ICTEAM, pôle d'Ingénierie Informatique
Ecole polytechnique de Louvain
Université catholique de Louvain
Louvain-la-Neuve, Belgium

Modelling and solving complex combinatorial optimization problems: quorumcast routing, elementary shortest path, elementary longest path and agricultural land allocation

Quoc-Trung BUI

December 2015

Thèse présentée en vue de
l'obtention du grade de Docteur en
Sciences de l'Ingénieur

Composition du jury:

Pr. Yves Deville (Promoteur)	ICTEAM, UCL, Belgium
Dr. Pham Quang Dung (Co-promoteur)	SoICT, HUST, Vietnam
Pr. Laurence Wolsey (Examinateur)	CORE, UCL, Belgium
Pr. Pierre Schaus (Examinateur)	ICTEAM, UCL, Belgium
Pr. Peter Van Roy (Président)	ICTEAM, UCL, Belgium
Pr. Bernard Fortz (Examinateur)	ULB, Belgium

ABBREVIATION

In this thesis, we use the following abbreviations:

- ALAP: agricultural land allocation problem
- ATSP: asymmetric traveling salesman problem
- B&B: branch-and-bound
- B&I: branch-and-infer
- B&C: branch-and-cut
- B&P: branch-and-price
- CBLS: constraint-based local search
- COP: combinatorial optimization problem
- COPs: combinatorial optimization problems
- CP: constraint programming
- CPP: constraint programming problem
- DP: dynamic programming
- ESPP: elementary shortest path problem on graphs with many negative-cost cycles
- ELPP: elementary longest path problem on graphs with many positive-cost cycles

- IP: integer programming
- LP: linear programming
- LPR: linear programming relaxation
- LS: local search
- MIP: mixed Integer programming
- QRP: quorumcast routing problem
- SECs: subtour elimination constraints
- STSP: symmetric traveling salesman problem

NOTATIONS

In this thesis, we use the notations $D = (V_D, E_D, c_D)$ for an undirected graph and $G = (V_G, A_G, c_G)$ for a directed graph (or digraph). An edge between two nodes u, v in an undirected graph is denoted by (v, u) or (u, v); while an arc from i to j is always denoted by (i, j). For the cost functions, an edge (v, u) or (u, v) is associated with a cost $c_D(v, u)$ or $c_D(u, v)(= c_D(v, u))$; and an arc (i, j) is associated with a cost $c_G(i, j)$. On an undirected graph $D = (V_D, E_D, c_D)$, we define the following notations:

- $E_D(S)$: All edges with both endpoints in the node set S
- $V_D(i)$: Set of nodes $\{j \in V_D | (i, j) \in E_D\}$
- $\delta_D(S)$: All edges with one endpoint in S and the other in $V_D \setminus S$

Given a directed graph $G = (V_G, A_G, c_G)$, we define the following notations:

- $A_G(S)$: All arcs with both endpoints in the node set S
- $V_G^-(S)$: Set of nodes $\{v \in V_G | u \in S, (v, u) \in A_G\}$
- $V_G^+(S)$: Set of nodes $\{v \in V_G | u \in S, (u, v) \in A_G\}$
- $\delta_G^-(S)$: All arcs going from a node not in S to a node in S
- $\delta_G^+(S)$: All arcs going from a node in S to a node not in S
- $\delta_G(S) = \delta_G^+(S) \cup \delta_G^-(S)$

- For any vector $x \in \mathbb{R}^{|A_G|}$ and a subset W of A, where each arc $(i, j) \in A$ is associated with an element x_{ij} of the vector, $x(W) = \sum_{(i,j)\in W} x_{ij}$.
- For any subset of arcs $X \subset A_G$, $c_G(X) = \sum_{(i,j)\in W} c_G(i, j)$

ABSTRACT

Combinatorial optimization is a branch of optimization. Its domain is Combinatorial Optimization Problems (COPs) where the set of feasible solutions is discrete and finite. Dealing with a COP is to find optimal solutions in the set of feasible solutions such that the value of a given cost function is minimized or maximized. In the literature, there exist both complete and incomplete methods for solving COPs. The complete (or exact) methods return the optimal solutions with the proof of the optimality, for example the branch-and-cut search method. Incomplete methods, such as local search, try to find hight-quality solutions which are as close to the optimal solutions as possible.

In this thesis we focus on solving four distinct COPs: the quorum-cast routing problem, the elementary shortest path problem on graphs with negative-cost cycles, the elementary longest path problem on graphs with positive-cost cycles, and the agricultural land allocation problem. In order to solve these problems with the complete methods, we use the branch-and-infer search method, the branch-and-cut search method, and the branch-and-price search method. We also solve the agricultural land allocation problem using incomplete methods, such as local search, Tabu search and constraints-based local search combined with metaheuristics. The experimental evaluations on well-known benchmarks show that all proposed algorithms for four COPs are better than the-state-of-the-art algorithms.

We introduce the agricultural land allocation problem, formulate it as a COP. To solve this problem, we divide it into three independent subproblems and propose several incomplete and complete methods for each subproblem. Our solution approach overcomes the limitations of the solution approach of the government.

ACKNOWLEDGEMENTS

First of all, I would like to thank Yves Deville and Quang Dung Pham for the four years they spent helping me. I am very lucky to have two great advisors. Yves Deville is a very demanding supervisor who wants every smallest detail to be clear and intuitive. His ideas and his criticisms sometimes made me nervous or even angry but I could not deny the truth that they were the best advices. Quang Dung Pham gave me the initial ideas and the background for almost all of my research subjects. He spent all his time and his energy for his students. I learned a lot from their honesty and their responsibility in both research and life style.

Many thanks goes to Trong Viet Ho who has shared the same office with me for the first year. He was like my brother in the BeCool team. He supported me a lot in research and daily life in Louvain-la-Neuve.

I want to warmly thank my other teammates in the BeCool group (Ho Trong Viet, Vianney le Clément, Florence Massen, Jean-Baptiste Mairy, Cyrille Dejemeppe, Michael Saint-Guillain, Ratheil Houndji, Francois Aubry, Khong Minh Thanh, Ha Quang Minh) for their ideas and constructive comments to this thesis. I will never forget these fruitful years of working with them - my second family.

Thanks goes as well to the whole INGI department and staff who organized frequently the different research and social activities that made working at UCL interesting and friendly.

I give my thanks to the member of the jury who accepted to read and to comment this thesis. Especially Laurence Wolsey who gave me many comments and advices on my works related to the integer programming and Pierre Schaus who proposed a CP model for a subproblem of the agricultural land allocation problem.

I thank all my Vietnamese friends in Louvain-la-Neuve for including me in a special community that gave me an interesting life out of research with different sport and cultural activities. They helped me a lot to relieve the nostalgia of my country - Vietnam.

The thanks cannot express my gratitude to my parents (Bui Trung Thanh and Nguyen Thi Thuyen) who raised me up and encouraged me in my whole life. They are really great examples for me. Thank you parents for everything that you do for me!

Last but not least, I would like to thank my wife Nguyen Thi Ha Nhan who is always there to support and to encourage me in every difficult moment of my life. I am very lucky to have her in my life.

This research has been supported by UCL (coopération et dévelopement).

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

1 INTRODUCTION

This thesis aims at modeling and solving completely combinatorial problems: QRP, ESPP, ELPP and ALAP. Except ALAP, these problems are complex problems dealing with paths and trees in graphs.

1.1 Combinatorial Optimization Problem

COPs appear in many fields in our real-life including routing, scheduling, packing, timetabling, sequencing, resources allocation, network design, etc. COP is modeled by a set of variables with discrete domains, and we have to find solutions satisfying a given set of constraints while optimizing an objective function. These problems are computationally very hard.

