論文 / 著書情報
Article / Book Information

題目(和文)	クリーク分割問題に対する整数計画アプローチにおける冗長性
Title(English)	Redundancy in Integer Programming Approach for the Clique Partitioning Problem
著者(和文)	鮭川矩義
Author(English)	Noriyoshi Sukegawa
出典(和文)	学位:博士（工学）, 学位授与機関:東京工業大学, 報告番号:甲第9892号, 授与年月日:2015年3月26日, 学位の種別:課程博士, 審査員:水野　眞治,宮川　雅巳,鈴木　定省,中田　和秀,松井　知己
Citation(English)	Degree:, Conferring organization: Tokyo Institute of Technology, Report number:甲第9892号, Conferred date:2015/3/26, Degree Type:Course doctor, Examiner:,,,,
学位種別(和文)	博士論文
Type(English)	Doctoral Thesis

Redundancy in Integer Programming Approach for the Clique Partitioning Problem

Noriyoshi Sukegawa

Advisor: Professor Shinji Mizuno

Department of Industrial Engineering and Management
Graduate School of Decision Science and Technology
Tokyo Institute of Technology

Abstract

In 1989, Grötschel and Wakabayashi [38] introduced a fundamental combinatorial optimization problem, called the *clique partitioning problem* (CPP) with a large number of real world applications related to *clustering* such as qualitative data analysis [14, 76, 88], group technology [72, 87], flight-gate scheduling [27], and community detection [2, 5, 30, 64].

In spite of its rich applications, the size of the standard *integer programming* (IP) formulation of CPP is known to be huge even for small-sized instances, which prevents us from efficient computation and limits the application range of CPP in practice [14].

In order to overcome this drawback, in this thesis, we develop methodologies for reducing the problem size of CPP. To this end, we focus on a *redundancy* in the standard IP formulation. More specifically, we aim at reducing the formulation size by efficiently and effectively removing redundancies in advance. The instances of reduced size can be solved efficiently by recent powerful optimization softwares.

First, we study on a *preprocessing* algorithm for detecting a redundancy in the decision variables, which rewrites a given instance of CPP into, often substantially, smaller but equivalent new one. The basic strategy is to strengthen a classical technique called the *pegging test* [67] by utilizing an underlying property of CPP.

Next, motivated by a recent work by Dinh and Thai [30], in a theoretical manner, we reveal a redundancy in the numerous constraints of the standard IP formulation, which yields a new formulation of CPP. Our *reformulation* enables us to easily reduce the number of the constraints required, often significantly, as well as computation time by the softwares.

Acknowledgment

I want to express my gratitude to Shinji Mizuno for giving me helpful advices, and for encouraging me during my doctoral studies. He is always passionate and active, and is my role model. My thanks go to Yoshitsugu Yamamoto for training me to be logical, and Tomomi Matsui for teaching me the aspects of fun and excitement in study activities.

I also would like to thank my co-authors. First of all, I want to express my gratitude to Atsushi Miyauchi and Liyuan Zhang for their helpful and meaningful discussions, and for sparing a large amount of time for discussing with me. Without them, I could not have done this study. Furthermore, I want to thank my co-authors Antoine Deza, Kazuhide Nakata, Tomonari Kitahara, Yuichi Takano, and Mirai Tanaka for delightful discussions. Finally, my thanks go to Masami Miyakawa and Sadami Suzuki for giving me helpful and valuable comments on this study, which improved the presentation of this thesis.

Contents

1 Introduction **1**
- 1.1 Introduction . . . 1
  - 1.1.1 Backgrounds . . . 1
  - 1.1.2 Motivation . . . 2
  - 1.1.3 Focus . . . 3
  - 1.1.4 Object . . . 3
  - 1.1.5 Results . . . 4
  - 1.1.6 Outline of Research . . . 5
- 1.2 Organization . . . 7

2 Problem **9**
- 2.1 Clique Partitioning Problem (CPP) . . . 9
- 2.2 Integer Programming (IP) Formulation . . . 11
- 2.3 Example: Régnier's problem . . . 13

3 Preprocessing **16**
- 3.1 Basic Idea . . . 17
  - 3.1.1 Lagrangian Relaxation . . . 18
  - 3.1.2 Pegging Test . . . 19
- 3.2 Algorithm . . . 20
  - 3.2.1 Basic Algorithm . . . 20
  - 3.2.2 Reduction Procedure . . . 23
- 3.3 Preliminary Numerical Experiments . . . 24
  - 3.3.1 On Basic Algorithm . . . 25
  - 3.3.2 On reduction procedure . . . 29
  - 3.3.3 Comparison with optimization software . . . 30
  - 3.3.4 Outlook . . . 31

4 Reformulation **33**
- 4.1 Motivation . . . 34
- 4.2 Redundant Constraints in CPP . . . 35
  - 4.2.1 Result . . . 35
  - 4.2.2 Preliminary Numerical Experiments . . . 37

4.2.3 Outlook 38
4.3 Linear Ordering Problem (LOP) 41
4.3.1 Problem Description 41
4.3.2 Formulations 42
4.3.3 Example: Kemeny's method 43
4.4 Redundant Constraints in LOP 43
4.4.1 Result 44
4.4.2 Preliminary Numerical Experiments 46
4.4.3 Outlook 48

5 Hardness Results **49**
5.1 Related Works 49
5.2 Maximum Edge CPP and Its Hardness 50

6 Conclusion **52**
6.1 Conclusion 52
6.2 Outlook 53
6.3 Future Work 54

A Proofs **56**
A.1 Proof of Theorem 2 56
A.2 Proof of Theorem 3 58
A.3 Proof of Theorem 4 and 5 60

Chapter 1

Introduction

1.1 Introduction

1.1.1 Backgrounds

Clustering is a fundamental problem to find an appropriate grouping of a given set of items into similar groups, called *clusters*, which appears in diverse fields including engineering, biology, physics, sociology, medical science, and so on. Therefore, algorithms for clustering have been studied extensively drawing upon the methodologies in statistics, mathematics, and computer science [43].

With respect to the type of data we deal with, clustering could be divided into two cases: *quantitative* data clustering and *qualitative* data clustering. However, the algorithms for clustering proposed so far are mainly designed for quantitative data from its ease of use.

As pointed out by Brusco and Köhn [14], on the other hand, we have few algorithms for qualitative data clustering. However, there are strong demands for appropriate and effective algorithms for qualitative data clustering from several areas of social science including economics, sociology, psychology, and zoology, as we frequently encounter qualitative data such as "blood-type", "religion", "feeling", "shape of ears", and so on, in these application area.

In 1965, Régnier [76] proposed an algorithm for qualitative data clustering, based on a combinatorial optimization problem, which is referred to as the *Régnier's problem*, or the *consensus clustering (median partitioning) problem* [31, 41].

In contrast to the existing approaches for qualitative data clustering, in Régnier [76]'s model, we can avoid an inappropriate conversion of the given qualitative data into some quantitative data. This is the most attractive point of Régnier's model. It should be noted that Régnier's model is still regarded as one of the most promising approaches for qualitative data clustering [14, 88].

The key idea is to assume that each attribute defines an equivalence relation on the target items. For instance, an attribute "feeling" can be cosidered to define an equivalence relation on the target items whose equivalence classes correspond to "happy", "sad", "nothing", and so on. As a result, we obtain a set of equivalence relations associated with the set of qualitative attributes.

Régnier [76] considers to "aggregate" such equivalence relations into one equivalence relation, where we need to solve some NP-hard [86] optimization problem, *Régnier's problem*.

In 1989, Grötschel and Wakabayashi [38] refined Régnier's problem to introduce a combinatorial optimization problem, which is referred to as the *clique partitioning problem* (CPP). Therefore, Régnier's problem is a special case of CPP, which implies that CPP is also NP-hard in general. It is known that CPP provides a general framework for many and diverse clustering algorithms based on an optimization approach.

As a matter of fact, in addition to Régnier's problem, CPP has been applied to many real world problems including flight-gate scheduling [27], microarray analysis [52], group technology problem [72, 87], and community detection [2, 5, 30, 64], known as the modularity maximization [70], to list a few, in the literature.

1.1.2 Motivation

On the other hand, in recent years, using optimization softwares is widely recognized as a promising strategy for solving real world problems. Regardless of commercial or non-commercial, we now have may powerful softwares such as CPLEX Optimizer, Gurobi Optimizer, and Xpress Optimization. With the substantial development of algorithms for solving optimization problems, the performance of these softwares also has been dramatically improved in the last decades.

The above optimization softwares are mainly designed for solving *integer programming* (IP) problems, or its generalization, *mixed integer programming* (MIP) problems. It is widely known that IP and MIP can describe many and diverse real world problems as they can model fundamental combinatorial structures including partition, order, route, subset, and so on, by its "integrality" [69].

As demonstrated by Grötschel and Wakabayashi [38], CPP can be formulated as a simple IP problem. Due to its simplicity, various algorithms have been proposed based on this standard IP formulation in the literature [2, 38, 44, 64, 72]. However, this standard IP formulation is known to suffer from numerous constraints as well as numerous decision variables even for relatively small instances of CPP, which prevents us from an efficient computation via the recent powerful optimization softwares mentioned above.

Furthermore, as a matter of fact, the instances solved to optimality in the literature are relatively small [28, 38, 72, 73], more specifically, up to $n = 158$ items. For instance, even when $n = 300$, there are already about 44 thousand decision variables and 13 million constraints in this standard IP formulation. Therefore, even for such middle-sized instances, preparing computational resources for describing the whole formulation would be hard in ordinary computational environments.

This drawback has already been pointed out by Kochenberger et al. [52, 87]. Due to the difficulties arising from this size issue of CPP, heuristic methods, which can not guarantee the optimality of the obtained solutions, have been extensively studied in the literature [6, 14, 19, 51, 52, 80, 87].

1.1.3 Focus

However, it is empirically known that optimization problems include a *redundancy*. For several decision variables, one may easily decide its values at optimal solutions. Then, such redundant decision variables can be regarded as constant terms in the model as we aim at finding a optimal solution. In a similar manner, there might be several redundant constraints. As we optimize the given objective function,

Furthermore, removing the redundancy often significantly shortens computation time as well as saves computational resources required, especially for problems arising from real world. According to a recent report by Achterberg et al. [1], *preprocessing*, also called the "presolve", is a very important process in order to achieve an efficient computation in their software, Gurobi Optimizer.

In preprocessing, we aim at detecting and removing the redundant parts, that is, redundant decision variables and constraints. For the basic but sophisticated techniques for general IP or MIP problems, see [7, 13, 42] for instance. Also, *reformulation* is becoming an important keyword as the computation time by the softwares depends on the formulation to some extent. Here, reformulation refers to an alternative formulation. Of course, reformulation should be more attractive in a sense, compared to the conventional one. In practice, we aim at finding a new formulation which is more excellent with the softwares.

With the substantial improvement of the performance of the optimization softwares in the last decades, methodologies for utilizing these softwares, including preprocessing and reformulation mentioned above, is becoming one of the important study topics in practice, as well as in theory.

1.1.4 Object

In order to make the CPP model applicable for larger, middle-sized, instances, in practice, in this thesis, we study on methodologies for reducing

the problem size of CPP. To this end, we focus on a *redundancy* in the standard IP formulation. Namely, we aim at reducing the problem size of CPP by removing a kind of redundancy based on a *preprocessing* and a *reformulation* mentioned above.

More specifically, we aim at developing an efficient algorithm, which detects and removes a redundancy for making the instances of CPP smaller as a *preprocessing*. The instances of reduced size can be solved efficiently by the recent powerful optimization softwares.

On the other hand, we also aim at revealing a redundancy in the standard IP formulation of CPP in a theoretical manner. Recall that the standard IP formulation has about 44 thousand decision variables and 13 million constraints when $n = 300$. Intuitively, the number of constraints is much larger than that of the decision variables. In this study, we give a simple sufficient condition for the redundant constraints, which yields a *reformulation* of CPP.

In contrast to existing studies [1, 7, 13, 42] focusing on a general setting, we utilize an underlying structure of CPP, "transitivity", for our purpose. Recall that CPP associated with clustering, and that a result of clustering can be seen as an equivalence relation. Therefore, in any solution of CPP, we encounter a transitivity: if i and j in the same cluster, and j and k in the same cluster, then, of course, i and k in the same cluster.

Also, in order to understand the limitation of our approach, we also address the *linear ordering problem* (LOP). LOP is an optimization problem on linear (or total) orders, which also satisfy a transitivity: if i is ordered higher than j, and j is ordered higher than k, then, of course, i is ordered higher than k. As a matter of fact, a natural and standard IP formulation of LOP introduced by Grötschel et al. [36, 63] has a similar structure as that of the standard IP formulation of CPP.

1.1.5 Results

We first develop a preprocessing algorithm for detecting redundant decision variables, which can be used as a preprocessing for the optimization softwares.

In this study, *redundant decision variables* refer to decision variables whose values at optimal are determined. Thus, the redundant variables are no longer "variables", and can be treated as constant terms in the problem. Therefore, our preprocessing algorithm can make the instances of CPP smaller by replacing the redundant variables with constant terms.

It the main part of the algorithm, we employ a classical technique called the *pegging test* with the *Lagrangian relaxation* [32, 34]. The pegging test is a powerful tool for detecting redundant variables, especially for problems with a simple structure such as the *knapsack problem* and its special variants [67, 68, 79, 89, 90]. In this study, we conceptually and experimentally show that

the pegging test is excellent with the transitivity in CPP.

Then, we address a reformulation. Our reformulation is based on the standard IP formulation of CPP. We give a theoretical characterization of the redundant constraints in the formulation.

In this study, *redundant constraints* refer to constraints such that removing them does not change the optimal solution set. Therefore, even if we disregard the redundant constraints, we can obtain an optimal solution of the original problem. Interestingly, we could confirm that a relatively large class of the constraints is easily revealed to be redundant. Moreover, the redundant constraints we found are characterized by a simple condition. Therefore, one can easily reduce the number of the constraints required, before we start to solve the problem.

1.1.6 Outline of Research

In what follows, we will briefly explain how we obtain the above results. To this end, several important existing studies related to our results will be explained.

On preprocessing

As mentioned above, the *pegging test* is known as an effective preprocessing, especially for the *knapsack problem* as demonstrated by Nauss [67]. As the name suggests, it can "peg" the values of several 0-1 variables at optimal. The usefulness of the pegging test has been confirmed for combinatorial optimization problems with simple structures, such as a capacitated facility location problem [68], and the knapsack problems with additional constraints [79, 90, 89].

On the other hand, we observe that the pegging test is also excellent with the transitivity in CPP. If we apply the pegging test to the standard IP formulation of CPP, we can find a pair of items $\{i, j\}$ such that i and j must be in the same cluster (clique) at any optimal solution. Now, suppose that we could show that the pair $\{i, j\}$ and the pair $\{j, k\}$ are in the same cluster (clique) at any optimal solution, respectively. Then, in order to meet the transitivity, we can conclude that the same thing holds for the pair $\{i, k\}$. Based on this simple observation, we develop an effective preprocessing algorithm for CPP.

In order to apply the pegging test, we need to solve some *relaxation problem*. In our algorithm, we employ the well-known method called the *Lagrangian relaxation* [32, 34]. The Lagrangian relaxation method is easy-to-implement, and is known to be excellent with the pegging test.

In the conventional and ordinary Lagrangian relaxation method, we need to deal with multipliers, called the *Lagrangian multipliers*, associated with the constraints of the original problem. Therefore, directly applying the

Lagrangian relaxation method to the standard IP formulation of CPP is prohibitive even for relatively small instances as it has numerous constraints. As a matter of fact, Charon and Hudry [20] encountered the same difficulty for LOP whose number of the constraints is too large, and failed to solve middle-sized instances.

Then, in order to make the Lagrangian relaxation method applicable for middle sized instances of LOP, Sukegawa et al. [84] proposed to simply modify an approach for the multipliers. A large part of multipliers are disregarded basically, but some of them will be added if needed. By this simple modification, they succeeded to solve middle-sized instances of LOP. Therefore, we employ their approach in our algorithm for CPP.

Our preliminary numerical experiments confirmed that the approach of Sukegawa et al. [84] is also effective for CPP. Especially, for the benchmark instances of Régnier's problem, our preprocessing algorithm efficiently reduces the problems size. The reduced problems can be solved by a powerful optimization software, Gurobi Optimizer, in seconds on an ordinary PC.

On reformulation

Our reformulation is motivated by a recent result in Dinh and Thai [30]. Dinh and Thai [30] dealt with the *modularity maximization problem* (MMP). As a matter of fact, MMP is known to be formulated as CPP. Therefore, several researchers tried to solve MMP via the standard IP formulation of CPP [2, 30, 64, 83].

Using a property of MMP in the objective function, Dinh and Thai [30] revealed a certain class of redundant constraints in the standard IP formulation of CPP, in a theoretical manner. Interestingly, in Dinh and Thai [30], the redundant constraints are characterized by a simple sufficient condition. Therefore, the number of the constraints can easily be reduced in advance. Moreover, their preliminary numerical experiments showed that this rewriting shortens computation time by a software, as well as saves computational resources required.

We address a generalization of the above result in Dinh and Thai [30]. Simplifying their analysis, we obtain a sufficient condition for the redundant constraints in the standard IP formulation of CPP, for any objective function. Our generalization enables us to use the approach of Dinh and Thai [30] for other important subclasses of CPP, such as Régnier's problem. Also, interestingly, we observe that our sufficient condition is slightly better than that of Dinh and Thai [30], even when the instances are limited to those of MMP.

Also, it should be noted that our and Dinh and Thai's approach is quite different from the existing studies. For problems with a large number of constraints, using the *cutting plane method* would be the most common approach. As a matter of fact, the exact algorithms proposed so far for

CPP [38, 72] are based on a cutting plane method.

In the cutting plane methods, basically, we aim at finding a set of "needed" constraints for solving the problem. In general, we solve many auxiliary problems, called the *relaxation problems*, in order to find the desired constraints. If an optimal solution for a relaxation problem is feasible at the original problem, then, we are done, that is, the current solution is optimal at the original problem. Otherwise, we need to find a new constraint which cuts off the current (infeasible) solution. Then, we again solve a new relaxation problem obtained from the previous one by adding several new constraints.

In contrast, our approach aims at theoretically characterizing "never needed", that is, redundant constraints using a problem structure. Therefore, the number of constraints treated in the algorithms can easily be reduced, without solving a large number of auxiliary problems like the cutting plane methods. This property is the biggest advantage of our approach.

1.2 Organization

The remainder of this thesis is organized as follows.

In Chapter 2, we introduce the clique partitioning problem (CPP) and its standard IP formulation, which is the main focus of this study. In order to show the utility of CPP, we show how Régnier's problem [76] reduces to CPP. The main contributions of this thesis are summarized in Chapter 3 and Chapter 4, respectively.

Chapter 3 summarizes our result on a preprocessing algorithm for CPP. First, we explain our basic idea. Then, we introduce the pegging test and the Lagrangian relaxation, which are the main parts of our algorithm. By this procedure, redundant decision variables are detected. After that, we introduce our reduction procedure. This reduction procedure rewrites the given instances into smaller but equivalent ones, based on the redundant variables. In order to see its performance, we conducted preliminary numerical experiments on several benchmark instances form real world, as well as randomly generated instances.

Chapter 4 summarizes our result on a reformulation of CPP. First, we introduce the recent result by Dinh and Thai [30], which is the motivation of our study. Then, we give a sufficient condition for the redundant constraints in the standard IP formulation of CPP, by generalizing and simplifying their analysis. Next, we consider to apply our analysis to the linear ordering problem (LOP). To this end, we explain LOP and its standard IP formulation. Then, we give several sufficient conditions for the redundant constraints in the formulation. In order to see how much we can detect the redundancy from real world instances, preliminary numerical experiments are conducted.

In Chapter 5, from a theoretical viewpoint of computational complexity, we show that an important subclass of CPP is intractable in theory, that is, NP-hard. The aim is to simply correct the proof in the recent work by Punnen and Zhang [75].

Finally, in Chapter 6, we summarize this thesis, and mention the future works, especially, focusing on a relationship with the new and active topic in the area of the combinatorial optimization [48].

