Contributions to graph partitioning problems under resource constraints

Dang Phuong Nguyen

THÈSE DE DOCTORAT DE
l'UNIVERSITÉ PIERRE ET MARIE CURIE

Spécialité

Informatique

École doctorale Informatique, Télécommunications et Électronique (Paris)

Présentée par

Dang Phuong NGUYEN

Pour obtenir le grade de

DOCTEUR de l'UNIVERSITÉ PIERRE ET MARIE CURIE

Sujet de la thèse :

Contributions à des problèmes de partitionnement de graphe sous contraintes de ressources

soutenue le dd mm 2016

devant le jury composé de :

M. Ridha Mahjoub	Rapporteur
Mme. Hoai An Le Thi	Rapporteur
M. Christophe Picouleau	Rapporteur
M. Patrice Perny	Examinateur
M. Michel Minoux	Diecteur de thèse
M. Viet Hung Nguyen	Co-Encadrant
M. Renaud Sirdey	Co-Encadrant
M. Thanh Hai Nguyen	Tuteur

L'école doctorale informatique, télécommunications et électronique (EDITE)

Contributions to graph partitioning problems under resource constraints

Document proposed for thesis defence

Dang Phuong Nguyen

Committee members :

Ridha Mahjoub[1]	Reviewer
Hoai An Le Thi[2]	Reviewer
Christophe Picouleau[3]	Reviewer
Patrice Perny[4]	Examiner
Michel Minoux[5]	Advisor
Viet Hung Nguyen[6]	Co-advisor
Renaud Sirdey[7]	Co-advisor
Thanh Hai Nguyen[8]	Tutor

[1]R. Mahjoub, Professeur, Université Paris Dauphine.
[2]H.A. Le Thi, Professeur, Université de Lorraine
[3]Ch. Picouleau, Professeur, CNAM
[4]P. Perny, Professeur, UPMC
[5]M. Minoux, Professeur Emérite, UPMC
[6]V.H. Nguyen, Maître de Conférences, UPMC
[7]R. Sirdey, HDR, CEA/LIST
[8]T.H. Nguyen, CEA/LIST

Résumé

Le problème de partitionnement de graphe est un problème fondamental en optimisation combinatoire. Le problème revient à décomposer l'ensemble des noeuds d'un graphe en plusieurs sous-ensembles disjoints de noeuds (ou clusters), de sorte que la somme des poids des arêtes dont les extrémités se trouvent dans différents clusters est réduite au minimum. Dans cette thèse, nous étudions le problème de partitionnement de graphes avec des poids (non négatifs) sur les noeuds et un ensemble de contraintes supplémentaires sur les clusters (GPP-SC) précisant que la capacité totale (par exemple, le poids total des noeuds dans un cluster, la capacité totale sur les arêtes ayant au moins une extrémité dans un cluster) de chaque groupe ne doit pas dépasser une limite prédéfinie (appelée limite de capacité). Ceci diffère des variantes du problème de partitionnement de graphe le plus souvent abordées dans la littérature en ce que:

- Le nombre de clusters n'est pas imposé (et fait parti de la solution),
- Les poids des noeuds ne sont pas homogènes.

Le sujet de ce travail est motivé par le problème de la répartition des tâches dans les structures multicoeurs. Le but est de trouver un placement admissible de toutes les tâches sur les processeurs tout en respectant leur capacité de calcul et de minimiser le volume total de la communication inter-processeur. Ce problème peut être formulé comme un problème de partitionnement de graphe sous contraintes de type sac-à-dos (GPKC) sur des graphes peu denses, un cas particulier de GPP-SC. En outre, dans de telles applications, le cas des incertitudes sur les poids des noeuds (poids qui correspondent par exemple à la durée des tâches) doit être pris en compte.

La première contribution de ce travail est de prendre en compte le caractère peu dense des graphes $G = (V, E)$ typiques rencontrés dans nos applications. La plupart des modèles de programmation mathématique existants pour le partitionnement de graphe utilisent $O(|V|^3)$ contraintes métriques pour modéliser les partitions de noeuds et donc supposent implicitement que G est un graphe complet. L'utilisation de ces contraintes métriques dans le cas où G n'est pas complet nécessite l'ajout de contraintes qui peuvent augmenter considérablement la taille du programme. Notre résultat montre que, pour le cas où G est un graphe peu dense, nous pouvons réduire le nombre de contraintes métriques à $O(|V||E|)$ [1], [4].

La deuxième contribution de ce travail est de calculer des bornes inférieures pour les graphes de plus grande taille. Nous proposons un nouveau modèle de programmation pour le problème de partitionnement de graphe qui fait usage de $O(m)$ variables seulement. Le modèle contient les inégalités de cycle et les inégalités

liées aux chemins dans le graphe pour formuler les partitions admisibles. Etant donné qu'il existe un nombre exponentiel de contraintes, la résolution du modèle a besoin d'un procédé de séparation pour accélérer les temps de calcul. Nous proposons une telle méthode de séparation qui utilise un algorithme de plus court chemin de toutes les paires de noeuds, et est donc en temps polynomial. Les résultats des calculs montrent que notre nouveau modèle et la méthode donne des bornes inférieures serrées pour des graphes d'assez grande taille allant jusqu'à quelques milliers noeuds.

La troisième contribution de ce travail est de prendre en compte l'incertitude sur le poids des noeuds. Nous proposons de modéliser les incertitudes en utilisant des contraintes en probabilité. Sous l'hypothèse que les poids de noeuds suivent une distribution gaussienne multivariée, les contraintes de type sac-à-dos stochastiques peuvent être reformulés comme des contraintes de cône du second ordre. Ces contraintes de cône du second ordre peuvent à leur tour être exprimés sous la forme de contraintes quadratiques et nous montrons comment gérer celle-ci par des méthodes de linéarisation. Nous fournissons des résultats de calcul montrant que la formulation quadratique peut être plus efficacement résolue (en utilisant un algorithme de branch-and-bound) que la formulation de cône du second ordre [2], [5].

Abstract

The graph partitioning problem is a fundamental problem in combinatorial optimization. The problem refers to partitioning the set of nodes of an edge weighted graph in several disjoint node subsets (or clusters), so that the sum of the weights of the edges whose end-nodes are in different clusters is minimized. In this thesis, we study the graph partitioning problem on graph with (non negative) node weights with additional set constraints on the clusters (GPP-SC) specifying that the total capacity (e.g. the total node weight, the total capacity over the edges having at least one end-node in the cluster) of each cluster should not exceed a specified limit (called capacity limit). This differs from the variants of graph partitioning problem most commonly addressed in the literature in that:

- The number of clusters is not imposed (and is part of the solution),
- The weights of the nodes are not homogeneous.

The subject of the present work is motivated by the task allocation problem in multicore structures. The goal is to find a feasible placement of all tasks to processors while respecting their computing capacity and minimizing the total volume of interprocessor communication. This problem can be formulated as a graph partitioning problem under knapsack constraints (GPKC) on sparse graphs, a special case of GPP-SC. Moreover, in such applications, the case of uncertain node weights (weights correspond for example to task durations) has to be taken into account.

The first contribution of the present work is to take into account the sparsity character of the graph $G = (V, E)$. Most existing mathematical programming models for the graph partitioning problem use $O(|V|^3)$ metric constraints to model the partition of nodes and thus implicitly assume that G is a complete graph. Using these metric constraints in the case where G is not complete requires adding edges and constraints which may greatly increase the size of the program. Our result shows that for the case where G is a sparse graph, we can reduce the number of metric constraints to $O(|V||E|)$ [1],[4].

The second contribution of present work is to compute lower bounds for large size graphs. We propose a new programming model for the graph partitioning problem that make use of only $O(m)$ variables. The model contains cycle inequalities and all inequalities related to the paths in the graph to formulate the feasible partitions. Since there are an exponential number of constraints, solving the model needs a separation method to speed up the computation times. We propose such a separation method that use an all pair shortest path algorithm thus is polynomial time. Computational results show that our new model and method can give tight lower bounds for large size graphs of thousands nodes.

The third contribution of the present work is to take into account the uncertainty over node weights. We propose to model uncertainties using chance constraints. Under the assumption that the node weights follow a multivariate Gaussian distribution, the stochastic knapsack constraints can be modeled as second order cone constraints. These second order cone constraints can in turn be reformulated as quadratic 0-1 constraints and we show how to handle the latter via linearization methods. We provide computational results showing that the quadratic 0-1 reformulation can be more efficiently solved (using branch-and-bound) than the second order cone formulation [2],[5].

Publications

1. D.P. Nguyen, T.H. Nguyen, P. Dubrulle. Hierarchical synchronization between processes in a high-performance execution support of dataflow process networks on many-core architectures. *8th International Conference on Complex, Intelligent and Software Intensive Systems (CISIS)*, Birmingham, UK, July 2014, p:439-444, doi: 10.1109/CISIS.2014.62.

2. V.H. Nguyen, M.Minoux, D.P. Nguyen. Improved compact formulations for metric and cut polyhedra. *Electronic Notes in Discrete Mathematics*, Volume 52, June 2016, Pages 125-132, ISSN 1571-0653, http://dx.doi.org/10.1016/j.endm.2016.03.017.

3. D.P. Nguyen, M. Minoux, V.H. Nguyen, T.H. Nguyen, and R. Sirdey. Stochastic graph partitioning: Quadratic versus SOCP formulations. *Optimization Letters*, p.1 - p.14, 2015, doi: 10.1007/s11590-015-0953-9

4. D.P. Nguyen, M. Minoux, V.H. Nguyen, T.H. Nguyen, and R. Sirdey. Improved compact formulations for a wide class of graph partitioning problems in sparse graphs. *Discrete Optimization*, Available online 30 May 2016, ISSN 1572-5286, http://dx.doi.org/10.1016/j.disopt.2016.05.003.

5. V.H. Nguyen, M.Minoux, D.P. Nguyen. Improved compact formulations for metric and cut polyhedra. *Submitted to Networks*, 2015.

Contents

Chapter 1

Introduction

Recently, with the arising of sites such as MySpace, Friendster, Orkut, Twitter, Facebook, etc. social networks have reached major popularity and another reason of social networks popularity is that they are easy to use. These networks make people of all over the world able to communicate with each other. One of the common features of these networks is called community structure which represents connected groups (clusters) that there should be many edges within each group and few between the groups. Resulted groups are fraction of individuals that have similar features or connected via relations. Groups in social networks are corresponding with social relations and are used for understanding the data structure such as organization structures, scientific collaboration and relations in telecommunication networks. Communities are useful in many applications. Web clients clustering (community detection) which have same or similar interests or are near together via location can improve the World Wide Web services performance. One of the community detection benefits is to provide better recommendation systems for efficient customer's guidance and increasing the business opportunities via representing the lists of retailer items which produces the clusters of customers with similar interests. The goal of graphs community detection is the identification of modules and their hierarchical structure by using the information which is encoded in graph topology.

In computer science we face a similar dilemma. Graphs are frequently used by computer scientists as abstractions when modeling an application problem. Cutting a graph into smaller pieces is one of the fundamental algorithmic operations. Even if the final application concerns a different problem (such as traversal, finding paths, trees, and flows), partitioning large graphs is often an important subproblem for complexity reduction or parallelization. With the advent of ever larger instances in applications such as scientific simulation, social networks, or road networks, graph partitioning therefore becomes more and more important, multifaceted, and challenging.

1.1 Research context

1.1.1 Graph partitioning problem under knapsack constraints (GPKC)

Until the 2000 decade, Moore's Law was a good indicator in the prediction of the performance of new microprocessors. For each new generation of hardware, new optimization challenges also appear for performance enhancement. With the end of the frequency version of Moore's law, new clusterized embedded parallel microprocessor architectures, known as manycores, have been developed. New challenges consist in applying combinatorial optimization techniques to problems in software compilation of applications, like in signal processing, image processing or multimedia, on these massively parallel architectures. Such applications can be represented under the static dataflow parallel programming model, in which one expresses computation-intensive applications as networks of concurrent processes (also called agents or actors) interacting through (and only through) unidirectional FIFO channels. They provide strong guarantee of determinism, absence of deadlocks and execution in bounded memory. The main difficulty in the development of this kind of applications for those architectures consists in handling resource limitations, a high exploitation difficulty of massive parallelism and global efficiency. On top of more traditional compilation aspects, compiling a dataflow program in order to achieve a high level of dependability and performance on such complex processor architectures involves solving a number of difficult, large-size discrete optimization problems among which graph partitioning, quadratic assignment and (constrained) multi-flow problems are worth mentioning [2].

Our applicative work focuses on the task allocation problem in multi-core structures. The goal is to find a feasible placement of all tasks to processors while respecting their computing capacity and minimizing the total volume of inter-processor communication. In this dissertation, the task allocation is processed under assignment and capacity constraints (of knapsack type). A task cannot be placed on several processors and the total amount of all tasks weights must not exceed the capacity of the processor.

1.1.2 Graph partitioning under capacity constraints (GPCC)

For various technological reasons, network operators often want to partition the node set V into clusters on which a certain network topology is imposed. For instance, in SONET/SDH optical networks, a common requirement is that every cluster is connected by a local network forming a cycle. Local networks are then interconnected by a secondary federal network which has one access node in each local network. Access nodes carry all the traffic internal to their local network and all the traffic exiting it but have a limited capacity. If we consider the traffic demand $t_{(u,v)}$ as the capacity of the edge (u, v), then the capacity of a local network (cluster) with node set $U \subset V$ follows our definition of capacity. As the topology and the capacity of local networks are imposed, the cost of these networks is almost fixed (except the cost of physical cables for building them) once the partition of V is

determined. Thus, the objective of the problem could be focused on minimizing either the number of local networks (clusters) or the cost of the federal network. For the latter, an objective function often used it to minimize the total length of the edges in the interconnection with lengths given by the product of the traffic and the distance between nodes.

The SONET/SDH network design problem minimizing the number of local networks has been introduced in 2003 by Goldschmidt et al. [32] under the name SRAP problem. Bonami et al. [11] modeled this problem as a variant of the graph partitioning problem that they call graph partitioning under capacity constraints (GPCC) where the constraints on the weights of the clusters are replaced with constraints related to the edges incident to the nodes of each cluster. Suppose that, each edge $e \in E$ is assigned a capacity $t_e \in \mathbb{Z}_+$. For any subset $U \subseteq V$, we define the capacity of U as the sum of the capacities of the edges incident to at least one node of U , i.e. the edges in $E(U) \cup \delta(U)$ where $E(U)$ is the set of the edges with both end nodes in U and $\delta(U)$ is the set of the edges with exactly one end in U. The capacity constraint is to bound the capacity of each cluster by a given constant T. The objective function we consider is to minimize the total length of the edges in the interconnection (with weights given by the lengths l_e) between the clusters.

1.2 A more general model: Graph partitioning problem under set constraints (GPP-SC)

We now introduce a more general model denoted GPP-SC (where SC stands for "set constraints") which encompasses GPKC and GPCC as special cases. GPP-SC can be mathematically defined as follows. Given an undirected connected graph $G = (V, E)$ where $V = \{1, \ldots, n\}$, $|E| = m$ and a length $l_e \in \mathbb{Z}_+$ is associated with each edge $e \in E$, find a partition of V into disjoint sets (or clusters) such that:

- every cluster $C \subset V$ satisfies a constraint of the form $\mathcal{G}(y^C) \leq 0$ where y^C is the incidence vector of C in $\{0, 1\}^n$ and $\mathcal{G} : \{0, 1\}^n \to \mathbb{R}$ is a given monotone nondecreasing pseudoboolean function (Note that $\mathcal{G}$ can also be viewed as a set function: $\mathcal{P}(V) \to \mathbb{R}$ which associates the real value $\mathcal{G}(y^C)$ with each subset $C \subset V$).

- the sum of the lengths of the edges having end-nodes in different clusters is minimized.

Note that in the above definition, the number k of clusters is not imposed and is a part of the output of the optimization process. The above class of problems is fairly general.

In mathematical form, the task allocation problem can be modeled as a special case of GPP-SC where each cluster C is required to satisfy a node weight constraint of the form $\sum_{v \in C} w_v \leq W$ where w_v for all $v \in V$ are given *nonnegative* node weights and W is a given upper limit on the total node weight of the cluster. This corresponds to considering the linear constraint $\mathcal{G}(y^C) \leq 0$ where $\mathcal{G}$ is the nondecreasing

pseudoboolean function defined by

$$\sum_{v \in V} w_v y_v^C - W \leq 0$$

Note that this special case of GPP-SC is the classical version of the graph partitioning problem defined in Garey and Johnson's book [31] which is known to be NP-hard [38].

The graph partitioning problem under capacity constraints can be modeled as a special case of GPP-SC where each cluster C is subject to a constraint of the form $\mathcal{G}(y^C) \leq 0$ where $\mathcal{G}$ is a given nondecreasing quadratic pseudoboolean function $\{0,1\}^n \to \mathbb{R}$ defined as:

$$\mathcal{G}(y^C) = \sum_{(u,v) \in E} t_{uv}(y_u^C + y_v^C - y_u^C y_v^C) - T \leq 0$$

for given $t_{uv} \in \mathbb{R}_+$ for all $(u,v) \in E$ and T a given positive constant. If one considers t_{uv} as the capacity of the edge (u,v), then $\sum_{(u,v) \in E} t_{uv}(y_u^C + y_v^C - y_u^C y_v^C)$ represents the total capacity over the edges having at least one end-node in the cluster C. The constraint limits this capacity to a constant T. It has been shown in [11] that GPCC is NP-hard.

A number of new results investigated in the present work will turn out to be applicable to general GPP-SC problem.

1.3 Motivation of this thesis and overview of our contributions

Since the general topic of the present study has been introduced, let us now mention some of the research questions which drew our attention and guided our research path:

1. The graphs of the task allocation problem and of SONET/SDH network design problem are often sparse. Thus what is the benefit of taking into account the sparsity characteristics of the graph in using existing programming models for graph partitioning problem?

2. Despite our increased understanding of the graph partitioning problem over the last decades, today's techniques seem insufficient for obtaining exact solutions for large graphs (from thousands to millions of nodes). Various heuristics can give upper bounds for the problem in such graphs, but how the validity and the quality of heuristics can be examined?

3. In practice, uncertainty is encountered each day, everywhere where the task allocation problem and the SONET/SDH network design problem are not exceptions. What is the benefit of taking into account uncertainty instead of solving the deterministic version ? And, which programming models can be used to solve the stochastic version efficiently?

The objective of this thesis is to improve our understanding of the deterministic/stochastic graph partitioning problem under set constraints by answering the above questions.

Addressing the first question, most existing mathematical programming models for the graph partitioning problem use $O(n^3)$ triangle constraints (see Chapter 2) to model the partition nodes and thus implicitly assume that G is a complete graph [33]. This requires their use in the case where G is not complete adding edges and constraints which greatly increases the size of the program, it was reported in the literature that it is rather difficult in the presence of all the $O(n^3)$ triangle constraints even for small values of n (i.e. $n \leq 20$). Indeed, the number of triangle constraints becomes quickly large as n increases and the program turns out to be difficult to solve even with the most efficient programming solvers. Several authors have tried to overcome this difficulty by dualizing all or only a subset of the triangle inequalities via a Lagrangian approach [30]. But beyond that, it would be interesting to be able to reduce intrinsically the number of triangle inequalities without weakening the relaxation quality, especially for sparse graphs (i.e. when $m = O(n)$). In Chapter 3, we will show that with only $O(mn)$ triangle constraints, instead of $O(n^3)$ as classically, we can obtain an 0/1 formulation equivalent to the problem, and more importantly an equivalent relaxation formulation. Moreover, we prove that the result holds even for a generic class of graph partitioning problem with additional constraints satisfying some monotonicity property. Computational results are discussed to show the benefit of using the reduced formulation in term of computational efficiency.

To answer the second question, exact solutions of the graph partitioning problem are often obtained by considering a 0/1 programming formulation to which a branch-and-bound based approach is applied. To validate the heuristics for large graphs, the linear relaxation of the 0/1 programming formulations is usually used to obtain lower bound for the problem. Among the programming models, the Node-Node model is known to give a good quality of the bound. This model is based on the choice of decision variables x_{ij} which are equal to 1 iff node i and node j are not in the same cluster thus the result is a 0/1 linear program and makes use of $O(n^2)$ variables and $O(n^3)$ triangle inequalities. Even when applying the results in Chapter 3 that can reduce the number of triangle inequalities in the Node-Node model to $O(nm)$ without weakening the linear relaxation, the number of variables is still in order $O(n^2)$, it quickly becomes extremely large as n increases and even the linear relaxation turns out to be difficult to solve. It is therefore interesting to be able to reduce intrinsically the number of variables of the problem in preserving the quality of the linear relaxation.

In order to do so, in Chapter 4, we consider an alternative programming model representing the node partitions of a graph partitioning problem that makes use of m decision variables $(x_e)_{e \in E}$ which are equal to 1 iff the end-nodes of e are in different clusters, together with the *cycle inequalities* defined on them [5], [15]. We prove that the node partitions and thus the knapsack constraints can be defined using all pair shortest path of G respected to the weights $(x_e)_{e \in E}$. The result is a m variable programming model for GPP-SC. This model is referred to as the *cycle model* and it is shown to be equivalent to the Node-Node model. It is shown that this model

yields significant improvement in case of sparse graphs where $m \ll \frac{n(n-1)}{2}$. Note that since there is a priori no known polynomial upper bound (in terms of n and m) on the number of cycles and of paths of the graph G, the cycle model have a priori an exponential number of inequalities thus a cutting plane algorithm is needed to speed up the computations. We introduce a such algorithm which can determine both violated cycle inequalities and violated knapsack constraints in using all pair shortest path of G respected to the weights $(x_e)_{e \in E}$. Computational experiments carried out instances of GPKC, show that, when applying cutting plane algorithm, the cycle model outperform the Node-Node model in terms of computation time.