COP = $\langle X, D, C, f \rangle$ where

- $X = \{x_1, ..., x_n\}$ is the set of variables;
- $D = \{D(x_1), ..., D(x_n)\}$ is the set of domains of variables, $D(X_n)$ is the domain of X_n;
- $C = \{C_1, ..., C_k\}$ is the set of constraints over variables;
- f is the objective function to be optimized.

Example Given a COP = $\langle X, D, C, f \rangle$ where

- $X = \{x_1, x_2\}$;
- $D(x_1) = \{0, 1, 2, 3\}, D(x_2) = \{0, 1, 2, 3, 4, 5\}$;
- $C_1 : 3x_1 + x_2 \leq 11, C_2 : -x_1 + 2x_2 \leq 5$;
- f: maximize $6x_1 + 5x_2$.

Clearly, the assignment $x_1 = 3, x_2 = 2$ is an optimal solution to that COP.

Approaches for solving COPs are divided into two categories: complete methods and incomplete methods. The aim of the complete methods is the find an optimal solution with the proof its optimality. Generally, the time complexity of these methods is exponential. The complete methods used in this thesis are B&I search method, B&C search method and B&P search method. The incomplete methods aim at finding high-quality solution in a reasonable time but without information on the quality of the solution. These include Greedy Algorithms, Approximation Algorithms, Local Search, Metaheuristic methods like Tabu Search, Simulated Annealing, Large Neighborhood Search, Very Large-Scale Neighborhood Search, Genetic Algorithms, Ant Colony Optimization, etc. Constraint-based Local Search and Tabu Search are the incomplete search methods that appear in this thesis.

In the following section, we describe problems to be solved in this thesis.

1.2 Applications

Some COPs, which are considered as applications in this thesis, are QRP, ESPP, ELPP and ALAP. Each of them is briefly described in a following subsection.

1.2.1 Quorumcast Routing Problem

Multicasting is the problem of delivering a message from a source to a given subset of nodes, called the multicast nodes, in a network. This thesis deals with QPR which is a generalization of multicasting. This multicasting is formulated as follows [23, 67, 34, 105, 85].

Given an undirected graph $D = (V_D, E_D, c_D)$, where V_D, E_D are respectively the set of nodes, the set of edges and each edge $(i, j) \in E_D$ is associated with a positive cost $c_D(i, j) \in \mathbb{R}^+$; a set of multicast nodes $S \subseteq V_D$; an integral value $q \leq |S|$; and a root node r; the objective of QRP is to find a minimum-cost tree T that spans r and at least q nodes of S.

QRP is NP-hard, as it reduces to the Steiner tree problem [41] when $q = |S|$. It appears in many distributed applications, for example, distributed synchronization and updating a replicated resource.

1.2.2 Elementary Shortest and Longest Path Problems

We consider in this thesis the elementary shortest path problem on graphs with negative-cost cycles and the elementary longest path problem on graphs with positive-cost cycles where the source and the destination node can be fixed or non-fixed. The most general version of the problem is stated as follows.

Given a directed graph (also called a digraph) $G = (V_G, A_G, c_G)$, where V_G is the set of nodes, A_G is the set of arcs and each arc $(u, v) \in A_G$ is associated with a cost $c_G(u, v)$ which can be negative, positive or zero; a set of source nodes $S \subseteq V_G$ and a set of destination nodes $T \subseteq V_G$; ESPP is to find an elementary shortest path starting from a node in S and terminating at a node in T and ELPP is to find an elementary longest path starting from a node in S and terminating at a node in T (a path is called elementary if only if it visit each node at most once).

1.2.3 Agricultural Land Allocation Problem

In most provinces of Vietnam, agricultural land is still fragmented. One household owns too many plots which belong to various land categories and locate at different fields. These plots are very small and scattered. For example, in Vinh Phuc province, one household has 47 plots, each of which has an area of only about ten square meters.

The land fragmentation results in a lot of difficulties for Vietnam. First, households can not use machines for cultivating their small plots, which leads to a high cost of production. Second, fragmented plots

require a very high cost for visiting and controlling them. Third, the excessive number of tracks between the plots give rise to a waste of agricultural land. Finally, projects of agricultural development are confronted with many difficulties caused by the huge number of small plots [83, 78, 48, 65, 72].

The government of Vietnam considers land fragmentation to be "a significant barrier to achieving further productivity gains in agriculture" [32]. So, the government promulgated a policy to do the land reform. This policy consists of reducing the number of land categories and of merging small plots into large fields then repartitioning these fields into larger plots for households in hope to reduce the number of separate plots. In provinces where the land reform was carried out, the results obtained were very promising. After the land reform, the number of plots held by a household markedly decreases (e.g., Bac Ninh [83], a reduction by a factor of 10) and the area of each plot increased, with the rice output increasing considerably (e.g., Quang Nam [83], an increase of 20%–25%). Today, this land reform process has only been applied in some provinces in Vietnam, where the land reform has not just done and where the agricultural land is changed frequently.

In the land reform problem in Vietnam, there are two distinct and consecutive tasks: the first task is to merge small plots into large fields and the second task is to split the large fields into plots. The first task is very easy. Consequently we can forgot the first task and focus on the second task which is officially named by Agricultural Land Allocation Problem (ALAP). This problem is briefly described as follows. In a region, such as a municipality, there are a set of agricultural fields, each field belongs to a land category, that is its land quality. We also have a set of households living in that region and owning those fields. For each land category, a household is associated with an expected area. It means that the household will be distributed one or several plots of the land category and the total area of these plots equal the expected area of the household. The objective of ALAP is to specify, for each household, the number of plots assigned to that household, the area of the plots and the geographical position of the plots. The objective is to minimize the number of plots. There is an additional constraint; a field should be split into smaller fields by long parallel tracks before dividing them into plots. This simplifies the use of machines in the inner plots

(the machines reach to plots by the long parallel tracks), see Figure 1.1 to understand well this constraint.

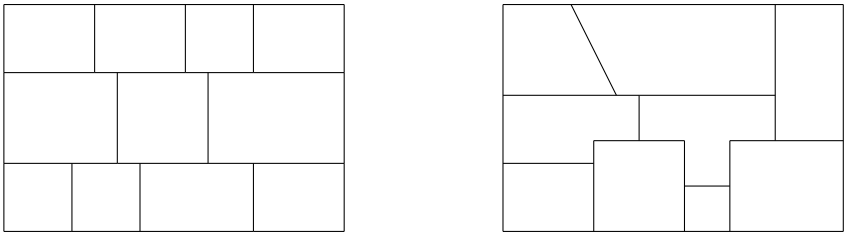

Figure 1.1: Households desire the solution on the left hand side of the figure. While the solution on the right hand side of the figure is not acceptable because the shape of some plots is not beautiful (not close to a square), the plots are not separated to others by parallel tracks.

1.3 Contributions

The main contributions of this thesis are as follows.

1. For QRP,
    - we propose four MIP formulations and use them to solve QRP by B&C algorithms that outperform the-state-of-the-art algorithm based on CP;
    - moreover through the experimental results, we show the effect of the values q and $|S|$ on the performance of the B&C algorithms.
2. For ESPP,
    - we introduce valid inequalities for ESPP, some of which are derived from ATSP;
    - we propose a separation algorithm for SECs which is simple and heuristic;

- we propose a B&C algorithm with a proposed cut filter that solves this problem more quickly than the-state-of-the-art algorithm in [33];
- we extend the decomposition technique in [84] to propose a DP algorithm for solving efficiently this problem on undirected graphs with many bridges and directed graphs whose corresponding undirected graphs consist of many bridges;
- we propose a decomposition technique and a DP algorithm for solving efficiently this problem on directed graphs with many strongly connected components.