Chapter 2

Problem

In this chapter, we introduce the clique partitioning problem (CPP). First, we give the formal definition of CPP. Then, we explain the standard integer programming (IP) formulation, which is the main focus of this study. In order to illustrate the usefulness of CPP, we also show how Régnier's problem [76] can be formulated as CPP.

2.1 Clique Partitioning Problem (CPP)

In this section, we define the clique partitioning problem. For simplicity, we define CPP as an optimization problem on graphs.

We denote an undirected graph G with vertex set V and edge set E by $G = (V, E)$. An edge $e \in E$ with endnodes $u, v \in V$ is denoted by $\{u, v\}$. For a subset $S \subseteq V$ of vertices, we denote the set of edges in G with both endnodes in S by $E(s)$, that is,

$$E(s) = \{\{u, v\} \in E \mid u, v \in S\}.$$

For a set of subsets $S_1, S_2, \ldots, S_k \subseteq V$ of vertices, let

$$E(S_1, S_2, \ldots, S_k) := \bigcup_{i=1}^{k} E(S_i).$$

A graph is called *complete* if every pair of its vertices is connected by an edge. A *clique* is a complete subgraph. We say that $\Gamma = \{W_1, W_2, \ldots, W_k\}$ is a partition of V if

$$\begin{aligned} &W_i \cap W_j = \emptyset \quad (i, j \text{ with } 1 \leq i < j \leq k), \\ &V = W_1 \cup W_2 \cup \cdots \cup W_k, \quad \text{and} \\ &W_i \neq \emptyset \quad (1 \leq i \leq k). \end{aligned}$$

A set A of edges in a graph $G = (V, E)$ is called a *clique partitioning* of G if there is a partition $\Gamma = \{W_1, W_2, \ldots, W_k\}$ of V such that

$$A = E(W_1, W_2, \ldots, W_k)$$

and the subgraph $(W_i, E(W_i))$ induced by W_i is a clique for $1 \leq i \leq k$. Note that k is not fixed. When G is complete, we observe that every partition of V induces a clique partitioning. In Figure 2.1, we show two examples of clique partitioning.

The clique partitioning problem (CPP) is formally defined as follows [38]. Given a complete graph $K = (V, E)$ with weights $c_e \in \mathbb{R}$ for all $e \in E$, find a clique partitioning $A \subseteq E$ such that

$$c(A) := \sum_{e \in A} c_e$$

is as large as possible.

In what follows, without loss of generality, we only focus on the cases when G is complete. When G is not complete, assigning a large negative weight c_e for each $e \notin E$, we could obtain an equivalent instance whose associated graph is complete.

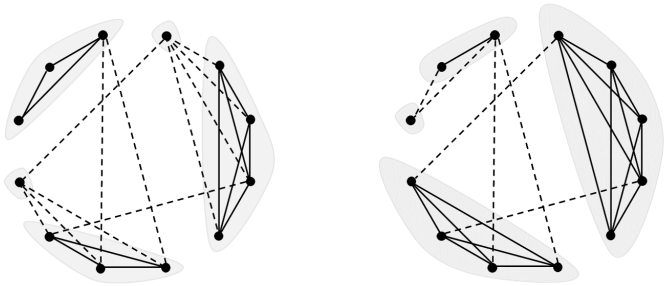
Interpretation and advantage

It should be noted that the edge weights c_e ($e \in E$) can take both negative and positive value. When applied to clustering tasks, the vertex set corresponds to the set of target items, and the edge weight c_e of $e = \{u, v\}$ simulates a "similarity" of item u and item v. Then, cliques are used to model "clusters", and hence we often call a clique cluster. We note that cluster does not necessarily means clique in general.

If c_e takes a large positive value, u and v are similar, while they are not similar if c_e takes a large negative value. Roughly speaking, in CPP, we want to take edges with large positive weights as much as possible within the cliques.

As we see from Figure 2.1, A clique partitioning corresponds to an equivalence relation, as each component, clique, can be regarded as representing an equivalence class. This is the reason why Régnier's problem, aggregation problem of equivalence relations, can be formulated as CPP, as we will confirm later.

Also, we observe that the number k of cliques, that is, clusters is not fixed in the definition of the clique partitioning. It is said that this property is useful in practice because we do not need to bother about the number of clusters in advance, and an appropriate number of clusters is automatically determined as we solved the problem.



$A_1 \subseteq E$ $A_2 \subseteq E$

- A_1: corresponds to a partition of V into 5 cliques
- A_2: corresponds to a partition of V into 4 cliques
- Dotted lines: edges not included in the clique partitioning

Figure 2.1: Two examples of clique partitioning A_1 and A_2 in an undirected graph $G = (V, E)$ with $|V| = 12$ vertices and $|E| = 23$ edges.

2.2 Integer Programming (IP) Formulation

In this section, we introduce the conventional integer programming (IP) formulation discussed in Grötschel and Wakabayashi [38], which is the focus of our study. For notational convenience, in what follows, let

$$\begin{aligned} &V := \{1, 2, \ldots, n\}, \\ &V^2_{\neq} := \{(i, j) \mid i, j \in V,\ i \neq j\}, \\ &V^3_{\neq} := \{(i, j, k) \mid i, j, k \in V,\ i \neq j, j \neq k, k \neq i\}. \end{aligned}$$

For a given clique partitioning A induced by $\Gamma = \{W_1, W_2, \ldots, W_k\}$, let x_{ij} be a 0-1 decision variable defined as

$$x_{ij} = \begin{cases} 1 & (i, j \in W_l \text{ for some } l \in \{1, 2, \ldots, k\}), \\ 0 & (\text{otherwise}) \end{cases} \qquad ((i, j) \in V^2_{\neq}),$$

then CPP is formulated as

$$\begin{aligned} \text{maximize} \quad & \frac{1}{2} \sum_{(i,j) \in V^2_{\neq}} c_{ij} x_{ij} & & \\ \text{subject to} \quad & x_{ij} \in \{0, 1\} & ((i, j) \in V^2_{\neq}) & \quad (\text{binary}), \\ & x_{ij} - x_{ji} = 0 & ((i, j) \in V^2_{\neq}) & \quad (\text{symmetry}), \\ & x_{ij} + x_{jk} - x_{ik} \leq 1 & ((i, j) \in V^3_{\neq}) & \quad (\text{transitivity}). \end{aligned}$$

Of course, the value of x_{ij} must coincide with that of x_{ji}. This is guaranteed by the *symmetry* constraint. Also, by the definition of the clique partitioning, if vertices i and j are in the same clique, and vertices j and k are in

the same clique, then vertices i and k must be in the same clique. In other words,

$$\{i,j\} \in A \wedge \{j,k\} \in A \;\Rightarrow\; \{i,k\} \in A.$$

This is guaranteed by the third constraints, called the *transitivity constraint*, as $x_{ij} = x_{jk} = 1$ implies $x_{ik} = 1$ from $x_{ij} + x_{jk} - x_{ik} \leq 1$.

We say that a solution $\boldsymbol{x} = (x_{ij})_{(i,j)\in V^2_{\neq}}$ is *feasible* if it satisfies all the constraints. Also, a feasible solution $\boldsymbol{x}$ is said to be *optimal* if it attains the maximum of the problem.

Halving the decision variables

Preparing both of x_{ij} and x_{ji} is unnecessary as they always take the same value. Therefore, in general, we substitute x_{ij} for x_{ji} for all $i, j \in V$ with $i < j$, and consider the following equivalent problem [38]:

(P_{CPP}) :

$$\begin{array}{lll}
\text{maximize} & \displaystyle\sum_{(i,j)\in V^2_<} c_{ij}x_{ij} & \\
\text{subject to} & x_{ij} \in \{0,1\} & ((i,j) \in V^2_<), \quad \text{(binary)},\\
& x_{ij} + x_{jk} - x_{ik} \leq 1 & ((i,j,k) \in V^3_<) \quad \text{(type } U\text{)},\\
& x_{ij} - x_{jk} + x_{ik} \leq 1 & ((i,j,k) \in V^3_<) \quad \text{(type } V\text{)},\\
& -x_{ij} + x_{jk} + x_{ik} \leq 1 & ((i,j,k) \in V^3_<) \quad \text{(type } W\text{)}.
\end{array}$$

where

$$V^2_< := \{(i,j) \mid i,j \in V,\ i < j\}, \quad \text{and}$$
$$V^3_< := \{(i,j,k) \mid i,j,k \in V,\ i < j < k\}.$$

We observe that the symmetry constraints vanished, and the number of the decision variables is halved. On the other hand, the inequality constraints separated into three types. We call them (type U), (type V), and (type W) respectively.

Existing studies

Grötschel and Wakabayashi [38] proposed a cutting plane algorithm based on their polyhedral study [39] on the *clique partitioning polytope*, which is the *convex hull* of the feasible region of (P_{CPP}). More specifically, they studied the facial structure of the clique partitioning polytope, and found several useful *facet defining inequalities* of the polytope. Facet defining inequalities are powerful constraints for cutting off the infeasible solutions generated by the cutting plane algorithms.

As a matter of fact, the transitivity constraint of (P_{CPP}) is known to be a fundamental facet defining inequality of the clique partitioning polytope [39].

Grötschel and Wakabayashi [38] confirmed that considering only a small fraction of the transitivity constraints often suffices to solve the instances of CPP from real world, which implies that there exists a great "redundancy" in the constraints in the formulation (P_{CPP}). For subsequent researches on exact methods for CPP, see [28, 73].

However, as pointed out by Oosten et al. [72], the transitivity constraints do not suffice to solve the instances of the *group technology problem*, an important subclass of CPP. Then, they addressed a better understanding of the facial structure of the clique partitioning polytope, and developed a sophisticated algorithm for efficiently solving relatively small instances of the group technology problem.

In light of the above difficulties, heuristic methods, which can not guarantee the optimality of the final solution, have been extensively studied in the literature [6, 14, 19, 51, 52, 80, 87].

Our motivation

The point is that the standard IP formulation (P_{CPP}) above has $n(n-1)/2 = O(n^2)$ binary decision variables, and $n(n-1)(n-2)/2 = O(n^3)$ inequality constraints, all of which grow very rapidly as n grows.

As pointed out by Kochenberger et al. [52, 87], a main drawback of the existing algorithms is on that they employed the standard IP formulation (P_{CPP}). However, intuitively, (P_{CPP}) may generally contain some redundant constraints. In other words, some constraints may be satisfied automatically as the objective function is maximized.

As a matter of fact, as mentioned above, Grötschel and Wakabayashi [38] experimentally confirmed that, for many relatively small instances arising from qualitative data analysis, their cutting plane algorithm terminates when only a small fraction of transitivity constraints are added. In other words, there might be a great redundancy in the transitivity constraints of the formulation (P_{CPP}).

In a similar manner, for some decision variables x_{ij} with a large positive or negative weight c_{ij}, we may easily estimate the value of x_{ij} at optimal. If we could find such variables, we may reduce the problem size as such redundant variables are no longer variables. The results obtained in this study are motivated by these simple intuitions.

2.3 Example: Régnier's problem

In this section, we explain Régnier's problem, an important special case of CPP, and show how it can be formulated as an instance of CPP. In our preliminary numerical experiments, several benchmark instances of Régnier's problem are employed.

Table 2.1: 6 wild cats characterized by 3 qualitative attributes [38].

	Qualitative Attributes		
Name	Pelt	Fur	Ears
Lion	uniform	short	round
Tigre	stripe	short	round
Jaguar	spots	short	round
Leopard	spots	short	round
Once	spots	long	round
Serval	spots	short	pointed

Suppose that we are given Table 2.1, which is a part of the table "Classification of wild cats" in Grötschel and Wakabayashi [38], and also that now we want to classify the wild cats into several similar groups, that is, clusters. The idea of Régnier [77] is to assume that each "qualitative" attribute defines an equivalence relation on the set of the target items

$$V = \{\text{Lion}, \text{Tigre}, \ldots, \text{Serval}\}.$$

For instance, one could say that the qualitative attribute "Pelt" defines three equivalence classes, "uniform", "stripe", and "spots", that is, the following partition of V,

$$\{\{\text{Lion}\}, \{\text{Tigre}\}, \{\text{Jaguar}, \text{Leopard}, \text{Once}, \text{Serval}\}\}.$$

Then, we could obtain three equivalence relations, $\sim^{\text{Pelt}}$, $\sim^{\text{Fur}}$, and $\sim^{\text{Ears}}$ corresponding to these qualitative attributes.

Formulation as CPP

As mentioned above, Régnier [77] proposes to aggregate the set of equivalence relations into one equivalence relation. Now, suppose that we are given q equivalence relations $\sim^1, \sim^2, \ldots, \sim^q$ on $V = \{1, 2, \ldots, n\}$, in general.

Régnier [77] defines a distance $d(\sim^s, \sim^t)$ of two equivalence relations $\sim^s$ and $\sim^t$ as their total number of disagreements, that is,

$$d(\sim^s, \sim^t) := |\{\{i, j\} \mid (i \sim^s j \wedge i \not\sim^t j) \vee (i \not\sim^s j \wedge i \sim^t j)\}|.$$

If we express equivalence relations $\sim^1, \sim^2, \ldots, \sim^q$ and a solution $\sim^X$ as binary vectors as follows

$$r_{ij}^k = \begin{cases} 1 & (i \sim^k j), \\ 0 & (\text{otherwise}) \end{cases} \quad (k \in \{1, 2, \ldots, q\}),$$

and

$$x_{ij} = \begin{cases} 1 & (i \sim^X j), \\ 0 & (\text{otherwise}), \end{cases}$$

for each $(i,j) \in V^2_<$, then we have

$$d(\sim^k, \sim^X) = \sum_{(i,j)\in V^2_<} \left(r^k_{ij} - x_{ij}\right)^2 = \sum_{(i,j)\in V^2_<} r^k_{ij} + \left(1 - 2r^k_{ij}\right) x_{ij}.$$

Recall that $V^2_< := \{(i,j) \mid i,j \in V,\ i < j\}$.

The Régnier's problem [77] aims at finding an equivalence relation $\sim^X$ that minimizes the total distance form the given equivalence relations $\sim^1$, $\sim^2, \ldots, \sim^q$, which can be simplified to

$$\sum_{k=1}^{q} d(\sim^k, \sim^X) = c + \sum_{(i,j)\in V^2_<} \left(\sum_{k=1}^{q} (1 - 2r^k_{ij})\right) x_{ij} = c + \sum_{(i,j)\in V^2_<} \tilde{c}_{ij} x_{ij}$$

where c is a constant term depending only on $\sim^1, \sim^2, \ldots, \sim^q$. In order to be an equivalence relation, $(x_{ij})_{(i,j)\in V^2_<}$ must satisfy the constraints of (P_{CPP}). Therefore, setting $c_{ij} = -\tilde{c}_{ij}$, we observe that the instances of Régnier's problem reduce to an instances of (P_{CPP}). This implies that Régnier's problem is a special case of CPP.

The values c_{ij} can be regarded as "similarity" of the two items i and j. For instance, in Table 2.1, this similarity for "Jaguar" and "Leopard" is 3, while that of "Once" and "Serval" is -1. All the values can be summarized as the following symmetric matrix:

$$\begin{pmatrix} - & 1 & 1 & 1 & -1 & -1 \\ & - & 1 & 1 & -1 & -1 \\ & & - & 3 & 1 & 1 \\ & & & - & 1 & 1 \\ & & & & - & -1 \\ & & & & & - \end{pmatrix}.$$

Finally, we note that Régnier's problem [77] is referred to as Zahn's problem [91] when the number of given equivalence relations, that is, $q = 1$, and the given relation is not necessarily an equivalence relation. It is not difficult to see that Zahn's problem is also formulated as an instance of CPP. For the details, see de Amorim et al. [6] for instance.

Chapter 3

Preprocessing

As we observed in the previous chapter, the standard IP formulation (P_{CPP}) suffers from numerous constraints as well as decision variables. This chapter discusses a methodology for detecting and removing a redundancy from decision variables, which leads to a problem size reduction.

Recall that *redundant decision variables* refer to decision variables whose values at optimal solutions are determined, in this study. Therefore, redundant decision variables can be treated as constant terms, which leads to a problem size reduction.

Contents

More specifically, in this chapter, we propose a specific preprocessing algorithm for reducing the problem size of CPP. Our preprocessing algorithm consists of two parts.

The first is an algorithm for detecting redundant decision variables. We call this algorithm *basic algorithm*. The second is a reduction procedure for rewriting a given instance into a smaller but equivalent one, based on the redundant variables detected by the basic algorithm.

Organization

First, we explain the basic idea for detecting the redundant decision variables. After that, the Lagrangian relaxation and the pegging test are introduced as specific tools for realizing the idea. So far, we do not use any structure of CPP.

Next, utilizing the structure of CPP, that is, transitivity, we propose a way for strengthening the conventional pegging test in our basic algorithm. Then, we explain our simple reduction procedure. Finally, to see its effectiveness, we report on our preliminary numerical experiments on several benchmark instances of Régnier's problem as well as randomly generated instances.

3.1 Basic Idea

In this section, we first explain the basic idea for finding redundant decision variables. After that, the Lagrangian relaxation and the pegging test are introduced as specific tools for realizing the idea.

Let (P) be some optimization (maximization) problem, and denote by $\omega(P)$ its optimal value. Suppose that there is a binary decision variable named x_i in (P). We denote by $(P \mid x_i = \xi)$ the subproblem obtained from P by setting $x_i = \xi \in \{0, 1\}$. Then, if we have

$$\omega(P \mid x_i = \xi) < \omega(P), \tag{3.1.1}$$

we can conclude that there is no optimal solution satisfying $x_i = \xi$. In other words, to solve (P), we can fix the value of x_i at $1 - \xi$. Therefore, we could reduce the number of the decision variables by one.

However, in general, computing $\omega(P \mid x_i = \xi)$ and $\omega(P)$ would be as hard as solving the original problem (P). Therefore, we cannot use the above discussion directly to a problem size reduction. On the other hand, one could consider to "indirectly" prove the above inequality (3.1.1) by estimating an upper and lower bounds of $\omega(P \mid x_i = \xi)$ and $\omega(P)$, respectively.

More specifically, if we know a value $\underline{\omega}$ with $\underline{\omega} \leq \omega(P)$, and a value $\overline{\omega}(P \mid x_i = \xi)$ with $\omega(P \mid x_i = \xi) \leq \overline{\omega}(P \mid x_i = \xi)$, then we can concluded that x_i must be $1 - \xi$ in any optimal solution if

$$\overline{\omega}(P \mid x_i = \xi) < \underline{\omega} \tag{3.1.2}$$

holds. This is the basic idea of the *pegging test*. Needless to say, the key for success depends on the qualities of $\overline{\omega}(P \mid x_i = \xi)$ and $\underline{\omega}$, and the computational burden for computing these two values.

How to get sharp $\overline{\omega}(P \mid x_i = \xi)$ and $\underline{\omega}$

To this end, the *Lagrangian relaxation* has been employed in the literature [83, 84, 89, 90] to calculate sharp $\overline{\omega}(P \mid x_i = \xi)$ efficiently. The Lagrangian relaxation method is known as an effective method for integer programming problems including (P_{CPP}) [32, 34, 54].

In the approaches based on the Lagrangian relaxation, in general, we first obtain an upper bound $\overline{\omega}(P)$. The strongest point of this approach is that $\overline{\omega}(P \mid x_i = \xi)$ can easily be computed from $\overline{\omega}(P)$ by subtracting some value for each decision variable x_i and for each $\xi \in \{0, 1\}$, as we will confirm later.

On the other hand, we also need to calculate sharp $\underline{\omega}$ efficiently. To this end, in our algorithm, we employ the *noising method* by Charon and Hudry [19], which is known to be very effective and stable for CPP.

3.1.1 Lagrangian Relaxation

Then, let us explain the details of the *Lagrangian relaxation*. Here, we show how the Lagrangian relaxation method woks for (P_{CPP}).

