Gottlieb and M. R. Rao. The generalized assignment problem: Valid inequalities and facets. *Mathematical Programming*, 16:31–52, 1990.
- [74] A. Göttmann. Kantenfärbung in Multigraphen. Master’s thesis, Technische Universität Darmstadt, April 2005.
- [75] D. A. Grable and A. Panconesi. Nearly optimal distributed edge colouring in $\mathcal{O}(\log \log n)$ rounds. In *Proceedings of the eighth annual ACM-SIAM symposium on Discrete algorithms*, pages 278–285, 1997.
- [76] M. Grötschel, L. Lovász, and A. Schrijver. *Geometric Algorithms and Combinatorial Optimization*. Springer, Berlin, 1988.
- [77] D. Gusfield, D. Naor, and C. Martel. A fast approximation algorithm for optimally increasing the edge-connectivity. *SIAM Journal on Computing*, 26(4):1139–1165, August 1997.
- [78] S. L. Hakimi and E. F. Schmeichel. Improved bounds for the chromatic index of graphs and multigraphs. *Journal of Graph Theory*, 32(4):311–326, 1999.
- [79] J. Hall, J. D. Hartline, A. R. Karlin, J. Saia, and J. Wilkes. On algorithms for efficient data migration. In *Symposium on Discrete Algorithms*, pages 620–629, 2001.
- [80] B. Hendrickson. Graph partitioning and parallel solvers: Has the emperor no clothes? (extended abstract). In *Lecture Notes in Computer Science, 1457*, pages 218–225. Springer-Verlag, 1998.
- [81] B. Hendrickson. Load balancing fictions, falsehoods and fallacies. *Appl. Math. Modelling*, 25:99–108, 2000.

- [82] B. Hendrickson and K. Devine. Dynamic load balancing in computational mechanics. *Comp. Meth. Applied Mechanics & Engineering*, 184(2–4):485–500, 2000.
- [83] B. Hendrickson and T. G. Kolda. Graph partitioning models for parallel computing. *Parallel Computing*, 26:1519–1534, 2000.
- [84] B. Hendrickson and R. Leland. A multilevel algorithm for partitioning graphs. *Proc. Supercomputing '95*, 1993.
- [85] B. Hendrickson and R. Leland. An empirical study of static load balancing algorithms. *Proc. Scalable High-Perf. Comput. Conf. IEEE*, pages 682–685, 1994.
- [86] B. Hendrickson and R. Leland. An improved spectral graph partitioning algorithm for mapping parallel computations. *SIAM Journal on Scientific Computing*, 16(2):452–469, 1995.
- [87] B. Hendrickson, R. Leland, and R. V. Driessche. Skewed graph partitioning. *Proc. Eighth SIAM Conf. Parallel Processing for Scientific Computing*, 1997.
- [88] B. Hendrickson and A. Pinar. Communication support for adaptive computation. *Proc. 10th SIAM Conf. Parallel Processing for Scientific Computing*, 2001.
- [89] B. Hendrickson and A. Pinar. Partitioning for complex objectives. *Proc. Irregular '01*, 2001.
- [90] A. Hertz and D. de Werra. Using tabu search techniques for graph colouring. *Computing*, 39:345–351, 1987.
- [91] M. Hilgemeier, N. Drechsler, and R. Drechsler. Fast heuristics for the edge coloring of large graphs. *Euromicro Symposium on Digital System Design (DSD'2003)*, pages 230–237, 2003.
- [92] D. S. Hochbaum, T. Nishizeki, and D. B. Shmoys. A better than “best possible” algorithm to edge color multigraphs. *Journal of Algorithms*, 7:79–104, 1986.
- [93] S. Holm and M. M. Sørensen. The optimal graph partitioning problem. *OR Spektrum*, 15:1–8, 1993.
- [94] I. J. Holyer. The NP-completeness of edge coloring. *SIAM Journal on Computing*, 10:718–720, 1981.
- [95] E. Johnson, A. Mehrotra, and G. Nemhauser. Min-cut clustering. *Mathematical Programming*, 62:133–152, 1993.
- [96] R. Jothi, B. Raghavachari, and S. Varadarajan. A $5/4$ -approximation algorithm for minimum 2-edge connectivity. In *Proceedings of the 14th ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 725–734. ACM press, Baltimore, MD, 2003.