3. For ELPP,

   - we adapt the B&C algorithm proposed in this thesis for ESPP to obtain a B&C algorithm for ELPP, which is faster than the-state-of-the-art exact algorithm based on CP in [84];
   - we adapt our previous preprocessing schema [84] that decomposes the undirected graph with many bridges into bridge-blocks so that it can be used for directed graphs with both positive-cost and negative-cost arcs;
   - by applying our B&C algorithm (instead of applying a CP-based algorithm) on each bridge-block, the overall performance is better than that of paper [84].

4. For ALAP,

   - we introduce this problem with an abstract single formulation;
   - we propose a formulation of ALAP in terms of three subproblems (PArea, PPos1 and PPos2) that are clearly formulated;
   - we propose various models for the subproblems (three CP models and five MIP models);
   - different optimization algorithms have been designed, using different optimization techniques (two B&I algorithms, four B&C algorithms, two B&P algorithms, four LS algorithms,

one CP large neighborhood search algorithm and one hybrid algorithm);

- experimental results compare the different approaches, they show that for the different subproblems, LS approach is here almost as accurate as the complete approaches, while being much more efficient in terms of computational time;
- experimental results compare the solutions computed by the government algorithm with ones computed by our approaches, they show that our solutions are much better than solutions of the government.

Lastly, all our models, algorithms, test instances and experiments are made *Open* and *Public*; and are available at http://becool.info.ucl.ac.be/

1.4 Publications

The content of this thesis leads to two papers accepted in refereed international conferences and two articles under review.

- Q.T. Bui, Q.D. Pham, Y. Deville. Solving the Quorumcast Routing Problem as a Mixed Integer Program, 11th International Conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems (CPAIOR 2014), Cork, Ireland, May 2014, Lecture Notes in Computer Science, Springer, 2014.

- Q.T. Bui, Q.P. Dung, Y. Deville. Solving the Agricultural Land Allocation Problem by Constraint-Based Local Search, 19th International Conference on Principles and Practice of Constraint Programming (CP 2013), Uppsala, Sweden, October 2013. Lecture Notes in Computer Science, Springer, 2013.

- Q.T. Bui, Q.P. Dung, Y. Deville. Exact Methods for Solving the Elementary Shortest and Longest Path Problems. Under review.

- Q.T. Bui, Q.P. Dung, Y. Deville. Optimization Approaches for the Agricultural Land Allocation Problem. Under review.

1.5 Outline

The remainder of this thesis is organized as follows:

Chapter 2 presents briefly search methods which are used in this thesis. Three famous complete methods, which are summarized here, are B&I search method, B&C search method and B&P search method. The principle of LS, CBLS, TB and metaheuristics are also presented in this chapter.

Chapter 3 focuses on solving QRP to optimality by branch-and-cut algorithms. In this chapter, four MIP formulations for this problem are proposed in Section; five branch-and-cut algorithms based on the proposed formulation are proposed in Section 3.3; and the experimental section compares the proposed algorithms to the-state-of-the-art algorithm which bases on CP approach and point out the influence of the parameter q and the size of the multicast node set on the performance of the proposed algorithms.

Chapter 4 aims at solving two problems: ESPP and ELPP. In this chapter, the equivalence between some problems is given in Section 4.1; while Section 4.2 presents the related works; all Section 4.3 is used to represent three MIP formulations for ESPP(s, t, G); some classes of inequalities that are valid for ESPP(s, t, G) are presented in Section 4.4; Section 4.5 describes in details our proposed algorithm for solving ESPP(s, t, G). Section 4.6 proposes decomposition techniques allowing the use of a dynamic programming schema for solving ESPP(s, t, G) and ELPP(G) on directed graphs with bridges; the experimental results are reported in Section 4.7.

Chapter 5 introduces and solves ALAP. After Section 5.1 describing the government's solution to ALAP and a short introduction to the different optimization techniques, this chapter dedicates one section to each of the three subproblems: PArea, PPos1 and PPos2. In each section, we present the problem formulation, propose algorithms for the problem, and give the experimental results.

The last chapter concludes this thesis and draw some research directions in the future.

2

OPTIMIZATION TECHNIQUES

This chapter aims at presenting briefly complete and incomplete search methods which will be used in the next chapters of the thesis.

- The complete search methods search for optimal solutions to COPs with the proof of the optimality. The complete search methods appearing in this thesis are B&I search method, B&C search method, and B&P search method.
- The objective of the incomplete search methods is to find hight-quality solutions to COPs which are as close to the optimal solutions as possible in reasonable time but without information on the quality of the solution. Local search, the constraint-based local search, Tabu search are the incomplete search methods used in this thesis.

2.1 Complete search methods

A search method is called complete or exact if only if it returns optimal solutions to given problems with the proof of the optimality. This section presents three well-known complete search methods: B&I search method, B&C search method and B&P search method. The common feature of these methods are all basing on branching search method which is presented in the following subsection.

2.1.1 Branching search method

Branching search method implements a recursive dived-and-conquer strategy. If the given problem P_0 is too hard to solve, then this method creates a series of subproblems $P_1, \ldots, P_m$ of P_0 and tries to solve each of them. This partition is called branching at P_0. Each P_i is solved, if possible. If a solution obtained, it becomes a candidate solution of P_0. The best candidate solution found so far is the *incumbent solution*. If P_i is too hard to solve, this method treats P_i in the same way as P_0(branching at P_i), and so on, recursively. This method begins with a single unprocessed problem (the original problem) and terminates when no unprocessed problem remains. When no unprocessed problem remains, the incumbent solution computed so far is an optimal solution [51].

Outline of the branching search method

Algorithm 1 summaries basic steps of a branching search algorithm for a minimum combinatorial optimization problem.

Algorithm 1: A basic branching search algorithm for a minimum COP

```
1 Let S ← {P_0} and v_UB ← ∞ ;
2 while S is nonempty do
3     Select a subproblem P ∈ S and remove P from S ;
4     if P is too hard to solve then
5         Define subproblem P_1, ..., P_m of P and add them to S ;
6     else
7         Let v be the optimal value of P and let
          v_UB ← min{v, v_UB} ;
8 The optimal value of P_0 is v_UB;
```

To ensure an exhaustive search, the subproblem $P_1, \ldots, P_m$ of P created should be exhaustive. Normally, their feasible sets also partition the feasible set of P (i.e., they are pairwise disjoint) [51].

To ensure that the search terminates, the branching mechanism must be designed so that problems become easy enough to solve as they are increasingly restricted. For instance, if the variable domains are finite, then branching on variables will eventually reduce the domains to singletons, thus fixing the value of each variable and making the restriction trivial to solve [51].

2.1.2 Branch-and-infer search method

B&I search method is a combination of the branching search method with *inference*, in which the inference is performed at each node of the search tree to infer new constraints that make the problem at hand easier. This method is widely in CP for solving not only COPs but also Constraint Satisfaction Problems (CSPs). In a CP model of a CSP or COP, there are a set of decision variables whose domain is finite and a set of constraints defined over the variables. Normally the inference component of this method tries to reduce the size of the variables' domain by exploiting the set of constraints. B&I search method is very useful to attack COPs where the constraints are hard to formulate, such as for example in scheduling problems.

Outline of the B&I search method

A simple outline of a B&I search algorithm is given in Algorithm 2.