In the Lagrangian relaxation method, we remove the transitivity constraints from the constraints, and add them to the objective function as a penalty function. More formally, we consider to solve the following *Lagrangian relaxation problem*:

$(LR_{\mathrm{CPP}}(\boldsymbol{u},\boldsymbol{v},\boldsymbol{w}))$:

$$\begin{aligned}
\text{maximize} \quad & \sum_{(i,j)\in V^2_<} c_{ij}x_{ij} \quad + \sum_{(i,j,k)\in V^3_<} u_{ijk}(1-x_{ij}-x_{jk}+x_{ik}) \\
& \qquad\qquad + \sum_{(i,j,k)\in V^3_<} v_{ijk}(1-x_{ij}+x_{jk}-x_{ik}) \\
& \qquad\qquad + \sum_{(i,j,k)\in V^3_<} w_{ijk}(1+x_{ij}-x_{jk}-x_{ik}) \\
\text{subject to} \quad & x_{ij}\in\{0,1\} \qquad ((i,j)\in V^2_<)
\end{aligned}$$

where

$$(\boldsymbol{u},\boldsymbol{v},\boldsymbol{w}) = \Big((u_{ijk})_{(i,j,k)\in V^3_<}, (v_{ijk})_{(i,j,k)\in V^3_<}, (w_{ijk})_{(i,j,k)\in V^3_<}\Big)$$

is a non-negative vector called the *Lagrangian multiplier vector*.

As the name suggests, the penalty function imposes a penalty for violating the transitivity constraints. It is not difficult to see that

$$\omega(LR_{\mathrm{CPP}}(\boldsymbol{u},\boldsymbol{v},\boldsymbol{w})) \geq \omega(P_{\mathrm{CPP}})$$

for any $(\boldsymbol{u},\boldsymbol{v},\boldsymbol{w}) \geq \boldsymbol{0}$ as any feasible solution $\boldsymbol{x}$ of (P_{CPP}) is also feasible at $(LR_{\mathrm{CPP}}(\boldsymbol{u},\boldsymbol{v},\boldsymbol{w}))$, and the objective value of $\boldsymbol{x}$ in $(LR_{\mathrm{CPP}}(\boldsymbol{u},\boldsymbol{v},\boldsymbol{w}))$ is grater than or equal to that of $\boldsymbol{x}$ in (P_{CPP}).

Let $r(\boldsymbol{u},\boldsymbol{v},\boldsymbol{w})_{ij}$ denote the coefficient of variable x_{ij} in the objective function of $(LR_{\mathrm{CPP}}(\boldsymbol{u},\boldsymbol{v},\boldsymbol{w}))$. More specifically, the coefficient $r(\boldsymbol{u},\boldsymbol{v},\boldsymbol{w})_{ij}$ is given by

$$\begin{aligned}
c_{ij} \quad & - \sum_{k:(i,j,k)\in V^3_<} u_{ijk} \quad - \sum_{k:(k,i,j)\in V^3_<} u_{kij} \quad + \sum_{k:(i,k,j)\in V^3_<} u_{ikj} \\
& - \sum_{k:(i,j,k)\in V^3_<} v_{ijk} \quad + \sum_{k:(k,i,j)\in V^3_<} v_{kij} \quad - \sum_{k:(i,k,j)\in V^3_<} v_{ikj} \\
& + \sum_{k:(i,j,k)\in V^3_<} w_{ijk} \quad - \sum_{k:(k,i,j)\in V^3_<} w_{kij} \quad - \sum_{k:(i,k,j)\in V^3_<} w_{ikj}.
\end{aligned}$$

We observe that the Lagrangian relaxation problem $(LR_{\mathrm{CPP}}(\boldsymbol{u},\boldsymbol{v},\boldsymbol{w}))$ is easily solved by just checking the signs of the above coefficients. However, needless to say, the quality of $\omega(LR_{\mathrm{CPP}}(\boldsymbol{u},\boldsymbol{v},\boldsymbol{w}))$, that is, the closeness to $\omega(P_{\mathrm{CPP}})$, depends on the choice of the Lagrangian multiplier vector

$(\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w})$. The problem to find a Lagrangian multiplier vector $(\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w})$ that minimizes the corresponding upper bound $\omega(LR_{\mathrm{CPP}}(\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w}))$ is referred to as the *Lagrangian dual* problem.

It is well-known that the *subgradient method* [32, 34, 54] efficiently (approximately) solves the Lagrangian dual problem. Therefore, we simply employ the subgradient method in our algorithm. Although there are several parameters for the method, we employ a most common approach. See Umetani and Yagiura [85] for the choice of parameters in the subgradient method.

A drawback and our approach

Charon and Hudry [20] proposes an algorithm using the Lagrangian relaxation method for the *linear ordering problem* (LOP). However, their algorithm was only applicable to relatively small instances, with up to 100 items, as we need to deal with numerous multipliers corresponding to the numerous constraints in its standard formulation.

In order to overcome this difficulty, Sukegawa et al. [84] proposed to simply modify the Lagrangian relaxation method in such a way that a large part of the multipliers are disregarded, that is, set to 0. Their algorithm also uses the pegging test in its main part, and succeeds to solve a real world problem with 347 items arising from a sports team ranking problem.

Since we also encounter a large number of multipliers in $(LR_{\mathrm{CPP}}(\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w}))$, we employ the approach in Sukegawa et al. [84], and make the Lagrangian relaxation method applicable to middle-sized instances of CPP.

3.1.2 Pegging Test

Suppose that $\bar{\boldsymbol{x}}$ is an optimal solution of $(LR_{\mathrm{CPP}}(\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w}))$. Furthermore, suppose that we have $\bar{x}_{ij} = 0$ for some $(i, j) \in V^2_<$. Then, the coefficient $r(\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w})_{ij}$ of x_{ij} in $(LR_{\mathrm{CPP}}(\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w}))$ must be non-positive.

In this case, we have

$$\omega(LR_{\mathrm{CPP}}(\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w}) \mid x_{ij} = 1) = \omega(LR_{\mathrm{CPP}}(\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w})) + r(\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w})_{ij}.$$

On the other hand, we observe that the left hand side $\omega(LR_{\mathrm{CPP}}(\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w}) \mid x_{ij} = 1)$ is an upper bound of the optimal value $\omega(P_{\mathrm{CPP}} \mid x_{ij} = 1)$ of the subproblem $(P_{\mathrm{CPP}} \mid x_{ij} = 1)$. Therefore, if we have

$$\underline{\omega} > \omega(LR_{\mathrm{CPP}}(\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w})) + r(\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w})_{ij}$$

for some lower bound $\underline{\omega}$ on the optimal value $\omega(P_{\mathrm{CPP}})$ of the original problem (P_{CPP}), we can conclude that there is no optimal solution $\boldsymbol{x}^*$ with $x^*_{ij} = 1$. In other words, we can fix the value of the decision variable x_{ij} at 0 without loss of optimality, which reduces the number of decision variables.

A similar argument holds for the case with $\bar{x}_{ij} = 1$. More formally, we have the following simple proposition.

Proposition 1. *Let $\bar{\boldsymbol{x}}$ be an optimal solution of the Lagrangian relaxation problem $(LR_{CPP}(\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w}))$. If*

$$\omega(LR_{CPP}(\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w})) - \underline{\omega} < |r(\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w})_{ij}| \tag{3.1.3}$$

holds, then $x^*_{ij} = \bar{x}_{ij}$ for any optimal solution $\boldsymbol{x}^*$ of (P_{CPP}).

We say that a decision variable x_{ij} is *pegged* at $\bar{x}_{ij}$ if the inequality (3.1.3) in Proposition 1 holds for some Lagrangian multiplier vector $(\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w})$. We observe that it is very easy to decide whether the above inequality holds or not, for each decision variables, once we solved the Lagrangian relaxation problem $(LR_{\mathrm{CPP}}(\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w}))$.

3.2 Algorithm

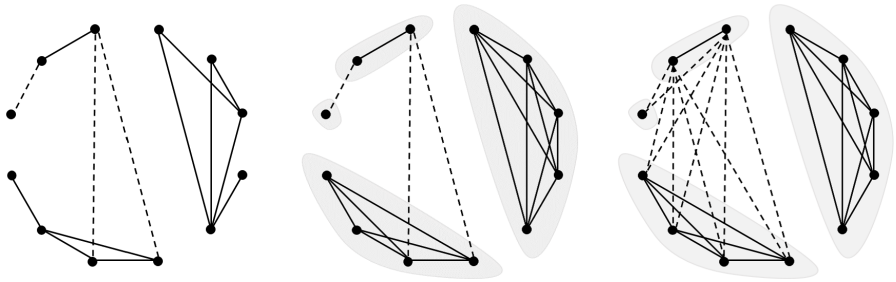
In this section, we explain our preprocessing algorithm for detecting the redundant decision variables in (P_{CPP}). As already mentioned, our preprocessing algorithm consists of two parts: *basic algorithm* and *reduction procedure*. We show how the transitivity of CPP can be utilized in the basic algorithm as well as the reduction procedure.

3.2.1 Basic Algorithm

Basically, we first obtain an (nearly) optimal Lagrangian multiplier vector $(\bar{\boldsymbol{u}}, \bar{\boldsymbol{v}}, \bar{\boldsymbol{w}})$ by the subgradient method, and then, apply the pegging test for this vector $(\bar{\boldsymbol{u}}, \bar{\boldsymbol{v}}, \bar{\boldsymbol{w}})$, that is, check the inequality (3.1.3) in Proposition 1 for each decision variable. The pegged variables are treated as constant terms in our algorithm. This is the basic procedure frequently employed in our algorithm.

However, as mentioned above, in order to make our algorithm applicable for middle-sized instances, we need to avoid for dealing with a large number of multipliers. To this end, like Sukegawa et al. [84], we consider to disregard a large part of the multipliers by fixing the values of the multipliers to zero. To this end, we introduce $U, V, W \subseteq V^3_<$ as sets of indices of the multipliers under consideration, for the transitivity constraints (type U), (type V), and (type W), respectively.

We start with (U, V, W) such that its number, $|U|+|V|+|W|$, is manageable size, and gradually make (U, V, W) larger. Once (U, V, W) is updated, we again apply the basic procedure, that is, the subgradient method and the pegging test, mentioned above.



(P_0, P_1): original (P_0, P_1'): Rule A (P_0', P_1'): Rule B

- Solid lines: a set of variables pegged at 1
- Dotted lines: a set of variables pegged at 0

Figure 3.1: Application of Rule A and Rule B for a graph G with (P_0, P_1).

Utilizing the transitivity for detecting more redundancy

Let $P_0 \subseteq E$ (resp. $P_1 \subseteq E$) be a set of edges corresponding to the decision variables pegged at 0 (resp. 1) by our algorithm. In order for a problem size reduction, we want to peg as many decision variables as possible. Namely, we want to make P_0 and P_1 as large as possible.

The point is how to utilize the structure of CPP for the pegged variables. We observe that P_0 and P_1 can be enlarged by the following simple two procedures, which are illustrated in Figure 3.1.

Rule A (enlargement of P_1):
Suppose that we have some variable, say x_{ij}, pegged at 1. In other words, now, we know that vertex i and j must be in the same clique in the optimal solutions. Furthermore, suppose that the same holds for j and k. Then, even if $\{i, k\} \notin P_1$, we could update P_1' as $P_1 \cup \{\{i, k\}\}$ due to the transitivity constraint $x_{ij} + x_{jk} - x_{ik} \leq 1$. Such a procedure can easily be done systematically by applying the *depth-first search* on a graph (V, P_1).

Rule B (enlargement of P_0):
Suppose that the graph (V, P_1') is divided into several connected components

$$\{(V^1, P_1^1), (V^2, P_1^2), \ldots, (V^l, P_1^l)\}.$$

Now, by the above procedure Rule A, any edge e connecting a pair of vertices in the same component satisfy $e \in P_1$. Suppose that $e = \{i, j\} \in P_0$ for some $i \in V^s$ and $j \in V^t$ with $s \neq t$. Then, to meet the transitivity constraints, $e' = \{i', j'\} \in P_0$ for each $i' \in V^s$ and $j' \in V^t$. Therefore, we can update P_0 to P_0' based on this simple fact.

Table 3.1: Formal description of the basic algorithm.

Basic Algorithm:

Input: an undirected complete graph $G = (V, E)$ and an edge weight function $c : E \to \mathbb{R}$.

Output: a pair $(P_0, P_1) \subseteq E \times E$ corresponding to the decision variables pegged at 0 and 1, respectively.

Step 0 (Initialization): Let (U, V, W) be some small set. Compute $\underline{\omega}$ by the noising method [19]. Below, disregard the multipliers whose corresponding indices are not in (U, V, W), that is, fix them at 0.

Step 1 (Lagrangian relaxation and pegging test): Apply the subgradient method [32, 34, 54] to obtain an (nearly) optimal Lagrangian multiplier vector $(\bar{\boldsymbol{u}}, \bar{\boldsymbol{v}}, \bar{\boldsymbol{w}})$ for the current (U, V, W). Perform the pegging test for $(\bar{\boldsymbol{u}}, \bar{\boldsymbol{v}}, \bar{\boldsymbol{w}})$ to obtain (P_0, P_1).

Step 3 (enlargement of (P_0, P_1) and termination): By Rule A and Rule B, obtain enlarged (P_0', P_1'). If the improvement of the upper bound becomes sufficiently small, stop.

Step 4 (update of (U, V, W)): Find transitivity constraints violated by an optimal solution of $(LR_{\mathrm{CPP}}(\bar{\boldsymbol{u}}, \bar{\boldsymbol{v}}, \bar{\boldsymbol{w}}))$ at hand, and go to Step 1 by adding the corresponding indices to the current (U, V, W).

In Table 3.1, we show the formal description of the basic algorithm. In Step 4 (update of (U, V, W)), for large n, adding all the violated transitivity constraints may run out of the computational resources immediately. Therefore, we impose a limit on the number of constraints added in each Rule As in Grötschel and Wakabayashi [38].

Finally, we note that the pegging test can easily be strengthened when P_0 or P_1 is non-empty. Suppose that x_{st} is not pegged yet, while we know subsets S and T of V such that $i \in S$ implies $\{i, s\} \in P_1$ and $j \in T$ implies $\{j, t\} \in P_1$. Now, assume that there is an optimal solution $\boldsymbol{x}^*$ with $x^*_{st} = 1$. Then, by the definition of P_1, we have $x^*_{ij} = 1$ for each $i \in S$ and for each $j \in T$. Therefore, we can conclude that

$$\omega(LR_{\mathrm{CPP}}(\boldsymbol{u}, \boldsymbol{v}, \boldsymbol{w}) \mid x_{st} = 1, \quad \begin{array}{l} x_{ab} = \xi \ (\forall \{a, b\} \in P_\xi), \\ x_{ij} = 1 \ (\forall i \in S, \forall j \in T)) \end{array} \tag{3.2.1}$$

is an upper bound of the subproblem with $x_{st} = 1$, while the naive pegging

test in Proposition 1 estimate it by

$$\omega(LR_{\mathrm{CPP}}(\boldsymbol{u},\boldsymbol{v},\boldsymbol{w}) \mid x_{st}=1, \quad x_{ab}=\xi \ (\forall\{a,b\}\in P_\xi)) \tag{3.2.2}$$

which is larger, that is, worse than (3.2.1) as (3.2.1) has more constraints than that of (3.2.2), in general. In sum, more formally, we have the following.

Theorem 1 (Sukegawa et al. [83])**.** *Let $\bar{\boldsymbol{x}}$ be an optimal solution of the Lagrangian relaxation problem $(LR_{CPP}(\boldsymbol{u},\boldsymbol{v},\boldsymbol{w}))$. If*

$$\omega(LR_{CPP}(\boldsymbol{u},\boldsymbol{v},\boldsymbol{w})) - \underline{\omega} < \sum_{(i,j)\in R_\xi} |r(\boldsymbol{u},\boldsymbol{v},\boldsymbol{w})_{ij}| \tag{3.2.3}$$

holds for some $\xi\in\{0,1\}$, then we have $x^*_{st}=1-\xi$ for any optimal solution $\boldsymbol{x}^*$ of the original problem, where

$$R_\xi = \{(i,j)\in ST_= \mid \bar{x}_{ij}=\xi\}\cup\{(i,j)\mid\{i,j\}\in P_{(1-\xi)}\}$$

and $ST_=$ is a set of indices corresponding to the edges connecting S and T.

Therefore, in the basic algorithm shown in Table 3.1, we employ the inequality (3.2.3) in place of (3.2.2) when P_0 or P_1 is non-empty. Our preliminary numerical experiments confirmed that (3.2.3) is considerably superior to (3.2.2) for finding redundant variables.

3.2.2 Reduction Procedure

Next, we consider to reduce the problem size based on (P_0,P_1) obtained from the basic algorithm in Table 3.1 as follows.

Rule A' (utilizing P_1):
We first apply a simple operator "shrink" to the vertices for utilizing P_1. Suppose that $\{u_1,u_2\}\in P_1$. In other words, we now know that vertex u_1 and u_2 must be in the same clique at any optimal solution. Then, we shrink these two vertices into a single vertex u without loss of optimality.

More specifically, we obtain a smaller but equivalent new instance $(G'=(V',E'),c')$ of CPP from the original one $(G=(V,E),c)$ as follows. Let $V'=V\cup\{u\}\setminus\{u_1,u_2\}$, $E'=\{\,\{i,j\}\mid i,j\in V', i\neq j\,\}$, and

$$c'_e=\begin{cases} c_{\{i,j\}} & (i\neq u \text{ and } j\neq u)\\ c_{\{u_1,j\}}+c_{\{u_2,j\}} & (i=u \text{ and } j\neq u)\end{cases} \quad (\forall e=\{i,j\}\in E').$$

The optimal values of these two instances differ by a constant $c_{u1,u2}$. It is not difficult to see that the following argument is easily generalized for the shrinking of any number of vertices.

Table 3.2: Formal description of the reduction procedure.

Reduction Procedure:

Input: Output of the basic algorithm, that is, a pair $(P_0, P_1) \subseteq E \times E$ corresponding to the decision variables pegged at 0 and 1, respectively.

Output: an undirected complete graph $G' = (V', E')$ and an edge weight function $c' : E \to \mathbb{R}$, and $P'_0 \subseteq E$ corresponding to the decision variables pegged at 0 in the reduced instance $(G' = (V', E'), c')$.

Step 0 (utilization of P_1): Obtain $(G' = (V', E'), c')$ by Rule A'.

Step 1 (utilization of P_0): By Rule B', obtain P'_0 from P_0.

Rule B' (utilizing P_0):
Let $(G' = (V', E'), c')$ be the reduced instance obtained from the original one $(G = (V, E), c)$ by Rule A'. Suppose that $\{s', t'\}$ is a pair of vertices of V' obtained by shrinking a set of vertices S and T of the original graph $G = (V, E)$, respectively.

If there is a pair of vertices $s \in S$ and $t \in T$ such that the corresponding variables are pegged at 0, that is, $(s, t) \in P_0$ or $(t, s) \in P_0$, then we can conclude that s' and t' cannot be in the same clique at any optimal solution of the reduced instance $(G' = (V', E'), c')$. In this way, we could obtain a set P'_0 of indices corresponding to the decision variables which can be pegged at 0 in the reduced instance $(G' = (V', E'), c')$.

One may consider to apply the above reduction procedure in the basic algorithm. However, frequently performing such a rewriting may cause a computational burden. Therefore, we propose to apply this reduction procedure only once after the application of the basic algorithm. In Table 3.2, we show the formal description of the reduction procedure.

3.3 Preliminary Numerical Experiments

In this subsection, we report on the preliminary numerical experiments of our preprocessing algorithm. We coded the preprocessing algorithm in Java, and run it on an Windows-based PC with an Intel Core 2, 1.79 GHz processor and 2.99 GB RAM.

In what follows, we briefly summarize this preliminary numerical experiments.

We first show the detailed results on the basic algorithm for the real world benchmark instances of Régnier's problem shown in Table 3.3. Then, in order to see its performance for larger, middle-sized instances, we introduce several randomly generated instances shown in Table 3.5, which imitates the real world instances in Table 3.4, and show the detailed results of the basic algorithm on these instances in Table 3.6.