Chapter 5 addresses the third question related to uncertainty which frequently arises in applications of GPP-SC. For instance in the task allocation problem, one of the main sources of uncertainties lies in the intrinsic indeterminism of duration of tasks for computing kernels of intermediate granularity. This indeterminism is due in part to some of the characteristics of the processor architecture such as the cache memories and memory access controllers and is also inherently due to data dependent control flows (conditional branches and loops). We consider GPKC in the case the node weights are uncertain hence this stochastic version of GPKC referred to as SGPKC. Dealing with uncertainty in an optimization problem can be done in various ways e.g using robust optimization [7],[9], or various stochastic programming models [60],[56]. We have chosen to handle uncertainty via chance constrained programming [14]. Given a probability level $\varepsilon \in (0, 1)$, the knapsack constraints in (i) can be formulated as chance constraints of the form $P(\sum_{v \in V_i} w_v \leq W) \geq 1 - \varepsilon$ for $i = 1, \ldots, k$. In chapter 5 we investigate SGPKC under the assumption that the node weights w follows a multivariate Gaussian distribution. In the case of individual chance constraints and with the probability level ε less than 0.5, the chance constraints can then be reformulated as *binary second order cone programming problem* (Binary SOCP) [47].

A first possible approach is thus to apply branch-and-bound algorithm (B&B) directly on this Bi-SOCP formulation. Our computational results (see Chapter 3.2) suggest however that only moderate efficiency can be expected from this approach. There are two main reasons for this: (a) in spite of the fact that the SOCP relaxation is a convex problem, it is still more computationally demanding than solving a linear program in each node of the B&B tree; (b) SOCP solvers do not offer warm-starting capabilities comparable to the simplex algorithm to speed up the B&B computation. This is why we have explored another approach which consists in: (i) reformulating Bi-SOCP constraints as quadratic 0-1 constraints; (ii) applying a linearization technique to solve the resulting quadratically constrained 0-1 problem. Several such linearization techniques are discussed with special emphasis on "compact" linearization which only moderately increase the number of necessary additional variables.

Finally Chapter 6 gives conclusions of the present work and several possible directions for future investigations.

Chapter 2

Literature review

2.1 Fundamental concepts

This section contains a collection of the most basic concepts that are used in the following chapters. It covers some fundamental concepts of graph theory, and computational complexity theory. The reader is assumed to be familiar with linear programming (LP) and integer linear programming including the dual simplex algorithm for linear programming and the branch-and-bound algorithm for integer linear programming.

2.1.1 Graph theory

A graph $G = (V, E)$ consists of a nonempty finite set of nodes V and a set of edges $E \subseteq \{(u, v)|u, v \in V\}$. If $e = (u, v)$ is an edge of E , each of the nodes u and v is called an end-node of e. For notational convenience the braces and the comma will usually be omitted when referring to an edge. An edge $(u, v) \in E$ is said to be incident to each of the end-nodes $u, v \in V$, and u and v are said to be adjacent or they are called neighbors with respect to E. Almost all graphs to be considered are assumed to be simple graphs. A graph is said to be simple if it does not contain multiple edges and loops. That is, every edge $e \in E$ with end-nodes u and v is unique $(u, v) = e = (v, u)$ and the end-nodes are distinct $u \neq v$.

A complete graph is a simple graph where every two distinct nodes are adjacent. The complete graph on n nodes is denoted by $K_n = (V_n, E_n)$, i.e. $E_n = \{(u, v)|u, v \in V_n\}$ and $|V_n| = n$. This subscript notation applies only to K , V , and E; that is, $G_i = (U_i, F_i)$ may be any graph, and i tells nothing about the number of nodes in G_i.

A digraph (directed graph) $D = (N, A)$ consists of a nonempty finite set of nodes N and a set of arcs $A \subseteq \{(u, v)|u, v \in N\}$ whose elements are ordered pairs of nodes. An arc $(u, v) \in A$ is said to "go from" node u to node v. Digraphs are not considered in this thesis.

The following notation will be used for some special edge sets. Let $G = (V, E)$ be a simple graph. For every subset $U \subseteq V$, the set of edges in G with both end-nodes in U is denoted by $E(U)$, i.e.

$$E(U) = \{(u, v) \in E|u, v \in U\}$$

If F is a subset of edges in a graph $G = (V, E)$, the set of end-nodes of the edges in F is denoted by $V(F)$ that is,

$$V(F) = \{u, v \in V | (u, v) \in F\}$$

If $G = (V, E)$ and $H = (U, F)$ are two graphs such that $U \subseteq V$ and $F \subseteq E(U)$, then H is called a subgraph of G . If $U \subseteq V$, then $H = (U, E(U))$ is the subgraph of G induced by U . This subgraph is also denoted by $G[U]$. Similarly, every edge set $F \subseteq E$ induces a subgraph $H = (V(F), F)$ of G.

If $G_1 = (U_1, F_1)$ and $G_2 = (U_2, F_2)$ are two graphs, the union of G_1 and G_2, denoted by $G_1 \cup G_2$, is obtained in the obvious way. That is, $G_1 \cup G_2 = (U_1 \cup U_2, F_1 \cup F_2)$. Similarly, the addition of edges to a graph $G = (V, E)$ is performed in the natural way. Suppose that $F \subseteq \{(u, v) | u, v \in V\}$ and $F \cap E = \emptyset$. Then $H = (V, E \cup F)$ is the graph obtained by adding to G the edges of F. If $U \subseteq V$ is a subset of nodes in G, $G - U$ denotes the subgraph obtained by deleting from G all nodes of U and all edges incident to the nodes in U , i.e. $G - U = (V \setminus U, E(V \setminus U))$.

A walk of length p in a graph is a sequence of p edges $e_1, \ldots, e_p$, where every edge is of the form $e_i = v_{i-1}v_i,\ i = 1, \ldots, p$. Let $v_0v_1, \ldots, v_{p-1}v_p$ be a walk of length p. Then the edge set $P = \{v_0v_1, \ldots, v_{p-1}v_p\}$ is called a path if all nodes $v_0, \ldots, v_p$ are distinct. If, on the other hand, the nodes $v_1, \ldots, v_p$ are all distinct ($p \geq 3$) and $v_0 = v_p$ then the edge set $C = \{v_0v_1, \ldots, v_{p-1}v_p, \}$ is called a cycle. The path P and the cycle C both have length p , and the path P is said to connect nodes v_0 and v_p. Note that all cycles are of length 3 or greater, while a path may be a single edge. If a cycle (path) has odd length, it is called an odd cycle (path). Even paths and cycles are defined analogously. A dipath in a digraph is analogous to a path in a simple graph, except that it has a direction.

A graph $G = (V, E)$ is said to be connected if, for every pair of nodes $u, v \in V$, there exists a path $P \subseteq E$ that connects u and v. A forest $F \subseteq E$ in G is an edge set that contains no cycle. If the subgraph $H = (V(F), F)$ induced by a forest F is connected, then F is called a tree. A tree T is a star if there is a node v such that every edge in T is incident to v. If $T \subseteq E$ is a tree in a graph $G = (V, E)$ such that $V(T) = V$, then T is said to be a spanning tree in G.

2.1.2 Graph partitioning and multicuts

A partition of a set S is a collection of subsets of S such that every element of S is contained in exactly one of the subsets. Let $G = (V, E)$ be a simple graph. The subsets in a partition of the node set V are called clusters. So a partition of V is a set of clusters $\{C_1, \ldots, C_p\}$ such that

$$\begin{aligned} &C_i \in V \text{ and } C_i \neq \emptyset && \forall i = 1 \ldots, p, \\ &C_i \cap C_j = \emptyset && \forall 1 \leq i < j \leq p, \\ &C_1 \cup \ldots \cup C_p = V. \end{aligned}$$

A partition of G, also called a graph partitioning, is a collection of disjoint subgraphs $\{G_1, \ldots, G_p\}$ induced by the clusters of a partition of V. That is, $G_i = G[C_i] = (C_i, E(C_i)),\ i = 1, \ldots, p$.

The edge set of a graph partitioning, referred to as the edges of a partition, is the set of all edges with both end-nodes in the same cluster. This set is written as $E(C_1 \ldots, C_p) := \cup_{i=1}^{p} E(C_i)$, where $\{C_1, \ldots, C_p\}$ is a partition of V. The complement, with respect to E, of the edges of a partition is called a multicut, i.e. the edge set $E \setminus E(C_1, \ldots, C_p)$. When $p = 2$ it is called a cut.

2.1.3 Problems, algorithms and complexity

This section gives an informal description of some basic concepts of computational complexity theory in order to clarify what is meant by a "hard" problem. A much more rigorous treatment of these concepts can be found in the book by Garey and Johnson [31]. The treatment given here corresponds to the one in Grotschel [33].

A problem is a general question to be answered. It is described by one or more parameters and a statement of what properties the answer (or solution) must satisfy. An instance of a problem is obtained by the specification of particular values for all the problem parameters.

Algorithms are general step-by-step procedures for solving problems. It is convenient to think of an algorithm as a computer program. An algorithm is said to solve a problem if the algorithm can be applied to any instance of the problem and it is guaranteed always to produce a solution for that instance.

The time requirements of an algorithm, i.e. the necessary number of elementary steps such as addition, subtraction, multiplication, comparison, etc., to solve a problem instance must be expected to vary roughly with the size of that particular instance. The size of a problem instance is the amount of input data needed to describe the instance. For example, the input to a computer is encoded as a binary string, and the size (or input length) of a particular problem instance is the number of zeros and ones used to encode the instance. The time complexity function for an algorithm expresses its time requirements by giving, for each possible input length, the largest amount of time needed by the algorithm to solve a problem instance of that size. This worst-case running time of the algorithm, expressed as a function $f(l)$ of the input length l, is said to be $O(g(l))$ (read: on the order of $g(l)$) if there exists a positive constant k such that $f(l) \leq kg(l)$ for all $l \geq 0$.

A polynomial time algorithm has a time complexity function $f(l)$ which is $O(p(l))$ for some polynomial function p . Any algorithm whose time complexity function cannot be bounded by a polynomial function is said to be an exponential (or superpolynomial) time algorithm. Polynomial time algorithms are considered much more efficient than exponential time algorithms (for large problem instances), because polynomial functions grow much slower than exponential functions. It follows that an exponential time algorithm, although it is finite, may require an inordinately huge amount of computation time even for moderate size problem instances.

A problem is said to be well-solved if a polynomial time algorithm for it is known. If a problem cannot possibly be solved by any polynomial time algorithm it must be considered intractable. However, in general it is not known how to prove that a problem is intractable in this strict sense. But there is a class of problems for which the existence of polynomial time algorithms is very unlikely. It is most convenient first to consider decision problems. A decision problem has two possible solutions,

either the answer "yes" or the answer "no". The class of all decision problems for which polynomial time algorithms exist is called the class P. There are many decision problems for which no polynomial time algorithm is known. In order to capture (some of) these problems another class NP is defined. The decision problems in NP share the common feature that, for any problem instance, there exists a polynomial time algorithm which can verify the truth or falsity of the claim that the answer for this instance is "yes".

All decision problems in NP can be solved by a nondeterministic polynomial time algorithm | hence the name NP. A nondeterministic algorithm is a hypothetical (and powerful) algorithm which is composed of two stages: a guessing stage and a checking stage. The first stage has the capability somehow to guess some structure, which is subsequently given as input (together with the problem instance) to the checking stage. In the checking stage the algorithm proceeds, in a normal deterministic manner, to find out if the structure provided by the guessing stage complies with the answer "yes".

1. if the answer is "yes", then there exists some structure that can be guessed, for which the checking stage will respond "yes",

2. if the answer is "no", then there exists no structure that can be guessed, for which the checking stage will respond "yes".

One should note the limited capability of a nondeterministic algorithm to halt with the answer "no". In fact, it may run forever if the answer is "no".

A nondeterministic algorithm that solves a decision problem is said to operate in polynomial time if, for every "yes"-instance, there is some structure that, when guessed, leads the checking stage to respond "yes" within time bounded by a polynomial in the input lengths of the problem instance and the guessed structure.

2.2 Overview of models and solution methods

There is a large amount of literature on methods that solve graph partitioning problem optimally. This includes methods dedicated to the bipartitioning case [12], [40], [27], [54], [3], [21], [20], [25], [35], [34], [46] and some methods that solve the general graph partitioning problem [28], [55], [58], [11]. Most of the methods rely on the branch-and-bound framework [22].

Bounds are derived using various approaches: Karisch et al. [40] use semi-definite programming, and Sellman et al. [54] and Sensen [55] employ multi-commodity flows. Linear programming is used by Brunetta et al. [12], Ferreira et al. [28], Lisser et al. [46] and by Armbruster et al. [3]. Felner et al. [27] and Delling et al. [20], [21] utilize combinatorial bounds. Delling et al. [20], [21] derive the bounds by computing minimum $s-t$ cuts between partial assignments (A, B) , i. e., $A, B \subseteq V$ and $A \cap B = \varnothing$. The method can partition road networks with more than a million nodes, but its running time highly depends on the bisection width of the graph.

In general, depending on the method used, two alternatives can be observed. Either the bounds derived are very good and yield small branch-and-bound trees but are hard to compute. Or the bounds are somewhat weaker and yield larger

trees but are faster to compute. The latter is the case when using combinatorial bounds. On finite connected subgraphs of the two dimensional grid without holes, the bipartitioning problem can be solved optimally in $O(n^4)$ time [25].

All of these methods can typically solve only very small problems requiring huge running times, or if they can solve large bipartitioning instances using a moderate amount of time [20], [21], highly depend on the bisection width of the graph. Methods that solve the general graph partitioning problem [Ferreira1998, Sensen2001] have immense running times for graphs with up to a few hundred nodes. Moreover, the experimental evaluation of these methods only considers small number of clusters $k \leq 4$.

We will discuss in the following the models and solutions methods that solve graph partitioning problem optimally in the literature and their application in the case GPP-SC.

2.2.1 Node-Node formulations

We first consider the node partitions formulation introduced by Chopra and Rao [16],[17]. Given a graph $G = (V, E)$ with $n = |V|$ and $m = |E|$. The nodes and the edges are respectively weighted by vectors $w \in \mathbf{R}^{|V|}$ and $l \in \mathbf{R}^{|E|}$, define $\pi = (V_i, i = 1, \ldots, r)$ to be a partition of G. We denote $E(\pi)$ the set of edges with endpoints in two different subsets in the partition π, i.e:

$$E(\pi) = \{(u, v) \in E | \{u, v\} \nsubseteq V_i \ \forall i \in \{1, \ldots, r\}\}$$

Define the incidence vector x as:

$$x_{uv} = \begin{cases} 1 \text{ for } (u, v) \in E(\pi) \\ 0 \text{ othewise} \end{cases} \tag{2.1}$$

Note that if G is a complete graph, $E(\pi)$ coincides with the set of all ordered pairs of V (i.e. $|E(\pi)| = \frac{|V|(|V|-1)}{2}$. If G is not complete, we have to add zero weight edges to compete the graph then the incidence vector x is defined for all ordered pairs of V.

Using the above variables, Chopra and Rao [16] used the *cycle inequalities* originally introduced by Barahona and Mahjoub [5], to define the partitions of G. Let $\mathcal{C}$ be the set of all cycles in G. For a vector $x \in \mathbf{R}^{|E|}$, x_e with $e \in E$ denotes the component of x associated with the edge $e \in E$ and for any subset $F \subseteq E$, let $x(F) = \sum_{e \in F} x_e$. The 0-1 formulation can be then written as follows:

$$\text{(CIP)} \begin{cases} x_e - x_{C \setminus e} \leq 0 & \forall e \in C, \forall C \in \mathcal{C} \\ x_e \in \{0, 1\} & e \in E \end{cases}$$

It has been shown by Chopra [15] that for series-parallel graphs, the linear programming relaxation of (CIP) characterize completely the convex hull of the incidence vectors of all the possible partitions of V, i.e. the vectors that are the solutions of CIP. Note that since there is a priori no known polynomial upper bound on the number of cycles, the above formulation CIP has a priori no known polynomial number

of inequalities. Moreover, using CIP becomes difficult in the presence of additional constraints (e.g. knapsack constraints and capacity constraints) since the decision variables (2.1) are often insufficient to formulate these constraints in a compact way. To overcome this difficulty, new variables are added in completing the graph by the edges of weight 0.

When $G = K_n = (V_n, E_n)$, the complete graph of n nodes, the decision variables are set for all pair of nodes (u, v) in E_n:

$$x_{uv} = \begin{cases} 1 \text{ iff } u \text{ and } v \text{ are in different clusters} \\ 0 \text{ othewise} \end{cases}$$

CIP is of polynomial size since in this case $\mathcal{C}$ reduces to the set of the triples $u \neq v \neq w \in V$. Concretely, let $\mathcal{T}$ be the set of all the (ordered) triples of distinct nodes $u, v, w \in V$, the following system:

$$(\text{IP}) = \begin{cases} x_{uv} + x_{uw} \geq x_{vw} & (u, v, w) \in \mathcal{T} \\ x_{uv} + x_{vw} \geq x_{uw} & (u, v, w) \in \mathcal{T} \\ x_{vw} + x_{uw} \geq x_{uv} & (u, v, w) \in \mathcal{T} \\ x_{uv} \in \{0, 1\} & (u, v) \in E_n \end{cases}$$

consisting of commonly called the triangle inequalities defines the partitions of G. This formulation was introduced by Grötschel and Wakabayashi in [33] to study the *clique partitioning problem*.

Theorem 2.2.1. *The incidence vector x defines a partition of the graph G if and only if x is a feasible point of* IP.

Proof. See Grötschel and Wakabayashi [33]. □

The number of inequalities of IP is $O(|V|^3) = O(n^3)$ whatever the value of $|E| = m$ may be. Using IP the graph partitioning problems (e.g. GPP-SC) can be easily formulated. More precisely, the node partitions can be defined as follows. For a node u in V that is assigned in the cluster C, a node $v \neq u$ in V is also assigned in C iff $x_{uv} = 0$. Moreover, the objective function is in linear form:

$$\min \sum_{(u,v) \in E_n} l_{uv} x_{uv}$$

The node-node formulation was used in most existing mathematical programming models for the graph partitioning problems. For instance, Sørensen [58] uses node-node formulation to formulate *simple graph partitioning problem* (a special case of GPKC where all node weights are equal to 1), Labbé et al. [43] uses node-node formulation to formulate the graph partitioning problem in which the cluster cardinality is bounded both from above and from below, Bonami et al. [11] and Goldschmidt et al. [32]) use node-node formulation to formulate GPCC.

2.2.2 Node-Cluster formulation

For the graph partitioning problem where the number of clusters is bounded from above by a constant k (note that even when k is not imposed as in GPP-SC, we can always set $k = n$), the Node-Cluster [32][11] can be used. This model is based on the choice of decision variables x_{iq} that for all node $i \in V$ and for all cluster $q \in \{1, \ldots, k\}$:

$$x_{iq} = \begin{cases} 1 \text{ node } i \text{ belongs to cluster } q \\ 0 \text{ othewise} \end{cases} \tag{2.2}$$

then the node partitions can be defined easily by:

$$\begin{cases} \sum_{q=1}^{k} x_{iq} = 1 \quad \forall i \in V \\ x_{iq} \in \{0, 1\} \quad \forall i \in V,\ 1 \leq q \leq k \end{cases} \tag{2.3}$$

the assignment constraints in 2.3 imposes that each node is assigned to exactly one cluster. The number of assignment constraints is n. However, this model is highly highly symmetric [48] (it is easy to see that the same partition has many representations in the model) and gives poor results in practice. Some constraints were proposed in [48] to remove some of the symmetry of the model. For the case $q \in \{1, \ldots, n\}$, Bonami et al. [11] proposes two families of constraints that remove all the symmetry related to having several different representations for the same partition. They impose that if the cluster indexed by q is not empty then the node of index q should be the smallest node contained in it by adding the constraints:

$$x_{iq} \leq x_{qq} \qquad \forall i = q+1, \ldots, n$$

$$x_{iq} = 0 \qquad \forall i = 1, \ldots, i-1$$

Note that with this modification, the upper bound constraint on the number of clusters can be modeled by the number of non-empty clusters (i.e. $\sum_{q=1}^{n} x_{qq} \leq k$).

the graph partitioning problems (e.g. GPP-SC) can be easily formulated using Node-Cluster model since the node partitions are well defined. However, the objective of minimization of the sum of the lengths of the edges having end-nodes in different clusters:

$$\min \sum_{(i,j) \in E} \sum_{q=1}^{k} (x_{iq}(1 - x_{jq}) + (1 - x_{iq})x_{jq}) l_{ij}$$

is in quadratic form. Solving Node-Cluster formulation in a LP solver requires some linearization techniques such as the classical linearization [29] and the linearization of Sherali and Smith [57]. Some authors tried to overcome these difficulties by mixing the Node-Node model and the Node-Cluster model.

2.2.3 Approaches of mixing Node-Node and Node-Cluster models

To obtain the mixed formulation for the node partitions, we use both decision variables of the Node-Node model and of the Node-Cluster model that are, for all node $i \in V$ and for all cluster $q \in \{1, \ldots, k\}$:

$$y_{iq} = \begin{cases} 1 \text{ node } i \text{ belongs to cluster } q \\ 0 \text{ othewise} \end{cases} \tag{2.4}$$

and for all edge $(i, j) \in E$:

$$x_{ij} = \begin{cases} 1 \text{ if } i \text{ and } j \text{ belong to different clusters} \\ 0 \text{ othewise} \end{cases} \tag{2.5}$$

then the node partitions can be defined as:

$$\begin{cases} \sum_{q=1}^{k} y_{iq} = 1 & \forall i \in V \\ y_{iq} - y_{jq} \leq x_{ij} & \forall (i,j) \in E,\ 1 \leq q \leq k \\ x_{ij} \in \{0,1\} & \forall (i,j) \in E \\ y_{iq} \in \{0,1\} & \forall i \in V,\ 1 \leq q \leq k \end{cases} \tag{2.6}$$

as compared with the Node-Cluster model, the mixed model contains more constraints to define the relationship between the variables x and y. These constraints express the fact that, if i belongs to cluster q and j does not belongs to cluster q then i et j belong to different clusters, and if i et j belong to the same cluster, then there exists a cluster q such that both i et j belong to the cluster q. Furthermore, the objective function becomes linear:

$$\min \sum_{(i,j) \in E} l_{ij} x_{ij}$$

as in the Node-Node model. However, the mixed model is also highly symmetric as the Node-Cluster model. Some authors tried to remove the symmetries of the model, e.g. Ferreira et al. [28] studies various valid inequalities, Kaibel et al. [39] proposes a new technique called *orbitopal fixing*. Note that the symmetries of the mixed model can also be removed in adding the triangle inequalities [37] but in this case, the mixed model can be considered as a variant of the Node-Node model in which the variables of the Node-Cluster model are considered as additional variables modeling the constraint on the number of clusters.