There are many strategies for branching (line 12), for example, domain splitting. And there are also many strategies for choosing the next node to exploit (line 5), for example, depth-first search [92]. The operation of the inference (line 6) bases on the constraint set of the problem. During the inference, a constraint may be considered many times in hope of reducing as much as possible the size of variable domain. After the inference step, one of three following situations can be happened:

- There exists a variable whose domain is empty. The current subproblem has no solution, so the algorithm tries to solve another subproblem in S.
- The domain of every variable remains only one value, so a feasible solution of P_0 is found, and this solution is compared to the

Algorithm 2: A general B&I search algorithm for a minimum COP

```
1  P_0 = ⟨X, D_0, C_0, f⟩ ← the original COP, in which X_0 is the set
   of variables, D_0 is the set of variable domains and C_0 is the set of
   new constraints ;
2  f ← objective function that is minimized;
3  S ← {P_0} and v_UB = +∞ ;
4  while S is nonempty do
5      Select a subproblem P = ⟨X, D_p, C_p, f⟩ ∈ S and remove P
       from S;
6      Perform the inference using C to infer a set of constraints C';
7      C_p ← C_p ∪ C' ∪ f ≤ v_UB;
8      if ∀x ∈ X, |D(x)| = 1 then
9          Let v be the solution value of P;
10         v_UB ← min{v, v_UB} ;
11     else if ∀x ∈ X, |D(x)| ≥ 1 and ∃x ∈ X, |D(x)| > 1 then
12         Define subproblem P_1, ..., P_m of P and add them to S;
           for each P_i, in addition to C, it has own branching
           constraints;
13 The optimal value of P_0 is v_UB;
```

current incumbent solution to ensure that the incumbent solution is always the best solution found so far.

- Every variable has a nonempty domain and there exists at least a variable whose domain contains more than one value, so the branching occurs.

A simple example

In order to illustrate the operation of the B&I search method, we consider an instance of the graph coloring problem. This instance can be described as follows. We need at most how many different colors to color the graph in Figure 2.1 such that there does not exist any arc connecting two nodes with the same color ? As the result of some heuristic

search in the preprocessing step, we know that for this instance, we do not need more than 4 different colors.

This instance is formulated as follows.
$COP = \langle X, D, C, f \rangle$, where

- $X = (x_1, \ldots, x_6)$ (one variable for each node of the graph);
- $D(x_i) = \{r, g, b, y\}$ with $i \in \{1, 2, \ldots, 6\}$ (we do not need more than 4 different colors, so the domain of the variable contains at most 4 values.);
- $C = \{c_1, \ldots, c_1 1\}$ where $c_1 : x_1 \neq x_2$, $c_2 : x_1 \neq x_3$, $c_3 : x_2 \neq x_3$, $c_4 : x_2 \neq x_4$, $c_5 : x_2 \neq x_5$, $c_6 : x_2 \neq x_6$, $c_7 : x_3 \neq x_5$, $c_8 : x_3 \neq x_6$, $c_9 : x_4 \neq x_5$, $c_{10} : x_4 \neq x_6$, and $c_{11} : x_5 \neq x_6$
- f: the number of colors used to color the graph is minimized.

A B&I search algorithm for solving this instance creates a part of the search tree that is given in Figure 2.2. And the domain of the variables at each node of this part of the search tree is given in Figure 2.1.

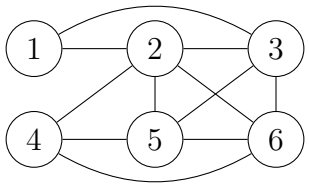

Figure 2.1: The graph of the instance

2.1.3 Branch-and-cut search method

B&C search method is a combination of branch-and-bound (B&B) search method with *cutting plane* method. This search is frequently used in integer programming. This method works by solving a sequence of LP relaxations of an integer programming problem. Cutting plane methods are invoked at each node of the search tree to find new cutting planes (constraints) in hope of improving the relaxation of the program to more closely approximate the integer programming problem [76].

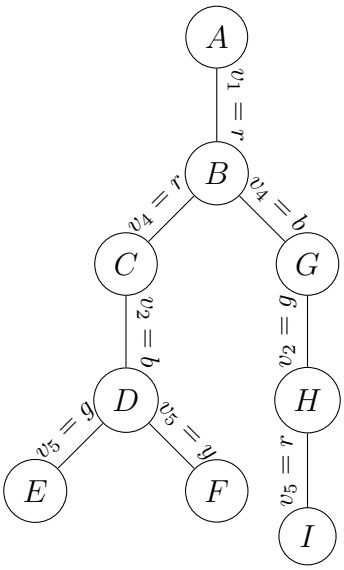

Figure 2.2: A part of the search tree constructed by the B&I search algorithm

	Domain of the variables					
Nodes	v_1	v_2	v_3	v_4	v_5	v_6
A	r,g,b,y	r,g,b,y	r,g,b,y	r,g,b,y	r,g,b,y	r,g,b,y
B	r	g,b,y	g,b,y	r,g,b,y	r,g,b,y	r,g,b,y
C	r	g,b,y	g,b,y	r	g,b,y	g,b,y
D	r	b	g,y	r	g,y	g,y
E	r	b	g	r	y	y
F	r	b	y	r	g	g
G	r	g,y	g,b,y	b	r,g,y	r,g,y
H	r	g	b,y	b	r,y	r,y
I	r	g	b	b	r	y

Table 2.1: Domain of the variables at the nodes of the part of the search tree in Figure 2.2

In the literature, many COPs can be formulated as an integer programming problems which can be well solved by the B&C search method.

The name "branch-and-cut" was coined by Padberg and Rinaldi [80]. The first and best known B&C search computer code for solving STSP was written by Hong [50, 7].

The B&B search method integrates the branching search method and *bounding procedure*. The bounding procedure calculates lower and/or upper bounds for the objective function value of the subproblem in hope of eliminating unpromising subproblems [26].

The cutting plane method for a general integer programming problem were firstly proposed by Gomory [45]. Unfortunately, the cutting planes proposed by Gomory are not very strong, consequently these algorithms converge slowly, so the algorithms were neglected for many years. However, the development of polyhedral theory makes a resurgence of cutting plane methods in the 1980's, and now they are the first choice for a vast variety of COPs. Typically, we can determine theoretically a partial polyhedral description of the convex hull of a set of integer feasible points [76].

Outline of the B&C search method

The schema of B&C method, which is given Algorithm 3, comes from John E. Mitchell's work [76].

In **Step 5** of this schema, a very large number of violated cutting planes are found, so they should be sorted in some ways, for example by violation, and just a subset should be added.

The objective of **Step 4** of the schema is to choose the next node to expand. A naive node selection strategy might lead to a huge search tree. By contrast, an intelligent node selection strategy leads quickly to a good feasible solution that sharply reduces the gap between the upper bound and lower bound and proves the optimality of the current incumbent. There exist some well-known node-selection strategies, such as depth-first-search, breadth-first-search, and best-bound search. In order to minimize the size of the search tree, the *best-bound search* strategy, which selects the node with the best lower bound, is carried out. Its advantage is that, for a fixed sequence of branching decisions, it minimizes the number of nodes that are explored, because all nodes that are explored would have been explored independently of the upper bound [13].

There exist many strategies for the branching (for **Step 9**). A simple branching strategy applied is to select the variable with the largest integer violation: this is known as *maximum fraction branching* [62, 9]. In practice, this rule is not efficient: it performs about as well as randomly selecting a branching variable [4]. The most successful branching strategies estimate the change in the lower bound after branching. Because we prune a node of the branch-and-bound tree whenever the lower bound of the node is greater than the current upper bound, we want to increase the lower bound as much as possible. In [4], Achterberg et al. also experimented with strong branching rules, pseudocost branching, strong branching, a hybrid of strong/pseudocost branching, pseudo cost branching with strong branching initialization, and reliability branching.

A simple example

This simple example comes from the article of John E. Mitchell in [76]. The integer programming problem is illustrated in Figure 2.3. The feasible integer points are marked. The LP relaxation is obtained by removing the integral constraints and is indicated by the polyhedron contained in the solid lines.

$$
\begin{aligned}
& min \quad z = -6x_1 - 5x_2 && \text{(2.1a)}\\
s.t \quad & 3x_1 + x_2 \leq 11 && \text{(2.1b)}\\
& -x_1 + 2x_2 \leq 5 && \text{(2.1c)}\\
& x_1, x_2 \geq 0, integer && \text{(2.1d)}
\end{aligned}
$$

A B&C search algorithm first solves the LP relaxation giving the point $(2\frac{3}{7}, 3\frac{5}{7}$, with objective value $-33\frac{1}{7}$. Now there is a choice: should the LP relaxation be improved by adding a cutting plane, for example, $x_1 + x_2 \leq 5$, or should the problem be divided into two by splitting on a variable?