Although the basic algorithm is not designed to solve the instances to optimality, but to reduce the problem size, we confirmed that it solves several instances we tested to optimality, see Table 3.4 and Table 3.6 for the details. For the instances which can not be solved by the basic algorithm, we apply the reduction procedure to obtain small reduced instances. The results are summarize in Table 3.7.

Finally, we show the results on the comparison of our approach with the naive one directly applying an optimization software. To this end, we employ the Gurobi Optimizer. More specifically, for each instance, we show the sum of the computation time for our preprocessing algorithm and that for solving the reduced instances (by Gurobi Optimizer) to optimality, and compare it by that for directly applying Gurobi Optimizer to the instance. See Table 3.8 for the details.

3.3.1 On Basic Algorithm

In order to evaluate the performance of the basic algorithm, we basically employ the following three criteria.

- # Redundant Variables (%): the percentage of the redundant variables detected by the basic algorithm, that is, $100(|P_0| + |P_1|)/|V_<^2|$, where the symbol "–" means that the basic algorithm solves the instance to optimality before finding any redundant variables.
- Comput. Time (s): the computation time in seconds required for the basic algorithm.
- Status: Solved (solved to optimality by using only the basic algorithm), Reduced (several redundant variables are found), or Unchanged (No redundant variables are found).

Benchmark instances

In Table 3.3, we show the details of the benchmark instance of Régnier's problem we tested. All of these instances are obtained from Grötschel and Wakabayashi [38] or Brusco and Köhn [14]. In Table 3.4, we show the results of our basic algorithm for these instances.

We observe that all the instances but `Workers` are solved to optimality by the basic algorithm. Although the basic algorithm is not designed to

Table 3.3: The benchmark instances of Régnier's problem.

Name	Source	# Vertices	# Relations	Correlatedness
Wildcats	[38]	30	14	0.76
Cars	[38]	33	13	0.78
Workers	[38]	34	13	0.73
Cetacea	[38]	36	15	–
Micro	[38]	40	14	0.71
Soybean	[14]	47	21	0.71
UNO	[38]	54	3	0.69
UNO1b	[38]	139	3	0.93
UNO2b	[38]	145	15	0.86
UNO1a	[38]	158	3	0.91
UNO2a	[38]	158	15	0.86

Table 3.4: Results of the basic algorithm for the instances in Table 3.3.

Instance		Results of Basic Algorithm		
Name	# Vertices	# Redundant Variables (%)	Comput. Time (s)	Status
Wildcats	30	89.1	0.02	Solved
Cars	33	–	0.07	Solved
Workers	34	92.3	0.12	Reduced
Cetacea	36	–	0.01	Solved
Micro	40	82.9	0.17	Solved
Soybean	47	–	0.03	Solved
UNO	54	–	0.01	Solved
UNO1b	139	–	0.12	Solved
UNO2b	145	98.6	0.63	Solved
UNO1a	158	–	0.18	Solved
UNO2a	158	99.9	0.60	Solved

solve the instance to optimality, interestingly, for these real world instances, the upper found by the basic algorithm successfully coincide with the lower bound. We observe that the basic algorithm succeeds to solve several instances before the pegging test finds the redundant variables, which are indicated as "–" in the column "# Redundant Variables (%)".

It should be noted that fast computation for the instances with more than 100 vertices is due to our modification of the subgradient method.

We confirmed that only a small fraction of the transitivity constraints are needed, which yields a high performance for these middle sized instances. The results in Table 3.4 show a potential applicability of our basic algorithm for larger, middle-sized instances of CPP. Then, we address larger, middle-sized instances, later.

As we will see later, the instance `Workers`, the only instance which cannot be solved by the basic algorithm, is reduced to a very small problem by our reduction procedure as 92.3% of the variables are pegged at either 0 or 1. The optimal solution of the reduced instance is immediately (within only 0.02 seconds) found by Gurobi Optimizer.

Randomly generated instances

Although the instances in Table 3.4 are benchmark instances of CPP, one may think that these are relatively easy. In addition to this, since the aim of our study is on the problem size reduction, we should address larger, middle-sized, instances of CPP. To this aim, we deal with randomly generated instances.

However, truly randomly generated instances have no structure for the items to be clustered appropriately, and the results for these instances do not offer an insight for real world applications of our preprocessing algorithm. Therefore, we propose to use instances shown in Table 3.5 which imitate the ones in Table 3.4 obtained from real world.

We observed that a set of equivalence relations appear in the benchmark instances of Régnier's problem shown in Table 3.4 are somewhat correlated. In order to confirm this in a mathematical manner, we define a *correlatedness* as follows. We first measure the *cosign similarity* of two equivalence relations for each pair of the given equivalence relations. The cosign similarity takes a value 1 if the two equivalence relations are the same, and takes 0 if they are totally different. Next, we calculate the geometric mean of these cosine similarities. One could say that this criteria captures how the given equivalence relations are correlated.

As a matter of fact, the instances in Table 3.4 have high correlatedness as we see from Table 3.3. Here, as the instance `Cetacea` has missing values, we omit its correlatedness. From Table 3.4, one could observe that correlatedness captures an easiness of the instances.

In Table 3.6, we show our results for the instances shown in Table 3.5. In Table 3.5, each instance `R`p`.`n`.`q stands for an instance with n vertices, q equivalence relations, and parameter p for controlling the correlatedness. For the details of the parameter p and the procedure for generating these correlated instances, see [83].

From Table 3.6, we observe that the basic algorithm works very well for these randomly generated but slightly correlated instances. Especially, for the `R7` type instances, the basic algorithm still efficiently and effectively

Table 3.5: The randomly generated instances of Régnier's problem.

Name	# Vertices	# Relations	Correlatedness
R6.200.5	200	5	0.70
R6.200.10		10	0.66
R6.200.15		15	0.65
R6.500.5	500	5	0.69
R6.500.10		10	0.67
R6.500.15		15	0.65
R7.500.5	500	5	0.75
R7.500.10		10	0.72
R7.500.15		15	0.72
R7.1000.5	1000	5	0.74
R7.1000.10		10	0.72
R7.1000.15		15	0.71

Table 3.6: Results of the basic algorithm for the instances in Table 3.5.

Instance		Results of Basic Algorithm		
Name	# Vertices	# Redundant Variables (%)	Comput. Time (s)	Status
R6.200.5	200	–	4.49	Solved
R6.200.10		93.6	10.96	Reduced
R6.200.15		99.2	5.41	Reduced
R6.500.5	500	99.3	1503.63	Reduced
R6.500.10		99.9	611.13	Reduced
R6.500.15		98.6	618.54	Reduced
R7.500.5	500	–	311.42	Solved
R7.500.10		0.1	205.91	Solved
R7.500.15		89.0	137.16	Solved
R7.1000.5	1000	0.0	$>$ 1 hour	Unchanged
R7.1000.10		99.6	2437.27	Reduced
R7.1000.15		99.9	975.93	Solved

reduces the problem size, even for the instances with $n \geq 500$, and succeeds to solve several instances to optimality. On the other hand, we observe that the basic algorithm fails to solve (or even reduce) R7.1000.5 within one hour and there still remains about 2% relative duality gap.

However, one could say that results in Table 3.6 show a potential of

Table 3.7: Results of the reduction procedure for the instances in Table 3.3 and Table 3.5.

Instance		Results of Reduction Procedure		
Name	# Vertices [Original]	# Vertices [Reduced]	Reduction Ratio (%)	Comput. Time (s)
`Workers`	34	7	79.4	0.09
`R6.200.10`	200	31	84.5	0.99
`R6.200.15`		21	89.5	1.02
`R6.500.5`	500	32	93.6	5.84
`R6.500.10`		21	95.8	9.79
`R6.500.15`		45	91.0	12.56
`R7.1000.10`	1000	28	97.2	40.77

the basic algorithm for efficiently solving or effectively reducing middle-sized instances of CPP. It should be noted that the standard formulation (P_{CPP}) has five hundred million constraints when $n = 1000$. As we will see in the next subsection, the instance `R7.1000.10` can be rewritten into a substantially smaller instance by the reduction procedure.

3.3.2 On reduction procedure

In this subsection, for the instances in Table 3.4 and Table 3.6, we will show how much our reduction procedure reduces the problem size based on the information (P_0, P_1) obtained from the basic algorithm.

In Table 3.7, the results of our reduction procedure are summarized, where we omit the results for the instances with status "Solved" or "Unchanged" by the basic algorithm as (P_0, P_1) is empty for these instances.

In order to evaluate the performance of the reduction procedure, we employ the following three criteria.

- # Vertices [Reduced]: the number of vertices in the reduced graph $(G' = (V', E'), c')$, that is, $n' = |V'|$.
- Reduction Ratio (%): the ratio of the reduction on the number of vertices in percentage, that is, $100(n - n')/n$, where n (resp. n') is the number of vertices in the original graph G (resp. the reduced graph G'). If this value is high, the corresponding instance is reduced to a substantially smaller one.
- Comput. Time (s): the computation time required for constructing the reduced instance $(G' = (V', E'), c')$ and P'_0 from the original instance

$(G = (V, E), c)$ using the information (P_0, P_1) obtained from the basic algorithm.

We observe that the size of the instances is reduced often considerably by our reduction procedure. Although `R7.1000.15` is already solved to optimality by the basic algorithm, we confirmed that the basic algorithm implicitly transformed `R7.1000.15` into a much smaller instance, one with only 14 vertices, which may contributes to the efficient computation in the basic algorithm.

As we will confirm in the next subsection, our preprocessing algorithm enables us to solve middle-sized instances such as `R6.500.5`, `R6.500.10`, `R6.500.15`, and `R7.1000.10` to optimality, because they are rewritten into significantly smaller instances by our reduction procedure, which are easy solved by powerful softwares including Gurobi Optimizer. In the the next subsection, we compare our approach with the one directly applying Gurobi Optimizer, in terms of their computation time.

3.3.3 Comparison with optimization software

Finally, we verify the validity of our approach in terms of its computation time, through a comparison with an approach that directly apply an optimization software, Gurobi Optimizer. More specifically, for each instance we tested, we show the sum of the computation time for our preprocessing algorithm and that for solving the reduced instance by Gurobi Optimizer to optimality, and compare it by that for directly applying Gurobi Optimizer to the instance.

The results are summarized in Table 3.8. The values in the three columns are as follows.

- Ours: the sum of the computation time required for our preprocessing algorithm, and that for solving the reduced instance by Gurobi Optimizer to optimality. In the column "Solving Time (s)", we omit its value if the corresponding instance is already solved to optimality by the basic algorithm.

- Naive: the computation time required for a naive approach that directly solving the instance (without our preprocessing algorithm) by Gurobi Optimizer, where "OM" means "over memory" due to numerous constraints.

- Reduction Ratio (%): the reduction in its computation time (in percentage) by using our approach, that is, $100(a - b)/a$ where a denotes the computation time for directly solving the instances by Gurobi Optimizer, and b denotes that for our approach.

Table 3.8: Comparison of our approach with a naive one.

	Ours			Naive	
Instance	Prepro. Time (s)	Solving Time (s)	Total Time (s)	Total Time (s)	Reduction Ratio (%)
Wildcats	0.02	–	0.02	0.34	94.1
Cars	0.07	–	0.07	0.52	86.5
Workers	0.21	0.02	0.23	0.66	65.2
Cetacea	0.01	–	0.01	0.61	98.4
Micro	0.17	–	0.17	0.94	81.9
Soybean	0.03	–	0.03	1.58	98.1
UNO	0.01	–	0.01	2.5	99.6
UNO1b	0.12	–	0.12	69.13	99.8
UNO2b	0.63	–	0.63	81.22	99.2
UNO1a	0.18	–	0.18	114.07	99.8
UNO2a	0.6	–	0.6	114.27	99.5
R6.200.5	4.49	–	4.49	139.3	96.8
R6.200.10	11.95	0.72	12.67	137.83	90.8
R6.200.15	6.43	0.17	6.6	137.96	95.2
R6.500.5	1509.47	0.83	1510.3	OM	–
R6.500.10	620.92	0.18	621.1	OM	–
R6.500.15	631.19	2.71	633.9	OM	–
R7.500.5	311.42	–	311.42	OM	–
R7.500.10	205.91	–	205.91	OM	–
R7.500.15	137.16	–	137.16	OM	–
R7.1000.10	2478.03	0.52	2478.55	OM	–
R7.1000.15	975.93	–	975.93	OM	–
Average					93.2

Here, we omit the results of the approach using only Gurobi Optimizer for the instances with $n \geq 500$ as numerous constraints run out of the computational resources we employed. We observe that there is a clear advantage of our approach. More formally, for the instances solved by Gurobi directly, the improvement, that is, the reduction ratio, is about 93.2% on average.

3.3.4 Outlook

The techniques employed in our algorithm, the pegging test and the Lagrangian relaxation, are known as classical but effective tools for integer programming problems [67, 32, 34]. In this chapter, taking advantage of the underlying structure of CPP, transitivity, we made an improvement on

the ordinary pegging test, which was experimentally confirmed to peg more variables than the naive one.

Advantages

Our preliminary numerical experiments showed that the preprocessing algorithm works well for middle-sized instances of Régnier's problem, which is one of the most important subclass of CPP. When the instances are not randomly generated ones, that is, the equivalence relations are correlated like the instances from real world in Table 3.4, our basic algorithm has a potential for efficiently solving them with up to $n = 1000$ items.

Furthermore, even if the basic algorithm could not solve an instance to optimality, we could obtain an equivalent but substantially smaller instance by our reduction procedure, when several redundant variables are found, that is, P_0 or P_1 is non-empty. One could say that our algorithm has considerably broaden the application area of CPP.

Although we omit the detailed results here, we confirmed that our preprocessing algorithm works well for the benchmark instances proposed in Amorim et al. [6]. More specifically, we confirmed that our algorithm solves the instances with up to $n = 1000$ items to optimality within only 300 seconds, by pegging about 75% of the decision variables [83].

Drawbacks and future works

However, on the other hand, it should be noted that there are several hard instances for our preprocessing algorithm. These instances are from the group technology problem [18, 50, 53, 58, 62, 72, 78, 87]. As demonstrated in Oosten et al. [72], several benchmark instances of the group technology problem cannot be solved to optimality if we only consider the transitivity constraints.

As a matter of fact, although our preprocessing algorithm finds optimal solutions of the half of the benchmark instances of the group technology problem efficiently, we could not peg the decision variables at all for the other half of the instances. See Sukegawa et al. [83] for the details. In order to make our algorithm stronger for these instances, we need to introduce several knowledge from the polyhedral study [39, 72]. This is one of the future works of this study.

Chapter 4

Reformulation

In the previous chapter, we experimentally confirmed that a large part of the transitivity constraints are unneeded, that is, redundant in our preprocessing. This chapter discusses a methodology for detecting and removing a redundancy in the transitivity constraints, in a theoretical manner. Recall that *redundant constraints* refer to constraints such that removing them does not change the optimal solution set, in this study.

Contents

More specifically, in this chapter, we give a simple sufficient condition for the redundant transitivity constraints in the IP formulation (P_{CPP}) of CPP. By just checking our condition, the number of the transitivity constraints handled in algorithms can easily be reduced, in advance. Therefore, our result could be regarded as a reformulation.

Organization

This chapter is organized as follows. First, we mention a recent work by Dinh and Thai [30], which is the motivation of our study. Then, we give a sufficient condition for the redundant transitivity constraints, and discuss its relationship to the results in Dinh and Thai [30]. In order to simply see the impact of our result, we conducted preliminary numerical experiments on the benchmark instances of Régnier's problem.

Also, in order to understand the limitation of our approach for redundant constraints, we also address the linear ordering problem (LOP) whose IP formulation has a structure very similar to that of CPP. To this end, we briefly explain LOP and its standard IP formulation, and give several sufficient conditions for the redundant transitivity constraints in LOP.

4.1 Motivation

Our results are motivated by a recent work by Dinh and Thai [30]. Dinh and Thai [30] deals with an optimization problem called the modularity maximization problem (MMP), which provides one of the most promising approaches for the community detection [70] in networks such as the *social network*.

The input of MMP is an undirected (not necessarily complete) graph $G = (V, E)$ with $|V| = n$ vertices and $|E| = m$ edges, where V simulates a set of persons and E simulates the relationship such as "friend" in practice. The aim of MMP is to find a grouping, that is, communities based on the relationship expressed by E. MMP is known to be formulated as CPP by setting

$$c_{ij} = A_{ij} - \frac{d_i d_j}{2m} \quad (i, j \text{ with } i \neq j) \tag{4.1.1}$$

where A_{ij} is a constant term which takes 1 if $\{i, j\} \in E$ and takes 0 otherwise, and d_i is the degree of vertex i in G, that is, the number of edges adjacent to vertex i in this case. For the details, see [30, 64] for instance.

By the above definition (4.1.1), we observe that $\{i, j\} \notin E$ implies $c_{ij} < 0$ as $A_{ij} = 0$ and d_i, d_j, and m are positive. If $d_i = 0$ holds for some vertex i, we could remove it from the model without loss of generality. On the other hand, even if $\{i, j\} \in E$, c_{ij} can be negative when d_i and d_j are sufficiently large.

Recall that the standard IP formulation of CPP is as follows.

(P_{CPP}) :

$$\begin{array}{lll}
\text{maximize} & \displaystyle\sum_{(i,j)\in V^2_<} c_{ij}x_{ij} & \\
\text{subject to} & x_{ij} \in \{0,1\} & ((i,j) \in V^2_<), \quad \text{(binary)}, \\
& x_{ij} + x_{jk} - x_{ik} \leq 1 & ((i,j,k) \in V^3_<) \quad \text{(type } U\text{)}, \\
& x_{ij} - x_{jk} + x_{ik} \leq 1 & ((i,j,k) \in V^3_<) \quad \text{(type } V\text{)}, \\
& -x_{ij} + x_{jk} + x_{ik} \leq 1 & ((i,j,k) \in V^3_<) \quad \text{(type } W\text{)}.
\end{array}$$

For the instances of MMP, using the properties associated with (4.1.1) in the objective function of the standard IP formulation (P_{CPP}) of CPP, Dinh and Thai [30] proved that

$$\begin{array}{lcll}
\{i,j\},\{j,k\} \notin E & \Rightarrow & x_{ij} + x_{jk} - x_{ik} \leq 1 \text{ is redundant} & \text{(type } U\text{)} \\
\{i,j\},\{i,k\} \notin E & \Rightarrow & x_{ij} - x_{jk} + x_{ik} \leq 1 \text{ is redundant} & \text{(type } V\text{)} \\
\{j,k\},\{i,k\} \notin E & \Rightarrow & -x_{ij} + x_{jk} + x_{ik} \leq 1 \text{ is redundant} & \text{(type } W\text{)}
\end{array}$$

and showed that their sufficient condition above satisfied by a large number of transitivity constraints in the instances of MMP, especially for the instances arising from the so-called *scale-free network*. See Dinh and Thai [30] for the details.

Moreover, interestingly, they experimentally confirmed that removing these redundant constraints shortens computation time as well as saves computational resources required.

Their analysis seems to utilize the structure of the given undirected graph $G = (V, E)$ as well as the property of MMP. However, once the instances of MMP are formulated as an instance of CPP, the information on $G = (V, E)$ and MMP has been lost partially.

Therefore, it would be natural to question whether we could extend their result for general instances of CPP, that is, we could obtain a similar result by just depending on, for instance, the sign pattern of the edge weight function $(c_{ij})_{(i,j)\in V^2_<}$. This is the motivation of our study.

4.2 Redundant Constraints in CPP

In this section, we give a sufficient condition for the transitivity constraints in the standard IP formulation (P_{CPP}) of CPP, for any objective function. As we will see later, this result can be seen as a generalization and an improvement of the result in Dinh and Thai [30].

4.2.1 Result

In this study, we theoretically show the existence of the redundant constraints in the standard IP formulation (P_{CPP}) of CPP, as well as its linear programming relaxation.