2.2.4 Semi-definite formulation

Semidefinite programming (SDP) is a subfield of convex optimization concerned with the optimization of a linear objective function over the intersection of the cone

of positive semidefinite matrices with an affine space. Semidefinite programming is a relatively new field of optimization which is of growing interest. Many practical problems in operations research and combinatorial optimization including the graph partitioning problem can be modeled or approximated as semidefinite programming problems.

To understand how the graph partitioning problem can be formulated in a semidefinite programming model, we start with the Node-Node model. If the decision variables are defined for all pair of nodes (u, v) such that $u, v \in V$:

$$x_{uv} = \begin{cases} 1 \text{ iff } u \text{ and } v \text{ are in different clusters} \\ 0 \text{ othewise} \end{cases}$$

then $(x_{uv})_{u,v \in V}$ forms a square matrix in $\mathcal{M}_n$. We note that if x is a feasible point of the graph partitioning problem then $\mathbb{1}_n - x$ where $\mathbb{1}_n$ is the square matrix in $\mathcal{M}_n$ in which all the entries are equal to 1, is a positive semidefinie matrix. Indeed, we suppose that x corresponds to a partition of G into k clusters, then the decision variables in the Node-Cluster model, for all node $i \in V$ and for all cluster $q \in \{1, \ldots, k\}$:

$$y_{iq} = \begin{cases} 1 \text{ node } i \text{ belongs to cluster } q \\ 0 \text{ othewise} \end{cases} \tag{2.7}$$

form a matrix in $\mathcal{M}_{n \times k}$. It is obvious that:

$$\mathbb{1}_n - x = yy^T$$

thus $\mathbb{1}_n - x$ is positive semidefinie matrix.

Hence, the semi-definite formulation for formulating the graph partitions can be obtained in adding the positive semidefinie constraint in the triangle inequalities system:

$$\begin{cases} x_{uv} + x_{uw} \geq x_{vw} & u, v, w \in V \\ \mathbb{1}_n - x \succeq 0 & \\ x_{uv} \in \{0, 1\} & u, v \in V \end{cases}$$

The objective function is also in linear form as in the Node-Node formulation:

$$\min \sum_{(u,v) \in E} l_{uv} x_{uv}$$

The semi-definite formulation is studied in [46] for the graph partitioning problem with additional balance constraints (i.e. equicut problem).

2.2.5 Discussion of the graph partitioning formulations

In this section, we give a brief discussion of the above graph partitioning formulations. We point out advantages and also drawback of each formulation as compared with each other.

The Node-Node model contains linear form objective, there are not symmetric. However, there are a huge number of triangle inequalities ($O(n^3)$).

The Node-Cluster model is highly symmetric, moreover the objective function in this model is in the quadratic form requiring some linearization techniques to apply in a LP solver. The authors in [11] shows that the Node-Node model outperforms the Node-Clusters model in term of the quality of the continuous relaxation.

The mixed model of Node-Node and Node-Cluster model has the objective function also in the linear form. However, as same as the Node-Cluster model, this model is highly symmetric and that make use of more variables than Node-Node and Node-Cluster model.

The semi-definite formulation contains an additional semi-definite constraint as compared with the Node-Node model. Moreover, the semi-definite programming still can not be compared with linear programming in term of computation times. The authors in [46] report that the semi-definite formulation is better than the Node-Node model with a few number of clusters, otherwise, when the number of clusters is bigger, the Node-Node model is preferable.

In all the investigation of the present work, we are interested only on the Node-Node formulations.

Chapter 3

Improved compact formulations for sparse graphs

Most existing mathematical programming models for the graph partitioning problem including GPP-SC use $O(|V|) = O(n^3)$ triangle constraints (2.2.1) to model the node partitions. Several works (e.g. [58], [43], [11], [32]) proposed branch-and-bound ou branch-and-cut algorithms which are all based on (IP) enhanced by several additional class of valid inequalities. Consequently, these algorithms have to solve repeatedly the continuous programming relaxation ($\overline{\text{IP}}$) of (IP). As the number of triangle inequalities in this model is $O(n^3)$, it quickly becomes extremely large as n increases and even the relaxations turn out to be difficult to solve. Some authors have tried to overcome this difficulty by dualizing all or only a subset of the triangle inequalities via a Lagrangian approach [30].

Another possible approach considered in this chapter, is to try and reduce the number of triangle inequalities without weakening the relaxation. We will show that with only $O(nm)$ triangle inequalities, instead of $O(n^3)$, we can obtain an equivalent formulation, not only for the Node-Node model for GPP-SC, but also for its relaxations. Obviously, such a reduction opens the way to considerable improvement in case of sparse graphs where $m \ll \frac{n(n-1)}{2}$.

Note that in the definition of GPP-SC, the number of clusters is not bounded and it is part of the output of the optimization process. However, we will show that this reduction of the triangle inequalities remains valid for GPP-SC even in the presence of an upper limit constraint on the number of clusters.

Computational results are discussed to show the benefit of using the reduced formulation in term of computational efficiency.

3.1 Basic 0/1 programming model for GPP-SC

The basic 0/1 programming model for GPP-SC which is considered in the sequel can be described as follows. We introduce $\frac{n(n-1)}{2}$ binary variables x_{uv} for all the pairs of nodes $u, v \in V$, $u < v$, such that

$$x_{uv} = \begin{cases} 0 & \text{if } u \text{ and } v \text{ belong to the same cluster,} \\ 1 & \text{otherwise.} \end{cases}$$

The triangle inequalities together with the binary constraints are often used to define the partitions of a graph G [33], [5]. They will be used here as part of the constraints for formulating GPP-SC. Denoting $\mathcal{T}$ the set of all triples (u, v, w) of nodes in V such that $u < v < w$ and E_n the set of all ordered pairs of node in V (i.e. $|E_n| = \frac{n(n-1)}{2}$), these constraints can be written as:

$$
(\mathrm{I}) \begin{cases} \forall (u,v,w) \in \mathcal{T} & & \\ x_{uv} + x_{uw} \geq x_{vw} & & (1) \\ x_{uv} + x_{vw} \geq x_{uw} & & (2) \\ x_{vw} + x_{uw} \geq x_{uv} & & (3) \\ x_{uv} \in \{0,1\} & \forall (u,v) \in E_n & \end{cases}
$$

The number of triangle inequalities in system (I) is $3\binom{n}{3}$ whatever the value of m may be. It has been shown by Chopra [15] that for series-parallel graphs, the linear programming relaxation of (I) characterize completely the convex hull of the incidence vectors of all the possible partitions of V, i.e. the vectors that are the solutions of (I).

The Node-Node model for GPP-SC can be described as follows. Denoting $\mathbb{1}$ the n-vector with all components equal to 1, for every node $u \in V$ and for the cluster C that contains u, the incidence vector y^C is equal to the n-vector $\mathbb{1} - \chi^u$, where χ^u is the n-vector with components defined for all $v \in V$ as:

$$
\chi^u_v = x_{|uv|} = \begin{cases} x_{uv} & \text{if } u < v, \\ x_{vu} & \text{if } u > v, \\ 0 & \text{if } u = v, \end{cases} \quad \forall v \in V.
$$

Now the constraint $\mathcal{G}(y^C) \leq 0$ to be satisfied by every cluster C in the solution can be equivalently reformulated as the n separate constraints $\mathcal{G}(\mathbb{1} - \chi^u) \leq 0$, for all $u \in V$. Denoting $g_u(x) = \mathcal{G}(\mathbb{1} - \chi^u)$, for all $u \in V$, the Node-Node model for GPP-SC is:

$$
(\mathrm{IP}) \begin{cases} \min \sum_{(u,v) \in E} l_{uv} x_{uv} & & \\ \text{s.t. } \forall (u,v,w) \in \mathcal{T} & & \\ \quad x_{uv} + x_{uw} \geq x_{vw} & & (1) \\ \quad x_{uv} + x_{vw} \geq x_{uw} & & (2) \\ \quad x_{vw} + x_{uw} \geq x_{uv} & & (3) \\ \quad g_u(x) \leq 0 & \forall u \in V & (4) \\ \quad x_{uv} \in \{0,1\} & (u,v) \in E_n & \end{cases}
$$

Observe that assuming $\mathcal{G}$ nondecreasing with respect to y^C implies that each pseudoboolean function $g_u : \{0,1\}^n \to \mathbb{R}_+$ should be *nonincreasing* with respect to x (note that g_u actually only depends on the subset of variables x_{ij} such that either $i = u$ or $j = u$). Let $(\overline{\mathrm{IP}})$ denote the continuous relaxation of (IP). This formulation has $O(n^2)$ variables and $O(n^3)$ constraints.

Variants of the above graph partitioning problem with various additional constraints have already been considered by several authors [33], [58], [43], [11]. These works

proposed branch-and-bound or branch-and-cut algorithms which are all based on (IP) enhanced by using various types of valid inequalities. Consequently, these algorithms have to solve repeatedly the continuous relaxation ($\overline{\text{IP}}$) of (IP). As already mentioned above, the (IP) model does not take advantage of the possible sparsity of the graph under consideration: it always requires $O(n^3)$ constraints, even in the case $m \ll \frac{n(n-1)}{2}$. Although the latter is just a standard linear program with a polynomial number of constraints, it was reported in the literature (e.g. [30],[11]) that it is rather difficult to solve in the presence of all the $3\binom{n}{3}$ triangle inequalities even for small values of n (i.e. $n \leq 20$). In the next section, we will show that we can obtain an equivalent formulation for (IP) and ($\overline{\text{IP}}$) with only $3m(n-2)$ triangle inequalities, instead of $3\binom{n}{3}$ as classically.

As will become apparent in the sequel, the above formulation (IP) is quite general and encompasses many possible variants of graph partitioning. Depending on the type of problem considered, the functions $g_u(x)$ involved in constraint (4) can be linear or nonlinear. In the latter case, solving (IP) with a MILP solver requires *linearization* of these functions (this will be the case of GPCC for instance, see section 6.2 in the computational section). However, it should be stressed that all the results in the forthcoming sections 3 and 4 concerning the reduction of the number of triangle inequalities will be applicable *independently of the linearization technique used.*

3.2 Improved 0/1 programming model for GPP-SC

We now process to show that, for the case of sparse graphs, the number of triangle inequalities can be significantly reduced while preserving the equivalence both for (IP) and ($\overline{\text{IP}}$). The idea is instead of considering all triples of $\mathcal{T}$, we only consider those such that at least one pair of nodes forms an edge in E. Precisely, let $\mathcal{T}'$ be the family of these triples, i.e. $\mathcal{T}' = \{(u,v,w) : u < v < w \in V$ and at least one of the edges (u,v), (u,w) and $(v,w) \in E\}$. Then the reduced programming model is obtained from (IP) with the triangle inequalities expressed only for the triples in $\mathcal{T}'$

$$
(\text{RIP}) \begin{cases}
\min \sum_{(u,v)\in E} l_{uv} x_{uv} & & \\
\text{s.t. } \forall (u,v,w) \in \mathcal{T}' & & \\
\quad x_{uv} + x_{uw} \geq x_{vw} & & (5) \\
\quad x_{uv} + x_{vw} \geq x_{uw} & & (6) \\
\quad x_{vw} + x_{uw} \geq x_{uv} & & (7) \\
\quad g_u(x) \leq 0 & \forall u \in V & (8) \\
\quad x_{uv} \in \{0,1\} & (u,v) \in E & \\
\quad x_{uv} \in [0,1] & (u,v) \in E_n \backslash E &
\end{cases}
$$

It is clear that $|\mathcal{T}'| \leq m(n-2)$ thus the number of triangle inequalities in (RIP) is at most $3m(n-2)$. The continuous relaxation of (RIP) is denoted $(\overline{\text{RIP}})$. Due to this reduction of the triangle inequalities, $(\overline{\text{RIP}})$ will obviously be more interesting than $(\overline{\text{IP}})$ for LP solvers when applied to sparse graphs.
Given a point $x \in \mathbb{R}^{|E_n|}$, we note x_E the restriction of x on $\mathbb{R}^{|E|}$ i.e. the components of x whose index are in E. We will show the following main lemma.

Lemma 3.2.1. *Given a point $x^r \in [0,1]^{|E_n|}$ satisfying the inequalities (5-8). There always exists a point $x \in [0,1]^{|E_n|}$ satisfying the inequalities (1-4) such that $x_E = x^r_E$.*

To prove Lemma 3.2.1, let us specify how to construct x from x^r:

> Let us consider the graph $G = (V, E)$ where the edges in E are weighted by x^r_E and let p_{uv} denote the value of the shortest path in G between u and v with respect to weights x^r_E. Then $x \in [0,1]^{|E_n|}$ is defined as: $x_{uv} = \min\{1, p_{uv}\}$ for all $(u,v) \in E_n$.

Before showing that the resulting x is feasible for $(\overline{\text{IP}})$, let us prove the following proposition.

Proposition 3.2.2. *$x_{uv} \geq x^r_{uv}$ for all $(u,v) \in E_n$.*

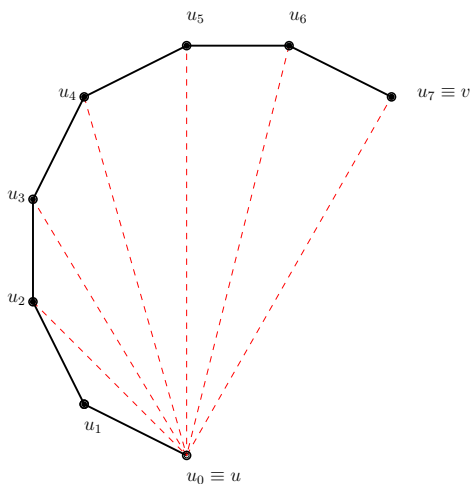


Figure 3.1: *A example of triangularization for a path uv.*

Proof. Let us consider any pair (u,v) and suppose that $\{u \equiv u_0, \ldots, u_p \equiv v\}$ is the shortest path in G between (u,v) with respect to weights x^r_E. Let us consider successively the triangles composed of vertex u_0 and an edge of the shortest path as showed in Figure 4.1. Since each of these triangles contains at least one edge in E, x^r satisfies the triangle inequalities (5-7) issued from these triangles. We want to show that $x^r_{|u_0u_p|} \leq \sum_{\eta=1}^{p} x^r_{|u_{\eta-1}u_\eta|}$. In fact, by induction:

- If $p = 1$, it is obvious that $x^r_{|u_0u_1|} = x^r_{|u_0u_1|}$.
- If $p = 2$, the inequality $x^r_{|u_0u_2|} \leq x^r_{|u_0u_1|} + x^r_{|u_1u_2|}$ is one of the triangle inequalities (5-7) for the triple (u_0, u_1, u_2) that is verified by x^r.

- Assume that the inequality was satisfied for $p = p_r$ for some $p_r \geq 2$, i.e. that

$$x^r_{|u_0 u_p|} \leq \sum_{\eta=1}^{p_r} x^r_{|u_{\eta-1} u_\eta|}$$

We will show that it is also true for $p = p_r + 1$. In fact, one of the triangle inequalities for the triple $(u_0, u_{p_r}, u_{p_r+1})$ gives

$$x^r_{|u_0 u_{p_r+1}|} \leq x^r_{|u_0 u_{p_r}|} + x^r_{|u_{p_r} u_{p_r+1}|}$$

By combining with the induction hypothesis we obtain

$$x^r_{|u_0 u_p|} \leq \sum_{\eta=1}^{p_r+1} x^r_{|u_{\eta-1} u_\eta|}.$$

As $u_0 = u$, $u_p = v$ and $\sum_{\eta=1}^{p} x^r_{|u_{\eta-1} u_\eta|} = p_{uv}$, we deduce that $x^r_{uv} \leq \min\{1, p_{uv}\} = x_{uv}$. □

We are now ready to provide a proof for Lemma 3.2.1.

Proof of Lemma 3.2.1. We prove that the point x constructed as above from x^r verifies the inequalities (1-4) and in addition $x_E = x^r_E$.
We prove first that x satisfies the triangle inequalities (1-3). We only need to prove that x satisfies (1), by symmetry, x satisfies also (2) and (3). Let us consider any triple (u, v, w) in $\mathcal{T}$ and suppose that p_{uv}, p_{vw} and p_{uw} are respectively the shortest path lengths in G between (u, v), (v, w) and (u, w) with respect to weights x^r_E. Note that the union of two shortest paths between (u, v) and (u, w) is also a path between (v, w), thus we have $p_{uv}+p_{uw} \geq p_{vw}$. Hence $\min\{1, p_{uv}\}+\min\{1, p_{uw}\} \geq \min\{1, p_{vw}\}$ which implies inequality (1).
We now show that x satisfies the inequalities (4). Indeed, as g is nonincreasing, we have $g_u(x) \leq g_u(x^r) \leq 0$ for all $u \in V$.
Finally, we show that $x_E = x^r_E$, i.e. we show that if (u, v) is an edge in E, the shortest path in G between u and v with respect to x^r_E is x^r_{uv}. This is shown by contradiction. Suppose that (u, v) is an edge in E and suppose that the shortest path in G between u and v with respect to x^r_E is $\{u \equiv u_0, \ldots, u_p \equiv v\} \neq (u, v)$. From the construction of x and Proposition 3.2.2 we have $p_{uv} = \sum_{\eta=1}^{p} x^r_{|u_{\eta-1} u_\eta|} > x^r_{uv}$. A contradiction. Hence we conclude that $x_E = x^r_E$. □

We now show the main result of this section:

Theorem 3.2.3. *($\overline{\text{IP}}$) and ($\overline{\text{RIP}}$) have the same optimal value.*

Proof. The result of the theorem readily follows from the Lemma 3.2.1 using the fact that $x_E = x^r_E$. □

A similar result holds for (IP) and (RIP) since in the construction of x from x^r, $x^r_E \in \{0, 1\}^{|E|}$ leads $x \in \{0, 1\}^{|E_n|}$.

Corollary 3.2.4. (IP) *and* (RIP) *have the same optimal value.*

3.3 Extension to the case when the number of clusters is bounded from above

In this section, it will be shown that part of the above results can be extended to the case when the number of clusters is bounded from above by a given constant. Note that the limitation on the number of clusters is somewhat conflicting with the monotone nondecreasing property of the cluster constraints in GPP-SC due to the fact that the number of clusters will increase as the number of nodes in the clusters is decreased. Therefore Theorem 3.2.3 seems to be non applicable in this case. Nevertheless, we prove in the following that the result of Corollary 3.2.4 is still applicable. To achieve this and exploiting some analogies with the analysis in [1], we introduce a vector $z \in \{0,1\}^{|V|}$ that is defined as follows: $z_i = 1$ if and only if i is the smallest node index in the cluster that contains i, in this case the node i is called a *representative*. Therefore z can be computed from x as, for all $i \in V$,

$$z_i = \begin{cases} 1 & \text{if } \sum_{j \in V,\ j<i} (1 - x_{ji}) = 0, \\ 0 & \text{otherwise.} \end{cases}$$

For a given positive integer k, the Node-Node model for GPP-SC with at most k clusters is then:

$$\text{(kIP)} \begin{cases} \min \sum_{(u,v)\in E} l_{uv} x_{uv} & & \\ \text{s.t. } \forall (u,v,w) \in \mathcal{T} & & \\ \quad x_{uv} + x_{uw} \geq x_{vw} & & \\ \quad x_{uv} + x_{vw} \geq x_{uw} & & \\ \quad x_{vw} + x_{uw} \geq x_{uv} & & \\ \quad g_u(x) \leq 0 & \forall u \in V & (8) \\ \quad z_u - x_{vu} \leq 0 & \forall u, v \in V,\ v < u & (9) \\ \quad z_u + \sum_{v=1}^{u-1} (1 - x_{vu}) \geq 1 & \forall u \in V & (10) \\ \quad \sum_{u=1}^{n} z_u \leq k & & (11) \\ \quad x_{uv} \in \{0,1\} & (u,v) \in E_n & \\ \quad z_u \in [0,1] & u \in V & \end{cases}$$

Constraints (8) are the same as in (IP) and as in (RIP) where $g_u : \{0,1\}^n \to \mathbb{R}_+$ is a nonincreasing pseudoboolean function. Constraints (9) ensure that every cluster contains no more than one representative (i.e. no more than one node i such that $z_i = 1$). Constraints (10) guarantee that a cluster contains at least one representative. Finally, constraint (11) ensures that the number of clusters is no more than a given positive integer number k. The *reduced programming model* (kRIP) is obtained by replacing the set of triples $\mathcal{T}$ with $\mathcal{T}'$.

Lemma 3.3.1. *Given* $(x^r, z^r) \in \{0,1\}^{|E_n|} \times \{0,1\}^{|V|}$ *a feasible point for* (kRIP). *It is always possible to find a feasible point* $(x, z) \in \{0,1\}^{|E_n|} \times \{0,1\}^{|V|}$ *for* (kIP) *such that* $x_E = x^r_E$.