Algorithm 3: A general B&C search algorithm for a minimum COP

1 **Step 1.** *Initialization:* Denote the initial integer programming problem by ILP^0 and set of the active nodes to be $L \leftarrow \{ILP^0\}$. Set the upper bound to be $\overline{z} \leftarrow +\infty$. Set $\underline{z_l} \leftarrow -\infty$ for the one problem $l \in L$.

2 **Step 2.** *Termination:* If $L = \emptyset$, then solution x^* which yielded the incumbent objective value $\overline{z}$ is optimal. If no such x^* exists (i.e., $\overline{z} = +\infty$) then ILP^0 is infeasible.

3 **Step 3.** *Problem selection:* Select and delete a problem ILP^l from L.

4 **Step 4.** *Relaxation:* Solve the LP relaxation of ILP^l. If the relaxation is infeasible, set $\underline{z_l} \leftarrow +\infty$ and go to **Step 6**. Let $\underline{z_l}$ denote the optimal objective value of the relaxation if it is finite and let x^{lR} be an optimal solution; otherwise set $\underline{z_l} = -\infty$.

5 **Step 5.** *Adding cutting planes:* If desired, search for cutting planes that are violated by x^{lR}; if any are found, add them to the relaxation and return to **Step 4**.

6 **Step 6.** *Fathoming and Pruning:*
(a) If $\underline{z_l} \geq \overline{z}$ go to **Step 2**.
(b) If $\underline{z_l} < \overline{z}$ and x^{lR} is integral feasible, update $\overline{z} = \underline{z_l}$, delete from L all problems with $\underline{z_l} \geq \overline{z}$, and go to **Step 2**.

7 **Step 7:** *Partitioning:* Let $S^{l1}, \ldots, S^{lk}$ be a partition of the constraint set S^l of problem ILP^l. Add $ILP^{li}, \ldots, ILP^{lk}$ to L, where ILP^{lj} is ILP^l with feasible region restricted to S^{lj} and $\underline{z}_{lj}$ for $j = 1, \ldots, k$ is set to the value of $\underline{z_l}$ for the parent problem l. Go to **Step 2**.

If the search splits on x_1, two new subproblems are obtained:

$$min \quad z = -6x_1 - 5x_2 \tag{2.2a}$$
$$s.t \qquad 3x_1 + x_2 \leq 11 \tag{2.2b}$$
$$-x_1 + 2x_2 \leq 5 \tag{2.2c}$$
$$x_1 \geq 3 \tag{2.2d}$$
$$x_1, x_2 \geq 0, integer \tag{2.2e}$$

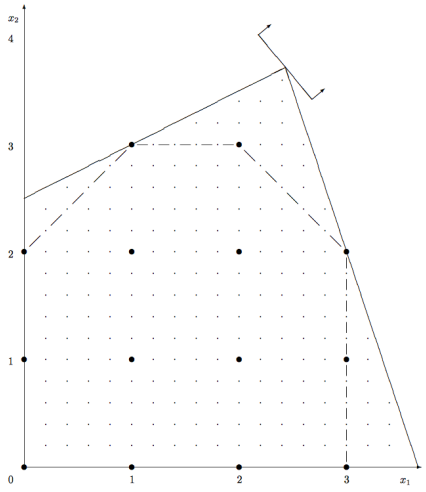

Figure 2.3: A two dimensional integer programming problem

and

$$min \;\; z = -6x_1 - 5x_2 \tag{2.3a}$$
$$s.t \qquad 3x_1 + x_2 \leq 11 \tag{2.3b}$$
$$-x_1 + 2x_2 \leq 5 \tag{2.3c}$$
$$x_1 \leq 2 \tag{2.3d}$$
$$x_1, x_2 \geq 0, integer \tag{2.3e}$$

The optimal solution to the original problem will be better than the solutions to these two subproblems. The solution to the linear relaxation of (2.2) is (3,2), with objective value -28. This solution is integral, so it solves (2.2), and becomes the incumbent best know feasible solution. The LP relaxation of (2.3) has optimal solution (2, 3.5), with objective value -29.5. This point is nonintegral, so it does not solve (2.3), and it must be attacked further.

Assume the B&C search algorithm uses a cutting plane method and adds the inequality $2x_1 + x_2 \leq 7$ to (2.3). This is a valid inequality, in that it is satisfied by every integral point that is feasible in (2.3).

Further, this inequality is violated by (2, 3.5), so it is a cutting plane. The resulting subproblem is

$$
\begin{aligned}
& min \;\; z = -6x_1 - 5x_2 && \text{(2.4a)}\\
s.t \quad & 3x_1 + x_2 \leq 11 && \text{(2.4b)}\\
& -x_1 + 2x_2 \leq 5 && \text{(2.4c)}\\
& x_1 \leq 2 && \text{(2.4d)}\\
& 2x_2 + x_2 \leq 7 && \text{(2.4e)}\\
& x_1, x_2 \geq 0, integer && \text{(2.4f)}
\end{aligned}
$$

The LP relaxation of (2.4) has optimal solution (1.8, 3.4), with objective value -27.8. Notice that the optimal value for this modified relaxation is larger than the value of the incumbent solution. The value of the optimal integral solution to the second subproblem must be at least as large as the value of the relaxation. Therefore, the incumbent solution is better than any feasible integral solution for (2.4), so it actually solves the original problem.

The progress of the B&C search algorithm is illustrated in Figure 2.4.

2.1.4 Branch-and-price search method

B&P search method integrates *column generation* and B&B search method. Similar to the B&C search method, this method is also for solving large integer programming problems, and it also solves a sequence of LP relaxations of the integer programming problem. At each node of the search tree, cutting planes (or constraints) may be generated to tighten the LP relaxation in the B&C search method, by contrast columns (or variables) must be generated to improve the LP relaxation in the B&P search method.

In the B&P search method, sets of columns (or variables) are left out of the LP relaxation of large integer programming problems because there are too many columns to handle efficiently and most of them might will be assigned to zero in the optimal solution. At each node of the search tree, in order to check for optimality, a subproblem, which is

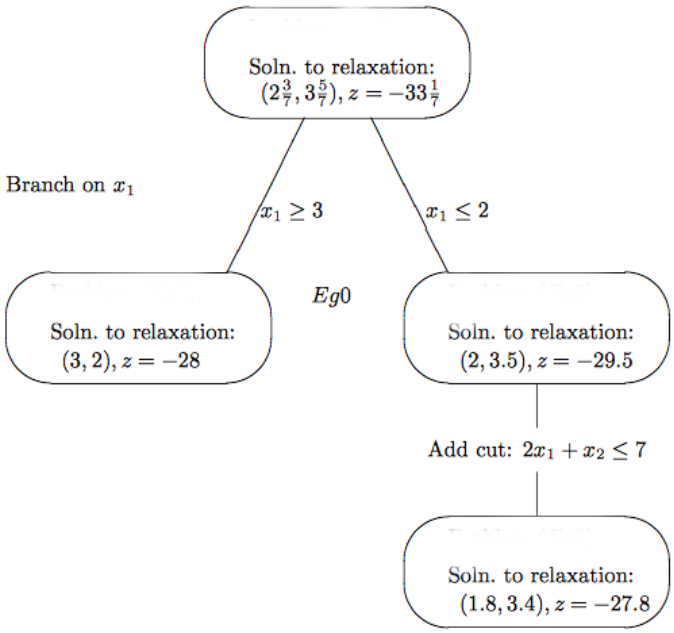

Figure 2.4: Progress of branch-and-cut on the two dimensional integer programming problem

also called the "pricing problem", is solved to identify columns to enter the basis. If such columns are found then the LP relaxation is reoptimized. The branching occurs when no columns "prices" out to enter the basis and the solution to the LP relaxation is not integral. Otherwise (no columns "prices" out to enter the basis and the solution to the LP relaxation is integral) a feasible solution is found. When all subproblems are already solved, the best solution found so far (incumbent) is an optimal solution.