Integer Programming

We first prove that

$$\begin{array}{lcll} c_{ij}, c_{jk} < 0 & \Rightarrow & x_{ij} + x_{jk} - x_{ik} \leq 1 \text{ is redundant} & (\text{type } U) \\ c_{ij}, c_{ik} < 0 & \Rightarrow & x_{ij} - x_{jk} + x_{ik} \leq 1 \text{ is redundant} & (\text{type } V) \\ c_{jk}, c_{ik} < 0 & \Rightarrow & -x_{ij} + x_{jk} + x_{ik} \leq 1 \text{ is redundant} & (\text{type } W) \end{array}$$

holds for the standard IP formulation (P_{CPP}) of CPP. Recall that $\{i, j\} \notin E$ implies $c_{ij} < 0$ in MMP, and also that c_{ij} can be negative even if $\{i, j\} \notin E$. Therefore, our result overrides the result in Dinh and Thai [30].

Roughly speaking, our sufficient condition can detect many redundant constraints when there are many negative edge weights, in other words, there are many clusters in the optimal solution.

By this generalization, we could make the idea of Dinh and Thai [30] applicable for other important subclasses of CPP such as Régnier's problem, and the special variants of MMP including the bipartite modularity maximization problem [9]. Moreover, as we only focus on the sign pattern of the edge weight function $(c_{ij})_{(i,j)\in V_<^2}$ in the objective function, the proof can further be simplified.

In order to prove our result, we introduce an IP problem (RP_{CPP}) obtained from (P_{CPP}) by deleting the transitivity constraints satisfying the above our sufficient conditions. More formally, (RP_{CPP}) is as follows.

(RP_{CPP}) :

$$
\begin{aligned}
\text{maximize} \quad & \sum_{(i,j)\in V_<^2} c_{ij}x_{ij} & & \\
\text{subject to} \quad & x_{ij} \in \{0,1\} & & ((i,j)\in V_<^2), \\
& x_{ij} + x_{jk} - x_{ik} \leq 1 & & ((i,j,k)\in V_<^3,\ c_{ij}\geq 0 \vee c_{ij}\geq 0), \\
& x_{ij} - x_{jk} + x_{ik} \leq 1 & & ((i,j,k)\in V_<^3,\ c_{ij}\geq 0 \vee c_{ik}\geq 0), \\
& -x_{ij} + x_{jk} + x_{ik} \leq 1 & & ((i,j,k)\in V_<^3,\ c_{jk}\geq 0 \vee c_{ik}\geq 0).
\end{aligned}
$$

Recall that *redundant constraints* refer to constraints such that removing them does not change the optimal solution set. Therefore, to prove our result formally, it suffices to show that (P_{CPP}) and (RP_{CPP}) have the same set of optimal solutions.

Theorem 2. *(P_{CPP}) and (RP_{CPP}) have the same set of optimal solutions.*

Proof. The basic idea is as follows. Recall that basically, we want to set as many decision variables x_{ij} with $c_{ij} \geq 0$ to 1 as possible, in (P_{CPP}). Therefore, in any optimal solution $\boldsymbol{x}^*$ of (RP_{CPP}), we observe that there are many decision variables x_{ij}^* such that $x_{ij}^* = 1$ and $c_{ij} \geq 0$.

Then, for each distinct pair of vertices, we could find a specific undirected "path" consisting of these decision variables, which plays an important role in our proof. More specifically, along with this path, we could show that the transitivity constraints not appearing in (RP_{CPP}) are automatically satisfied by the constraints appearing in (RP_{CPP}). See Appendix A.1 for the complete proof. □

Linear Programming Relaxation

The simplest way to utilize our result in practice is to use $(\overline{RP}_{\mathrm{CPP}})$ in place of (P_{CPP}) in the softwares. However, in almost all the softwares, we need to frequently solve a *relaxation problem* but the original problem. To this end, the *linear programming* (LP) relaxation problem is employed in many softwares.

In general, the redundant constraints in an IP formulation are not necessarily also redundant in its LP relaxation problem. Therefore, it would be natural to ask whether the redundant constraints in Theorem 2 is also redundant in its LP relaxation problem.

In this study, we prove that the redundant constraints we revealed for (P_{CPP}) are also redundant in its LP relaxation problem. Let $(\overline{P}_{\mathrm{CPP}})$ and $(\overline{RP}_{\mathrm{CPP}})$ denote the linear programming relaxations which are derived by replacing the set of constraints $x_{ij} \in \{0,1\}$ by $x_{ij} \in [0,1]$ in (P_{CPP}) and (RP_{CPP}), respectively. Then we have the following.

Theorem 3. *$(\overline{P}_{CPP})$ and $(\overline{RP}_{CPP})$ have the same set of optimal solutions.*

Proof. See Appendix A.2 for the complete proof. □

It should be noted that Dinh and Thai [30] proved a similar result on the LP relaxation problem. As in Theorem 2, Theorem 3 overrides their result on the LP relaxation. Also, it should be noted that this result can be used when developing algorithms based on the LP relaxation for CPP, like the ones in [2, 30, 64].

4.2.2 Preliminary Numerical Experiments

In this section, we report on the preliminary numerical experiments of our reformulation (RP_{CPP}).

In Table 4.1, we show the results of the computation time by (P_{CPP}) and (RP_{CPP}) using Gurobi Optimizer (Gurobi 5.6.0) for the benchmark instances of Régnier's problem used in the previous chapter. Here, we newly add a middle sized instance `Human` from benchmark instances in UCI Irvine Machine Learning Repository [10].

- Solving Time (s): the computation time required for solving the instance through the given formulation.
- # Redundant Constr. (%): the percentage of the redundant transitivity constraints which satisfies our sufficient condition. Namely, (100 - (this value)) % of the transitivity constraints in the original formulation (P_{CPP}) appear in the formulation (RP_{CPP}).
- Reduction Ratio (%): the improvement with respect to its computation time in percentage, that is, $100(a-b)/a$ where a denotes the computation time for (P_{CPP}) and b denotes that for (RP_{CPP}).

We observe that our sufficient condition could reveal about 30% of the constraints to be redundant on average, which also contributes the improvement on the computation time, about 30% on average. Also, We observe that the computation time of (RP_{CPP}) is always less than those by (P_{CPP}).

Table 4.1: Comparison of our reformulation (RP_{CPP}) with the original formulation (P_{CPP}).

	Original	Ours		
Name	Solving Time (s)	# Redundant Constr. (%)	Solving Time (s)	Reduction Ratio (%)
`Wildcats`	0.34	17.6	0.31	8.82
`Cars`	0.52	10.2	0.47	9.62
`Workers`	0.66	17.1	0.58	12.12
`Cetacea`	0.61	55.2	0.31	49.18
`Micro`	0.94	25.1	0.73	22.34
`Soybean`	1.58	35.9	1.00	36.71
`UNO`	2.50	38.6	1.59	36.40
`Human`	4618.19	59.9	1242.36	73.09
`UNO1b`	69.13	30.7	46.33	32.98
`UNO2b`	81.22	12.3	71.98	11.38
`UNO1a`	114.07	39.9	62.87	44.88
`UNO2a`	114.27	20.3	90.24	21.03
Average		30.2		29.88

Especially for the instances `Cetacea` and `Human`, more than 50% of the constraints are revealed to be redundant, which contributes the reduction in computation time. For instance, for `Human`, 59.9% of the transitivity constraints are removed by our reformulation, and the improvement on the computation time by the optimization software, Gurobi Optimizer, is about 73.09%. Here, the instance `Human` is a newly added instance which can be obtained from UCI Machine Learning Repository [10].

4.2.3 Outlook

The analysis used in our proof is motivated by that of Dinh and Thai [30] for MMP, which is a special case of CPP. Although the essential approach is the same, we could simplify their proof to obtain a (slightly) better sufficient condition for the redundant constraints in CPP. Namely, our contribution is on the point that we made the approach in Dinh and Thai [30] applicable for more general cases.

Advantage

We note that the strong point of our approach is its simplicity. We could reduce the number of constraints as well as the computation time by a simple

Table 4.2: Comparison of our reformulation ($\overline{RP}_{\mathrm{CPP}}$) with the original formulation ($\overline{P}_{\mathrm{CPP}}$) for the benchmark instances of the modularity maximization problem (MMP) reported in Dinh and Thai [30].

		Original	Ours		
ID	# Vertices	Solving Time (s)	# Redundant Constr. (%)	Solving Time (s)	Reduction Ratio (%)
1	34	0.21	91.9	0.02	90.4
2	62	3.85	94.9	0.11	97.1
3	77	13.43	97.1	0.08	99.4
4	105	60.4	94.6	1.76	97.1
5	115	106.27	91.1	13.98	86.8
6	332	OM	98.7	197.03	–
7	512	OM	99.5	53.18	–
8	1589	OM	99.9	2.94	–
Average			96.0		94.2

operation in advance. Compared to the implementation of the sophisticated algorithms, using our result is much simpler, and hence would be more attractive especially for the users of CPP model in practice.

Discussion

For a comparison, in Table 4.2, we simply show the results reported in Dinh and Thai [30] for the modularity maximization problem (MMP). See Dinh and Thai [30] for the further details on the computational environment. They dealt with 8 well-known benchmark instances of MMP, and showed how many constraints are revealed to be redundant by their condition. We observe that the instances in Table 4.2 have many redundant constraints, compared to the benchmark instances of Régnier's problem we tested.

One could observe that this difference is caused by the number of edges with negative weights because our sufficient condition is satisfied by many transitivity constraints when there are many negative edge weights. For instance, in `UNO2a` in Table 4.1, only 35.5% of the edges have negative weights, which makes it difficult to detect the redundant constraints. On the other hand, in the instance `8` in Table 4.2, surprisingly, about 99.8% of the edges have negative weights.

In sum, our results seem to have an impact for the instances of CPP with a large number of "negative" edge weights, including MMP and the special variants of MMP including the bipartite modularity maximization

c^1 c^2

- Solid lines: edges with non-negative weights
- Dotted lines: edges with negative weights

Figure 4.1: Images of c^1 and c^2.

problem [9]. As a matter of fact, for the instances shown in Table 4.1, the coefficient of correlation between the number of edges with negative weights and the number of redundant constraints detected by our sufficient condition is about 0.96.

Here we simply note that the configuration of the edges with negative weights also affects the number of redundant constraints detected by our condition. For instance, see the two edge weights c^1 and c^2 shown in Figure 4.1. We observe that the number of redundant constraints detected by our condition is 2 for c^1 and 3 for c^2 respectively,

Finally, for the instances with a large number of positive edge weights, we propose to apply the preprocessing algorithm in advance. Then, one would expect to obtain a reduced graph with many negative edge weights as the edges with positive edge weights may be pegged at 1 and deleted by the shrinking in our reduction procedure. As a matter of fact, we could confirm that a large number of the edges in the instances of reduced size in Table 3.7 have negative weights.

Future works

We think that there are two future directions. The first direction is to strengthen our results, that is, to derive a larger class of redundant constraints. For instance, we believe that

$$\begin{aligned} c_{ij} + c_{jk} < 0 \quad &\Rightarrow \quad x_{ij} + x_{jk} - x_{ik} \leq 1 \text{ is redundant} \quad (\text{type } U) \\ c_{ij} + c_{ik} < 0 \quad &\Rightarrow \quad x_{ij} - x_{jk} + x_{ik} \leq 1 \text{ is redundant} \quad (\text{type } V) \\ c_{jk} + c_{ik} < 0 \quad &\Rightarrow \quad -x_{ij} + x_{jk} + x_{ik} \leq 1 \text{ is redundant} \quad (\text{type } W) \end{aligned}$$

are true as we could not find a counterexample in our preliminary numerical experiments.

The second future direction is to apply our analysis to IP formulations of other optimization problems. There are a large number of problems which are formulated by using transitivity constraints. Then, is it possible to derive a class of redundant constraints in such formulations by similar analysis? In order to answer this question partially, in what follows, we address the linear ordering problem (LOP).

4.3 Linear Ordering Problem (LOP)

In this section, in order to understand the limitation of our approach for redundant constraints, we also address the linear ordering problem (LOP) whose standard IP formulation has a very similar structure as that of the standard IP formulation (P_{CPP}) of CPP.

As a matter of fact, we will observe that the standard IP formulation of LOP also has the transitivity constraints like CPP. In what follows, we first briefly explain the problem and its IP formulation. After that, we get into the explanation of our sufficient conditions for the redundant transitivity constraints in LOP.

4.3.1 Problem Description

We define LOP as an optimization problem on a graph. We denote a simple directed complete graph G with vertex set V and arc set A by $G = (V, A)$. Namely, $A = \{(i, j) : i, j \in V,\ i \neq j\}$.

A spanning subgraph $G[T] = (V, T)$ of G induced by a set $T \subseteq A$ of arcs is called *tournament* if either $(i, j) \in T$ or $(j, i) \in T$ holds, but not both, for each distinct pair of vertices $u, v \in V$. In addition, if there is no directed cycle, that is, an ordered sequence of vertices $v_1, v_2, \ldots, v_l$ satisfying

$$(v_1, v_2), (v_2, v_3), \ldots, (v_{l-1}, v_l), (v_l, v_1) \in T,$$

then we call $G[T]$ *acyclic tournament*.

We observe that every acyclic tournament $G[T]$ uniquely determines a *linear ordering* of V, that is, a bijection $\pi : V \to \{1, 2, \ldots, n\}$ satisfying

$$(i, j) \in T \Leftrightarrow \pi(i) < \pi(j).$$

For this, we say that i is *before* j (denoted by $i \succ j$) in an acyclic tournament $G[T]$ if $(i, j) \in T$. In the linear ordering problem, we are given an arc weight function $w : N \to \mathbb{R}$ and seek an acyclic tournament that maximizes the total weight

$$\sum_{(i,j)\in T} w_{ij}.$$

One could say that CPP and LOP are similar in the sense that both of them can be regarded as an optimization problem on a *transitive relation*.

CPP corresponds to an equivalence relation, while LOP corresponds to a linear order.

4.3.2 Formulations

Next, we introduce the conventional integer programming (IP) formulation employed in existing algorithms for LOP in the literature [20, 36, 63].

If we let a 0-1 decision variable x_{ij} take 1 if $(i, j) \in T$ and take 0 otherwise, the LOP is formulated as the following simple integer (linear) programming problem:

(P_{LOP}) :

$$
\begin{array}{lllll}
\text{maximize} & \displaystyle\sum_{(i,j)\in A} w_{ij}x_{ij} & & & \\
\text{subject to} & x_{ij} \in \{0,1\} & ((i,j) \in A) & \text{(binary)}, \\
& x_{ij} + x_{ji} = 1 & ((i,j) \in A) & \text{(anti-symmetry)}, \\
& x_{ij} + x_{jk} + x_{ki} \leq 2 & ((i,j,k) \in V^3_{\neq}) & \text{(transitivity)}
\end{array}
$$

where

$$V^3_{\neq} := \{(i, j, k) \mid i, j, k \in V,\ i \neq j, j \neq k, k \neq i\}.$$

In order to see the correctness of the above formulation, letting

$$T_{\boldsymbol{x}} = \{(i, j) : x_{ij} = 1\}$$

denote a set of arcs corresponding to a given feasible solution $\boldsymbol{x} = (x_{ij})_{(i,j)\in A}$ of the above formulation, we will check that $G[T_{\boldsymbol{x}}]$ is an acyclic tournament.

The anti-symmetry constraints ensure that $G[T_{\boldsymbol{x}}]$ is a tournament. Also, we observe that the transitivity constraints rule out directed cycles of length 3 in $G[T_{\boldsymbol{x}}]$. In other words, they say that if i is before j and j is before k, then i must be before k in the ordering. For this, these constraints are referred to as the *transitivity constraints*. Combining with the equality constraints, the transitivity constraints can also rule out directed cycles of length k with $k \geq 4$. Hence, $G[T_{\boldsymbol{x}}]$ is an acyclic tournament.

We observe that the above conventional IP formulation (P_{LOP}) for LOP has a similar structure as that of CPP, that is, (P_{CPP}), and suffers from numerous constraints as well as decision variables. For instance, if we consider to order 300 items, then the formulation has about 44 thousand decision variables and 9 million constraints, which are large but slightly less than those of CPP.

Also, one could consider to halve the number of the decision variables using the anti-symmetry constraints, however, for notational ease in our analysis, we omit this rewriting.

Existing Studies

Compared to CPP, LOP has many theoretically attractive properties, especially, on the associated polytope, which is referred to as the *linear ordering polytope* [11, 26, 37, 55, 71], and defined as the convex hull of the feasible region of (P_{LOP}). These theoretical results are utilized in [36, 63] to develop exact methods.

LOP is also known to be intractable, in theory, even when the input instances satisfy a special structure [4], while it has a large number of applications such as ordering problem of ancestry items [35], ranking method for sports team [74, 84], analysis of Input-Output matrices [23] originated by Leontief [56], and a voting system known as the *Kemeny's method* [49], a "relaxed" approach for Arrow's impossibility theorem [8]. For the detailed history as well as the conjectures and open problems of LOP, see survey papers [21, 22, 77].

4.3.3 Example: Kemeny's method

The linear ordering problem is frequently used as a mathematical framework for determining an optimal ordering of a set of items based on their pairwise comparisons. One of the well-known applications of this problem would be a voting system called the *Kemeny method* [49].

In this method, we use V to simulate a set of candidates for some election. Now, suppose that there are several voters who have rankings, that is, linear orders on V. Then, we could calculate the number w_{ij} of voters who agree with that i should be ranked higher than j. In the Kemeny's method [49], we aim at finding a linear order $\pi : V \to \{1, 2, \ldots, n\}$ that maximizes the sum of the agreements to $(w_{ij})_{(i,j)\in V^2_<}$, that is,

$$\sum_{\pi(i)<\pi(j)} w_{ij}.$$

We observe that this can be seen as an instance of LOP.

It is known that Kemeny's method [49] satisfies the Condorcet criterion, which is one the desirable criteria for voting systems. Also, one could observe that Kemeny's method solves a problem of aggregation of linear orders, which can be seen as a special variant of Régnier's problem.

4.4 Redundant Constraints in LOP

In this section, we give sufficient conditions for the redundant constraints in the standard IP formulation (P_{LOP}) of LOP. The proofs for these sufficient conditions are motivated by our proof for Theorem 2.

4.4.1 Result

Given arc weights w, we focus on a set A^+ of arcs corresponding to "win" or "tie". More formally, $A^+ = \{(u, v) : w_{uv} \geq w_{vu}\}$. Using this definition, let us classify the set of transitivity constraints $x_{uv} + x_{vw} + x_{wu} \leq 2$ into 4 classes depending on an integer value

$$l_{uvw} = |\{(u, v), (v, w), (w, u)\} \cap A^+|,$$

which ranges from 0 to 3 and satisfies $l_{uvw} = l_{vwu} = l_{wuv}$.

One could say that l_{uvw} captures the degree of "bothersome" when we aim at finding an optimal solution. For instance, we observe that a triplet with $l_{uvw} = 3$ has a relationship like "rock-paper-scissors", that is, u is superior to v ($w_{uv} \in A^+$), v is superior to w ($w_{vw} \in A^+$) but w is superior to u ($w_{wu} \in A^+$).

Our first result states that the transitivity constraints with $l_{uvw} = 0$ are redundant. For convenience, let us introduce a relaxation problem $(RP^{[q]}_{\mathrm{LOP}})$ obtained from the original formulation (P_{LOP}) by dropping all the transitivity constraints with $l_{ijk} \leq q$, that is,

$(RP^{[q]}_{\mathrm{LOP}})$:

$$\begin{array}{lll} \text{maximize} & \displaystyle\sum_{(i,j)\in A} w_{ij}x_{ij} & \\ \text{subject to} & x_{ij} \in \{0, 1\} & ((i, j) \in A), \\ & x_{ij} + x_{ji} = 1 & ((i, j) \in A), \\ & x_{ij} + x_{jk} + x_{ki} \leq 2 & ((i, j, k) \in V^3_{\neq},\ l_{ijk} > q). \end{array}$$

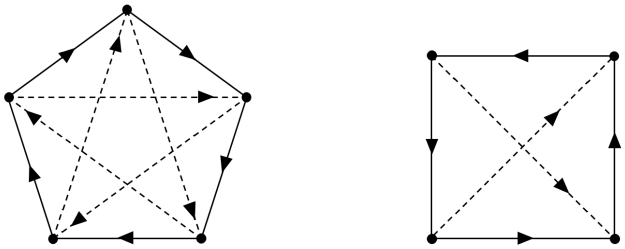
Then, we have the following.