The main idea of the proof of this lemma is quite similar to the one given in Section 3.2, except for the construction of (x, z) from (x^r, z^r) which is now done as follows:

> Assume that (x^r, z^r) is a feasible point of (kRIP). We can construct a partition P of V as follows. For node i from 1 to $n = |V|$, if $z_i = 1$ (i.e. i is a representative) then for all node j from $i+1$ to n, j is assigned to the same cluster as i if and only if j has not been assigned before and $x^r_{ij} = 0$. The point $(x, z) \in \{0,1\}^{E_n} \times [0,1]^V$ is then computed from the partition P as:
>
> $$\text{for all } (i,j) \in E_n,\ x_{ij} = \begin{cases} 0 & \text{if } i \text{ and } j \text{ belong to the same cluster,} \\ 1 & \text{otherwise.} \end{cases}$$
>
> and
>
> $$\text{for all } i \in V,\ z_i = \begin{cases} 1 & \text{if } i \text{ is the smallest node index in the} \\ & \text{cluster that contains the node } i\ , \\ 0 & \text{otherwise.} \end{cases}$$

To show that such a point (x, z) is feasible point for (kIP), we need the following proposition :

Proposition 3.3.2.

(1) Node $j > i$ is assigned to the same cluster as the representative i if and only if i is the smallest index of a representative such that $x^r_{ij} = 0$.

(2) $z = z^r$.

(3) For every representative $i \in V$ (i.e. $z_i = 1$), $x_{ji} = x^r_{ji} = 1$, for all $j \in V,\ j < i$ and $x_{ij} \geq x^r_{ij}$, for all $j \in V,\ j > i$.

Proof. The proof directly follows from the construction of (x, z). □

We are now ready to provide a proof for Lemma 3.3.1.

Proof of Lemma 3.3.1. We prove that (x, z) constructed as above from (x^r, z^r) is a feasible point of (kIP) and in addition that $x_E = x^r_E$.
The point (x, z) is deduced from the partition construction and thus x satisfies the triangle inequalities whereas z satisfies the representative constraints (9) and (10). As z^r satisfies the constraint (11) in (kRIP) and $z = z^r$ (Proposition 3.3.2 item (2)), z also satisfies this constraint. For every representative $i \in V,\ z_i = 1$, Proposition 3.3.2 item (3) guarantees that constraints (8) are satisfied for i, since $g_i(x^r) \leq 0$ and g_i is a nonincreasing function. For every node u that is not a representative, assume that u belong to the cluster with the representative i, we have $g_u(x) = g_i(x) \leq 0$ as they are defined on the same cluster. Thus all the constraints of (kIP) are satisfied by (x, z) and we can conclude that (x, z) is a feasible point for (kIP).
We now show that $x_E = x^r_E$ (i.e. $x_{uv} = x^r_{uv}$, for all $(u, v) \in E$). Indeed, for all $(u, v) \in E$:

- if $x^r_{uv} = 1$, u and v are not in the same cluster since otherwise, there would exist a cluster with i as representative such that $x^r_{iu} = x^r_{iv} = 0$, and the triangle inequality $x^r_{iu} + x^r_{iv} \geq x^r_{uv}$ would be violated. Hence $x_{uv} = x^r_{uv} = 1$.

- if $x^r_{uv} = 0$, u and v will be assigned to the same cluster. Indeed, we remark that for any representative i such that $x^r_{iu} = 0$, the triangle inequality $x^r_{iu}+x^r_{uv} \geq x^r_{iv}$ implies $x^r_{iv} = 0$, and reciprocally for any representative i such that $x^r_{iv} = 0$, the triangle inequality $x^r_{iv} + x^r_{uv} \geq x^r_{iu}$ implies $x^r_{iu} = 0$. Due to this remark, if i is the smallest representative index such that $x^r_{iu} = 0$, then i is also the smallest representative index such that $x^r_{iv} = 0$. By Proposition 3.3.2 item (1), u and v are then assigned to the same cluster with i as representative and hence $x_{uv} = x^r_{uv} = 0$.

□

A similar result as Corollary 3.2.4 now holds for (kIP) and (kRIP).

Theorem 3.3.3. (kIP) *and* (kRIP) *have the same optimal value.*

Proof. The proof of the theorem follows trivially from the Lemma 3.3.1 by the fact that $x_E = x^r_E$. □

3.4 Computational experiments

In this section, we present computational results obtained with the improved compact formulation (RIP) as compared with the standard compact formulation (IP) for both variants GPKC and GPCC of GPP-SC. To carry out the computational comparisons, a sufficiently diverse set of test instances of relatively small size (typically less than 100 nodes) was needed. However **(a)** such instances for GPKC are only very scarce in the existing literature (e.g only four instances with $n < 100$ can be found in the DIMACS data set [4]); **(b)** there is no data set corresponding to the sparse instances for GPCC. Therefore it was decided to generate the required test set for GPKC and GPCC in the following way:

- For a choice of graph type and for a given number of nodes n and number of edges m, the graph is firstly generated. To verify our results in the present paper, four well known sparse graph types are chosen to be generated that are : series-parallel graph, planar grid graph, toroidal grid graph and random graph. Except the planar grid graphs and the toroidal grid graphs that have fixed structure, the series-parallel graphs are generated using a generator named Task Graphs For Free (TGFF) [53] and the random graphs are generated by picking edges uniformly at random until the number of edges reaches m, and testing connectedness.

- The edge weights $t_{uv},\ (u, v) \in E$ and the node weights $w_u,\ u \in V$ are drawn independently and uniformly from the interval $[1, 1000]$.

- The upper bounds of the knapsack constraints W and of the capacity constraints T are chosen in such a way as to ensure that the generated instances will not be "too easy" to solve. More precisely we used METIS[41] to create an "optimal" partition of the graph with k clusters that we call the initial partition, we then do 1000 random perturbations of this partition. The bounds W and T are then chosen so that only 10% of these partitions correspond to feasible solutions.

All experiments are run on a machine with Intel Core i7-3630QM 2.40GHz processor and 16 GB of RAM. The solver CPLEX 12.6 is used to solve respectively ($\overline{\text{IP}}$), (IP), ($\overline{\text{RIP}}$) and (RIP). CPLEX pre-solve is switched off as is classically done to avoid possible undesirable side-effects due to the uncontrolled behaviour of this "black-box" procedure. All computation times are CPU seconds and the computation times of (IP) and (RIP) are subject to a *time limit of 12000 seconds* while the computation times of ($\overline{\text{IP}}$) and ($\overline{\text{RIP}}$) are subject to a *time limit of 3600 seconds.*

3.4.1 Experimental results for GPKC

The graph partitioning problem under knapsack constraints (GPKC) has been defined in Introduction section as a special case of GPP-SC. The problem contains knapsack constraints of type $\sum_{v \in C} w_v \leq W$ for each cluster C in the partition. In the Node-Node model, knapsack constraints can be written as:

$$\sum_{v \in V} (1 - x_{|uv|}) w_v \leq W, \quad \forall u \in V \qquad (12)$$

These constraints express that for all $u \in V$, the total weight of the cluster that contains u is bounded by a given constant W. Note that the total weight of each cluster is equal to sum of the weights of nodes contained in it. Note that in our formulation, all pairs in E_n are ordered thus the nodes u and v are in the same cluster or not, determined by $x_{|uv|}$, i.e. x_{vu} if $v < u$, x_{uv} if $v > u$ and 0 if $u \equiv v$.
In Table 3.1 we report the results obtained for (IP) and (RIP) when applying on GPKC instances, denoted (IPKC) and (RIPKC) respectively and for their linear relaxations ($\overline{\text{IPKC}}$) and ($\overline{\text{RIPKC}}$). In our experiments, each instance belongs to one of four graph types: series-parallel graphs, planar grid graphs, toroidal grid graphs and random graphs. Series-parallel graphs are highly sparse while random graphs are denser graphs with $\frac{m}{n} \approx$ (4 to 8). As for each value of n, m, we have five instances, the first two columns in this table report the average CPU time to obtain the solution of (IPKC) and its linear relaxation ($\overline{\text{IPKC}}$), the third and fourth columns show the average CPU time to obtain the solution of (RIPKC) and its continuous relaxation ($\overline{\text{RIPKC}}$), and the last column gives the average value of the integrality gap.
As can be seen from Table 3.1, (RIPKC) and ($\overline{\text{RIPKC}}$) are much better than (IPKC) and ($\overline{\text{IPKC}}$) in terms of computation times. The difference becomes more significant as the number of nodes increases. With series-parallel graphs, the gain of (RIPKC) and ($\overline{\text{RIPKC}}$) is extremely clear as we report a reduction of solution time by a factor 10 to 50 for ($\overline{\text{RIPKC}}$) as compared with ($\overline{\text{IPKC}}$) and by a factor 3 to 60 for (RIPKC) as compared with (IPKC). The difference naturally decreases as m

increases. For instance with random graphs, we report a reduction of solution time by a factor 3 to 15 for ($\overline{\text{RIPKC}}$) as compared with ($\overline{\text{IPKC}}$) and by a factor 3 for (RIPKC) as compared with (IPKC). There are also instances for which CPLEX is not even capable to solve the continuous relaxation with the classical formulation within the prescribed time-limits but succeeds at finding an optimal integer solution with our reduced formulation (series-parallel (80,130) and bigger, planar grid 8×10 and larger, toroidal grid 8×10). With (RIPKC) we can solve exactly large instances, e.g, ($n = 140$) for series-parallel graphs, ($n = 110$) for planar grid graphs, ($n = 80$) for toroidal grid graphs and ($n = 50$) for random graphs.

Also it can be seen that the continuous relaxation ($\overline{\text{RIPKC}}$) is rather strong for GPKC, especially for instances related to series-parallel and planar grid graphs (6.5% on average). For toroidal grid and random graphs, the gaps is slightly bigger (10.5% on average).

Compared efficiency with and without upper bound on the number of clusters

Computation experiments are also made to compare the efficiency of solving GPKC with and without upper bound on the number of clusters. To highlight the differences between them, the experiments are done in the following way. For any GPKC instance i, we first solve (RIPKC), the corresponding number of clusters of the optimal solution is denoted k_i^*. We then solve (kRIPKC) (i.e. (kRIP) applying on GPKC instances) for the same instance i in adding the upper bound on the number of clusters k_i. It is obvious that if $k_i \geq k_i^*$, the solutions of (RIPKC) and (kRIPKC) are identical. Computational results are shown in Table 3.2 for $k_i = k_i^* - 1$ and $k_i = k_i^* - 2$. For each value of n, m, five instances are solved. The third and the fourth columns in the table report the CPU time to obtain the optimal solution using (RIPKC) and the number of clusters in the optimal solution, the fifth and the sixth columns report the CPU time to obtain the optimal solution using ((k*-1)RIPKC) and ((k*-2)RIPKC) respectively. The acronym "NF" indicates a non-feasible instance, and the computation time to prove the infeasibility is shown in parenthesis.

As can be seen from Table 3.2, while all instances can be solved for (RIPKC), there are only part of instances can be solved for (kRIPKC) within the time limit of 12000 seconds. Moreover, for the instances that can be solved for both models, (kRIPKC) is from 3 to 6 times slower than (RIPKC). A possible explanation of this is as follows. From the results in Table 3.2, it is seen that for all instances considered, the number of clusters k^* in the optimal solution to (RIPKC) is close to its minimum possible value. Indeed, for $k = k^* - 1$, about 50% of the instances become infeasible, and for $k = k^* - 2$, about 90% of the instances turn out to be infeasible. The observed increase in the computation times is thus due at least in part, to the fact that, when decreasing k, the instances become close to the boundary between feasibility and infeasibility.

3.4.2 Experimental results for GPCC

In this section we present the computation results for (IP) and (RIP) on GPCC instances, denoted (IPCC) and (RIPCC) respectively, and for their continuous relaxation ($\overline{\text{IPCC}}$) and ($\overline{\text{RIPCC}}$). Let us rewrite the capacity constraints proposed in [32] in the Node-Node model:

$$\sum_{(v,w)\in E} t_{vw} x_{|uv|} x_{|uw|} \geq \sum_{(v,w)\in E} t_{vw} - T, \quad \forall u \in V \qquad (13)$$

It expresses the fact that the complement of the capacity of the cluster containing u should be greater than the total capacity of the graph minus T. This is equivalent to say that the capacity of the cluster containing u should be bounded by T.
We observe that the capacity constraint (13) is in non-convex quadratic form for which CPLEX cannot be used. We therefore have to transform the capacity constraints to linear form via some linearization techniques. We note that the results of the present paper remain applicable whatever linearization technique used. In the computational experiments presented below, we use the classical Fortet linearization [29], which is both simple and known to achieve a good compromise between the number of additional variables needed and the strength of the resulting relaxation. Application of this technique to GPCC leads to introduce new variables y_{uvw} to represent each product $x_{|uv|}x_{|uw|}$ thus the constraint (13) becomes:

$$\begin{cases} \sum_{(v,w)\in E} t_{vw} y_{uvw} \geq \sum_{(v,w)\in E} t_{vw} - C, \quad \forall u \in V \\ \max\left\{0, x_{|uv|} + x_{|uw|} - 1\right\} \leq y_{uvw} \leq \min\left\{x_{|uv|}, x_{|uw|}\right\} \end{cases}$$

Hence our quadratic 0-1 formulations for GPCC become a mixed integer linear program that can be solved more easily by CPLEX. Note that the constraint (13) is the same for (IPCC), (RIPCC), ($\overline{\text{IPCC}}$) and ($\overline{\text{RIPCC}}$) so that the linearization does not influence the comparison between them.
Since usually the traffic matrix for a SONET/SDH optical networks are quite dense, we generate instances for GPCC with number of edges $\frac{m}{n} \approx$ (1.5 to 8), the results are shown in Table 3.3.
As we can see in Table 3.3 , (RIPCC) and ($\overline{\text{RIPCC}}$) is much better than (IPCC) and ($\overline{\text{IPCC}}$) in terms of computation times. We report a reduction of solution time by a factor 3 to 18 for ($\overline{\text{RIPCC}}$) as compared with ($\overline{\text{IPCC}}$) and by a factor about 4 for (RIPCC) as compared with (IPCC). The difference is even more clear when increasing the number of nodes. The instances used in this section feature higher density than those in the previous section thus we can only solve those up to $n = 80$.
We finally observe that the quality of the continuous relaxation of the Node-Node model for GPCC is not as good as for the GPKC problem: the value of the gap is observed to be in the range 10.4% to 18.1%.

3.5 Conclusions

We have given an improved compact formulation for a wide class of graph partitioning problem using $O(nm)$ triangle inequalities instead of $O(n^3)$ in the classical

formulation, while preserving equivalence, both in the integral version and in the relaxed version. Numerical experiments comparing our improved formulation with the classical formulation have been presented for two problems: the graph partitioning problem under knapsack constraints and the graph partitioning problem under capacity constraints. These numerical results have shown that solution times are reduced drastically from 3 to 50 times with our improved formulation. There are instances for which the LP solver is not even capable of solving the continuous relaxation with the classical formulation but succeeds at finding optimal integer solutions with our reduced formulation.

The largest instances solved to exact optimality with the approach of the present chapter are limited in size ($n \leq 140,\ m \leq 280$) while practical applications would require handling much bigger problems. For bigger problems, only approximate (heuristic) solutions can be hoped for, and a key issue in this context is to validate the approximate solutions obtained by computing lower bounds. This is the subject addressed in the next chapter.

Table 3.1: Computation results of (IPKC) and (RIPKC) for sparse graphs.

graph types	n,m	(IPKC)	($\overline{\text{IPKC}}$)	(RIPKC)	($\overline{\text{RIPKC}}$)	Cont.Rlx
		CPU	CPU	CPU	CPU	GAP(%)
series-parallel	22, 38	2.18	0.22	0.65	0.02	6.7
series-parallel	25, 40	3.43	0.55	0.67	0.04	8.4
series-parallel	27, 45	6.36	0.63	0.38	0.05	6.1
series-parallel	30, 50	19.36	1.96	2.12	0.10	5.6
series-parallel	35, 60	34.25	2.04	3.10	0.13	4.3
series-parallel	40, 65	114.3	6.40	5.13	0.27	4.5
series-parallel	45, 75	486.1	11.63	8.28	0.35	7.2
series-parallel	50, 80	-	23.48	15.32	0.85	4.7
series-parallel	60, 90	-	186.33	33.61	3.45	4.2
series-parallel	80, 130	-	-	83.38	10.67	5.8
series-parallel	100, 170	-	-	95.66	25.12	4.1
series-parallel	120, 180	-	-	588.58	68.26	5.2
series-parallel	130, 200	-	-	3276.3	98.27	2.7
series-parallel	140, 210	-	-	8783.4	155.66	5.9
planar grid 4×10	40, 66	120.3	5.78	5.41	0.36	6.3
planar grid 5×10	50, 85	-	26.96	16.8	0.89	6.3
planar grid 6×10	60, 104	-	174.31	61.4	1.72	5.7
planar grid 7×10	70, 123	-	951.72	123.94	3.26	6.1
planar grid 8×10	80, 142	-	-	427.7	12.19	7.5
planar grid 9×10	90, 161	-	-	1478.2	15.77	8.0
planar grid 10×10	100, 180	-	-	2147.2	26.51	5.4
planar grid 11×10	110, 199	-	-	8044.8	40.78	7.3
toroidal grid 4×10	40, 80	424.4	6.39	18.37	0.44	10.1
toroidal grid 5×10	50, 100	-	36.68	117.29	1.03	10.6
toroidal grid 6×10	60, 120	-	199.63	431.51	2.08	8.2
toroidal grid 7×10	70, 140	-	1233.61	3361.7	3.66	9.7
toroidal grid 8×10	80, 160	-	-	10934.5	14.23	11.6
random graph	22, 120	58.8	0.54	16.75	0.15	9.3
random graph	25, 150	120.2	1.13	34.34	0.7	12.5
random graph	27, 150	379.5	1.25	139.7	0.49	10.8
random graph	30, 200	1436.8	2.79	458.5	0.98	8.4
random graph	35, 250	-	7.97	945.6	2.65	10.5
random graph	40, 280	-	14.5	3326.9	4.97	10.2
random graph	45, 200	-	19.0	3884.2	1.47	12.7
random graph	50, 200	-	35.88	9024.8	2.32	11.4

Table 3.2: Comparison between (RIPKC) and (kRIPKC) for random sparse graphs. NF indicates non-feasible instances.

n,m	Instance	(RIPKC)		((k*-1)RIPKC)	((k*-2)RIPKC)
	number	CPU	k*	CPU	CPU
	#1	276	3	NF(0.12)	NF(0.13)
	#2	402	5	NF(0.14)	NF(0.12)
30, 200	#3	452	5	1395	NF(0.15)
	#4	507	6	1558	NF(0.17)
	#5	558	8	1681	1847
	#1	733	4	NF(0.22)	NF(0.17)
	#2	911	5	NF(0.21)	NF(0.21)
35, 250	#3	906	5	3286	NF(0.38)
	#4	992	6	NF(0.25)	NF(0.20)
	#5	1071	7	4492	NF(0.28)
	#1	2954	4	NF(0.30)	NF(0.20)
	#2	3038	5	NF(0.29)	NF(0.31)
40, 280	#3	3353	6	9822	NF(0.35)
	#4	3827	7	9376	NF(0.35)
	#5	4158	9	>12000	>12000
	#1	3535	5	NF(0.39)	NF(0.33)
	#2	3642	5	NF(0.39)	NF(0.30)
45, 200	#3	3701	7	>12000	NF(0.43)
	#4	3936	9	>12000	NF(0.41)
	#5	4053	10	>12000	>12000

Table 3.3: Computation results of (IPCC) and (RIPCC) for random graphs.

n,m	(IPCC)	($\overline{\text{IPCC}}$)	(RIPCC)	($\overline{\text{RIPCC}}$)	Cont.Rlx
	CPU	CPU	CPU	CPU	GAP(%)
22, 120	68.3	0.64	17.8	0.15	15.0
25, 125	156.9	1.1	37.3	0.32	10.4
27, 150	344.5	1.4	107.3	0.55	13.7
30, 200	1944.2	3.64	438.5	1.23	14.8
35, 200	-	9.56	1025.9	3.65	16.2
40, 250	-	20.67	3853.4	6.73	18.1
45, 200	-	28.39	3328.5	1.81	16.7
50, 200	-	45.31	11038.6	2.88	15.2
60, 150	-	327.63	5291.4	4.36	12.3
70, 140	-	-	10038.2	5.05	11.8
80, 120	-	-	8615.7	9.49	14.1

Chapter 4

Cutting plane approach for large size graphs: Lower bounds and efficient generation of heuristic solutions

Graph partitioning problem (e.g. GPP-SC) is known to be NP-hard in general. So in the last years a lot of effort has been spent in the development of fast and good heuristics for the problem, a recent survey is given in [13]. These heuristics often can handle rather large graphs with thousand nodes and deliver good solutions. As a natural requirement, the validation of the heuristics need to be verified.

Exact solutions can be used as important factors to validate of heuristics. However, only a little expense has been done in the development of exact algorithms [28], [58], [39], [11]. From the NP-hardness fact it is clear that generally only relatively small graphs can be solved exactly. To verify the validation of heuristics for large graphs, the linear relaxation of the 0/1 programming formulations is usually used to obtain lower bound for the problem. Among the programming models, the Node-Node model is known to give a good quality of the bound [11], [58], [43], [51]. Although Chapter 3 shows that one can reduce the number of triangle inequalities in the Node-Node model to $O(nm)$ without weakening the linear relaxation, the number of variables is still in order $O(n^2)$ that quickly becomes extremely large as n increases and even the continuous relaxation turns out to be difficult to solve. It is therefore interesting to be able to reduce intrinsically the number of variables of the problem in preserving the quality of linear relaxation.