- [97] J. Kahn. Asymptotics of the chromatic index for multigraphs. *Journal of Combinatorial Theorie*, B(68):233–254, 1996.
- [98] V. Kaibel, M. Peinhardt, and M. E. Pfetsch. Orbitopal fixing. *Optimization Online*, 2006.
- [99] G. Karypis. Multi-constraint mesh partitioning for contact/impact computations. Technical Report 03-022, Department of Computer Science & Engineering/Army HPC Research Center, University of Minnesota, 2003.
- [100] G. Karypis and V. Kumar. A fast and high quality multilevel scheme for partitioning irregular graphs. *SIAM Journal on Scientific Computing*, 1998.
- [101] G. Karypis and V. Kumar. Multilevel algorithms for multi-constraint graph partitioning. Technical Report TR 98-019, Department of Computer Science, University of Minnesota, 1998.
- [102] G. Karypis, V. Kumar, and K. Schloegel. A new algorithm for multi-objective graph partitioning. Technical Report 99-003, University of Minnesota, Department of Computer Science and Army HPC Center, 1999.
- [103] G. Karypis, V. Kumar, and K. Schloegel. Parallel multilevel algorithms for multi-constraint graph partitioning. Technical Report 99-031, University of Minnesota, Department of Computer Science and Army HPC Center, 1999.
- [104] G. Karypis, V. Kumar, and K. Schloegel. A unified algorithm for load-balancing adaptive scientific simulations. In *Proceedings of the 2000 ACM/IEEE conference on Supercomputing (CDROM)*, 2000.
- [105] G. Karypis, V. Kumar, and K. Schloegel. Graph partitioning for high-performance scientific simulations. In J. Dongara, I. Foster, G. Fox, W. Gropp, K. Kennedy, L. Torczon, and A. White, editors, *CRPC Parallel Computing Handbook*, chapter 18, pages 491–541. Morgan Kaufmann, San Francisco, CA, 2002.
- [106] A. V. Karzanov. Determining the maximal flow in a network by the method of preflows. *Soviet Mathematics Doklady*, 15:434–437, 1974.
- [107] A. V. Karzanov and E. A. Timofeev. Efficient algorithms for finding all minimum edge cuts of a nonoriented graph. *Cybernetics*, 22:156–162, 1986.
- [108] L. G. Khachiyan. A polynomial algorithm in linear programming. *Soviet Mathematics Doklady*, 20:191–194, 1979.
- [109] L. G. Khachiyan. Polynomial algorithms in linear programming. *USSR Computational Mathematics and Mathematical Physics*, 20:53–72, 1980.

- [110] S. Khuller. Approximation algorithms for finding highly connected subgraphs. Technical Report CS-TR-3398, University of Maryland Institute for Advanced Computer Studies, 1995.
- [111] S. Khuller, Y.-H. Kim, and Y.-C. J. Wan. Algorithms for data migration with cloning. *SIAM Journal on Computing*, 33(2):448–461, 2004.
- [112] S. Khuller and B. Raghavachari. Improved approximation algorithms for uniform connectivity problems. *Journal of Algorithms*, 21(2):434–450, 1996.
- [113] S. Khuller and U. Vishkin. Biconnectivity approximations and graph carvings. *Journal of the ACM*, 41(2):214–235, 1994.
- [114] A. D. King, B. A. Reed, and A. R. Vetta. An upper bound for the chromatic number of line graphs. In S. Felsner, editor, *2005 European Conference on Combinatorics, Graph Theory and Applications (EuroComb '05)*, volume AE of *DMTCS Proceedings*, pages 151–156. Discrete Mathematics and Theoretical Computer Science, 2005.
- [115] S. Kirkpatrick, C. Gelatt, and M. Vecchi. Optimization by simulated annealing (1983). *Science*, 220:671 – 680, 1983.
- [116] W. Klotz. Graph coloring algorithms. *Mathematik-Bericht (TU Clausthal)*, 2:1–9, 2002.
- [117] C. E. Lemke. The dual method of solving the linear programming problem. *Naval Research Logistics Quarterly*, 1:36–47, 1954.
- [118] M. Leyzorek, R. S. Gray, A. A. Johnson, W. C. Ladew, S. R. Meaker, R. M. Petry, and R. N. Seitz. Investigation of model techniques — first annual report — 6 june 1956 – 1 july 1957 — a study of model techniques for communication systems. Technical report, Case Institute of Technology, Cleveland, Ohio, 1957.
- [119] V. M. Malhotra, M. P. Kumar, and S. N. Maheshwari. An $\mathcal{O}(|V|^3)$ algorithm for finding maximum flows in networks. *Information Processing Letters*, 7:277–278, 1978.
- [120] M. V. Marathe, A. Panconesi, and L. D. Risinger. An experimental study of a simple, distributed edge coloring algorithm. *ACM Symposium on Parallel Algorithms and Architectures, Proceedings of the twelfth annual ACM symposium on Parallel algorithms and architectures*, pages 166–175, 2000.
- [121] H. Marchand and L. A. Wolsey. Knapsack problem with a single continuous variable, 1997.
- [122] H. Marchand and L. A. Wolsey. The 0-1 knapsack problem with a single continuous variable. *Mathematical Programming*, 85:15–33, 1999.