Theorem 4. *(P_{LOP}) and $(RP^{[0]}_{LOP})$ have the same set of optimal solutions.*

Proof. The essential idea is the same as that of Theorem 2. Recall that basically, we want to set as many decision variables x_{ij} with $(i, j) \in A^+$ to 1 as possible, in (P_{LOP}). Therefore, in any optimal solution $\boldsymbol{x}^*$ of (RP_{LOP}), we could observe that there are many decision variables x^*_{ij} such that $x^*_{ij} = 1$ and $(i, j) \in A^+$.

Then, as in the proof of Theorem 2, for each pair of vertices, we could find a specific directed "path" consisting of these decision variables, which plays an important role in our proof. The proof for the existence of such a path is motivated by a lemma used in the proof of Theorem 2. See Appendix A.3 for the complete proof. □

If we make some assumption on the structure of (V, A^+) then we have the following.



$G_1 : T_1 \subseteq A_1$ $(d = 5)$ $\qquad$ $G_2 : T_2 \subseteq A_2$ $(d = 4)$

- Solid lines: A^+ with sufficiently large weights, say 100
- Dotted lines: $A \setminus A^+$ with weights 0
- The other arcs which do not appear in the figures have weights 1

Figure 4.2: Figures of T_1 and T_2.

Theorem 5. *Let d be the length of a longest directed cycle in (V, A^+). When $d = 4$, $(RP^{[1]}_{LOP})$ and (P_{LOP}) have the same set of optimal solutions. When $d = 3$, the same thing holds for $(RP^{[2]}_{LOP})$ and (P_{LOP}). Finally, if there is no directed cycle in (V, A^+), the same thing holds for $(RP^{[3]}_{LOP})$ and (P_{LOP}).*

Proof. See Appendix A.3 for the complete proof. □

As we see from Theorem 4 and Theorem 5, we could show that there is also a redundancy in the transitivity constraints in (P_{LOP}) like (P_{CPP}). However, compared to the case for (P_{CPP}), Theorem 4 is not attractive as the number of the transitivity constraints satisfying $l_{ijk} = 0$ is not that big in general. Moreover, to show Theorem 5, we need to make several assumption on the input graphs.

Therefore, it would be natural to ask whether there is a better sufficient condition for the redundant constraints. However, we observe that the above sufficient conditions are tight in a sense, as follows.

Remark 1. *There is an instance $G_1 = (V_1, A_1)$ such that $d = 5$ and an optimal solution T_1 of $(RP^{[1]}_{LOP})$ is NOT feasible for the original problem (P_{LOP}). This means that the formulation $(RP^{[1]}_{LOP})$ does NOT necessarily gives us an optimal and feasible solution of the original problem (P_{LOP}), in general, when $d \geq 5$. In a similar manner, there is an instance $G_2 = (V_2, A_2)$ such that $d = 4$ and an optimal solution T_2 of $(RP^{[2]}_{LOP})$ is NOT feasible for the original problem (P_{LOP}). Figures of T_1 and T_2 are shown in Figure 4.2.*

Table 4.3: Comparison of our reformulation ($RP^{[0]}_{\mathrm{LOP}}$) setting $q = 0$ with the original formulation (P_{LOP}).

		Original	Ours ($RP^{[0]}_{\mathrm{LOP}}$)		
Name	# Vertices	Solving Time (s)	# Redundant Constr. (%)	Solving Time (s)	Reduction Ratio (%)
be75eec	50	0.66	5.0	0.31	52.4
be75np	50	1.81	4.8	1.55	14.6
be75oi	50	0.35	4.5	0.34	4.1
be75tot	50	0.35	5.5	0.32	7.5
stabu70	60	1.40	6.3	1.41	-1.4
stabu74	60	0.92	5.9	0.83	9.6
stabu75	60	0.84	6.2	0.91	-7.6
t59b11xx	44	0.32	2.4	0.29	6.7
t59d11xx	44	0.20	2.2	0.2	-0.3
t59f11xx	44	0.20	2.3	0.19	1.4
Average			4.5		8.7

4.4.2 Preliminary Numerical Experiments

In this section, we briefly verify how often the sufficient conditions are satisfied in the real-world instances. To this end, we use the benchmark instances arising from an analysis of `Input/Output` matrices which can be obtained in *Linear Ordering library*[1] for instance. In the analysis of the Input/Output matrices, the vertex set V simulates a set of sectors in some country, and each directed arc (i, j) has a weight w_{ij} which expresses the amount of financial flows from sector i to sector j. Then the LOP model is employed to assess its degree of maturation based on these weights.

If the optimal value for the given instance is high, then one could say that the degree of maturation of economics in the country is relatively low, because the financial flows are in one direction. On the other hand, if the optimal value is relatively low, we say that the country is developed.

In Table 4.3 and Table 4.4, we summarize the results. Although we report only on the results for the first 10 instances in the *Linear Ordering library*, similar results are obtained for the remaining instances.

From Table 4.3 and Table 4.4, we could observe how the number of transitivity constraints (%) and the solving time changes as we change the value q of ($RP^{[q]}_{\mathrm{LOP}}$) for $q \in \{0, 1\}$. We confirmed that the length of the

[1] http://www.optsicom.es/lolib/

Table 4.4: Comparison of our reformulation ($RP^{[1]}_{\mathrm{LOP}}$) setting $q = 1$ with the original formulation (P_{LOP}).

		Original	Ours ($RP^{[1]}_{\mathrm{LOP}}$)		
Name	# Vertices	Solving Time (s)	# Redundant Constr. (%)	Solving Time (s)	Reduction Ratio (%)
be75eec	50	0.66	50.0	0.15	77.9
be75np	50	1.81	50.0	0.60	66.8
be75oi	50	0.35	50.0	0.17	52.4
be75tot	50	0.35	50.0	0.15	56.7
stabu70	60	1.40	50.0	0.62	55.7
stabu74	60	0.92	50.0	0.40	56.3
stabu75	60	0.84	50.0	0.37	55.8
t59b11xx	44	0.32	50.0	0.16	48.1
t59d11xx	44	0.20	50.0	0.09	53.3
t59f11xx	44	0.20	50.0	0.09	54.1
Average		–	50.0	–	57.7

longest cycles, that is, d in Theorem 5, of these instances are equal to the number of vertices, that is, n. Hence, sufficient conditions in Theorem 5 are not satisfied for these instances.

From Table 4.3, we observe that only a small fraction of the constraints are removed when $q = 0$. More specifically, on average, there are only about 4.5% transitivity constraints satisfying $l_{ijk} = 0$. Therefore, although the computation time for ($RP^{[0]}_{\mathrm{LOP}}$) is smaller than those for (P_{LOP}) on average, the difference is not that big.

On the other hand, from Table 4.4, we observe that the computation time is substantially reduced if we use the formulation ($RP^{[1]}_{\mathrm{LOP}}$) in place of (P_{LOP}). It is not difficult to see that exactly half, that is, 50% of the transitivity constraints are redundant in the formulation ($RP^{[1]}_{\mathrm{LOP}}$) when there is no tie in the given objective function, that is, $w_{ij} \neq w_{ji}$ for each i and j with $i \neq j$.

Although our sufficient condition in Theorem 5 is not satisfied for all the instances we tested, interestingly, we confirmed that ($RP^{[1]}_{\mathrm{LOP}}$) always gave us an optimal solution of the original problem (P_{LOP}) for all the instances we tested. In sum, in other words, there might be a better sufficient condition. On the other hand, the optimal solutions of ($RP^{[2]}_{\mathrm{LOP}}$) and ($RP^{[3]}_{\mathrm{LOP}}$) are always not feasible.

4.4.3 Outlook

We could show that the approach for revealing the redundant constraints for CPP is partially applicable to LOP. As a matter of fact, in the proofs, we use the same technique, that is, the existence of a specific desired "path" in the optimal solutions.

However, compared to CPP, the results we obtained for LOP are not that attractive. Even worse, although we could show that $(RP^{[0]}_{\mathrm{LOP}})$ can be used in place of the original formulation (P_{LOP}) in a theoretical manner, the computation time by Gurobi Optimizer can not be shortened for many instances we tested.

On the other hand, we could observe that our results are on the limitation in a sense, as demonstrated in Remark 1. However, one could say that the example shown in Figure 4.2, G_1 for Remark 1 is "pathological" as it can be regarded as a chain of triplets with $l_{uvw} = 3$, that is, triplets with a "rock-paper-scissors" relationship.

Therefore, one would expect that we rarely encounter such a pathological structure in the instances from real world. As a matter of fact, we confirmed that instances in Table 4.4 include only a few pathological cycles.

Future work

As a matter of fact, carefully reading our proof in Appendix A.3, one may notice that $(RP^{[1]}_{\mathrm{LOP}})$ and (P_{LOP}) have the same set of optimal solutions if there exists a desired directed "path" (used in our proof) such that it is not a part of any pathological cycle. However, for now, we have no good sufficient condition for ensuring this situation.

We observe that $(RP^{[1]}_{\mathrm{LOP}})$ has just half of the transitivity constraints of (P_{LOP}), and the computation time is also about half of that of (P_{LOP}). Moreover, as already mentioned in our preliminary numerical experiments, the optimal solutions of $(RP^{[1]}_{\mathrm{LOP}})$ are also optimal at the original problem (P_{LOP}) for all instances we tested.

Therefore, we think that it would be an interesting future work to find a better sufficient condition for $(RP^{[1]}_{\mathrm{LOP}})$ to share the same optimal solution set as that of (P_{LOP}). As a promising approach, one can address a limitation on the arc weights. For instance, if we limit $w_{ij} \in \{0, 1\}$ for each $(i, j) \in A$, the corresponding instance can be regarded as an instance of *feedback arc set problem*. One could observe that the "pathological" situation in Figure 4.2 for Remark 1 cannot be happen when $w_{ij} \in \{0, 1\}$ for each $(i, j) \in A$.

Chapter 5

Hardness Results

As already mentioned, CPP is intractable, that is, NP-hard in theory, as its special case, Régnier's problem is known to be NP-hard [86]. As CPP has many applications, for important subclasses of CPP, many researchers tried to understand the border between the tractable (polynomial time solvability) and intractable (NP-hard) cases, from a theoretical viewpoint.

One of the most common approaches is to identify whether the problem is *NP-hard* or not. NP-hardness is a well known criterion for the difficulty of optimization problems. Many researchers believe that there is no efficient (polynomial time) algorithm to solve these NP-hard problems. For this reason, several important special cases of CPP have been studied from this viewpoint in the literature.

Contents

In this chapter, we show that an important subclass of CPP is intractable in theory, that is, NP-hard. This problem is referred to as the *maximum edge clique partitioning problem* (Max-ECPP). Max-ECPP is originally introduced by Dessmark et al. [29] as an useful framework for DNA clone classification, and is discussed from a viewpoint of approximability and inapproximability in the literature [27, 75].

Although Max-ECPP is announced to be NP-hard in general in a recent paper by Punnen and Zhang [75], their proof has an error as demonstrated by Sukegawa and Miyauchi [81]. The aim of this chapter is to give a simple and correct proof for the NP-hardness of Max-ECPP.

5.1 Related Works

As mentioned above, for important subclasses of CPP, many researchers tried to identify its theoretical hardness or easiness, that is, NP-hardness or polynomial time solvability.

For instance, the problem of aggregation of equivalence relations, the Régnier's problem is known to be NP-hard in general as demonstrated by Wakabayashi [86]. It should be noted that the problem is trivial when the number q of the given equivalence relations is at most two. However, the complexity for $q = 3$ cases is an open problem [31]. Also, a special variant of Régnier's problem, Zahn's problem [91], is also known to be NP-hard. For the details, see de Amorim et al. [6] for instance.

The modularity maximization problem, which was the focus of Dinh and Thai [30], is also known to be NP-hard as demonstrated by Brandes et al. [12]. Zhan et al [92] showed that a special variant of the modularity maximization problem, the bipartite modularity maximization problem introduced by Barber [9], is NP-hard. However, there is an error in their proof as pointed out by Costa and Hansen [24, 25]. Recently, Miyauchi and Sukegawa [65] give a correct proof for the NP-hardness of the problem.

5.2 Maximum Edge CPP and Its Hardness

Max-ECPP is defined as follows. In this problem, we are given undirected graph $G = (V, E)$. A partition $\{V_1, V_2, \ldots, V_t\}$ of V is called a *clique partitioning* if V_i is a clique of G for each $i \in \{1, 2, \ldots, t\}$, where t is not fixed. Then Max-ECP is to find a clique partitioning that maximizes the number of edges within the cliques, that is,

$$\sum_{i=1}^{t} |\{\{u, v\} | u, v \in V_i\}|.$$

We observe that Max-ECP is formulated as CPP by setting $c_{uv} = 1$ if $\{u, v\} \in E$ and c_{uv} to a sufficiently large negative value otherwise.

The *clique number* of G is the number of vertices in a maximum clique of G and is denoted by $\omega(G)$. When $\omega(G) = 1$, the problem is trivial as G has no edge. When $\omega(G) = 2$, one could observe that Max-ECP can be reduced to the *maximum cardinality matching problem*, where we aim at finding a maximum cardinality set of edges such that no two edges share a common vertex. The maximum cardinality matching problem is known to be solved in polynomial time.

Recently, Punnen and Zhang [75] mentioned that a decision version of Max-ECP (that is, a problems to decide whether there exists a solution (clique partitioning) which attains some specific value) is NP-complete when $\omega(G) = 3$, which implies the NP-hardness of Max-ECP when $\omega(G) = 3$. For more formal definitions, see Garey and Johnson [33].

The proof in Punnen and Zhang [75] uses the *3-dimensional matching problem* (3DM), which is known to be NP-complete. The idea is to show that Max-ECP can solve 3DM. More specifically, we construct a function f converting any instance I of 3DM into an instance $f(I)$ of Max-ECP, such

that the computation time for calculating $f(I)$ is polynomially-bounded by the size of I, and the answer for I in 3DM is "yes" if and only if the answer for $f(I)$ in the decision version of Max-ECP is "yes".

However, as demonstrated by Sukegawa and Miyauchi [81], the "if" part is incorrect. More specifically, they give a simple counterexample to the "if" part. See Sukegawa and Miyauchi [81] for the details of their counterexample.

As we will see below, the proof can easily be corrected by replacing 3DM with the triangle cover problem (TCP). TCP is also known to be NP-complete even when $\omega(G) = 3$ [33]. In TCP, we are given a simple and undirected graph $G = (V, E)$ with $|V| = 3q$ for some positive integer q. The problem is to determine an existence of a clique partitioning $\{V_1, V_2, \ldots, V_t\}$ of V such that $|V_i| = 3$ (triangle) for each $i \in \{1, 2, \ldots, q\}$.

When $\omega(G) = 3$, for any solution, that is, clique partitioning of Max-ECP, the number of vertices is at most three (triangle) for each clique. Now, suppose that the optimal value of Max-ECP on G attains $3q$. Then, for any optimal solution $\{V_1, V_2, \ldots, V_t\}$ of Max-ECP on G, we have $|V_i| = 3$ for each $i \in \{1, 2, \ldots, t\}$. This implies that TCP has a solution on G. It is not difficult to see the reverse of the above argument also holds. In sum, we have the following.

Theorem 6. *The decision version of Max-ECP is NP-complete even when the input graphs are limited to the cases where $\omega(G) = 3$, which implies the NP-hardness of Max-ECP in general.*

Proof. See Sukegawa and Miyauchi [81] for the details. □

Chapter 6

Conclusion

In this chapter, we summarize this thesis, and mention the outlook.

6.1 Conclusion

In this thesis, we address the clique partitioning problem (CPP) defined by Grötschel and Wakabayashi [38] in 1989. CPP is a fundamental combinatorial optimization problem with a large number applications including flight-gate scheduling [27], microarray analysis [52], group technology problem [72, 87], and community detection [2, 5, 30, 64], known as the modularity maximization [70]. Due to its simplicity, CPP provides a general framework for diverse clustering tasks from real world. Notably, CPP provides one of the most promising and attractive approaches for qualitative data clustering, known as Régnier's problem [14].

In spite of its rich applications, the conventional and standard integer programming (IP) formulation of CPP is known to be very large, which prevents us from efficient computation in practice [52, 87]. However, on the other hand, it is empirically known that optimization problems include "redundant" parts, especially for the problems arising from real world.

In this thesis, we develop a methodology for removing a redundancy from the IP formulation of CPP, which ease the difficulty arising from the massiveness of the formulation. More specifically, we develop a practical preprocessing algorithm and a theoretical characterization for removing a redundancy, both of which lead to a problem size reduction of CPP. Our result contributes to efficient computation through the recent powerful optimization softwares, in practice.

The key of our algorithm and analysis was based on the underlying structure of CPP, transitivity. Therefore, in order to understand the limitation of our approach, we also address the linear ordering problem (LOP), which has a transitivity similar to that of CPP. LOP is also known to suffer from difficulties arising from the massiveness of its IP formulation. We observed

that a part of our approach is also applicable to LOP.

Like our study, some authors tried to utilizes a structure common to CPP and LOP. As a matter of fact, our preprocessing for CPP, was motivated by a recent work by Sukegawa et al. [84], who utilized the transitivity of LOP to construct an efficient preprocessing algorithm. Also, Ailon et al. [3] developed a simple and efficient approximation algorithm for Régnier's problem [76] (a special case of CPP) and Kemeny method [49] (a special case of LOP), based on a structure common to these two problems.

6.2 Outlook

In this study, we focused only on the standard IP formulation (P_{CPP}) discussed in Grötschel and Wakabayashi [38], and considered to improve it. However, one could address another formulation, that is, a reformulation.

For instance, Kochenberger et al. [52, 87] pointed out a limitation of exact approaches in [38, 61, 72] based on the standard IP formulation, and proposed a reformulation using a quadratic term in the objective function as follows:

CP[Node] :

$$\begin{array}{lll} \text{maximize} & \displaystyle\sum_{(i,j)\in V^2_<} w_{ij} \sum_{k\in K} x_{ik}x_{jk} & \\ \text{subject to} & x_{ik} \in \{0,1\} & (i \in V, k \in K), \\ & \displaystyle\sum_{k\in K} x_{ik} = 1 & (i \in V) \end{array}$$

where K is a set of clusters. Namely, the decision variable x_{ik} takes 1 if vertex i is assigned to k-th cluster, and takes 0 otherwise.

In the standard IP formulation proposed in Grötschel and Wakabayashi [38], the decision variables are associated with the set of edges. Therefore, Kochenberger et al. [52, 87] called the formulation (P_{CPP}) discussed in this thesis, *edge-based* formulation, and denoted this by CP[Edge]. In contrast, as they associate the decision variables with the assignment of vertex (node) to the clusters, they call the above formulation *node-based* formulation, denoted by CP[Node].

The formulation in Kochenberger et al. [52, 87] could be applied for relatively large instances as the number of decision variables and constraints are much smaller than those of Grötschel and Wakabayashi [38]. However, the formulation in Kochenberger et al. [52, 87] is not adequate for solving the instances of CPP to optimality, due to the quadratic term in its objective function.

Even worse, we observe that the above formulation in Kochenberger et al. [52, 87] suffers from a *symmetry*, that is, there exist a lot of expressions

for each solution. This is due to the *indistinguishability* of the names of the clusters. As also pointed out by Kaibel et al. [47], such a property is known to be a great disadvantage when using the ordinary softwares. Therefore, one could say that the reformulation in Kochenberger et al. [52, 87] also has a kind of "redundancy". On the other hand, we observe that the standard IP formulation of Grötschel and Wakabayashi [38] does not suffer from such a symmetry.

Recently, interestingly, Matsui et al. [59] demonstrated that nicely modifying the objective function of the standard IP formulation (P_{CPP}) of CPP yields an attractive new formulation for the *vertex coloring problem* (VCP) whose conventional IP formulation suffers from a great symmetry [57]. More specifically, they succeeds to count the number of clusters from the decision variables in (P_{CPP}) via a formula, which can be expressed by a mixed integer linear programming (MILP) model.