Outline of the B&P search method

Outline of a B&P search algorithm is given in Algorithm 4. In comparison with the outline of a B&C algorithm (Algorithm 3), we can easily find out two following significant differences.

1. In **Step 5**, the cutting plane method in the B&C search algorithm tries to find valid inequalities (row generation) which are violated by the optimal solution to the LP relaxation. While, in the B&P search algorithm, a subproblem (differing from the restriction of the original problem) is solved to find new variables to enter the basic (column generation).

2. In **Step 4**, the LP relaxation of a restricted master problem in the B&P search algorithm is always feasible, while this property is not true for the B&C search algorithm.

A simple example

To understand well the operation of a B&P search algorithm, let us solve an instance of the Generalized Assigment Problem by a simple B&P search algorithm. The objective of this task is to maximum profit assignment of 3 jobs to 2 machines, such that each job is assigned to precisely one machine subject to capacity restrictions on the machines.

The instance details: the profit matrix is given Table 2.2, the capacities of machines (C_i) are given in Table 2.3, and the amount of capacity of machine i taken up by each job j is also given in Table 2.4.

We denote k_i the set of possible feasible assignments to machine i, thus $k_1 = \{(1,0,0),(0,1,0),(0,0,1),(1,0,1)\}$ and $k_2 = \{(1,0,0),(0,1,0),(0,0,1),(1,1,0),(0,1,1),(1,0,1)\}$. As the results, the **Master Problem** is now formulated as follows:

$$max\ z = 5y_1^1 + 7y_2^1 + 3y_3^1 + 8y_4^1 + 2y_1^2 + 10y_2^2 + 5y_3^2 + 12y_4^2 + 15y_5^2 + 7y_6^2 \tag{2.5}$$

s.t.

$$y_1^1 + y_4^1 + y_1^2 + y_4^2 + y_6^2 = 1 \qquad (u_1) \tag{2.6}$$

$$y_2^1 + y_2^2 + y_4^2 + y_5^2 = 1 \qquad (u_2) \tag{2.7}$$

$$y_3^1 + y_4^1 + y_3^2 + y_5^2 + y_6^2 = 1 \qquad (u_3) \tag{2.8}$$

Algorithm 4: A general B&P search algorithm for a minimum COP

1 **Step 1.** *Initialization:* $MP \leftarrow$ the master problem; $RMP^0 \leftarrow$ the initial restricted master problem; set of active nodes, $L \leftarrow RMP^0$; the upper bound, $\overline{z} \leftarrow +\infty$; for each $l \in L$, $\underline{z_l} \leftarrow -\infty$.

2 **Step 2.** *Termination:* If $L = \emptyset$, then the current incumbent solution x^* with the objective value $\overline{z}$ is optimal. If no such x^* exists then MP is infeasible.

3 **Step 3.** *Problem selection:* Select and delete a restricted master problem RMP^l from L.

4 **Step 4.** *Relaxation:* Solve the LP relaxation of RMP^l. Let $\underline{z_l}$ and x_l^R be the value of the optimal solution to the relaxation and the optimal solution, respectively.

5 **Step 5.** *Generating and adding columns:* Generate dual values from the restricted master problem. Solve the subproblem using the dual values in hope of identifying new columns to add to the restricted master problem. If there exist any columns found by solving the subproblem that have positive reduced cost, add them to the restricted master problem RMP^l and go to **Step 4**. If no such column exists, the algorithm goes to one of two following situations. If x_l^R is an integral solution, go to **Step 6**. Otherwise, x_l^R is not integral, go to **Step 7**.

6 **Step 6.** *Fathoming and Pruning:* If $\underline{z_l} \geq \overline{z}$ go to **Step 2**. If $\underline{z_l} \geq \overline{z}$ and x_l^R is integral, update $\overline{z} \leftarrow \underline{z_l}$, delete from L all problems with $\underline{z_l} \geq \overline{z}$, and go to **Step 2**.

7 **Step 7:** *Branching:* Let $S^{l1}, \ldots, S^{lk}$ be a partition of the constraint set S^l of problem RMP^l. Add $RMP^{li}, \ldots, RMP^{lk}$ to L, where RMP^{lj} is RMP^l with feasible restricted to S^{lj} and $\underline{z_l}_j$ for $j = 1, \ldots, k$ is set to the value of $\underline{z_l}$ for the parent problem l. Go to **Step 2**.

$$y_1^1 + y_2^1 + y_3^1 + y_4^1 \leq 1 \qquad (v_1) \qquad (2.9)$$

	Job (j)		
	1	2	3
Machine i = 1	5	7	3
Machine i = 2	2	10	5

Table 2.2: Profit matrix

i	C_i
1	5
2	8

Table 2.3: The capacity of the machines

	Job (j)		
	1	2	3
Machine i = 1	3	4	2
Machine i = 2	4	3	4

Table 2.4: The matrix of the capacity of machine by job

$$y_1^2 + y_2^2 + y_3^2 + y_4^2 + y_5^2 + y_6^2 \leq 1 \quad (v_2) \tag{2.10}$$

In this master problem, the first three constraints indicate that each job must be taken by machines; the last two constraints present that each machine can be used at most once; u_j and v_i denote the dual values associated with job j and machine i respectively.

Let us arbitrarily choose column y_1^1 and y_5^2, which is a feasible solution. Now, the Restricted Master Problem (RMP) is as follows:

$$max \;\; z = 5y_1^1 + 15y_5^2 \tag{2.11}$$

s.t.

$$y_1^1 + 0 = 1 \qquad (u_1) \tag{2.12}$$

$$0 + y_5^2 = 1 \qquad (u_2) \tag{2.13}$$

$$0 + y_5^2 = 1 \tag{2.14}$$

$$y_1^1 + 0 = 1 \tag{2.15}$$

$$0 + y_5^2 = 1 \tag{2.16}$$

The last three constraints are redundant, hence their corresponding dual values are 0.

Now $B = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$, therefore $B^{-1} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ and $c_B B^{-1} = \begin{bmatrix} 5 & 15 \end{bmatrix}$

Hence $u_1 = 5, u_2 = 15, v_1 = 0, v_2 = 0$.

Subproblem for Machine 1

max $(5-5)x_1^1 + (7-15)x_1^2 + (3-0)x_1^3$

s.t. $3x_1^1 + 4x_1^2 + 2x_1^3 \leq 5$

Optimal solutions are (1,0,1) and (0,0,1) with $z = 3$

Hence the reduced cost $z(KP_1) - v1 = 3 - 0 = 3$

Subproblem for Machine 2

max $(2-5)x_2^1 + (10-15)x_2^2 + (3-0)x_2^3$

s.t. $5x_2^1 + 3x_2^2 + 4x_2^3 \leq 8$

Optimal solution is (0,0,1) with $z = 5$

Hence the reduced cost $z(KP_2) - v2 = 5 - 0 = 5$ Since the reduced cost for machine 2 is higher, so sequence (0,0,1) for machine 2 is the column generated.

Now the new Restricted Master Problem is:

$$max \;\; z = 5y_1^1 + 15y_5^2 + 5y_3^2 \tag{2.17}$$

s.t.

$$y_1^1 + 0 + 0 = 1 \qquad (u_1) \tag{2.18}$$

$$0 + y_5^2 + 0 = 1 \qquad (u_2) \tag{2.19}$$

$$0 + y_5^2 + y_3^2 = 1 \qquad (u_3) \tag{2.20}$$

$$y_1^1 + 0 + 0 = 1 \tag{2.21}$$

$$0 + y_3^2 y_5^2 = 1 \tag{2.22}$$

The last two constraints are redundant, hence their corresponding dual values are 0.