- [123] F. Margot. Pruning by isomorphism in branch-and-cut. Technical Report Research Report 2001-08, Department of Mathematics, University of Kentucky, July 2001.
- [124] F. Margot. Exploiting orbits in symmetric ilp. *Mathematical Programming*, 98:3–21, 2003.
- [125] S. Martello and P. Toth. *Knapsack Problems: Algorithms and Computer Implementations*. John Wiley and Sons, Ltd., New York, 1990.
- [126] A. Martin. General mixed integer programming: Computational issues for branch-and-cut algorithms. In D. Naddef and M. Juenger, editors, *Computational Combinatorial Optimization*. Springer, Berlin, 2001.
- [127] A. Martin, M. Möller, and S. Moritz. Mixed integer models for the stationary case of gas network optimization. *Mathematical Programming*, 105:563 – 582, 2006.
- [128] A. Mehrotra and M. Trick. A column generation approach for graph coloring. *INFORMS Journal on Computing*, 8(4):344–353, 1996.
- [129] E. F. Moore. The shortest path through a maze. In H. Aiken, editor, *Proceedings of an International Symposium on the Theory of Switching*, pages 285–292. Harvard University Press, Cambridge, Massachusetts, 1959.
- [130] H. Nagamochi and T. Watanabe. Computing k -edge-connected components of a multigraph. *IEICE Trans. Fundamentals*, E76-A(4):513–517, April 1993.
- [131] G. Nemhauser and S. Park. A polyhedral approach to edge coloring. *Operations Research Letters*, 10:315–322, 1991.
- [132] G. Nemhauser and L. Wolsey. *Integer Combinatorial Optimization*. Wiley, New York, 1988.
- [133] T. Nishizeki and K. Kashiwagi. On the 1.1-edge-coloring of multigraphs. *SIAM Journal on Discrete Mathematics*, 3(3):391–410, 1990.
- [134] T. Nishizeki and M. Sato. An algorithm for edge-coloring multigraphs. *Trans. Inst. of Electronics and Communication Eng.*, J67-D(4):466–471, 1984.
- [135] T. Nishizeki and X. Zhou. Edge-coloring and f -coloring for various classes of graphs. *J. Graph Algorithms Appl.*, 3(1):1–18, 1999.
- [136] O. Ore. The four-color problem. *Academic Press*, 1967.
- [137] M. Plantholt. A sublinear bound on the chromatic index of multigraphs. *Discrete Mathematics*, 202:201–213, 1999.
- [138] M. J. Plantholt. An order-based upper bound on the chromatic index of a multigraph. *J. Combin. Inform. System Sci.*, 16:271–280, 1991.