As the standard IP formulation of CPP does not suffer from the symmetry, our reformulation [59] for VCP also does not suffer from the symmetry. As a matter of fact, it succeeded to find an optimal solution of a well-known hard instance, `DSJC125.9` from *DIMACS Implementation Challenge*, within only one minute, on an ordinary PC by Gurobi Optimizer. It should be noted that the optimal value of `DSJC125.9` was an open problem until very recently. See [40, 57] for instance.

In addition, we observe that the idea in Matsui et al. [59] enables us to control the number of clusters in the standard IP formulation of CPP, which yields a new formulation for a clustering problem with a cluster number constraint. Such special variants of CPP also have been studied extensively in the literature [45, 46]. Our preliminary numerical experiments confirmed that the computation time of this new formulation is much superior to the conventional and naive one proposed in Johnson et al. [45, 46].

Like the study in Matsui et al [59], there are many studies on reformulation for the vertex coloring problem [15, 16, 17, 51, 60]. In this way, considering how to formulate a given problem as an IP or MILP problem would become a more and more important study topic, with the development of the usability and the performance of MILP softwares.

6.3 Future Work

On the other hand, the "size" of the formulation has been one of the hottest topic in the area of oprimizarion, from a theoretical viewpoint. For instance, there is a study on the formulation size as the *linear programming* (LP). It is well known that there exists an efficient, that is, polynomial-time algorithm for solving LP problems.

Therefore, if we could formulate some problem as an LP problem whose size is polynomially bounded by the original size, then we could conclude

that the original problem can be solved in polynomial-time. In sum, there is an interesting relationship between the size of the formulation and its theoretical computational complexity.

In this study, we tried to reduce the formulation size of the standard IP formulation of CPP, focusing on a redundancy. Like our study, recently, Kaibel and Weltge [48] address the formulation size of the *traveling salesman problem* (TSP) as an integer programming problem. TSP is a well-known example such that its conventional IP formulation becomes huge. The problem is very simple: Given a set of n cities and a distance for each pair of the cities, ind a shortest route that starts from some city and back to the original city.

More specifically, in spite of its simplicity, the size of its standard IP formulation is known to require $O(n^2)$ decision variables and $O(2^n)$ constraints, which causes a shortage of computational resources even for relatively small n. Interestingly, Kaibel and Weltge [48] theoretically demonstrated that any naive IP formulation for TSP "must" require $O(2^n)$ constraints.

However, we observe that their analysis only focus on the system of the constraints, and does not utilizes the information from the objective function. In contrast to their approach, in this study, we could reveal a class of redundant constraints by utilizing the information from the objective function. Therefore, one could expect that there must be redundant constraints in the standard IP formulations for TSP. In light of this, we are now interested in the formulation size of TSP when we take the objective function into account in the analysis.

Appendix A

Proofs

A.1 Proof of Theorem 2

We note that the proof below is largely based on our original paper [66].

Preliminaries

We begin by introducing some notations. Let $\boldsymbol{x}^* = (x^*_{ij})_{(i,j)\in V^2_<}$ be an optimal solution of (RP_{CPP}). Now, the goal is to show that $\boldsymbol{x}^*$ is a feasible solution of the original problem (P_{CPP}).

If this shown, we can conclude that any optimal solution of (RP_{CPP}) is also feasible and optimal at (P_{CPP}). Conversely, any optimal solution of (P_{CPP}) is also optimal at (RP_{CPP}) as we observe that the optimal values of (RP_{CPP}) and (P_{CPP}) coincide in this case. Therefore, in what follows, we show the feasibility of $\boldsymbol{x}^*$ at the original problem (P_{CPP}).

For convenience, define

$$E_+ = \{\{i,j\} \in E \mid c_{ij} \geq 0\} \text{ and } E^* = \{\{i,j\} \in E \mid x^*_{ij} = 1\}.$$

Furthermore, let

$$\{(V_1, E^*_1), (V_2, E^*_2), \ldots, (V_p, E^*_p)\}$$

denote the set of the connected components of (V, E^*). In order to show the feasibility of $\boldsymbol{x}^*$ at (P_{CPP}), it suffices to show that each component is clique, that is,

$$E^*_r = \{\{i,j\} \mid i,j \in V_r, i \neq j\} \quad (\forall r \in \{1,2,\ldots,p\}).$$

We now present the following lemma.

Lemma 1. *$(V_l, E^*_l \cap E_+)$ is connected for each $l \in \{1,2,\ldots,p\}$. Namely, for each distinct pair $\{u,v\}$ of vertices of V_l, there is a path on $E^*_l \cap E_+$ from u to v.*

Proof. It suffices to show that for any partition $\{S, T\}$ of V_l, there exists an edge in $E_l^* \cap E_+$ whose one endpoint is in S and the other is in T. From the definition of V_l, there exists at least one edge in E_l^* between S and T. Let $\emptyset \neq E_{ST} \subseteq E_l^*$ denote the set of edges between S and T.

Now, suppose that all these edges $e \in E_{ST}$ are not in E_+, that is, the edge weights $w_e < 0$ for all $e \in E_{ST}$. Let us focus on a solution $\boldsymbol{x}' = (x'_{ij})_{(i,j)\in V_<^2}$ of (RP_{CPP}) obtained from $\boldsymbol{x}^*$ by changing the values of variables corresponding to $e \in E_{ST}$ from 1 to 0. Namely,

$$x'_{ij} = \begin{cases} 0 & (e \in E_{ST}) \\ x^*_{ij} & (e \notin E_{ST}) \end{cases} \quad (\forall e = \{i, j\} \in E \text{ with } i < j).$$

This operation can be seen as a partitioning V_l into S and T. Therefore $\boldsymbol{x}'$ is feasible for (RP_{CPP}) as there is no restriction of the number of components in the solutions of CPP.

Suppose that there is a transitivity constraint, say $x_{ij} + x_{jk} - x_{ik} \leq 1$, violated by $\boldsymbol{x}'$, but not violated by $\boldsymbol{x}^*$. Then, $x'_{ij} = x'_{jk} = 1$ and $x'_{ik} = 0$. however, this cannot happen because $x'_{ik} = 0$ together with $i, j, k \in V_l$ implies $x'_{ij} = 0$ or $x'_{jk} = 0$ by the definition of $\{S, T\}$.

On the other hand, we see that the objective value of $\boldsymbol{x}'$ is strictly greater than that of $\boldsymbol{x}^*$ as we have $w_e < 0$ for all $e \in E_{ST}$. This contradicts the optimality of $\boldsymbol{x}^*$ at (RP_{CPP}). □

Proof of Theorem 2

Then, let us get into the main part of our proof. As already mentioned, it suffices to show that the optimal solution $\boldsymbol{x}^*$ of (RP_{CPP}) is also feasible at (P_{CPP}), which is equivalent to the statement that (V_l, E_l^*) is complete, that is clique, for each $l \in \{1, 2, \ldots, p\}$.

From Lemma 1, for any distinct pair of vertices $i, j \in V_l$, there exists a path $i = u_0, u_1, \ldots, u_q = j$ on $E_l^* \cap E_+$. Using the definition of E_+, we observe that (RP_{CPP}) can be rewritten as

(RP_{CPP}) :

$$\begin{array}{lll}
\text{maximize} & \displaystyle\sum_{(i,j)\in V_<^2} c_{ij} x_{ij} & \\
\text{subject to} & x_{ij} \in \{0, 1\} & ((i, j) \in V_<^2), \\
& x_{ij} + x_{jk} - x_{ik} \leq 1 & ((i, j, k) \in V_<^3,\ \{i, j\} \in E_+ \vee \{j, k\} \in E_+), \\
& x_{ij} - x_{jk} + x_{ik} \leq 1 & ((i, j, k) \in V_<^3,\ \{i, j\} \in E_+ \vee \{i, k\} \in E_+), \\
& -x_{ij} + x_{jk} + x_{ik} \leq 1 & ((i, j, k) \in V_<^3,\ \{j, k\} \in E_+ \vee \{i, k\} \in E_+).
\end{array}$$

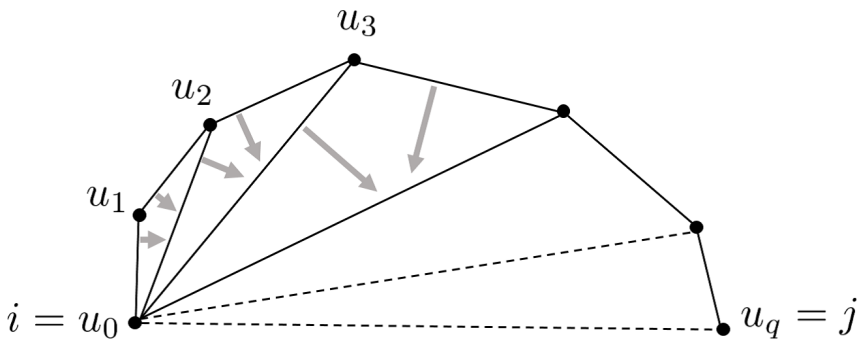


Figure A.1: An image of the procedure used in the proof of Theorem 2

Therefore, since $\{u_0, u_1\} \in E_+$, the transitivity constraint

$$x_{u_0u_1} + x_{u_1u_2} - x_{u_0u_2} \leq 1$$

is contained in (RP_{CPP}). (Note that in this notation it is necessary that $u_0 < u_1 < u_2$ holds, that is, $(u_0, u_1, u_2) \in V^3_<$. If it is not the case, one should swap the order of the indices of the variables appropriately. In what follows, we frequently use this argument.) Thus, by substituting

$$x^*_{u_0u_1} = x^*_{u_1u_2} = 1$$

to the above constraint $x_{u_0u_1} + x_{u_1u_2} - x_{u_0u_2} \leq 1$, we have $x^*_{u_0u_2} = 1$. In a similar manner, since $\{u_2, u_3\} \in E_+$, the transitivity constraint

$$x_{u_0u_2} + x_{u_2u_3} - x_{u_0u_3} \leq 1$$

is also contained in (RP_{CPP}). Thus, by substituting

$$x^*_{u_0u_2} = x^*_{u_2u_3} = 1$$

to the above constraint $x_{u_0u_2} + x_{u_2u_3} - x_{u_0u_3} \leq 1$, we have $x^*_{u_0u_3} = 1$. Repeatedly applying this operation, we derive $x^*_{u_0u_q} = x^*_{ij} = 1$. The above procedure is illustrated in Figure A.1. This implies that (V_l, E^*_l) is complete, that is, clique, which means that $\boldsymbol{x}^*$ is feasible at (P_{CPP}).

A.2 Proof of Theorem 3

We note that the proof below is largely based on our original paper [66].

Preliminaries

We begin by introducing some tools and notations. Let $\boldsymbol{x}^* = (x^*_{ij})_{1 \leq i < j \leq n}$ be an arbitrary optimal solution of $(\overline{RP}_{\mathrm{CPP}})$. Like the proof of Theorem 2, the goal is to show the feasibility of $\boldsymbol{x}^*$ at $(\overline{P}_{\mathrm{CPP}})$.

For simplicity, we focus on the residual graph of $\boldsymbol{x}^*$, denoted by $\boldsymbol{d}^*$. Namely,

$$\boldsymbol{d}^* = (d^*_{ij})_{(i,j) \in V^2_<} = \mathbf{1} - \boldsymbol{x}^*.$$

Then, we see that the transitivity constraints for $\boldsymbol{x}^*$ in $(\overline{P}_{\mathrm{CPP}})$ correspond to the triangle inequalities for $\boldsymbol{d}^*$. For instance,

$$x^*_{ij} + x^*_{jk} - x^*_{ik} \leq 1 \;\Leftrightarrow\; d^*_{ik} \leq d^*_{ij} + d^*_{jk}$$

holds for the first set of transitivity constraints. Here, we set

$$\overline{E}^* = \{\{i,j\} \in E \mid d^*_{ij} < 1 \;(\Leftrightarrow x^*_{ij} > 0)\}.$$

Also, let

$$\{(V_1, \overline{E}^*_1), (V_2, \overline{E}^*_2), \ldots, (V_p, \overline{E}^*_p)\}$$

denote the set of the connected components of $(V, \overline{E}^*)$. Like Lemma 1, we have the following fact. The proof is omitted as it is essentially the same as that of Lemma 1.

Lemma 2. *$(V_l, \overline{E}^*_l \cap E_+)$ is connected for each $l \in \{1, 2, \ldots, p\}$.*

Proof of Theorem 3

Then, let us get into the main part of our proof. It suffices to show that any optimal solution $\boldsymbol{x}^* = (x^*_{ij})_{(i,j) \in V^2_<}$ of $(\overline{RP}_{\mathrm{CPP}})$ is feasible for $(\overline{P}_{\mathrm{CPP}})$. As mentioned above, it is equivalent to the statement that the residual graph $\boldsymbol{d}^*$ satisfies all the triangle inequalities corresponding to the transitivity constraints in $(\overline{P}_{\mathrm{CPP}})$.

Then, it suffices to confirm that the triangle inequalities for all triples of vertices in each connected component of $(V, \overline{E}^*)$ are satisfied. The reason is that the other inequalities are always satisfied because there are at least two terms equal to 1 in each inequality. Here we used the fact that we have $d^*_{ij} = 1$ for any $i, j \in V$ in different connected components by the definition of $(V, \overline{E}^*)$.

From Lemma 2, for any distinct vertices $i, j \in V_l$, there exists at least one path on $\overline{E}^*_l \cap E_+$. The shortest one of these paths and its length are denoted by $i = u_0, u_1, \ldots, u_q = j$ and d'_{ij}, respectively.

Now, repeatedly applying the triangle inequalities, corresponding to the transitivity constraints in $(\overline{RP}_{\mathrm{CPP}})$, to $\boldsymbol{d}^*$ as

$$\begin{aligned} d'_{ij} &= d^*_{u_0u_1} + d^*_{u_1u_2} + \cdots + d^*_{u_{q-1}u_q} \\ &\geq d^*_{u_0u_2} + d^*_{u_2u_3} + \cdots + d^*_{u_{q-1}u_q} \\ &\geq \cdots \geq d^*_{u_0u_{q-1}} + d^*_{u_{q-1}u_q} \geq d^*_{u_0u_q} = d^*_{ij}, \end{aligned}$$

we have $d'_{ij} \geq d^*_{ij}$. Here, let $d_{ij} = \min\{d'_{ij}, 1\}$ and construct $\boldsymbol{d} = (d_{ij})_{(i,j)\in V^2_<}$ by gathering d_{ij} for all $i, j \in V_l$. Clearly, $d_{ij} \geq d^*_{ij}$ as $d^*_{ij} \leq 1$ by its definition. Furthermore, we see that $\boldsymbol{d}$ satisfies the triangle inequalities because

$$\begin{aligned} d_{ij} + d_{jk} &= \min\{d'_{ij}, 1\} + \min\{d'_{jk}, 1\} \\ &\geq \min\{d'_{ij} + d'_{jk}, 1\} \\ &\geq \min\{d'_{ik}, 1\} = d_{ik}. \end{aligned}$$

The first inequality follows from the non-negativity of $\boldsymbol{d}' = (d'_{ij})_{(i,j)\in V^2_<}$. The second inequality follows as $\boldsymbol{d}'$ satisfies the triangle inequalities.

As a matter of fact, $d_{ij} = d^*_{ij}$ for all $i, j \in V_l$. If this is shown, then we see that $\boldsymbol{d}^*$ satisfies the triangle inequalities in each component, namely, the proof is completed.

Suppose that there exists a pair of vertices $i, j \in V_l$ such that $d_{ij} \neq d^*_{ij}$. Recalling that $d_{ij} \geq d^*_{ij}$, we have $d_{ij} > d^*_{ij}$. Then, since $d^*_{ij} < d_{ij} \leq 1$, we obtain $\{i, j\} \in \overline{E}^*_l$.

Now, suppose that $\{i, j\}$ is also contained in E_+. Then, $\{i, j\}$ is one of the paths between i and j on $\overline{E}^*_l \cap E_+$. Since d'_{ij} is the length of the shortest path, we have

$$d_{ij} \leq d'_{ij} \leq d^*_{ij}.$$

Putting this condition together with $d_{ij} \geq d^*_{ij}$, we obtain $d_{ij} = d^*_{ij}$. Thus, under the first assumption $d_{ij} \neq d^*_{ij}$, we have $\{i, j\} \notin E_+$. Now, let us focus on a solution

$$\boldsymbol{x}' = \begin{cases} 1 - d_{ij} & \text{if } i, j \in V_l, \\ x^*_{ij} & \text{otherwise,} \end{cases}$$

of $(\overline{RP}_{\mathrm{CPP}})$. This solution is feasible for $(\overline{RP}_{\mathrm{CPP}})$ because, as observed above, $\boldsymbol{d}$ satisfies the triangle inequalities.

By a simple calculation, we can confirm that the objective value of $\boldsymbol{x}'$ is strictly greater than that of $\boldsymbol{x}^*$. This contradicts the optimality of $\boldsymbol{x}^*$ at $(\overline{RP}_{\mathrm{CPP}})$.

A.3 Proof of Theorem 4 and 5

We note that the proof below is largely based on our original but unpublished paper [82].

Preliminaries

The following simple lemma plays an important role to show the results. We note that the following can be seen as an analogy of Lemma 1 for CPP.

Lemma 3. *Let $\boldsymbol{x}$ be an optimal solution for $(RP^{[q]}_{LOP})$ for some $q \in \{0,1,2,3\}$ and $T_{\boldsymbol{x}} = \{(u,v) \mid x_{uv} = 1\}$ be a set of arcs corresponding to $\boldsymbol{x}$. If there is a directed cycle in $(V, T_{\boldsymbol{x}})$, then there also exists one in $(V, T_{\boldsymbol{x}} \cap A^+)$.*

Proof. Let s, t be a distinct pair of vertices such that there is a directed path from s to t in $(V, T_{\boldsymbol{x}})$. Here, we show that there also exists a directed path from s to t in $(V, T_{\boldsymbol{x}} \cap A^+)$. It is easy to see that this observation completes the proof.

Suppose, to the contrary, that there is no directed path from s to t in $(V, T_{\boldsymbol{x}} \cap A^+)$. Let S be a set of vertices which are reachable from s by a directed path in $(V, T_{\boldsymbol{x}} \cap A^+)$. Furthermore let $T = V \setminus S$ and

$$F = \{(u,v) \mid u \in S, v \in T,\ x_{uv} = 1\}.$$

Therefore, every arc $(u,v) \in F$ must be an element of $A \setminus A^+$. Now, we focus on a solution $\boldsymbol{x}'$ obtained from $\boldsymbol{x}$ by flipping the arcs in F. Namely,

$$x'_{uv} = \begin{cases} 0 & ((u,v) \in F), \\ 1 & ((v,u) \in F), \\ x_{uv} & (\text{otherwise}) \end{cases}$$

for each $(u,v) \in A$. It is not difficult to see that this operation never violate the transitivity constraints which are satisfied by $\boldsymbol{x}$. Hence, $\boldsymbol{x}'$ is a feasible solution of $(RP^{[q]}_{\mathrm{LOP}})$.

On the other hand, as mentioned above, we have $(u,v) \in A \setminus A^+$, that is, $w_{uv} < w_{vu}$ for each $(u,v) \in F$. This means that $\boldsymbol{x}'$ attains a strictly better objective value that of $\boldsymbol{x}$. This contradicts to the optimality of $\boldsymbol{x}$ at $(RP^{[q]}_{\mathrm{LOP}})$. □

Proof of Theorem 4

Let $\bar{\boldsymbol{x}}$ be an optimal solution of $(RP^{[0]}_{\mathrm{LOP}})$ and $T_{\bar{\boldsymbol{x}}}$ be a set of arcs corresponding to $\bar{\boldsymbol{x}}$. Now, the goal is to show that there is no directed cycle in $(V, T_{\bar{\boldsymbol{x}}})$, which implies the feasibility of $\bar{\boldsymbol{x}}$ at the original problem (P_{LOP}).