We consider in this chapter an alternative programming model representing the node partitions of a graph partitioning problem that makes use of m decision variables $(x_e)_{e\in E}$ which are equal to 1 iff the end-nodes of e are in different clusters, together with the *cycle inequalities* defined on them [5], [15]. We prove that the node partitions and thus the knapsack constraints can be defined using all pair shortest path of G respected to the weights $(x_e)_{e\in E}$. The result is a m variable programming model for GPP-SC. This model is referred to as the *cycle model* and it is shown to be equivalent to the Node-Node model. Obviously, this model yields considerable improvement in case of sparse graphs where $m \ll \frac{n(n-1)}{2}$. Note that since there is

a priori no known polynomial upper bound (in terms of n and m) on the number of cycles and of paths of the graph G, the cycle model have a priori an exponential number of inequalities thus a cutting plane algorithm is needed to speed up the computations. We introduce a such algorithm which can determine both violated cycle inequalities and violated knapsack constraints in using all pair shortest path of G respected to the weights $(x_e)_{e\in E}$.

Computational experiments show that, when applying cutting plane algorithm, the cycle model outperform the Node-Node model with triangle inequalities in terms of computation time. With the new model, the linear relaxation of GPKC can be solved optimally for the large size graph of thousands nodes (see Section 4.4).

In Section 4.3, we study a progressive aggregation procedure that can build feasible partitions of the graph G satisfying the knapsack constraints in GPKC. Using this method and the new cycle model, we introduce an efficient generation of heuristic solutions. Computational results are discussed to show the quality of the new generation. We are not aware of any existing heuristic approach capable of consistently producing solutions with such guarantees of quality for graphs of comparable size.

In this chapter, we focus our studies on the special case GPKC of GPP-SC. However, it should be stressed that all the results in the forthcoming Section 4.2 and 4.3 will be applicable for all the class GPP-SC.

4.1 Preliminary: Node-Node formulation for GPKC

The Node-Node model for GPKC can be obtained as a special case of the Node-Node model for GPP-SC that was described in Chapter 3. The model makes use of $\frac{n(n-1)}{2}$ binary variables x_{uv} for all the pairs of nodes $u, v \in V$, $u < v$, such that

$$x_{uv} = \begin{cases} 0 & \text{if } u \text{ and } v \text{ belong to the same cluster,} \\ 1 & \text{otherwise.} \end{cases}$$

GPKC contains additional node weights constraints (of knapsack type) of the form $\sum_{v\in V_i} w_v \leq W$ for all $i = 1, \ldots, r$ and W is a given upper limit of the total node weight of the cluster. Using the above variable, the knapsack constraints can be formulated as:

$$\sum_{j\in V,\ j<i} w_j(1 - x_{ji}) + \sum_{j\in V,\ j>i} w_j(1 - x_{ij}) \leq W - w_i \quad \forall i \in V \tag{4.1}$$

Hence the Node-Node formulation for GPKC is:

$$
(\mathrm{NN})\begin{cases}
\min \sum_{(u,v)\in E} t_{uv}x_{uv} & \\
\text{s.t. } \forall (u,v,w)\in \mathcal{T} & \\
\quad x_{uv}+x_{uw}\geq x_{vw} & \\
\quad x_{uv}+x_{vw}\geq x_{uw} & \\
\quad x_{vw}+x_{uw}\geq x_{uv} & \\
\quad \sum_{j\in V,\ j<i} w_j(1-x_{ji}) + \sum_{j\in V,\ j>i} w_j(1-x_{ij}) \leq W - w_i & \forall i\in V \\
\quad x_{uv}\in\{0,1\} & \forall (u,v)\in E_n
\end{cases}
$$

Let LNN be the linear relaxation of NN. The number of constraints in NN is $O(nm)$ and thus it represents a compact 0-1 formulation for GPKC.

4.2 A m-variable formulation for GPKC and its solution via cutting-planes

4.2.1 A m-variables formulation for GPKC

Define $\pi = (V_i, i = 1,\ldots,r)$ to be a partition of G. We denote $E(\pi)$ the set of edges whose end-nodes are in two different clusters in the partition π, i.e.:

$$E(\pi) = \{(u,v)\in E | \{u,v\} \nsubseteq V_i, \forall i\in\{1,\ldots,r\}\}$$

Define the incidence vector $x(\pi)$ as:

$$x_e(\pi) = \begin{cases} 1 & \text{for } e\in E(\pi) \\ 0 & \text{otherwise.}\end{cases}$$

Reciprocally, given an incidence vector $(x_e)_{e\in E}$, we can build a partition π as follows:

> Let G' be a copy of G where the edges are weighted by $(x_e)_{e\in E}$. For all pair of node i and j, let P_{ij} be the set of all possible $i-j$ paths in G'. For all part $p_{ij}\in P_{ij}$, let $x_{p_{ij}}$ be its total path weight defined as $x_{p_{ij}} = \sum_{(u,v)\in p_{ij}} x_{uv}$. Hence node i and node j are assigned to the same cluster iff $\min_{p_{ij}\in P_{ij}} x_{p_{ij}} = 0$.

The so-called *cycle inequalities* together with the binary constraints can be used to define the partitions of a graph G [5]. They will be used here as part of the constraints for formulating GPKC. Denoting $\mathcal{C}$ the set of all cycles of G, these constraints can be written as:

$$(\mathrm{IC})\begin{cases} x_e - x_{C\setminus e}\leq 0 & \forall e\in C,\ \forall C\in\mathcal{C} \\ x_e\in\{0,1\} & \forall e\in E\end{cases}$$

Note that there is no known polynomial upper bound (in terms of n and m) on the number of cycles of G thus there is no known polynomial upper bound on the

number of cycle inequalities in (IC). It has been shown by Chopra [15] that for series-parallel graphs, the linear programming relaxation of (IC) characterize completely the convex hull of the incidence vectors of all the possible partitions of V, i.e. the vectors that are the solutions of (IC).
GPKC contains additional node weight constraints (of knapsack type) of the form $\sum_{v \in V_i} w_v \leq W$ for all $i = 1, \ldots, r$ and W is a given upper limit of the total node weight of the cluster. Using the above construction of the partition from a incidence vector, the knapsack constraints can be formulated as, for all node $i \in V$:

$$\sum_{j \in V,\ j \neq i} w_j (1 - \min\{1, \min_{p_{ij} \in P_{ij}} x_{p_{ij}}\}) \leq W - w_i \tag{4.2}$$

Let p_0 be a *virtual* path that is defined as $x_{p_0} = 1$, then (4.2) can be rewritten as, for all node $i \in V$:

$$\sum_{j \in V,\ j \neq i} w_j (1 - \min_{p_{ij} \in P_{ij} \cup \{p_0\}} x_{p_{ij}}) \leq W - w_i \tag{4.3}$$

Note that the left-hand side of (4.3) is a nonincreasing function with respect to $x_{p_{ij}}$, thus (4.3) is equivalent to the following linear system, for all node $i \in V$:

$$\sum_{j \in V,\ j \neq i} w_j (1 - x_{p_{ij}}) \leq W - w_i, \quad \forall p_{ij} \in P_{ij} \cup \{p_0\} \tag{4.4}$$

Hence the 0/1 linear formulation for GPKC is:

$$\text{(ICP)} \begin{cases} \min \sum_{e \in E} t_e x_e & \\ \text{s.t. } x_e - x_{C \setminus e} \leq 0 & \forall e \in C,\ \forall C \in \mathcal{C} \\ \quad \sum_{j \in V,\ j \neq i} w_j (1 - x_{p_{ij}}) \leq W - w_i & \forall i \in V,\ \forall p_{ij} \in P_{ij} \cup \{p_0\} \\ \quad x_e \in \{0, 1\} & \forall e \in E \end{cases}$$

Let LCP be the linear relaxation of ICP. Note that the number of knapsack constraints in LCP is extremely large due to a large number of paths in G and due to an exponential number of choices for the paths in each knapsack constraint. Hence LCP turns out to be difficult to solve. A separation method may help to speed up computation times.

Lemma 4.2.1. *Given $x \in [0,1]^{|E_n|}$ a feasible point of* LNN *then x_E is a feasible point of* LCP*. Reciprocally, given x^c a feasible point of* LCP*, there always exists $x \in [0,1]^{|E_n|}$ a feasible point of* LNN *such that $x_E = x^c$.*

Proof. "$\Rightarrow$" Given $x \in [0,1]^{|E_n|}$ a feasible point of LNN, Theorem 3.2.1 in Chapter 3 shows that, using x_E, we can build a partition of G satisfying the knapsack constraints in GPKC. Hence x_E is a feasible point of LCP.

"$\Leftarrow$" Given x^c a feasible point of LCP, we can always build a partition of G using the above construction. Constraints 4.4 in LCP show that this partition satisfies the knapsack constraint in GPKC. Hence the point $x \in [0,1]^{|E_n|}$ associated with this partition is a feasible point of LNN. It is obvious that $x_E = x^c$. □

Using the Lemma 4.2.1, we can show the equivalent of LNN and LCP in the following theorem:

Theorem 4.2.2. LNN *and* LCP *have the same optimal value.*

4.2.2 Solving the separation subproblem via shortest path computations

We introduce in this section a polynomial separation method both for cycle constraints and for knapsack constraints in LCP. Given an intermediate incidence vector $(\bar{x}_e)_{e\in E} \in [0,1]^{|E|}$, the method is to determine the constraints that are most violated by $(\bar{x}_e)_{e\in E}$. Note that the cycle constraints and the knapsack constraints can be written as:

$$\begin{cases} x_{ij} - \min\limits_{p_{ij}\in P_{ij}} x_{p_{ij}} \leq 0 & \forall (i,j) \in E \\ \sum\limits_{j\in V,\ j\neq i} w_j(1 - \min\limits_{\substack{p_{ij}\in P_{ij} \\ x_{p_{ij}}<1}} x_{p_{ij}}) \leq W - w_i & \forall i \in V \end{cases} \tag{4.5}$$

The system (4.5) contains $m+n$ constraint (polynomial in terms of m and n). Let $\bar{G}$ be a copy of G where the edges are weighted by $(\bar{x}_e)_{e\in E}$, the most violated constraints in (4.5) can be determined using the all pair shortest paths on $\bar{G}$ that can be computed in polynomial time. The violated constraints are integrated to the intermediate formulation as follows:

- If some cycle constraint in (4.5) is violated for a edge $(i,j) \in E$, let $\bar{p}_{ij}$ be the path that verifies $\bar{x}_{\bar{p}_{ij}} = \min\limits_{p_{ij}\in P_{ij}\setminus\{(ij)\}} x_{p_{ij}}$ (i.e. shortest path), hence the cycle constraint $x_{ij} - x_{\bar{p}_{ij}} \leq 0$ is appended to the current formulation.
- If some knapsack constraint in (4.5) is violated for a node $i \in V$, for all $j \in V$ and $j \neq i$, let $\bar{p}_{ij}$ be the path that verifies $\bar{x}_{\bar{p}_{ij}} = \min\limits_{\substack{p_{ij}\in P_{ij} \\ x_{p_{ij}}<1}} x_{p_{ij}}$ (i.e. shortest path), hence the knapsack constraint $\sum_{j\in V,\ j\neq i} w_j(1 - x_{\bar{p}_{ij}}) \leq W - w_i$ is appended to the current formulation.

The number of violated constraints which are added in each iteration is at most $m+n$ and moreover these are the most violated constraints.

4.2.3 Efficient implementation of the cutting plane algorithm

4.2.3.1 Eliminating constraints featuring largest slacks

Cutting-plane is a method frequently used to solve large linear problems featuring huge number of constraints. Experiments show that, using a cutting-plane algorithm on a linear formulation, the solvers make use of fewer constraints, as compared with the complete formulation. Practically, the violated constraints are determined in each iteration and part of them (or all of them) are integrated to the next iteration.

By doing so, we expect that only necessary constraints are used to solve the problem. However, experiments show that, many unnecessary constraints are also included in the course of the computations. We suggest here a method to remove as much as possible the unnecessary constraints in each iteration and then limit the number of constraints.

We focus our improvement on the knapsack constraints. The main idea being to ensure that the number of knapsack constraints never exceeds $k \times m$ for same parameter k, typically chosen in the range $[3, 5]$. This is achieved by applying the following simple rule : if, at some stage of the procedure, p knapsack constraints are already present in the current formulation, and q violated inequalities are generated, then if $p + q \geq k \times m$ then $p + q - k \times m$ constraints having largest slacks are deleted.

4.2.3.2 Truncated shortest path computations

We provided a polynomial separation method for LCP using the computation of all pair shortest path of G with respect to weights x. Although the all pair shortest path can be computed in polynomial time, but when repeating this algorithm in each iteration of the separation procedure, this may cause of large solution time for the model. Computation results in Section 4.4 show that the shortest path computation times are accounted for about 2/3 of the total computation times. Reducing the computation time of shortest path algorithm is thus a necessity.

Note that the cycle constraints and the knapsack constraints can be written as:

$$\begin{cases} x_{ij} - \min\limits_{\substack{p_{ij} \in P_{ij} \\ x_{p_{ij}} < 1}} x_{p_{ij}} \leq 0 & \forall (i,j) \in E \\ \sum\limits_{j \in V,\ j \neq i} w_j (1 - \min\limits_{\substack{p_{ij} \in P_{ij} \\ x_{p_{ij}} < 1}} x_{p_{ij}}) \leq W - w_i & \forall i \in V \end{cases} \tag{4.6}$$

It can be observed that only the paths that have weights less than 1 are necessary for the separation procedure. This observation leads us to modify the shortest path algorithm in the following ways:

- All paths are set to be 1 at the input.
- Only the paths that have weights less than 1 are computed.
- The paths that are not computed (i.e. the weights are more than 1), remain of weight 1.

For all the numerical experiments in Section 4.4, we make use of Dijkstra's shortest path algorithm [**?**]. The truncated version of this algorithm (the original pseudo-code can be found in [**?**]) can be presented as follows.

```
Result: Truncated-Dijkstra(Graph, source)
dist[source] ⟵ 0
create vertex set Q
for vertex v in Graph do
    if v ≠ source then
        dist[v] ⟵ 1
        prev[v] ⟵ UNDEFINED
    end
    Q.add-with-priority(v, dist[v])
end
while Q is not empty do
    u ⟵ Q.extract-min()
    if dist[u]=1 then
        break the while loop
    end
    for neighbor v of u do
        alt = dist[u] + length(u, v)
        if alt < dist[v] then
            dist[v] ⟵ alt
            prev[v] ⟵ u
            Q.decrease-priority(v, alt)
        end
    end
end
return dist[], prev[]
```

Algorithm 1: Truncated Dijkstra's algorithm

This modification is denoted truncated shortest path algorithm. Generally, the truncated algorithm contains less operations than the original ones thus it can be computed in polynomial time. The complexity is unchanged since in the worst case, all the paths are computed. However, in practice, the computation times are significantly reduced (see Section 4.4).
In the computational experiments discussed in Section 4.4 below, the implementation of LCP using the techniques in Section 4.2.3.1 and Section 4.2.3.2 is referred to as iLCP ("improved LCP").

4.3 Efficient computation of upper bounds: heuristic solutions

NN and its equivalent ICP give a good quality of continuous relaxation (i.e. LNN and LCP) [51], our experiments (see section 4.4) show that the gaps are less than 10% for series-parallel graphs and less than 20% for random graphs. Experiments also show that for series-parallel graphs, the fractional solution vector x of LCP contains many integer components (i.e. 0 and 1). This phenomenon leads us, in this section, to find an efficient method of building feasible partitions (i.e. integer points $x^{\dagger}$) of ICP from a solution incidence vector x of LCP on series-parallel graphs. Base

on $x^\dagger$, we obtain an upper bound for ICP and hence for GPKC.

4.3.1 Building feasible partitions: upper rounding procedure (UR)

Let us show a basic deduction of $x^\dagger$ from x as follows, for all $x_e,\ e \in E$:

$$x_e^\dagger = \begin{cases} 0 & \text{if } x_e = 0 \\ 1 & \text{if } x_e > 0 \end{cases}$$

We prove that $x^\dagger$ obtained form such construction is a feasible point of ICP. Indeed, by the construction, we have trivially $x^\dagger \geq x$, thus $x^\dagger$ satisfies the knapsack constraints since these constraints form nonincreasing functions with respect to x. It remains to show that $x^\dagger$ satisfies also the cycle inequalities, indeed, for all $x_e,\ e \in E$:

- if $x_e = 0$, $x_e^\dagger = 0$ and the cycle inequality $x_e^\dagger - x_{C\backslash e}^\dagger \leq 0$ is satisfied trivially.
- if $x_e > 0$, $x_e^\dagger = 1$. The cycle inequality $x_e - x_{C\backslash e} \leq 0$ is satisfied thus there exist an edge $e^{'} \in C\backslash e$ such that $x_{e'} > 0$. We deduce that $x_{e'}^\dagger = 1$ hence the cycle inequality $x_e^\dagger - x_{C\backslash e}^\dagger \leq 0$ is satisfied.

The above deduction of $x^\dagger$ form x is called upper rounding procedure (UR). It seems to be too trivial and cannot convince ourselves that it is the method leading to the most optimized results. Moreover, this method is reasonable only in the case when there are a large number of integer components in x (e.g. for series-parallel graph). We aim in the following, to study a more advanced method that can be applied for all graph types and it will give better upper bounds than UR.

4.3.2 Building feasible partitions: progressive aggregation procedure (PA)

We study in this section a more advanced method to build feasible partitions of GPKC based on the relaxed solution x. Experiments in Section 4.4 report that x give good lower bounds for GPKC and many components of x close to integer values (i.e. 0 or 1). This report leads us to have two important remarks as follows:

- For an edge $(u, v) \in E$ such that x_{uv} close or equal to 0, there is large possibility that u and v are in the same cluster.
- For two edges (i, j) and (u, v) in E such that $x_{ij} < x_{uv}$, the possibility that i and j are in the same cluster is higher than the one of u and v.

The remarks are important basis for us to establish a progressive aggregation procedure (PA) for feasible partitions of GPKC that use the components of the current fractional solution to define the order according to which the aggregation takes place. Throughout the procedure, the knapsack constraints are verified. The procedure can be stated as follows.

```
Result: Progressive-aggregation(Graph, x)      # Graph G=(V,E)
create partition P of V #i.e.
  P = {V1, ..., Vk} | V1 ∪ ... ∪ Vk = V, Vi ∩ Vj = ∅ ∀1 ≤ i ≠ j ≤ k
P ⟵ {{1}, {2}, ..., {n}}          #P is the trivial partition
x^s ⟵ sort(x)                      # Use sort(x) as priority indicator array
E^s ⟵ edge-array(x^s)              # Extract edge array in order of x^s
for edge e in order in E^s do
    (u, v) ⟵ edge-end-nodes(e)      #Extract end-nodes of the edge e
    s_u ⟵ find-set(P, u)        #Find cluster s_u in P that contains u
    s_v ⟵ find-set(P, v)        #Find cluster s_v in P that contains v
    if s_u ≢ s_v then
        temp ⟵ fusion(s_u, s_v)  # Union of two cluster
        temp-weight ⟵ weight(temp)   # Weight of the temporary cluster
          temp
        if temp-weight ≤ W then
                                          #Verify the knapsack constraints
            remove(P, s_u)                #remove cluster s_u
            remove(P, s_v)                #remove cluster s_v
            add(P, temp)                  #add the union of cluster s_u and s_v
        end
    end
end
return P                    # Return the partition matrix
```

Algorithm 2: Progressive aggregation procedure

This progressive aggregation is an improvement of the upper rounding procedure. Indeed, in PA the relaxed solution are sorted to as the priority indicator array thus the edges e such that $x_e = 0$ are considered first to the aggregation of nodes. Hence UR is a first part of running PA. We conclude that PA can only produce upper bounds atleast as good as with UR.

4.4 Numerical experiments

In this section, numerical experiments are done to verify our results in Section 4.2 and Section 4.3. We used the same instances generated as in Chapter 3.
All experiments are run on a machine with Intel Core i7-3630QM 2.40GHz processor and 16 GiB of RAM. The solver GUROBI 6.5 is used to solve the programming models while GUROBI pre-solve switched off. All computation times are CPU seconds and the computation times are subject to a *time limit of 3600 seconds.*

4.4.1 Experiments for the cycle model using the cutting plane algorithms

In this section, we show the comparison between the Node-Node model and the cycle model when applying the cutting plane algorithms. The number of constraints in the

final iteration of cutting plane algorithms are also presented beside the computation times (CPU).
We first solve LCP using the original cutting plane algorithm proposed in Section 5.4 (i.e. without the improvement techniques in Section 4.2.3). As shown in Table 5.1, LCP outperform LNN in terms of computation times and it is obvious that they have the same gaps. In terms of number of constraints in the final iteration, LCP use much less than LNN. For large instances or for more complex graphs, the number of constraints in LNN becomes extremely large and it obviously cannot be solved on the time limit (3600 seconds). LNN can only solve small instances: maximum 500 nodes for series-parallel graphs, 400 nodes for planar grid graphs and 225 nodes for toroidal grid graphs.

Table 4.1: Computation results of LCP and LNN for sparse graphs.

graph types	n,m	(LNN)		(LCP)		Cont.Rlx
		CPU	Nb Ctr	CPU	Nb Ctr	GAP(%)
SP	50, 78	0.85	4211	0.3	1053	5.8
SP	100, 164	14.6	18241	1.2	3124	8.1
SP	150, 237	23.1	36733	4.8	6228	7.9
SP	200, 300	44.7	65164	11.5	7634	-
SP	300, 474	148	148325	34.2	13875	-
SP	400, 598	295	251456	61.2	21066	-
SP	500, 753	573	398427	117	30169	-
SP	600, 956	>3600	-	264	42352	-
SP	800, 1221	>3600	-	661	55738	-
PG 8×8	64, 112	1.9	7162	0.6	1512	6.9
PG 10×10	100, 180	19.5	20045	2.1	3730	8.3
PG 15×15	225, 420	103	96281	27.4	11422	-
PG 20×20	400, 760	1376	311765	159	28218	-
PG 25×25	625, 1200	>3600	-	3383	63870	-
TG 8×8	64, 128	1.9	7564	0.9	2031	11.5
TG 10×10	100, 200	28.1	21544	2.5	4587	10.2
TG 15×15	225, 450	431	109362	31.2	12178	-
TG 20×20	400, 800	>3600	-	267	30572	-

To show more the benefit of using the cycle model as compared with the Node-Node model, we experience LNN using simple separation method obtained by numbering the triangle inequalities. This separation method can be described as follows.