- [139] M. J. Plantholt. An improved order-based upper bound on the chromatic index of a multigraph. *Congressus Numerantium*, 103:129–141, 1994.
- [140] M. J. Plantholt and L. Eggan. The chromatic index of nearly bipartite multigraphs. *J. Combin. Theory Ser. B*, 40:71–80, 1986.
- [141] M. J. Plantholt and S. K. Tipnis. Regular multigraphs of high degree are 1-factorizable. *Journal of the London Mathematical Society*, 44(2):393–400, 1991.
- [142] M. J. Plantholt and S. K. Tipnis. The chromatic index of multigraphs of order at most 10. *Discrete Mathematics*, 177:185–193, 1997.
- [143] M. J. Plantholt and S. K. Tipnis. All regular multigraphs of even order and high degree are 1-factorable. *The Electronic Journal of Combinatorics*, 8, 2001.
- [144] A. Pothen. Graph partitioning algorithms with applications to scientific computing. In D. Keyes, A. Sameh, and V. Venkatakrisnan, editors, *Parallel Numerical Algorithms*. Kluwer Academic Press, 1996.
- [145] C. Rhee. On the chromatic index of graphs with $2m + 1$ vertices and $2m^2$ edges. *Information Processing Letters*, 66:115–118, 1998.
- [146] G. T. Ross and R. M. Soland. A branch-and-bound algorithm for the generalized assignment problem. *Mathematical Programming*, 8:91–103, 1975.
- [147] J. Rothe. Heuristics versus completeness for graph coloring. *Chicago Journal of Theoretical Computer Science*, 2000.
- [148] J. Saia. *Algorithms for managing data in distributed systems*. PhD thesis, University of Washington, 2002.
- [149] P. Sanders and D. Steurer. An asymptotic approximation scheme for multigraph edge coloring. In *Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms*, pages 897–906, 2005.
- [150] M. W. P. Savelsbergh. A branch-and-price algorithm for the generalized assignment problem. *Operations Research*, 45:831–841, 1997.
- [151] A. Schrijver. *Theory of Linear and Integer Programming*. Wiley, Chichester, 1986.
- [152] A. Schrijver. *Combinatorial Optimization—Polyhedra and Efficiency*. Springer, 2003.
- [153] A. S. Schulz and M. Skutella. Scheduling unrelated machines by randomized rounding. *SIAM Journal of Discrete Mathematics*, 15(4):450–469, 2002.
- [154] P. D. Seymour. Some unsolved problems on one-factorizations of graphs. In J. A. Bondy and U. Murty, editors, *Graph Theory and Related Topics*, pages 367–368. Academic Press, 1979.

- [155] C. E. Shannon. A theorem on coloring the lines of a network. *J. Math. Phys.*, 28:148–151, 1949.
- [156] H. D. Sherali and J. C. Smith. Improving discrete model representations via symmetry considerations. *Management Science*, 47:1396–1407, 2001.
- [157] W. Spears, K. DeJong, T. Baeck, D. Fogel, and H. de Garis. An overview of evolutionary computation. In *Proceedings of the European Conference on Machine Learning (ECML-93)*, pages 442 – 459, 1993.
- [158] I. Stoykov. An approximation algorithm for edge-coloring of multigraphs. Master’s thesis, Technische Universität Darmstadt, April 2005.
- [159] R. E. Tarjan. A simple version of Karzanov’s blocking flow algorithm. *Operations Research Letters*, 2:265–268, 1984.
- [160] V. G. Vizing. The chromatic class of a multigraph. *Cybernetics*, 3:32–41, 1965.
- [161] E. Zemel. Easily computable facets of the knapsack polytope. *Mathematics of Operations Research*, 14(4):760–765, 1989.
- [162] X. Zhou. *Efficient Algorithms for Edge-Coloring Graphs*. PhD thesis, Tohoku University, Japan, 1995.
- [163] A. Zhu. A uniform framework for approximating weighted connectivity problems. B.Sc. thesis, University of Maryland, MD, May 1999.
- [164] W. Zuo. Introduction of computational fluid dynamics. 2005.

Akademischer Werdegang

Daniel Junglas

geboren am 6. August 1978 in Freiburg im Breisgau

1985 – 1989	Katholische Martinus Grundschule in Mainz
1989 – 1998	Rabanus-Maurus-Gymnasium Mainz
1998	Abitur
1998 – 2003	Studium Diplom Mathematik mit Nebenfach Informatik im Studiengang „Mathematics with Computer Science“ an der Technischen Universität Darmstadt
März 2003	Diplom in Mathematik
2003 – 2006	Stipendiat im Graduiertenkolleg 853, „Modellierung, Simulation und Optimierung von Ingenieursanwendun- gen“