Suppose, to the contrary, that there is a directed cycle in $(V, T_{\bar{\boldsymbol{x}}})$. Then, by Lemma 3, there is a directed cycle $v_1, v_2, \ldots, v_k$ in $(V, T_{\bar{\boldsymbol{x}}} \cap A^+)$. Since $(v_1, v_2), (v_2, v_3) \in A^+$, we have $l_{v_1 v_2 v_3} \geq 2 > 0$, which means that $(RP^{[0]}_{\mathrm{LOP}})$ has the transitivity constraints

$$x_{v_1 v_2} + x_{v_2 v_3} + x_{v_3 v_1} \leq 2.$$

Substituting $\bar{x}_{v_1v_2} = \bar{x}_{v_2v_3} = 1$ to the above constraint, we have $\bar{x}_{v_3v_1} = 0$, which implies that

$$\bar{x}_{v_1v_3} = 1$$

by the anti-symmetry constraint $x_{v_1v_3} + x_{v_3v_1} = 1$. Next, let us focus on the transitivity constraint

$$x_{v_1v_3} + x_{v_3v_4} + x_{v_4v_1} \leq 2.$$

Since at least $(v_3, v_4) \in A^+$, we have $l_{v_1v_3v_4} \geq 1 > 0$, which means that $(RP^{[0]}_{\mathrm{LOP}})$ has this transitivity constraints. Again, substituting $\bar{x}_{v_1v_3} = \bar{x}_{v_3v_4} = 1$ to the above constraint, we have $\bar{x}_{v_1v_4} = 1$.

Applying similar arguments along this directed cycle, we finally have $\bar{x}_{v_1v_k} = 1$, which contradicts to that $v_1, v_2, \ldots, v_k$ is a directed cycle in $(V, T_{\boldsymbol{x}})$. One could observe that the above procedure is similar to the one used in our proof of Theorem 2 illustrated in Figure A.1.

Proof of Theorem 5

Since the last statement in Theorem 5 is clear, we only give proofs of the first two statements. As in the proof of Theorem 4, letting $\bar{\boldsymbol{x}}$ be an optimal solution of $(RP^{[1]}_{\mathrm{LOP}})$ or $(RP^{[2]}_{\mathrm{LOP}})$ and

$$T_{\bar{\boldsymbol{x}}} = \{(u, v) \mid \bar{x}_{uv} = 1\}$$

be a set of arcs corresponding to this solution, we show that there is no directed cycle in $(V, T_{\boldsymbol{x}})$ under the assumptions $d = 4$ or $d = 3$. Suppose, to the contrary, that there is a directed cycle in $(V, T_{\bar{\boldsymbol{x}}})$. Then, again, by Lemma 3, there is a directed cycle $v_1, v_2, \ldots, v_k$ in $(V, T_{\bar{\boldsymbol{x}}} \cap A^+)$.

Let us consider the case when $d = 4$. In this case, the length k of the directed cycle is 3 or 4. Let $\bar{\boldsymbol{x}}$ be an optimal solution of $(RP^{[1]}_{\mathrm{LOP}})$. When $k = 3$, since $l_{v_1v_2v_3} = 3 > 1$, $(RP^{[1]}_{\mathrm{LOP}})$ has the transitivity constraint

$$x_{v_1v_2} + x_{v_2v_3} + x_{v_3v_1} \leq 2$$

ruling out a directed cycle v_1, v_2, v_3 in $(V, T_{\bar{\boldsymbol{x}}})$, which yields a contradiction. When $k = 4$, since $l_{v_1v_2v_3} \geq 2 > 1$ and $l_{v_1v_3v_4} \geq 2 > 1$, $(RP^{[1]})$ has both of the transitivity constraints

$$x_{v_1v_2} + x_{v_2v_3} + x_{v_3v_1} \leq 2 \text{ and } x_{v_1v_3} + x_{v_3v_4} + x_{v_4v_1} \leq 2.$$

Substituting $\bar{x}_{v_1v_2} = \bar{x}_{v_2v_3} = 1$ to the former constraint, we have $\bar{x}_{v_3v_1} = 0$. Then, now, in order to meet the anti-symmetry constraint $x_{v_1v_3} + x_{v_3v_1} = 1$, we must have $\bar{x}_{v_1v_3} = 1$. This implies that v_1, v_3, v_4 is a directed cycle in $(V, T_{\bar{\boldsymbol{x}}})$. However this is a contradiction as the directed cycle v_1, v_3, v_4 cannot occur in $(V, T_{\bar{\boldsymbol{x}}})$ due to the transitivity constraint

$$x_{v_1v_3} + x_{v_3v_4} + x_{v_4v_1} \leq 2.$$

Next, let us consider the case when $d = 3$. Then, the length k of the directed cycle is 3. Let $\bar{\boldsymbol{x}}$ be an optimal solution of $(RP^{[2]}_{\mathrm{LOP}})$. As $l_{v_1v_2v_3} = 3 > 2$, $(RP^{[2]}_{\mathrm{LOP}})$ has the transitivity constraint

$$x_{v_1v_2} + x_{v_2v_3} + x_{v_3v_1} \leq 2,$$

which means that v_1, v_2, v_3 cannot be a directed cycle in $(V, T_{\bar{\boldsymbol{x}}})$. This is a contradiction.

Bibliography

[1] T. Achterberg, R. E. Bixby, Z. Gu, E. Rothberg, and D. Weninger: Multi-row presolve reductions in mixed integer programming, *Proceedings of the Twenty-Sixth RAMP Symposium* (2014) 181–196.

[2] G. Agarwal and D. Kempe: Modularity-maximizing graph communities via mathematical programming, *European Physical Journal B* **66** (2008) 409–418.

[3] N. Ailon, M. Charikar, and A. Newman: Aggregating inconsistent information: ranking and clustering, *Journal of the ACM* **55** (2008) 23.

[4] N. Alon: Ranking tournaments, *SIAM Journal on Discrete Mathematics* **20** (2006) 137–142.

[5] D. Aloise, S. Cafieri, G. Caporossi, P. Hansen L. Liberti, and S. Perron: Column generation algorithms for exact modularity maximization in networks, *Physical Review E* **82** (2010) 046112.

[6] S. G. de Amorim, J-P. Barthélemy, and C. C. Riberio: Clustering and clique partitioning: simulated annealing and tabu search approaches, *Journal of Classification* **9** (1992) 17–41.

[7] E. D. Andersen and K. D. Andersen: Presolving in linear programming, *Mathematical Programming* **71** (1995) 221–245.

[8] K. J. Arrow: *Social Choice and Individual Values* (New York: Wiley, 1963).

[9] M. J. Barber: Modularity and community detection in bipartite networks, *Physical Review E* **76** (2007) 066102.

[10] C. L. Blake and C. J. Merz: UCI Irvine Machine Learning Repository [http://archive.ics.uci.edu/ml/] (last accessed on Apr. 17, 2014), University of California, Irvine, School of Information and Computer Sciences (1998).

[11] G. Bolotashvili, M. Kovalev, and E. Girlich: New facets of the linear ordering polytope, *SIAM Journal of Discrete Mathematics* **12** (1999) 326–336.

[12] U. Brandes, D. Delling, M. Gaertler, R. Gorke, M. Hoefer, Z. Nikoloski, and D. Wagner: On modularity clustering, *Knowledge and Data Engineering, IEEE Transactions on* **20** (2008) 72–188.

[13] A. L. Brearley, G. Mitra, and H. P. Williams: Analysis of mathematical programming problems prior to applying the simplex algorithm, *Mathematical Programming* **8** (1975) 54–83.

[14] M. J. Brusco and H. F. Köhn: Clustering qualitative data based on binary equivalence relations: neighborfood search heuristics for the clique partitioning problem, *Psychometrika* **74** (2009) 685–703.

[15] E. K. Burke, J. Mareček, A. J. Parkes, and H. Rudová: A supernodal formulation of vertex colouring with applications in course timetabling, *Annals of Operations Research* **179** (2010) 105–130.

[16] M. Campêlo, V. A. Campos, and R. C. Corrêa: On the asymmetric representatives formulation for the vertex coloring problem, *Discrete Applied Mathematics* **156** (2008) 1097–1111.

[17] M. Campêlo, R. C. Corrêa, and Y. Frota: Cliques, holes and the vertex coloring polytope, *Information Processing Letters* **89** (2004) 159–164.

[18] H. M. Chan and D. A. Milner: Direct clustering algorithm for group formulation in cellular manufacturing, *Journal of Manufacturing Systems* **1** (1982) 65–74.

[19] I. Charon and O. Hudry: Noising methods for a clique partitioning problem, *Discrete Applied Mathematics* **154** (2006) 754–769.

[20] I. Charon and O. Hudry: A branch-and-bound algorithm to solve the linear ordering problem for weighted tournaments, *Discrete Applied Mathematics* **154** (2006) 2097–2116.

[21] I. Charon and O. Hudry: A survey on the linear ordering problem for weighted or unweighted tournaments, *A Quarterly Journal of Operations Research* **5** (2007) 5–60.

[22] I. Charon and O. Hudry: An updated survey on the linear ordering problem for weighted or unweighted tournaments, *Annals of Operations Research* **175** (2010) 107–158.

[23] H. B. Chenery and T. Watanabe: International comparisons of the structure of production, *Econometrica* **26** (1956) 487–521.

[24] A. Costa and P. Hansen: Comment on "Evolutionary method for finding communities in bipartite networks," *Physical Review E* **84** (2011) 058101.

[25] A. Costa and P. Hansen: A locally optimal hierarchical divisive heuristic for bipartite modularity maximization, *Optimization Letters* **8** (2014) 903–917.

[26] J. P. Doignon, S. Fiorini, and G. Joret: Facets of the linear ordering polytope: A unification for the fence family through weighted graphs, *Journal of Mathematical Psychology* **50** (2006) 251–262.

[27] U. Dorndorf, F. Jaehn, and E. Pesch: Modeling robust flight-gate scheduling as a clique partitioning problem, *Transportation Science* **42** (2008) 292–301.

[28] U. Dorndorf and E. Pesch: Fast clustering algorithms, *ORSA Journal on Computing* **6** (1994) 141–153.

[29] A. Dessmark, J. Jansson, A. Lingas, E. M. Lundell, and M. Persson: On the approximability of maximum and minimum edge clique partition problems, *International Journal of Foundations of Computer Science* **18** (2007) 217–226.

[30] T. N. Dinh and M. T. Thai: Towards optimal community detection: From trees to general weighted networks, *Internet Mathematics* (accepted pending revision).

[31] V. Filkov and S. Skiena: Integrating microarray data by consensus clustering, *International Journal on Artificial Intelligence Tools* **13** (2004) 863–880.

[32] M. L. Fisher: The Lagrangian relaxation method for solving integer programming problems, *Management Science* **27** (1981) 1–18.

[33] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness* (W. H. Freeman & Co., 1979).

[34] A. M. Geoffrion: Lagrangian relaxation for integer programming, *Mathematical Programming Study* **2** (1974) 82–114.

[35] F. Glover, T. Klastorin, and D. Kongman: Optimal weighted ancestry relationships, *Management Science* **20** (1974) 1190–1193.

[36] M. Grötschel, J. Jünger, and G. Reinelt: A cutting plane algorithm for the linear ordering problem, *Operations Research* **32** (1984) 1195–1220.

[37] M. Grötschel, J. Jünger, and G. Reinelt: Facets of the linear ordering polytope, *Mathematical Programming* **33** (1985) 43–60.

[38] M. Grötschel and Y. Wakabayashi: A cutting plane algorithm for a clustering problem, *Mathematical Programming* **45** (1989) 59–96.

[39] M. Grötschel and Y. Wakabayashi: Facets of the clique partitioning polytope, *Mathematical Programming* **47** (1990) 367–387.

[40] S. Gualandi and F. Malucelli: Exact solution of graph coloring problems via constraint programming and column generation, *INFORMS Journal on Computing* **24** (2012) 81–100.

[41] A. Guenoche: Consensus of partitions: a constructive approach, *Advances in data analysis and classification* **5** (2011) 215–229.

[42] M. Guignard and K. Spielberg: Logical reduction methods in zero-one programming – minimal preferred variables, *Operations Research* **29** 49–74.

[43] P. Hansen and B. Jaumard: Cluster analysis and mathematical programming, *Mathematical Programming* **79** (1997) 191–215.

[44] F. Jaehn and E. Pesch: New bounds and constraint propagation techniques for the clique partitioning problem, *Discrete Applied Mathematics* **161** (2013) 2025–2037.

[45] X. Ji and J. E. Mitchel: Branch-and-price-and-cut on the clique partitioning problem with minimum clique size requirement, *Discrete Optimization* **4** (2007) 87–102.

[46] E. L. Johnson, A. Mehrotra, and G. L. Nemhauser: Min-cut clustering, *Mathematical Programming* **62** (1993) 133–151.

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

[48] V. Kaibel and S. Weltge: Lower bounds on the sizes of integer programs without additional variables, *Integer Programming and Combinatorial Optimization*, Lecture Notes in Computer Science **8494** (2014) 321–332.

[49] J. G. Kemeny: Mathematics without numbers, *Daedalus* **4** (1959) 577–591.

[50] K. R. King: Machine component group formation in group technology, *Omega* **8** (1980) 193–199.

[51] G. A. Kochenberger, F. Glover, B. Alidaee, and C. Rego: An unconstrained quadratic binary programming approach to the vertex coloring problem, *Annals of Operations Research* **139** (2005) 229–241.

[52] G. Kochenberger, F. Glover, B. Alidaee, and H. Wang: Clustering of microarray data via clique partitioning, *Journal of Combinatorial Optimization* **10** (2005) 77–92.

[53] R. K. Kumar, A. Kusiak, and A. Vannelli: Grouping of parts and components in flexible manufacturing systems, *European Journal of Operational Research* **24** (1986) 387–397.

[54] T. Larsson, M. Patriksson, and A-B. Strömberg: Conditional subgradient optimization – Theory and applications, *European Journal of Operational Research* **88** (1996) 382–403.

[55] J. Leunga and J. Lee: More facets from fences for linear ordering and acyclic subgraph polytopes, *Discrete Applied Mathematics* **50** (1994) 185–200.

[56] W. Leontief: *Structure of American Economy 1919-1929* (Harverd University Press, Cambridge, 1941).

[57] E. Malaguti and P. Toth: A survey on vertex coloring problems, *International Transactions in Operational Research* **17** (2010) 1–34.

[58] B. Malakooti and Z. Yang: Multiple criteria approach and generation of efficient alternatives for machine-part family formulation in group technology, *IIE Transactions* **34** (2002) 837–846.

[59] T. Matsui, N. Sukegawa, and A. Miyauchi: Fractional programming formulation for the vertex coloring problem, *Information Processing Letters* **114** (2014) 706–709.

[60] A. Mehrotra and M. A. Trick, A column generation approach for graph coloring, *INFORMS Journal on Computing* **8** (1996) 344–354.

[61] A. Mehrotra and M. A. Trick: Cliques and clustering: A combinatorial approach, *Operations Research Letters* **22** (1998) 1–12.

[62] J. Miltenburg and W. Zhang: A comparative evaluation of nine well-known algorithms for solving the cell formation in group technology, *Journal of Operations Management* **10** (1991) 44–72.

[63] J. E. Mitchell and B. Borchers: Solving real-world linear ordering problem using a primal-dual interior point cutting plane method, *Annals of Operations Research* **62** (1996) 253–276.

[64] A. Miyauchi and Y. Miyamoto: Computing an upper bound of modularity, *European Physical Journal B* **86** (2013) 302.

[65] A. Miyauchi and N. Sukegawa: Maximizing Barber's bipartite modularity is also hard, *Optimization Letters*, accepted.

[66] A. Miyauchi and N. Sukegawa: Redundant constraints in the standard formulation for the clique partitioning problem, *Optimization Letters*, online available; DOI: 10.1007/s11590-014-0754-6.

[67] R. M. Nauss: An efficient algorithm for the 0-1 knapsack problem, *Management Science* **23** (1976) 27–31.

[68] R. M. Nauss: An improved algorithm for the capacitated facility location problem, *Journal of the Operational Research Society* **29** (1978) 1195–1201.

[69] L. G. Nemhauser and L. A. Wolsey: *Integer and combinatorial optimization* (New York: Wiley, 1988).

[70] M. E. Newman and M. Girvan: Finding and evaluating community structure in networks, *Physical review E* **69** 026113.

[71] A. Newman and S. Vempala: Fences are futile: on relaxations for the linear ordering problem, *Integer Programming and Combinatorial Optimization*, Lecture Notes in Computer Science **2081** (2001) 333–347.

[72] M. Oosten, J. H. G. C. Rutten, and F. C. R. Spieksma: The clique partitioning problem: Facets and patching facets, *Networks* **38** (2001) 209–226.

[73] G. Palubeckis: A branch-and-bound approach using polyhedral results for a clustering problem, *INFORMS Journal on Computing* **9** (1997) 30–42.

[74] K. Pedings, A. N. Langville, and Y. Yamamoto: A minimum violations ranking method, *Optimization and Engineering* **13** (2012) 349–370.

[75] A. P. Punnen and R. Zhang: Analysis of an approximate greedy algorithm for the maximum edge clique partitioning problem, *Discrete Optimization* **9** (2012) 205–208.

[76] S. Régnier: Sur quelques aspects mathematiques des problemes de classification automatique, *I.C.C. Bulletin* **4** (1965) 175–191.

[77] G. Reinelt: *The Linear Ordering Problem: Algorithms and Applications* (Helderman: Berlin, 1985).

[78] D. F. Rogers and S. S. Kulkarni: Optimal bivariate clustering and a genetic algorithm with an application in cellular manufacturing, *European Journal of Operational Research* **160** (2005) 423–444.

[79] N. Samphaiboon and T. Yamada: Heuristic and exact algorithms for the precedence-constrained knapsack problem, *Journal of Optimization Theory and Applications* **105** (2000) 659–676.

[80] D. Snyers: Clique partitioning problem and genetic algorithms, *Artificial Neural Nets and Genetic Algorithms*, Proceedings of the International Conference in Innsbruck, Austria (1993) 352–360.

[81] N. Sukegawa and A. Miyauchi: A note on the complexity of the maximum edge clique partitioning problem with respect to clique number, *Discrete Optimization* **10** (2013) 331–332.

[82] N. Sukegawa and S. Mizuno: Redundancy of transitivity constraints in the linear ordering problem, *Technical Reports in Industrial Engineering and Management of Tokyo Institute of Technology* (2014) 2014-2.

[83] N. Sukegawa, Y. Yamamoto, and L. Zhang: Lagrangian relaxation and pegging test for the clique partitioning problem, *Advances in Data Analysis and Classification* **7** (2013) 363–391.

[84] N. Sukegawa, Y. Yamamoto, and L. Zhang: Lagrangian relaxation and pegging test for linear ordering problems, *Journal of the Operations Research Society of Japan* **54** (2011) 142–160.

[85] S. Umetani and M. Yagiura: Relaxation heuristics for the set covering problem, *Journal of the Operations Research Society of Japan* **50** (2007) 350–375.

[86] Y. Wakabayashi: Aggregation of Binary Relations: Algorithmic and Polyhedral Investigations, Ph.D. Thesis (Universität Augsburg, West Germany, 1986).

[87] H. Wang, B. Alidaee, F. Glover, and G. Kochenberger: Solving group technology problems via clique partitioning, *International Journal of Flexible Manufacturing Systems* **18** (2006) 77–97.

[88] H. Wang, T. Obremski, B. Alidaee, and G. Kochenberger: Clique partitioning for clustering: a comparison with K-means and latent class analysis, *Communications in Statistics-Simulation and Computation* **37** (2007) 1–13.

[89] T. Yamada and T. Takeoka: An exact algorithm for the fixed-charge multiple knapsack problem, *European Journal of Operational Research* **192** (2009) 700–705.

[90] B. You and T. Yamada: A pegging approach to the precedence-constrained knapsack problem, *European Journal of Operational Research* **183** (2007) 618–632.

[91] C. T. Zahn: Approximating Symmetric Relations by Equivalence Relations, *SIAM Journal on Applied Mathematics* **12** (1964) 840–847.

[92] W. Zhan, Z. Zhang, J. Guan, and S. Zhou: Evolutionary method for finding communities in bipartite networks, *Physical Review E* **83** (2011) 066120.