Now $B = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

Therefore $B^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$ and $c_B B^{-1} = \begin{bmatrix} 5 & 15 & 5 \end{bmatrix}$

Hence $u_1 = 5, u_2 = 15, u_3 = 5, v_1 = 0, v_2 = 0$.

Subproblem for Machine 1

max $(5-5)x_1^1 + (7-15)x_1^2 + (3-5)x_1^3$

s.t. $3x_1^1 + 4x_1^2 + 2x_1^3 \leq 5$

Optimal solutions are (1,0,0) and (0,0,0) with $z = 0$

Hence the reduced cost $z(KP_1) - v1 = 0 - 0 = 0$

Subproblem for Machine 2

max $(2-5)x_2^1 + (10-15)x_2^2 + (5-5)x_2^3$

s.t. $5x_2^1 + 3x_2^2 + 4x_2^3 \leq 8$

Optimal solutions are (0,0,1) and (0,0,0) with $z = 0$

Hence the reduced cost $z(KP_2) - v2 = 0 - 0 = 0$

Therefore, the reduced costs for all columns are 0. Hence the solution $y_1^1 = 1, y_5^2 = 1, y_3^2 = 0$ is optimal.

2.2 Local search

In the literature, there exist many incomplete methods. However, in this thesis, we only focus on LS for solving COPs. LS is an efficient method that can find high quality solutions to COP in polynomial time [74]. A LS algorithm for a COP typically starts from an initial solution

and moves from solutions to neighboring solutions by changing some features of the solutions. The goal of the movement is to obtain solutions which are better than solutions found so far. The search process terminates when some criteria are met, for example no one constraint is violated. In this section, we first discus the principles of a LS algorithm, then we present CBLS, lastly we describe some fundamental metaheuristics which can be combined with LS to improve the LS performance.

2.2.1 Local search algorithm

We give here some discussion about designing an efficient LS algorithm for a general COP.

Mathematical problem formulation

The first step of designing a LS algorithm for the COP is to give a mathematical formulation for that problem. For a problem, there may exist many different formulations. Two LS algorithms, which are suitable for two different formulations, may be very distinct. A good formulation helps the corresponding LS algorithm find out high quality solutions in a small amount of computation time.

Normally, proposing a formulation for a COP contains two consecutive steps. In the first step, we have to define clearly decision variables and their domain. We formulate all constraints and the objective function of the COP over the decision variables in the second step.

Initial solution generation

From the mathematical problem formulation proposed in the previous step, we define a solution to problem. A solution of the problem is an assignment of values to the decision variables. A feasible solution is a solution of the problem which does not violate any constraint of the problem. A LS algorithm starts from an initial solution, so we need an initial solution generation.

Initial solutions can be generated randomly or by applying some heuristics in hope of obtaining high quality initial solutions. Initial solutions should be on different regions of the solution space. When the

constraints of the problem are many and complicated, random initial solutions may violated many constraints, as consequently we may loose a large amount of computational time to obtain a feasible solution, in this case initial solutions should be generated by applying some heuristics.

Neighborhood construction

The main step of a LS algorithm is moving from a solution to a neighboring solution. So, the definition of neighboring solutions of a solution is very important in designing a LS algorithm. A neighbor can be obtained by changing some components (or parts or features) of the given solution. Moreover, the size of neighborhood has a strong influence on the performance of the LS algorithm. A small-size neighborhood may lead to the LS algorithm be trapped on low-quality solutions. Otherwise a large-size neighborhood takes a costly computation time to select a neighboring solution to move on.

Neighbor selection

The task of this component is to select a neighboring solution from one or various neighborhoods. This step can be done randomly or based on a quality evaluation of each neighboring solution. There are some heuristic strategies for the selection, such as first improvement, best improvement, random selection. A good-design LS algorithm should keep the balance between the size of neighborhoods and the time complexity of neighbor selection.

Metaheuristics are often implemented in this component to guide the search. Several metaheuristics will be described later in this chapter.

Move

Move step transforms the current solution into its neighboring solution which is already selected by the neighbor selection component. This step is done by local modifications of solution components and of the value of the objective function.

Termination criteria

This component decides when the LS algorithm should be ended, for example, a given number of iterations has been exceeded. In order to obtain hight quality solutions in an appropriate amount of computation time, the termination criteria should be carefully decided, for example a LS algorithm can not obtain good solution if it ends before the algorithm converge.

2.2.2 Constraint-based local search

CBLS uses constraints to describe and control local search [103]. In a COP, in addition to an objective functions, there is a set of constraints, so a solution to the COP have to satisfy the set of constraints and have an optimal objective function value. CBLS for a COP, the search is directed by a clever combination of the constraints with the objective function.

2.2.3 CP-Large neighborhood search

The goal of LNS is to avoid being stuck in a region of the search tree for too long by restarting frequently to explore other regions. The principle is the following [99]:

- Keep a current best solution to your problem, repeat the following two steps (relax + restart) until a stopping criteria is met.
- Relax: relax the current best solution (typically by fixing only a fraction of the decision variables to their value in the current best solution).
- Restart: Try to improve the current best solution with CP and replace it if a better solution can be found. This restart has a limited number of failures.

2.2.4 Metaheuristics

The goal of heuristics in LS is to reach hight quality local minima quickly. Metaheuristics have a fundamentally role: they aim at escaping

these local minima and at directing the search toward global optimality [103]. This objective can be reached in many different ways.

Iterated local search

Iterated Local Search (ILS) is a ubiquitous metaheuristic that iterates a specific local search from different starting points in order to sample various regions of the search space and to avoid returning a low-quality local minimum [103]. Different starting points increase the diversification the search. Different heuristics can be used to generate starting points. Figure 2.5 depicts an example of ILS. With the different restart points, LS can explore different regions of the search space. Obviously, the opportunity to achieve the global optimum is larger.

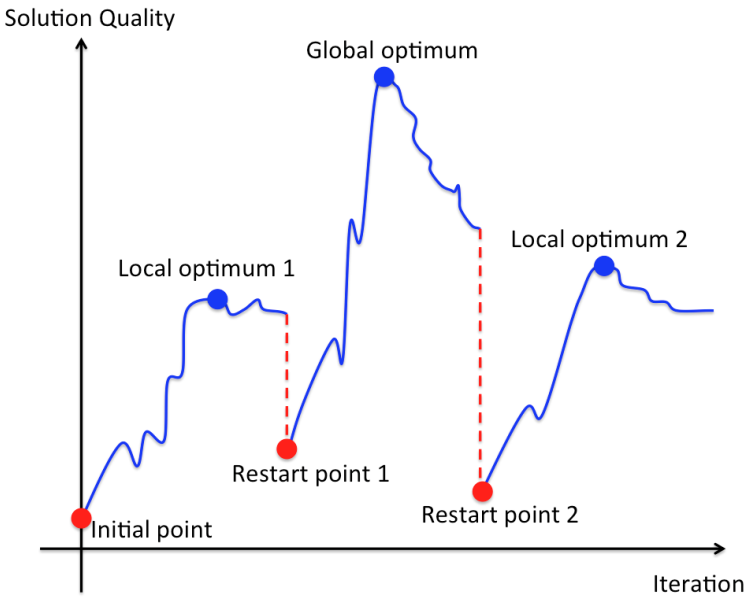

Figure 2.5: Iterated Local Search

Tabu Search

Tabu Search (TS) [42] is a metaheuristic offering a diversified search in each of its search steps. It is very popular and effective. TS tries

to prevent the search from visiting the same points in the search space. There are two interesting features of tTS: first, the definition of legal moves imposes no constraint on the objective value and allows LS to select moves degrading the quality of the current solution, thus escaping local minima; second, the greedy nature of TS ensures that the objective function does not degrade too much at any step [103].