> Given an intermediate incidence vector $(x_{uv})_{(u,v)\in E_n} \in [0,1]^{\frac{(n-1)n}{2}}$, the method is to determine the inequalities that are violated by $(x_{uv})_{(u,v)\in E_n}$. The number of triangle inequalities is $O(nm)$ thus polynomial in terms of n, the violated inequalities are therefore separable in polynomial time. we propose a choice of violated triangle inequalities as follows. For each edge (u,v) in E, if w is a random node in $V\backslash\{u,v\} = \{1,\ldots,n\}\backslash\{u,v\}$ such that one of three triangle inequalities corresponding to the triple

(u, v, w) are violated, then all three inequalities are chosen (valid inequalities are also added to the intermediate formulation). Note that in each iteration, the number of additional inequalities is at most $|E| = 3m$ and they are unbiasedly spread over the edges in E.

We note SNN as LNN when applying in the cutting plane algorithm. Computational results comparing LCP and SNN are shown in the Table 4.2. As we can see in the table, LCP outperforms SNN in term of computational times, specially for the planar grid graphs and the toroidal grid graphs. LCP can solve larger instances than SNN although the latter is already better than LNN. In terms of the number of constraints at the final iteration, LCP contains much less than SNN that seems to be the reason why it run faster.

Table 4.2: Computation results of LCP and SNN for sparse graphs.

graph types	n,m	(SNN)		(LCP)	
		CPU	Nb Ctr	CPU	Nb Ctr
SP	100, 164	0.9	3241	0.8	3124
SP	200, 300	8.9	15164	8.5	7634
SP	400, 598	58.0	81456	51.2	21066
SP	600, 956	344	210583	264	42352
SP	800, 1221	>3600	-	661	55738
PG 10×10	100, 180	3.3	12336	2.1	3730
PG 20×20	400, 760	451	227178	159	28218
PG 25×25	625, 1200	>3600	-	3383	63870
TG 10×10	100, 200	8.2	21544	2.5	4587
TG 15×15	225, 450	299	139362	31.2	12178
TG 20×20	400, 800	>3600	-	267	30572

As we indicated in Section 4.2.3, the computation of shortest path algorithm may be the cause of a large computation time for optimal solution. As we report in Table 4.3, the shortest path algorithm can be accounted from 57% to 73% total resolution times of LCP. This report leads us to do the experiments in the next section to show the benefit of using the improvement techniques represented in Section 4.2.3 on the cutting plane algorithm.

4.4.2 Experiments for the cycle model using efficient implementation of the cutting plane algorithm

The numerical results in this section show the benefit of using improved cutting plane algorithm for LCP, that we note as iLCP, as compared to the original one (i.e. LCP) both in terms of computation times and of number of constraints. For all the computation of iLCP represented in this section, the threshold slacks S_k were chosen in order to limit the number of knapsack constraints by $3 \times m$.
As shown in Table 5.2, iLCP outperforms LCP. When applying iLCP, a large number of unnecessary constraints are removed in the formulation, the number of constraints in iLCP is from two to seven times less than LCP. As a consequence, iLCP is much

Table 4.3: Report on the resolution times

graph types	n,m	(LCP)		
		CPU	% separation	% short path
SP	50, 78	0.1	62	57
SP	100, 164	0.8	73	60
SP	150, 237	3.0	75	68
SP	200, 300	8.4	80	70
SP	300, 474	19.9	78	70
SP	400, 598	28.2	71	65
SP	500, 753	43.5	76	66
SP	600, 956	118	80	72
SP	800, 1221	306	68	59
SP	1000, 1569	918	70	65
SP	2000, 3117	3208	78	73

faster than LCP in terms of computation times. iLCP can solve larger instances than using LCP: up to 2000 nodes for series-parallel graphs, 625 nodes for toroidal grid graphs.

4.4.3 GPKC upper bound computation

In this section, we present the computational results on the upper bounds for GPKC on series-parallel graphs using the upper rounding procedure and the progressive aggregation procedure that are presented in Section 4.3. To show the quality of our method, we compute the gap between the lower bounds given by iLCP and the upper bound deduced from its solution, we note this gap as *Res* in the table of results.

As shown in Table 4.5, PA give better upper bounds than UR. The relaxation solution of planar grid instances and toroidal grid instances contain too few integer components thus UR is not applicable. PA shows the quality for all the graph types. As we can see in the table, the gaps are low, specially for the case of series-parallel graphs (less than 10%). We report also that the running times for both two methods are almost for free.

We note that the progressive aggregation procedure can be applied not only on the relaxed solution but also on any value of the priority indicator array. This remark leads us to improve PA in finding the best heuristic solution given when applying PA on each rounds of the constraint-generation procedure. The new method is denoted iPA ("improved PA") and the numerical experiments are shown in Table 4.6.

As can be seen in Table 4.6, iPA gives slightly better heuristic solutions than PA. On the other hand, iPA runs significantly lower than PA. This report make iPA poor practical. However, we report also that the best heuristic solutions can be obtained on several last rounds of the constraint-generation procedure thus we only need to apply the progressive aggregation procedure on these rounds and the the computation times can be significantly reduced. However due to lack of time, this implementation can only be done in future works.

Table 4.4: Improved versus original cutting plane algorithm

graph types	n,m	(LCP)		(iLCP)	
		CPU	Nb Ctr	CPU	Nb Ctr
SP	50, 78	0.3	1053	0.1	432
SP	100, 164	1.2	3124	0.4	838
SP	150, 237	4.8	6228	1.0	1037
SP	200, 300	11.5	7634	3.6	1576
SP	300, 474	34.2	13875	7.9	2602
SP	400, 598	61.2	21066	12.7	3488
SP	500, 753	117	30169	19.1	4384
SP	600, 956	264	42352	38.3	5182
SP	800, 1221	661	55738	134	7333
SP	1000, 1569	>3600	-	449	9566
SP	2000, 3117	>3600	-	1892	17594
PG 8×8	64, 112	0.6	1512	0.1	591
PG 10×10	100, 180	2.1	3730	0.7	969
PG 15×15	225, 420	27.4	11422	7.8	2501
PG 20×20	400, 760	159	28218	39	4733
PG 25×25	625, 1200	3383	63870	342	7427
TG 8×8	64, 128	0.9	2031	0.4	671
TG 10×10	100, 200	2.5	4587	1.1	1203
TG 15×15	225, 450	31.2	12178	8.4	2405
TG 20×20	400, 800	267	30572	54	5341
TG 25×25	625, 1250	>3600	-	511	7759

4.4.4 Convergence profile of the cutting plane algorithm

In this section, we illustrate the convergence of LCP using the improvement techniques proposed in Section 4.2.3 (i.e. iLCP). The relaxed solutions given by iLCP can be used as lower bounds for the problem. Note that in using cutting plane algorithms, a lower bound of GPKC can usually be obtained even if the computation process is stopped before a solution is found, and we gain much of computation time. The quality of this bound depends on the time of which the computation process is stopped. Hence to evaluate the quality of a cutting plane algorithm, the convergence profile is also an important reference.
Figure 4.1 shows the convergence of isCP in terms of factors on the number of variables (i.e. number of edges). The Figure shows that the separation procedure approaches the relaxed solution rapidly.

4.5 Conclusion

In the present chapter we investigate a new formulation for the graph partitioning problem under knapsack constraints (GPKC) that makes use of only $O(m)$ variables. As there are an exponential number of constraints in the formulation, we introduce a separation method to speed up the computation times. The separation method

Table 4.5: Progressive aggregation versus upper rounding for GPKC.

graph types	n,m	(UR)		(PA)		Cont.Rlx
		CPU	Res(%)	CPU	Res(%)	GAP(%)
SP	50, 78	0.1	10.2	0.1	8.4	5.8
SP	100, 164	0.4	17.1	0.4	9.3	8.1
SP	150, 237	1.0	16.6	1.0	10.4	7.9
SP	200, 300	7.9	18.8	8.1	8.2	-
SP	400, 598	12.7	11.5	13.0	9.9	-
SP	500, 753	19.1	20.2	19.2	7.5	-
SP	600, 956	38.3	14.3	38.5	5.9	-
SP	800, 1221	134	15.4	136	8.9	-
SP	1000, 1569	449	13.4	451	9.2	-
SP	2000, 3117	1892	16.2	1901	7.0	-
PG	64, 112	-	-	0.1	14.2	10.1
PG	100, 180	-	-	0.7	11.4	-
PG	400, 760	-	-	40	10.1	-
PG	625, 1200	-	-	345	15.6	-
TG	64, 128	-	-	0.4	16.8	11.7
TG	100, 200	-	-	1.2	13.1	-
TG	225, 450	-	-	8.6	19.0	-
TG	400, 800	-	-	55	12.8	-
TG	625, 1250	-	-	515	15.2	-

makes use of all pair shortest path algorithm thus is in polynomial time. Numerical experiments comparing our new formulation with the classical Node-Node formulation have been presented to compute lower bounds for GPKC. These numerical results show that, the new formulation outperforms the Node-Node formulation in term of resolution times, while preserving the quality of lower bounds. The new formulation succeeds at finding relaxed solutions for large size graphs up to 2000 nodes.

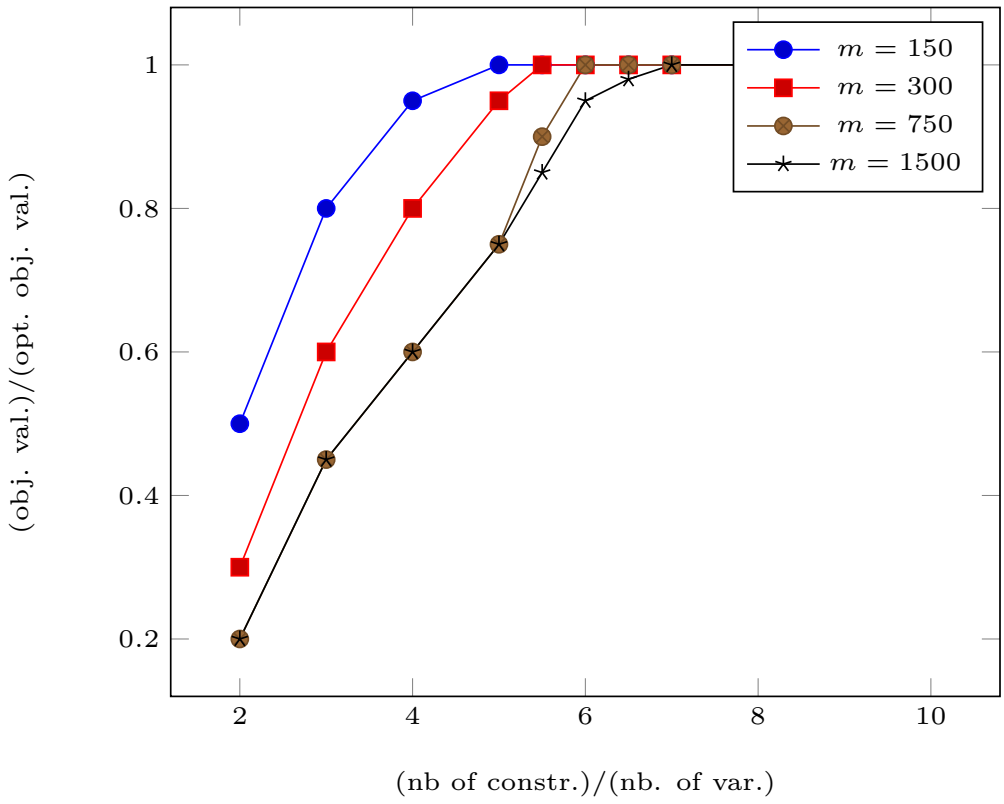
In addition, we introduce a generation of feasible partitions for GPKC based on the relaxed solution. This generation follows a progressive aggregation procedure that makes use of the relaxed solution as priority indicator. Numerical experiments show that our generation gives tight upper bounds for GPKC, the gaps are less than 20% in general and less than 10% for series-parallel graphs.

Although in the present chapter, only GPKC has been investigated, but all the results will turn out to be applicable to general GPP-SC problem.

Table 4.6: Improved progressive aggregation procedure for GPKC.

graph types	n,m	(PA)		(iPA)	
		CPU	Res(%)	CPU	Res(%)
SP	50, 78	0.1	8.4	0.2	8.4
SP	100, 164	0.4	9.3	0.6	9.0
SP	150, 237	1.0	10.4	1.5	10.4
SP	200, 300	8.1	8.2	11.8	7.8
SP	400, 598	13.0	9.9	20	9.9
SP	500, 753	19.2	7.5	27.1	7.5
SP	600, 956	38.5	5.9	51.7	5.9
SP	800, 1221	136	8.9	187	8.0
SP	1000, 1569	451	9.2	655	8.5
SP	2000, 3117	1901	7.0	2470	7.0
PG	64, 112	0.1	14.2	0.1	13.5
PG	100, 180	0.7	11.4	1.0	10.6
PG	400, 760	40	10.1	58.5	10.1
PG	625, 1200	345	15.6	463	15.0
TG	64, 128	0.4	16.8	0.6	15.1
TG	100, 200	1.2	13.1	1.7	13.0
TG	225, 450	8.6	19.0	11.3	17.4
TG	400, 800	55	12.8	75.5	12.8
TG	625, 1250	515	15.2	714	14.9

Figure 4.1: Convergence of iLCP



Chapter 5

Stochastic graph partitioning

5.1 Stochastic programming

5.1.1 Optimization under uncertainty, an overview

A large majority of algorithms and methods intended to solve combinatorial optimization problems suppose that input data are known precisely. As such, a generic way of mathematically representing an optimization problem is as follows:

$$\begin{aligned} \min_{x} \quad & f(x) \\ \text{s. t.:} \quad & G_i(x, \xi_i) \leq 0 \quad i \in \{1, \ldots, m\} \end{aligned} \tag{5.1}$$

where $x \in \mathbb{R}^n$ is the design parameter, $f(x) \in \mathbb{R}$ is the objective function and we have m inequality constraints $G(x, \xi) \in \mathbb{R}$ with $\xi = (\xi_1, \ldots, \xi_m)$ a m-dimensional parameter vector.

However, for real-world optimization problems, one might ask if the formulation above is as general and practical as it seems since the design space is often characterized by data which are uncertain or inexact.

Beginning with the seminal works of Dantzig [19], Charnes and Cooper [14], Miller and Wagner [49], Bellman and Zadeh [6], optimization under uncertainty is an extremely active domain of research, both in theory and algorithms, and thanks to recent studies, there is an increased regain of interest. The recent case study of Ben-Tal and Nemirovski [8] on a collection of 90 problems from NETLIB library showed that systems optimized in the classical sense (see formulation 5.1) can be very sensitive to small changes and that only 1% perturbation of the data can severely affect the feasibility properties of deterministic solutions.

One such example from Ben-Tal et al. [8] is an antenna design problem in which only 5% errors can entirely destroy the radiation characteristics established during nominal optimization. Another example analyzed in [8] is a LP program PILOT4

from Netlib library with 1000 variables and 410 constraints, constraint j being:

$$\begin{aligned}[Aj]Tx \equiv -15.79081x_{826} - 8.598819x_{827} - 1.88789x_{828} - 1.362414x_{829} \\ -1.526049x_{830} - 0.031883x_{849} - 28.725555x_{850} - 10.792065x_{851} \\ -0.19004x_{852} - 2.757176x_{853} - 12.290832x_{854} + 717.562256x_{855} \\ -0.057865x_{856} - 3.785417x_{857} - 78.30661x_{858} - 122.163055x_{859} \\ -6.46609x_{860} - 0.48371x_{861} - 0.615264x_{862} - 1.353783x_{863} \\ -84.644257x_{864} - 122.459045x_{865} - 43.15593x_{866} - 1.712592x_{870} \\ -0.401597x_{871} + x_{880} - 0.946049x_{898} - 0.946049x_{916} \\ \geq b \equiv 23.387405\end{aligned}$$

. with $A_j \in \mathbb{R}^n$ the line j of the constraints matrix and $x \in \mathbb{R}^n$. This kind of "ugly" coefficients could model certain technological processes and we could make the hypothesis that they cannot be specified with high accuracy and thus, they are uncertain and have inaccurate last digits. For the optimal solution x^* when the uncertain coefficients are perturbed within 0.01% margin by independent random perturbations, distributed uniformly, the constraint is violated by at most 150% of the right hand side with a probability of 0.18. In the worst case (all uncertain coefficients are perturbed with 0.01%), the constraint is violated in x^* by 450% of the right hand side.

Let us give another simple example illustrating that the optimal solution of problem 5.1 might actually be unfeasible if uncertainty on the parameter vector ξ is ignored. Let $\xi = (\xi_1, \ldots, \xi_m)$ with $\xi_1, \ldots, \xi_m$, m independent observations of a standard normal distribution, $x \in \mathbb{R}$, $f(x) = x$ and $G_i(x, \xi) = \xi_i - x$, for all $i = 1, \ldots, m$. If the uncertainty on the parameter is ignored and ξ is substituted by the expected value $E(\xi)$ in problem 5.1, then the optimal solution is obtained for $x^* = 0$. However, the probability that $x^* = 0$ is a feasible solution equals to

$$\mathbb{P}_i\{G_i(x^*, \xi_i) \leq 0, \forall i \in \{1, \ldots, m\}\} = \mathbb{P}\{x^*\xi_i, \forall i \in \{1, \ldots, m\}\} = 0.5^m$$

As the value of m increases, this probability becomes very small (e.g. for $m = 7$, it is less than 0.01).

As shown by the previous case studies, taking into account uncertainty impacting the parameters required for optimization is necessary in order to find optimal solutions which are feasible in a meaningful sense. Nevertheless, as we will point out in the next section, optimizing under uncertainty induces several additional difficulties and a crucial point is the way uncertainty is formalized and the underlying assumptions.

5.1.2 Chance constrained programming

Chance constrained programming [14] is one of the standard methods for handling uncertainty in optimization, dealing with constraints of the form:

$$\mathbf{P}(g(x, \xi) \leq 0) \geq 1 - \varepsilon \tag{5.2}$$

where $x \in \mathbf{R}^n$ is the decision vector, $\xi \in \mathbf{R}^a$ a random variable and $g : \mathbf{R}^n \times \mathbf{R}^a \to \mathbf{R}^b$ a constraint mapping. The level $\varepsilon \in (0, 1)$ is user given and defines

the preference for safety of the decision x. The constraint (5.2) means that we wish to take a decision x that satisfies the b-dimensional random inequality system $g(x, \xi) \leq 0$ with high enough probability. To investigate with the constraint (5.2) using chance constrained programming, ξ is usually assumed well-characterized and that knowledge of the distribution is available. Such a situation naturally arises when ξ has been investigated by statisticians.

Two conceptually different versions of (5.2) exist and are referred to as Individual Chance Constraints (ICC) or Joint Chance Constraints (JCC). The equation appearing in (5.2) is a version of a Joint Chance Constraint. A deduced set of Individual Chance Constraints would be:

$$\mathbf{P}(g_i(x, \xi) \leq 0) \geq 1 - \varepsilon_i, \ i = 1, \ldots, b \tag{5.3}$$

where g_i refers to the i-th component of the mapping g and ε_i are arbitrary choices. The situation of (5.3) refers to a situation wherein we wish to satisfy each individual equation in the random b-dimensional inequality system $g(x, \xi) \leq 0$ with high enough probability, but we make no request on the system as a whole.

5.1.3 Convexity studies

Some of the earliest studies from stochastic optimization were interested in establishing conditions in which the probabilistic distributions and the functions defining the constraints define a convex feasible space. As such, almost all exact solutions existing for chance constrained programs require a continuous distribution and a convex structure of the problem.

Charnes and Cooper [14] studied the case of single chance constraints ($m = 1$) when the continuous random variables are only on the right hand-side of the constraints (i.e. completely decoupled of the decision variables) and proposed a deterministic nonlinear equivalent problem. Also, when $m = 1$ and the randomness is continuous and on the left hand-side, Kataoka [42] proved that these chance constrained programs are convex for independently normal distribution and $\varepsilon \geq 0.5$.

For joint chance constrained programs with more than one constraint, the most difficult case to solve is the one in which the random distributions are affecting the left hand-side. Instead, for random right hand-side parameters, Prékopa [52] showed that if the random variables have a log-concave distribution (e.g. the multivariate normal distribution, uniform distribution are log-concave), then the initial probabilistic program can be rewritten as a convex deterministic equivalent problem. Prékopa also proved that for normal distributed left hand-side parameters, if all covariance and cross-variance matrices for columns or rows are proportional between them, then the problem is convex. A later study of Henrion [36] gives convexity conditions in which program for left hand-side random parameters normally distributed with independent components, and improved convexity conditions have been investigated in [50].

To the best of our knowledge, existing studies determined convexity conditions only for linear probabilistic constraints with normal distributions on the left hand-side or log-concave distributions on the right hand-side.

5.1.4 Stochastic graph partitioning

Previous work related to the stochastic form of the problem treated in the present dissertation is quite scarce. Fan et Pardalos [23] studied a problem relatively close to ours: partition the vertex set of a graph into several disjoints subsets so that the sum of weights of the edges between the disjoint subsets is minimized, with a cardinality constraint on each subset and the uncertainty affecting the edge weights. In Fan et al. [23], assuming there is no information on the probability distribution other than that the weights on the links are independent and bounded in known intervals, they formulate the problem using a robust optimization model, similar to Bertsimas et al. [10]. The equivalent linear programming formulation is then solved by an algorithm based on a decomposition method. In a more recent study Fan et al. [24], introduce a two-stage stochastic graph partitioning problem, assuming that the distribution of edge weights has finite explicit scenarios. Having as objective to minimize the expected weight of edges in the set of cuts over all scenarios, they present a nonlinear stochastic mixed integer model and propose an equivalent integer programming formulation for solving the problem using CPLEX. Taskin et al. [61] study the stochastic edge-partition problem, where the edge weights are uncertain, and are realized only after the node-to-subgraph assignments have been made. They introduce a two-stage cutting plane algorithm with integer variables in both stages and, to overcome the computational difficulties, they also prescribe a hybrid integer/constraint programming method as an alternative.

The approaches above differ in several aspects from our study. First, in our case, the problem formulation is not the same, dealing with multidimensional capacity constraints on the nodes instead of cardinality constraints. Consequently, uncertainty is addressed in a different manner, the assumption of uncertainty being made on the weights of the nodes rather than on the weights of the edges. Finally, we remark that the existing methods are exact and thus, mostly suited for small-size instances of the problem, the numerical experiments being performed on graphs with at most 100 nodes. On the contrary, for the processes placement problem, we are interested in practice to partition much larger graphs.

In this chapter, we focus our studies on the case of GPKC with the presence of uncertainty on the node weights. We note that all the results in this chapter remain applicable for GPP-SC.

5.1.5 Stochastic graph partitioning under knapsack constraint formulation

In this thesis, we chose dealing with the uncertainties by Individual Chance Constrained programming, hence the stochastic knapsack constraints (5.6) can be reformulated as :

$$\mathbf{P}(\sum_{\substack{v \in V \\ v \neq u}} (1 - x_{vu}) w_v \leq W) \geq 1 - \varepsilon \quad \forall u \in V \tag{5.4}$$

where we chose the same probability level for all constraints.

Hence the stochastic graph partitioning under knapsack constraints is reformu-

lated as follows :

$$
(\text{SGPKC}) = \begin{cases}
\min & \sum_{(u,v)\in E} t_{uv} x_{uv} \\
\text{s. t.:} & x_{uv} + x_{uw} \geq x_{vw} & (u,v,w) \in \mathcal{T} \\
& x_{uv} + x_{vw} \geq x_{uw} & (u,v,w) \in \mathcal{T} \\
& x_{vw} + x_{uw} \geq x_{uv} & (u,v,w) \in \mathcal{T} \\
& \mathbf{P}(\sum_{\substack{v\in V \\ v\neq u}} (1 - x_{vu}) w_v \leq W) \geq 1 - \varepsilon & u \in V \\
& x_{uv} \in \{0,1\} & (u,v) \in E_n
\end{cases} \tag{5.5}
$$

.

To investigate the formulation (5.5) in an efficient way, we have to overcome two main difficulties : the large number of triangle constraint ($O(n^3)$) and the complexity of the chance constraints. In Chapter 3.1 we propose an improved formulation for GPKC in the case of sparse graphs; in Chapter 3.2 we introduce several reformulations of the stochastic graph partitioning problem (SGPKC) and compare them computationally.

5.2 Problem of partitioning process networks under uncertain processing time

As a typical example of application of GPKC, we can mention a problem arising in the field of compilation for real-time embedded systems, the partitioning of process networks on a clusterized parallel architecture. This problem is an extension of the standard problem of graph partitioning, which is known to be NP-hard [31]. The specific class of partitioning problems considered in this paper consists in assigning the weighted nodes of a graph to clusters (representing the processors), in order to minimize the sum of costs for edges having their endpoints in different clusters, without exceeding the limited capacity of each clusters (e.g. the memory footprint) and by taking into account uncertainty affecting on vertex weights. Known for the one-dimensional and deterministic case as the Node Capacitated Graph Partitioning problem [28], the stochastic version of the problem does not seem to have been investigated so far except from a non-parametric and approximate resolution view point [59].

In this problem, one of the main sources of uncertainties lies in the intrinsic indeterminism of execution times for computing kernels of intermediate granularity. This indeterminism is due in part to some of the characteristics of the processor architecture such as the cache memories and memory access controllers and is also inherently due to data dependent control flows (conditional branches and loops). The distributions of processing times are often complex, sometimes giving use to multimodal distributions (due to the presence of data dependent control).

The example above shows that the uncertainties can affect the node weights of the graph G hence we have to deal the stochastic version of GPKC (i.e. SGPKC). To handle the problem with chance constrained programming, we assume that the

uncertainties nodes weights w following a multivariate Gaussian distribution. Moreover, even when the Gaussian assumption on the random variables is not verified, the use of multivariate Gaussian approximation is still reasonable, based on the extended central limit theorem. Indeed if we assume that for a given u, the number of x_{vu} variables equal to 0 (i.e the cardinality of the cluster containing u) is sufficiently large (typically more than 30-40), the stochastic knapsack constraints

$$\sum_{\substack{v \in V \\ v \neq u}} (1 - x_{vu}) w_v \leq W \quad \forall u \in V \tag{5.6}$$

involve a combination of sufficiently many random variables, which can be approximated as a normal random variable under some conditions that the means and the variances have to satisfy. These conditions were introduced in the Lindeberg-Feller theorem [45][26] and its corollary, the Liapounov's theorem [44]. This condition was studied for sums of N independent random variables $(X_i)_{1 \leq i \leq N}$ with means $(m_i)_{1 \leq i \leq N}$ and variances $(a_i^2)_{1 \leq i \leq N}$. Let $s_n^2 = \sum_{i=1}^N a_i^2$ then :

$$\lim_{N \to \infty} \frac{1}{s_n^2} \sum_{i=1}^N \mathbf{E}\left[(X_i - m_i)^2 \mathbb{1}_{\{|X_i - m_i| > \epsilon s_n\}}\right] = 0, \ \forall \epsilon > 0 \Longrightarrow \frac{\sum_{i=1}^N (X_i - m_i)}{s_n} \xrightarrow{\mathbb{P}} \mathcal{N}(0,1) \tag{5.7}$$

Lindeberg's condition is sufficient, but not in general necessary. However if

$$\lim_{N \to \infty} \max_{i=1,\ldots,N} \frac{a_i^2}{s_n^2} \to 0,$$

this condition is both sufficient and necessary. For dependent random variables, the central limit theorem remains valid under conditions investigated in [18]. We have carried out some systematic experiments showing that, for sum of N multimodal random variables (three modes were considered in our experiments), good approximations of the Gaussian cdf are obtained as soon as N exceeds typically 30 to 40.

In this chapter, we investigate SGPKC under the assumption that the node weights w follows a multivariate Gaussian distribution. In the case of individual chance constraints and with the probability level ε less than 0.5, the chance constraints can then be reformulated as *binary second order cone constraints* (Binary SOCC) [47]. We study a comparison of several alternative techniques for solving SGPKC: First, we consider the second-order cone formulation for the chance constraint which handle SGPKC as a binary second-order cone program (Binary SOCP). The CPLEX solver is used to solve this program. Second, we consider the quadratic formulation for the chance constraint which handle SGPKC as a binary quadratic constrained program. Several linearization techniques are discussed to transform this binary quadratic program into binary linear program. In particular, we consider the classical linearization technique (Fortet [29]) and the linearization using bilinear forms given by Sherali-Smith [57]. Note that contrary to the former, the latter uses much fewer additional variables. Again the CPLEX solver is used for solving the resulting binary linear programs.

The numerical results obtained show that the solution technique using Sherali-Smith linearization provides better efficiency for SGPKC, than the one using binary

SOCP; the latter in turn outperforms the classical linearization technique. This shows that despite the fact that the quadratic formulation of chance constraint when relaxed is not convex (contrary to the second-order formulation), it turns out to provide better efficiency as compared with the second-order cone formulation when the variables are binary and a branch-and-bound procedure has to be applied.

5.3 Second order cone formulation

We assume that the probability distribution of node weights follows a multivariate Gaussian distribution with given means $(\bar{w}_i)_{1\leq i\leq n}$ and covariance matrix $(\sigma_{ij})_{1\leq i,j\leq n}$. We therefore reformulate the chance constraints (5.4) by the Binary SOCCs as follows. For all clusters $i = 1, \ldots, n$,

$$\sum_{u=1}^{n}(1-x_{ui})\bar{w}_u+\gamma\sqrt{\sum_{u=1}^{n}\sigma_{uu}(1-x_{ui})^2+2\sum_{u=1}^{n-1}\sum_{v=u+1}^{n}\sigma_{uv}(1-x_{ui})(1-x_{vi})} \leq W \quad (5.8)$$

where $\gamma = \mathcal{F}^{-1}(1-\varepsilon)$, $\mathcal{F}$ denoting the cumulative distribution function of $\mathcal{N}(0,1)$ (e.g $\gamma \simeq 1.685$ for $\varepsilon = 0.05$).

Then the resulting model for SGPKC is the following Binary SOCP (Bi-SOCP) program :

$$\begin{cases} \min & \sum_{(i,j)\in E} t_{ij}x_{ij} & \\ \text{s. t.:} & x_{ij}+x_{ik}\geq x_{jk}, & \forall(i,j,k)\in\mathcal{T}' \\ & x_{ij}+x_{jk}\geq x_{ik}, & \forall(i,j,k)\in\mathcal{T}' \\ & x_{ik}+x_{jk}\geq x_{ij}, & \forall(i,j,k)\in\mathcal{T}' \\ & \sum_{u=1}^{n}(1-x_{ui})\bar{w}_u+ & \\ & \quad +\gamma\sqrt{\sum_{u=1}^{n}\sigma_{uu}(1-x_{ui})^2+2\sum_{u=1}^{n-1}\sum_{v=u+1}^{n}\sigma_{uv}(1-x_{ui})(1-x_{vi})}\leq W & i=1,\ldots,n \\ & x_{uu}=0 & u=1,\ldots,n \\ & x_{ij}\in\{0,1\} & (i,j)\in E_n \end{cases}$$

We can see that the continuous relaxation of (Bi-SOCP) is a second-order cone program thus it can be solved using SOCP. SOCP has shown its effectiveness in solving nonlinear convex problem that include linear and (convex) quadratic programs as special cases. Several efficient primal-dual interior-point for SOCP have been developed in the last few years which share many of the features of primal-dual interior-point methods for linear programming (LP). However, the algorithmic efficiency of SOCP solvers when embedded into a tree search branch-and-bound to handle Bi-SOCP problem partly remains an open research problem.

5.4 Quadratically constrained 0-1 programming reformulation and linearization techniques

As an alternative to Binary SOCP, we investigate here a quadratic formulation for the SGPKC. To achieve this, we only need to replace the SOCCs (5.8) in (Bi-SOCP) with their equivalent quadratic forms :

$$\begin{cases} -2\sum_{u=1}^{n-1}\sum_{v=u+1}^{n}(\bar{w}_u\bar{w}_v-\gamma^2\sigma_{uv})(1-x_{ui})(1-x_{vi})+\sum_{u=1}^{n}(2W\bar{w}_u+\gamma^2\sigma_{uu}-\bar{w}_u^2)(1-x_{ui})\leq W^2 \\ \sum_{u=1}^{n}(1-x_{ui})\bar{w}_u\leq W \end{cases} \tag{5.9}$$

Since the quadratic formulation is difficult to handle directly, we will consider reformulations using various linearization techniques. The first technique discussed below is basically the classical linearization technique [29] and the second one is the linearization technique proposed by Sherali and Smith [57].

We first simplify (5.9) by setting :

$$\begin{cases} q_{uv}=2(\bar{w}_u\bar{w}_v-\gamma^2\sigma_{uv}) \\ d_u=2W\bar{w}_u+\gamma^2\sigma_{uu}-\bar{w}_u^2 \end{cases} \tag{5.10}$$

then the quadratic constraint in (5.9) reads :

$$\sum_{u=1}^{n}d_u(1-x_{ui})-\sum_{u=1}^{n-1}\sum_{v=u+1}^{n}q_{uv}(1-x_{ui})(1-x_{vi})\leq W^2 \tag{5.11}$$

5.4.1 Classical linearization technique

Introducing variable y_{uvi} to represent each product $x_{ui}x_{vi}$ then (5.11) is replaced with :

$$\begin{cases} \sum_{u=1}^{n}d_u(1-x_{ui})-\sum_{u=1}^{n-1}\sum_{v=u+1}^{n}q_{uv}y_{uvi}\leq W^2 \\ \max\{0,1-x_{ui}-x_{vi}-\}\leq y_{uvi}\leq\min\{1-x_{ui},1-x_{vi}\} \end{cases} \tag{5.12}$$

Using the classical linearization technique, SGPKC can be reformulated as fol-

lows :

$$
(\mathrm{CL})\begin{cases}
\min & \displaystyle\sum_{(i,j)\in E} t_{ij}x_{ij} & \\
\text{s. t.:} & x_{ij}+x_{ik}\geq x_{jk}, & \forall (i,j,k)\in\mathcal{T}' \\
& x_{ij}+x_{jk}\geq x_{ik}, & \forall (i,j,k)\in\mathcal{T}' \\
& x_{ik}+x_{jk}\geq x_{ij}, & \forall (i,j,k)\in\mathcal{T}' \\
& \displaystyle\sum_{u=1}^{n}(1-x_{ui})\bar{w}_u\leq W & i=1,\ldots,n \\
& \displaystyle\sum_{u=1}^{n}d_u(1-x_{ui})-\sum_{u=1}^{n-1}\sum_{v=u+1}^{n}q_{uv}y_{uvi}\leq W^2 & i=1,\ldots,n \\
& y_{uvi}\leq 1-x_{ui} & \forall u,v,i=1,\ldots,n \\
& y_{uvi}\leq 1-x_{vi} & \forall u,v,i=1,\ldots,n \\
& \max\{0,1-x_{ui}-x_{vi}\}\leq y_{uvi} & \forall u,v,i=1,\ldots,n \\
& x_{uu}=0 & u=1,\ldots,n \\
& x_{ij}\in\{0,1\} & (i,j)\in E_n
\end{cases}
$$

It can easily be shown that the constraints $\max\{0,1-x_{ui}-x_{vi}\}\leq y_{uvi}$ is redundant and can be removed. A drawback of the above formulation (referred to as an "extended formulation" because of the introduction of the extra variables y_{uvi}) is the large number of variables and constraints it requires. Note that in our graph partitioning problem, for a complete graph with n nodes the quadratic formulations we study already have $O(n^2)$ variables and the extended formulation $O(n^3)$ variables and also $O(n^3)$ added constraints. This can become rapidly unpractical. In the following subsection, we discuss an alternative linearization technique requiring fewer additional variables and additional constraints.

5.4.2 Sherali-Smith's linearization technique

This linearization technique has been introduced in [57]. The basic idea underlying this technique is :

- To transform each quadratic form into a bilinear form using $O(n)$ additional variables: applied to the quadratic constraint (5.11), it consists in introducing variables λ_{ui} to represent each sum $\displaystyle\sum_{v=u+1}^{n} q_{uv}(1-x_{vi})$. If a lower bound λ_{ui}^{min} and an upper bound λ_{ui}^{max} are known for λ_{ui} then the quadratic constraint (5.11) can be rewritten as:

$$
\begin{cases}
\sum_{u=1}^{n} d_{ui}(1-x_{ui})-\sum_{u=1}^{n-1}(1-x_{ui})\lambda_{ui}\leq W^2 & \\
\sum_{v=u+1}^{n} q_{uv}(1-x_{vi})=\lambda_{ui}, & \forall u=1,\ldots,n-1 \\
\lambda_{ui}^{min}\leq\lambda_{ui}\leq\lambda_{ui}^{max}, & \forall u=1,\ldots,n-1 \\
x\in\{0,1\}^n &
\end{cases}
$$

- Linearizing the various bilinear terms resulting from the above transformation: this is done by introducing a variable z_{ui} to represent each product $(1-x_{ui})\lambda_{ui}$:

$$\begin{cases} \sum_{u=1}^{n} d_{ui}(1-x_{ui}) - \sum_{u=1}^{n-1} z_{ui} \leq W^2 & \\ \sum_{v=u+1}^{n} q_{uv}(1-x_{vi}) = \lambda_{ui}, & \forall u = 1,\dots,n-1 \\ \lambda_{ui}^{min}(1-x_{ui}) \leq z_{ui} \leq \lambda_{ui}^{max}(1-x_{ui}), & \forall u = 1,\dots,n-1 \\ \lambda_{ui}^{min} x_{ui} \leq \lambda_{ui} - z_{ui} \leq \lambda_{ui}^{max} x_{ui}, & \forall u = 1,\dots,n-1 \\ x \in \{0,1\}^n & \end{cases}$$

- In the case of our problem for the uncertain weights, we assume that the variances are small compared to the means, so that $q_{uv} = 2(\bar{w}_u\bar{w}_v - \gamma^2\sigma_{uv}) > 0, \forall 1 \leq u < v \leq n$. Therefore $\lambda_{ui}^{min} = 0$ and the constraints $\lambda_{ui}^{min} x_{ui} \leq \lambda_{ui} - z_{ui} \leq \lambda_{ui}^{max} x_{ui}$, $\forall u = 1,\dots,n-1$ can be removed.

Applying the various transformations above to constraints (5.11) in the quadratic formulation of SGPKC, we can reformulate it as follows:

$$(\text{SS}) \begin{cases} \min & \displaystyle\sum_{(i,j)\in E} t_{ij}x_{ij} & \\ \text{s. t.:} & x_{ij} + x_{ik} \geq x_{jk}, & \forall (i,j,k) \in \mathcal{T}' \\ & x_{ij} + x_{jk} \geq x_{ik}, & \forall (i,j,k) \in \mathcal{T}' \\ & x_{ik} + x_{jk} \geq x_{ij}, & \forall (i,j,k) \in \mathcal{T}' \\ & \displaystyle\sum_{u=1}^{n}(1-x_{ui})\bar{w}_u \leq W & i = 1,\dots,n \\ & \displaystyle\sum_{u=1}^{n} d_u(1-x_{ui}) - \sum_{u=1}^{n-1} z_{ui} \leq W^2 & i = 1,\dots,n \\ & \displaystyle z_{ui} \leq \sum_{v=u+1}^{n} q_{uv}(1-x_{vi}) & i,u = 1,\dots,n \\ & 0 \leq z_{ui} \leq \lambda_{ui}^{max}(1-x_{ui}) & i,u = 1,\dots,n \\ & x_{uu} = 0 & u = 1,\dots,n \\ & x_{ij} \in \{0,1\} & (i,j) \in E_n \end{cases}$$

The above formulation (SS) requires fewer variables and constraints when compared with the classical linearization (CL). For instance, in our problem for a graph with n nodes, the number of added variables and the number of added constraints are $O(n^2)$. However as shown in [57], this is at the expense of weaker relaxation as compared with the classical linearization technique.

The last unknown parameters in this formulation are the bounds λ_{ui}^{max} that can be estimated based on the definition of added variables $(z_{ui})_{1\leq i\leq n-1}$. We will consider two such estimates in the following, two variants of the Sherali-Smith reformulation will thus be obtained.

Simple bounds on λ_{ui}^{max}

Note that since $q_{uv} \geq 0,\ \forall(u,v)$, we can take

$$\lambda_{ui}^{max} = \sum_{v=u+1}^{n} q_{uv} \tag{5.13}$$

It is observed that λ_{ui}^{max} does not depend on i, so at most n values have to be computed. Using these bounds for λ_{ui}^{max} in (SS), we obtain a first formulation based on Sherali-Smith's technique.

Improved bounds on λ_{ui}^{max}

The application of of Sherali-Smith technique can be made more efficient if we can obtain a better estimate of the bounds λ_{ui}^{max}. The idea is the fact that we can get stronger bounds of each sum $\sum_{v=u+1}^{n} q_{uv}(1-x_{vi})$ by adding valid inequalities in the process of computing them. In our problem, one of the valid inequalities that can be chosen is the stochastic knapsack constraint (5.8), whereby the bounds can be estimated, for all $i = 1, \ldots, n$ as :

$$\lambda_{ui}^{max} = \begin{cases} \max & \sum_{v=u+1}^{n} q_{uv}(1-x_v) = \max \sum_{v=u+1}^{n} 2(\bar{w}_u\bar{w}_v - \gamma^2\sigma_{uv})(1-x_v) \\ \text{s. t.:} & \sum_{v=1}^{n} \bar{w}_v(1-x_v) + \\ & + \gamma\sqrt{\sum_{v=1}^{n}\sigma_{vv}(1-x_v)^2 + 2\sum_{v=1}^{n-1}\sum_{v'=v+1}^{n}\sigma_{vv'}(1-x_v)(1-x_{v'})} \leq W \\ & x \in \{0,1\}^n \end{cases} \tag{5.14}$$

Again λ_{ui}^{max} does not depend on i so at most n problems of the form (5.14) have to be solved. Using the improved bounds λ_{ui}^{max} deduced from (5.14), we obtain the improved Sherali-Smith formulation (ISS). In addition, we chose the continuous relaxation version of (5.14) to estimate the bounds λ_{ui}^{max} because our experiments shows that using the integer formulation (5.14) would not lead to a significant improvement in the quality of the bounds.

5.5 Computational results

In this section, we present computational results for the SGPKC comparing the formulations that were discussed in Section 5.3 and Section 5.4. The formulations are compared in both computation times and gaps.

For a given number of nodes n and number of edges m, we generate five instances by picking edges uniformly at random until the number of edges reaches m. The edge weights t and the means of node weights $\bar{w}$ are drawn independently and uniformly from the interval $[1, 1000]$, the covariance matrix σ of node weights is generated as

diagonally dominant matrix where each point σ_{ii} $\forall i = 1, \ldots, n$ in the diagonal is drawn independently and uniformly from the interval $[1, 20\% \bar{w}_i]$. For each instance, we calibrate the upper bounds of knapsack constraints W in order to ensure that the generated instances will be not "easy" to solved. To achieve this, we used METIS [41] to estimate the solution with the number of clusters $k = 4$ (or 6, 8, 12) that we call the initial partition, we then do 1000 perturbations of this partition. These bounds W then were chosen so that only 10% of these partitions are satisfied. Finally the probability level ε was chosen to be 0.05 (= 5%) a fairly standard value in practice.

All experiments are run on a machine with Intel Core i7-3630QM 2.40GHz processors and 16 GiB of RAM. The solver CPLEX 12.6 is used to solve respectively (Bi-SOCP), (CL), (SS) and (ISS) and to ensure that comparisons will be not biased we switch off CPLEX pre-solve. All computation times are CPU seconds and the computation times are subject to a time limit of 7200 seconds.

In Table 5.1 we report the results obtained with (Bi-SOCP), (CL), (SS) and (ISS). In our experiments, each instance belongs to one of four graph types: series-parallel graph (SP), planar grid graph (PG), toroidal grid graph (TG) and random graph (RG). Series-parallel graphs are highly sparse while random graphs are denser graphs with $m \approx (4 \text{ to } 8) \times n$. As for each value of n, m, we have five instances, the first column of each technique in this table report the average CPU time to obtain the solution, the second column of each technique report the average gaps at root node. For the (ISS), the CPU time to calculate the bounds is included in the total CPU time to obtain the solution. For a specific technique and for the instances that this technique guarantee a solution, we note the CPU time as "Nopt" and we indicate the residual gap in parenthesis.

As can be seen from Table 1, (ISS) has the highest performance while (CL) is the least efficient technique in term of computation times. We can arrange the order of effectiveness of these techniques as (ISS)$\succ$(SS)$\succ$(Bi-SOCP)$\succ$(CL).

The main observations which arise from the results are the following :

- As discussed above, the (CL) technique does not perform well for large instances due to a large number of added variables and added constraints. We report the solution times for (CL) are worse by a factor 2 to 4 than the (Bi-SOCP).

- For the small instances, (SS) does not outperform (Bi-SOCP). However for the larger instances, (SS) is slightly faster than (Bi-SOCP) in spite of the fact that the number of nodes in the search tree for (Bi-SOCP) is quite reduced. A possible explanation of this is that : **a)** the computational effort required for solving each node is significant, and **b)** SOCP solvers do not enjoy the same warm-starting capabilities as simplex-based LP solvers.

- (ISS) clearly dominates the other techniques as we can see in the table, it is faster by a factor 1.5 to 3 than (SS) in term of solution times.

- There are also instances for which CPLEX is not even capable to solve (Bi-SOCP) and (CL) but succeeds to find optimal integer solution with (SS) and (ISS). With (ISS) we can solve larger instances, e.g, ($n = 80$) for series-parallel

graphs, ($n = 70$) for planar grid graphs, ($n = 60$) for toroidal grid graphs and ($n = 60$) for random graphs.

- We also report the average gaps at root node for each solution technique. (Bi-SOCP) provides the best gaps, sightly better than (CL) and (SS) and (ISS), however the difference is small (<5%). This is not surprise due to the fact that (CL), (SS) and (ISS) may be considered as relaxations of (Bi-SOCP). Another observation is that although improved bounds λ^{max} have been used in (ISS), the gaps at root node have not changed. However the number of nodes in the tree search is reduced in most cases, leading to the reduction of CPU times.

Finally, Table 5.2 shows the impact of computing the improved bounds for λ_{ui}^{max} on total CPU times for (ISS). The computation times of the bounds reported in the third column are very small as compared to the total solution times of (ISS), while the average improvement on the bounds is quite significant.

5.6 Conclusion

In this chapter, a stochastic version of the node weighted graph partitioning problem has been investigated. Practical application in the context of partitioning process network problem has been discussed. It has been shown that transforming an initial SOCP based formulation into quadratic constraints and applying some linearization techniques can be more efficient than solving the usual binary SOCP formulation.

Table 5.1: Comparison of the various solution techniques. Nopt indicates that the exact optimal solution could not be found within the imposed time limit (7200s). In such cases, the value of the relative residual gap is shown in parenthesis.

Instances		Bi-SOCP		CL		SS		ISS	
types	n,m	CPU	GAP (Nodes)	CPU	GAP (Nodes)	CPU	GAP (Nodes)	CPU	GAP (Nodes)
SP	25, 40	4.3	16.2 (6)	12.6	17.0 (11)	4.4	18.3 (22)	2.2	18.3 (15)
SP	30, 50	30.8	11.4 (17)	100.9	13.1 (142)	27.5	15.6 (366)	15.6	15.6 (254)
SP	35, 60	40.4	10.5 (32)	177.3	15.4 (336)	43.5	15.9 (829)	29.3	15.9 (599)
SP	40, 65	126.3	9.8 (59)	513.4	11.3 (573)	122.4	12.7 (1136)	63.8	12.7 (729)
SP	45, 75	665.2	13.2 (88)	1539	14.6 (612)	595.0	17.2 (1387)	336.5	17.2 (874)
SP	50, 80	862.8	9.1 (117)	2238	10.4 (935)	978.7	13.3 (1843)	517.2	13.3 (1311)
SP	60, 90	3127	10.6 (156)	Nopt (6.8%)	12.3 (1033)	1844	16.5 (2552)	699.3	16.5 (1566)
SP	80, 130	Nopt (2.4%)	12.7 (187)	Nopt (7.3%)	14.8 (1426)	Nopt (2.8%)	17.3 (3136)	5273	17.3 (3629)
PG	30, 47	105.4	13.2 (49)	256.8	14.6 (395)	97.2	15.3 (866)	41.1	15.3 (542)
PG	40, 66	583.4	12.6 (74)	1727	14.1 (528)	556.2	15.0 (1344)	236.5	15.0 (836)
PG	50, 85	2193	11.4 (125)	Nopt (3.7%)	12.8 (1057)	2386	13.8 (2546)	827.7	13.8 (1888)
PG	60, 104	Nopt (0.2%)	13.2 (146)	Nopt (5.6%)	14.4 (1791)	6216	15.8 (4728)	2397	15.8 (3352)
PG	70, 123	Nopt (2.4%)	12.0 (168)	Nopt (7.1%)	12.2 (2032)	Nopt (3.2%)	14.7 (4759)	6163	14.7 (5292)
TG	30, 60	359.2	14.9 (68)	1044	15.2 (539)	332.7	18.8 (1353)	154.3	18.8 (877)
TG	40, 80	989.5	12.2 (128)	5268	13.2 (572)	937.2	15.4 (1736)	443.4	15.4 (1255)
TG	50, 100	5453	14.3 (183)	Nopt (6.9%)	16.7 (1682)	4829	18.3 (3357)	1537.3	18.3 (2172)
TG	60, 120	Nopt (3.7%)	13.8 (175)	Nopt (8.6%)	14.2 (1843)	Nopt (3.2%)	16.5 (3776)	5234	16.5 (4275)
			(55)		(474)		(973)		(666)
RG	25, 150	442.3	15.0 (72)	945.2	16.2 (553)	489.1	17.1 (1296)	226.4	17.1 (924)
RG	30, 200	Nopt (0.2%)	15.5 (127)	Nopt (4.1%)	16.1 (1183)	7111	17.9 (6421)	3123	17.9 (4318)
RG	40, 120	3612	13.2 (196)	Nopt (3.3%)	14.8 (974)	3828	16.4 (3689)	1805	16.4 (3150)
RG	50, 120	Nopt (2.3%)	14.7 (188)	Nopt (8.6%)	15.6 (2158)	Nopt (2.6%)	17.2 (4523)	6006	17.2 (5298)
RG	60, 100	Nopt (3.8%)	16.2 (209)	Nopt (9.9%)	17.2 (1542)	Nopt (3.6%)	19.4 (4175)	6419	19.4 (4941)

Table 5.2: Computation times of the bounds λ_{ui}^{max} by (5.14)

n,l	CPU	Percentage(%) of total CPU	average improvement on bounds(%)
25, 55	0	0	40
30, 70	0.1	0.7	39
35, 80	0.2	0.6	43
40, 75	0.4	0.7	36
45, 95	0.6	0.6	48
50, 80	0.8	0.5	45
60, 90	1.0	0.2	50
80, 130	1.8	0	43

Chapter 6

Conclusions and perspectives

6.1 Conclusions

The graph partitioning problem under set constraints (GPP-SC) has been investigated in this thesis. Practical applications in the context of partitioning process network problem and the SONET/SDH network design problem have been discussed.

We have given an improved compact formulation for the problem problem using $O(|V||E|)$ triangle inequalities instead of $O(|V|^3)$ triangle inequalities as usual. Numerical results have shown that solution times are reduced drastically from 3 to 50 times with our improved formulation. There are instances for that CPLEX is even not capable to solve the continuous relaxation with the classical formulation but succeed to find optimal integer solution with our reduced formulation.

For large size graphs, only approximate (heuristic) solutions can be hoped for, and the key issue in this context is to validate the approximate solutions obtained by computing lower bounds. We investigate a new formulation for the problem that makes use of only $O(|E|)$ decision variables. As there are an exponential number of constraints in the formulation, a separation method is needed to speed up the computation times. We introduce such a separation that makes use of all pair shortest path algorithm to determine violate constraints, thus the separation is in polynomial time. Numerical results have shown that lower bounds for large size problems can be given by our new formulation with up to 2000 nodes. In addition, we introduce also an efficient generation of heuristic solutions base on the relaxed solution given by the new formulation. Numerical experiments show that our generation gives tight upper bounds for the problem with the gaps less than 20%, specially for series-parallel graphs with the gaps less than 10%.

We have also investigated a stochastic version of the node weighted graph partitioning problem. It has been shown that transforming an initial SOCP based formulation into quadratic constraints and applying some linearization techniques can be more efficient than solving the usual binary SOCP formulation.

6.2 Perspectives

As a possible direction for future investigation, exhibiting further constraint structures, lending themselves to significant reduction of the number of triangle inequal-

ities would be worth considering. We mention the search for additional polyhedral results related to the formulations of the Chapter 3, possibly leading to improved computational efficiency of branch-and-bound based procedures. We note also that carrying out an extensive comparison in terms of efficiency of Node-Cluster versus Node-Node models remains a potentially interesting subject for future work in the area.

In regarding the results in Chapter 4, we plan to expand the investigation of the cycle model for general GPP-SC problem. In addition, we mention the search for additional solution techniques related to the separation procedure, possibly leading to speed up even more the computation times. We plan also to improve the generation of heuristic solutions, in applying the progressive aggregation procedure on several last rounds of the constraint-generation procedure.

We will expand our research on SGPKC for others probability distributions on the node weights such as uniform distribution or truncated Gaussian distribution.

Bibliography

[1] Z. Ales, A. Knippel, and A. Pauchet. On the polyhedron of the K-partitioning problem with representative variables. *ArXiv e-prints*, November 2014.

[2] V. Alexandrov, M. Lees, V. Krzhizhanovskaya, J. Dongarra, P.M.A. Sloot, P. Aubry, P.-E. Beaucamps, F. Blanc, B. Bodin, S. Carpov, L Cudennec, V. David, P. Dore, P. Dubrulle, B. Dupont de D., F. Galea, T. Goubier, M. Harrand, S. Jones, J.-D. Lesage, S. Louise, N.M. Chaisemartin, T.H. Nguyen, X. Raynaud, and R. Sirdey. 2013 international conference on computational science extended cyclostatic dataflow program compilation and execution for an integrated manycore processor. *Procedia Computer Science*, 18:1624 – 1633, 2013.

[3] M. Armbruster, M. Fügenschuh, C. Helmberg, and A. Martin. *A Comparative Study of Linear and Semidefinite Branch-and-Cut Methods for Solving the Minimum Graph Bisection Problem*, pages 112–124. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[4] D.A. Bader, H. Meyerhenke, P. Sanders, and D. Wagner, editors. *Graph Partitioning and Graph Clustering*. 10th DIMACS Implementation Challenge Workshop. February 13-14, 2012. Georgia Institute of Technology, Atlanta, GA. Contemporary Mathematics 588. American Mathematical Society and Center for Discrete Mathematics and Theoretical Computer Science, 2013.

[5] F. Barahona and A.R. Mahjoub. On the cut polytope. *Mathematical Programming*, 36(2):157–173, 1986.

[6] R.E. Bellman and L.A. Zadeh. Decision-making in a fuzzy environment. *Management Science*, 17(4):B–141–B–164, 1970.

[7] A. Ben-Tal and A. Nemirovski. Robust convex optimization. *Math. Oper. Res.*, 23(4):769–805, November 1998.

[8] A. Ben-Tal and A. Nemirovski. Robust optimization – methodology and applications. *Mathematical Programming*, 92(3):453–480, 2002.

[9] D. Bertsimas and M. Sim. The price of robustness. *Oper. Res.*, 52(1):35–53, January 2004.

[10] D. Bertsimas and M. Sim. The price of robustness. *Oper. Res.*, 52(1):35–53, January 2004.

[11] P. Bonami, V.H Nguyen, M. Klein, and M. Minoux. On the solution of a graph partitioning problem under capacity constraints. In A.R. Mahjoub, V. Markakis, I. Milis, and V.Th. Paschos, editors, *ISCO*, volume 7422 of *Lecture Notes in Computer Science*, pages 285–296. Springer, 2012.

[12] L. Brunetta, M. Conforti, and G. Rinaldi. A branch-and-cut algorithm for the equicut problem. *Mathematical Programming*, 78(2):243–263, 1997.

[13] A. Buluc, H. Meyerhenke, I. Safro, P. Sanders, and C. Schulz. Recent Advances in Graph Partitioning. *ArXiv e-prints*, November 2013.

[14] A. Charnes and W. Cooper. Chance-constrained programming. *Management Science*, 6(1):73–79, 1959.

[15] S. Chopra. The graph partitioning polytope on series-parallel and4-wheel free graphs. *SIAM J. Discret. Math.*, 7(1):16–31, February 1994.

[16] S. Chopra and M.R. Rao. The partition problem. *Mathematical Programming*, 59(1-3):87–115, 1993.

[17] S. Chopra and M.R. Rao. Facets of the k-partition polytope. *Discrete Applied Mathematics*, 61(1):27 – 48, 1995.

[18] W. J. Cocke. Central limit theorems for sums of dependent vector variables. *The Annals of Mathematical Statistics*, 43(3):pp. 968–976, 1972.

[19] G.B. Dantzig. Linear programming under uncertainty. *Management Science*, 1(3-4):197–206, 1955.

[20] D. Delling, A.V. Goldberg, I. Razenshteyn, and R.F. Werneck. Exact combinatorial branch-and-bound for graph bisection. In *Proceedings of the Meeting on Algorithm Engineering & Expermiments*, ALENEX '12, pages 30–44, Philadelphia, PA, USA, 2012. Society for Industrial and Applied Mathematics.

[21] D. Delling and R.F. Werneck. Better bounds for graph bisection. In *Proceedings of the 20th Annual European Conference on Algorithms*, ESA'12, pages 407–418, Berlin, Heidelberg, 2012. Springer-Verlag.

[22] A. G. Doig, Bya H. Land, and A. G. Doig. An automatic method for solving discrete programming problems. *Econometrica*, pages 497–520, 1960.

[23] N. Fan and P.M. Pardalos. *Robust Optimization of Graph Partitioning and Critical Node Detection in Analyzing Networks*, pages 170–183. Springer Berlin Heidelberg, Berlin, Heidelberg, 2010.

[24] N. Fan, Q.P. Zheng, and P.M. Pardalos. On the two-stage stochastic graph partitioning problem. In *Proceedings of the 5th International Conference on Combinatorial Optimization and Applications*, COCOA'11, pages 500–509, Berlin, Heidelberg, 2011. Springer-Verlag.

[25] A.E. Feldmann and P. Widmayer. An $o(n^4)$ time algorithm to compute the bisection width of solid grid graphs. *Algorithmica*, 71(1):181–200, January 2015.

[26] W. Feller. Über den zentralen grenzwertsatz der wahrscheinlichkeitsrechnung. *Mathematische Zeitschrift*, 40(1):521–559, 1936.

[27] A. Felner. Finding optimal solutions to the graph partitioning problem with heuristic search. *Annals of Mathematics and Artificial Intelligence*, 45(3-4):293–322, December 2005.

[28] C.E. Ferreira, A. Martin, C.C. de Souza, R. Weismantel, and L.A. Wolsey. The node capacitated graph partitioning problem: A computational study. *Mathematical Programming*, 81(2):229–256, 1998.

[29] R. Fortet. L'algèbre de boole et ses applications en recherche operationnelle. *Trabajos de Estadistica*, 11(2):111–118, 1960.

[30] A. Frangioni, A. Lodi, and G. Rinaldi. New approaches for optimizing over the semimetric polytope. *Mathematical Programming*, 104(2-3):375–388, 2005.

[31] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1979.

[32] O. Goldschmidt, A. Laugier, and E.A. Olinick. Sonet/sdh ring assignment with capacity constraints. *Discrete Applied Mathematics*, 129(1):99 – 128, 2003. Algorithmic Aspects of Communication.

[33] M. Grötschel and Y. Wakabayashi. A cutting plane algorithm for a clustering problem. *Math. Program.*, 45(1):59–96, August 1989.

[34] W.W. Hager and Y. Krylyuk. Graph partitioning and continuous quadratic programming. *SIAM J. Discret. Math.*, 12(4):500–523, October 1999.

[35] W.W. Hager, D.T. Phan, and H. Zhang. An exact algorithm for graph partitioning. *Mathematical Programming*, 137(1):531–556, 2013.

[36] R. Henrion and C. Strugarek. Convexity of chance constraints with independent random variables. *Computational Optimization and Applications*, 41(2):263–276, 2008.

[37] S. Holm and M.M. Sørensen. The optimal graph partitioning problem. *Operations-Research-Spektrum*, 15(1):1–8, 1993.

[38] L. Hyafil and R.L. Rivest. Graph partitioning and constructing optimal decision trees are polynomial complete problems. Technical Report Rapport de Recherche no. 33, IRIA – Laboratoire de Recherche en Informatique et Automatique, Domaine de Voluceau, Rocquencourt 78150 - Le Chesnay, France, 1973.

[39] V. Kaibel, M. Peinhardt, and M.E. Pfetsch. Orbitopal fixing. *Discrete Optimization*, 8(4):595 – 610, 2011.

[40] S.E. Karisch, F. Rendl, and J. Clausen. Solving graph bisection problems with semidefinite programming. *INFORMS J. on Computing*, 12(3):177–191, July 2000.

[41] G. Karypis and V. Kumar. MeTis: Unstructured Graph Partitioning and Sparse Matrix Ordering System, Version 4.0, 2009.

[42] S. Kataoka. A stochastic programming model. *Econometrica*, 31(1/2):181–196, 1963.

[43] M. Labbé and F.A. Özsoy. Size-constrained graph partitioning polytopes. *Discrete Mathematics*, 310(24):3473 – 3493, 2010.

[44] A.M. Liapunov. *Collected Works of Academician A.M. Lyapunov*. Number v. 1-2 in Collected Works of Academician A.M. Lyapunov. Translation Division, Foreign Technology Division, 1967.

[45] J.W. Lindeberg. Eine neue herleitung des exponentialgesetzes in der wahrscheinlichkeitsrechnung. *Mathematische Zeitschrift*, 15(1):211–225, 1922.

[46] A. Lisser and F. Rendl. Graph partitioning using linear and semidefinite programming. *Mathematical Programming*, 95(1):91–101, 2003.

[47] M.S. Lobo, L. Vandenberghe, S. Boyd, and H. Lebret. Applications of second-order cone programming. *Linear Algebra and its Applications*, 284(1-3):193 – 228, 1998. International Linear Algebra Society (ILAS) Symposium on Fast Algorithms for Control, Signals and Image Processing.

[48] E.M. Macambira, N. Maculan, and C.C. de Souza. A column generation approach for sonet ring assignment. *Networks*, 47(3):157–171, 2006.

[49] B.L. Miller and H.M. Wagner. Chance constrained programming with joint constraints. *Operations Research*, 13(6):930–945, 1965.

[50] M. Minoux and R. Zorgati. Convexity of gaussian chance constraints and of related probability maximization problems. *Computational Statistics*, 31(1):387–408, 2016.

[51] D.P. Nguyen, M. Minoux, V.H. Nguyen, T.H. Nguyen, and R. Sirdey. Improved compact formulations for a wide class of graph partitioning problems in sparse graphs. *Discrete Optimization*, pages –, 2016.

[52] A. Prékopa. *Stochastic Programming*. Springer, 1995.

[53] D. Rhodes, R. Dick, and K. Vallerio. Task graphs for free. `http://ziyang.eecs.umich.edu/~dickrp/tgff/`.

[54] M. Sellmann, N. Sensen, and L. Timajev. *Multicommodity Flow Approximation Used for Exact Graph Partitioning*, pages 752–764. Springer Berlin Heidelberg, Berlin, Heidelberg, 2003.

[55] N. Sensen. *Lower Bounds and Exact Algorithms for the Graph Partitioning Problem Using Multicommodity Flows*, pages 391–403. Springer Berlin Heidelberg, Berlin, Heidelberg, 2001.

[56] A. Shapiro, D. Dentcheva, and A. Ruszczyński. *Lectures on Stochastic Programming*. Society for Industrial and Applied Mathematics, 2009.

[57] H.D. Sherali and J.C. Smith. An improved linearization strategy for zero-one quadratic programming problems. *Optimization Letters*, 1(1):33–47, 2007.

[58] M.M. Sørensen. Facet-defining inequalities for the simple graph partitioning polytope. *Discrete Optimization*, 4(2):221 – 231, 2007.

[59] O. Stan, R. Sirdey, J. Carlier, and D. Nace. The robust binomial approach to chance-constrained optimization problems with application to stochastic partitioning of large process networks. *Journal of Heuristics*, 20(3):261–290, 2014.

[60] W.W. Stein and T.Z. William, editors. *Applications of Stochastic Programming*. Society for Industrial and Applied Mathematics, 2005.

[61] Z.C. TaşKiN, J.C. Smith, S. Ahmed, and A.J. Schaefer. Cutting plane algorithms for solving a stochastic edge-partition problem. *Discret. Optim.*, 6(4):420–435, November 2009.