TS can not keep track of all visited solutions (lacks of memory and costly time computation for checking revisiting), so TS uses a short-term memory to prevent the search from returning the recently visited solutions. A popular technique is the transition abstraction that stores the transitions, not the states (solutions) when the difference between states is only a few variables.

Since TS stores a sequence of abstractions and not a sequence of states, it may forbid transition to a solution that has not been visited before and may be quite desirable. To overcome this limitation, TS often features an *aspiration* criterion that specifies when the tabu status may be overridden [103].

Other metaheuristics are often used for solving the combinatorial optimization problems such as Genetic Algorithms, Variable Neighborhood Search, Simulated Annealing or Ant Colony Optimization.

3

QUORUMCAST ROUTING PROBLEM

In this chapter, we solve QRP as different IPPs and MIPPs by B&C algorithms. The experimental results show that these B&C algorithms outperform the-state-of-the-art algorithm that solved this problem as a CPP.

Moreover, in this chapter, a sensitivity analysis is also performed on values of m (the number of multicast nodes in the given graph) and q (the at least number of multicast nodes in the solution).

3.1 Related works

QRP is NP-hard, as it reduces to the Steiner tree problem [41] when $q = |S|$. QRP appears in many distributed applications, for example, distributed synchronization and updating a replicated resource (see [23] for more details).

For solving QRP, various incomplete approaches to computing an approximation of the optimal solution have been proposed in [23, 34, 105, 85], in which the CBLS algorithm in [85] is currently the-state-of-the-art incomplete algorithm. In addition, two exact algorithms in [67, 85] have been proposed for solving this problem to optimality. In [67], a partial solution is defined to be a set of sub-trees that spans the root and

some multicast nodes; a partial solution is extended by adding one edge at each step until a feasible solution is constructed; a *Confined Area Pruning* scheme was introduced that allows reducing that search space. CP approach in [85] is currently the-state-of-the-art exact algorithm. In this chapter, we will compare our approach with this CP approach, where the problem is modeled as follows [85].

Two variables are defined for each node $v \in V_D$,

- $x(v)$ represents the successor of v on the unique path from v to the root r on the solution tree T; the domain of $x(v)$ is denoted by $D(x(v))$ and is defined to be the set of adjacent nodes of v on graph D plus $\perp$: $D(x(v)) = \{u \in V_D | (u, v) \in \delta_D(v)\} \cup \{\perp\}$. If v is not in T, then $x(v) = \perp$. Moreover, for the root r, we have $x(r) = r$.

- $y(v)$ represents the length of the path (number of arcs) from v to r on T. It is undefined if $v \notin T$.

$$minimize \sum_{v \in V \setminus \{r\}} c_D(v, x(v)) \tag{3.1a}$$

$$x(v) \neq \perp \Rightarrow y(v) = y(x(v)) + 1, \forall v \in V_D \setminus \{r\} \tag{3.1b}$$

$$x(u) = \perp \Rightarrow x(v) \neq u, \forall u, v \in V_D \setminus \{r\} \tag{3.1c}$$

$$\sum_{v \in S} (x(v) \neq \perp) \geq q \tag{3.1d}$$

$$y(r) = 0 \tag{3.1e}$$

$$x(r) = r \tag{3.1f}$$

In the objective function we assume that $c_D(v, \perp) = 0, \forall v \in V_D$. The constraint (3.1b) plays the role of eliminating cycles. Constraint (3.1c) specifies that if a node u is not included in the solution, then it cannot be a successor of any other node v. The constraint (3.1d) states that the number of multicast nodes must be at least q (called the quorum constraint). By convention, constraints (3.1d) and (3.1f) impose the successor of the root and the length of the path from the root to itself.

3.2 Mathematical models

In this section, we propose four mathematical models for QRP. One is proposed directly on the undirected graph $D = (V_D, E_D, c_D)$, and the others are proposed on the corresponding directed graph of $G = (V_G, A_G, c_G)$ that is formed by replacing each edge of D by two opposite arcs with the same cost as the original edge.

These models can exploit the properties of QRP solutions. Let T be a solution of an instance of QRP with r, q, S on a graph D. One can easily show that

1. all leaf nodes of T are multicast nodes [67];

2. T spans exactly q multicast nodes.

3.2.1 Edge-based formulation: Model 1

In this section, we propose a formulation on the undirected graph $D = (V_D, E_D, c_D)$, called "edge-based formulation". Many problems have been modeled by similar formulations [6, 71, 49, 40, 69, 5]. This model introduces binary variables y_i stating whether node i is in T and the binary variables x_e stating whether edge $e \in E$ is in the solution tree T.

In this model (Figure 4.1), the constraints (3.2b) are connectivity constraints (or SECs). The constraints (3.2c) and (3.2d) ensure that if $v \in T$, then $y_v = 1$, and if $v \notin T$, then $y_v = 0$. The constraint (3.2e) presents a basic property of a tree that requires the relation between the number of nodes and the number of edges. The constraints (3.2g) ensure that T includes exactly q multicast nodes (Property (2)). Finally, the constraint (3.2f) ensures that the root r is always in T. Notice that Property (1), stating that all leaf nodes are multicast nodes, could also be included, but experimental results have shown that it is useless here as well as in all subsequent models. It is therefore not considered.

In this model, there are $|E_D| + |V_D|$ variables and an exponential number of constraints in the number of nodes.

$$Minimize \sum_{e \in E} c_D(e) x_e \tag{3.2a}$$

$$\sum_{e \in \delta_D(W)} x_e \leq |W| - 1, \forall W \subset V_D, 2 \leq |W| \leq |V_D| - 1 \tag{3.2b}$$

$$\sum_{e \in \delta_D(i)} x_e \geq y_i, \forall i \in V_D \tag{3.2c}$$

$$\sum_{e \in \delta_D(i)} x_e \leq (|V_G| - 1) y_i, \forall i \in V_D \tag{3.2d}$$

$$1 + \sum_{e \in E_D} x_e = \sum_{v \in V_D} y_v \tag{3.2e}$$

$$y_r = 1 \tag{3.2f}$$

$$\sum_{v \in S} y_v = q \tag{3.2g}$$

$$x_e \in \{0, 1\}, \forall e \in E_D \tag{3.2h}$$

$$y_i \in \{0, 1\}, \forall i \in V_D \tag{3.2i}$$

Figure 3.1: Edge-based formulation

$$Minimize \sum_{(i,j)\in A_G} c_G(i,j)x_{ij} \tag{3.3a}$$

$$\sum_{(r,i)\in A_G} (y^k_{ri} - y^k_{ir}) \leq 1, \forall k \in V_G \tag{3.3b}$$

$$\sum_{k\in S, k\neq r, (r,i)\in A_G} (y^k_{ri} - y^k_{ir}) = q - 1 \tag{3.3c}$$

$$\sum_{(k,i)\in A_G} (y^k_{ki} - y^k_{ik}) \geq -1, \forall k \in V_G \tag{3.3d}$$

$$\sum_{k\in S, k\neq r, (k,i)\in A_G} (y^k_{ki} - y^k_{ik}) = -(q - 1) \tag{3.3e}$$

$$\sum_{(j,i)\in A_G} (y^k_{ij} - y^k_{ji}) = 0, \forall k \in V_G, i \in V_G \setminus \{k, r\} \tag{3.3f}$$

$$y^k_{ij} \leq x_{ij}, \forall (i,j) \in A_G, \forall k \in V_G \cup \{r\} \tag{3.3g}$$

$$y^k_{ij} \geq 0, \forall (i,j) \in A_G, \forall k \in V_G \cup \{r\} \tag{3.3h}$$

$$x_{ij} \in \{0,1\}, \forall (i,j) \in V_G \tag{3.3i}$$

Figure 3.2: Multi-commodity flows formulation

3.2.2 Multi-commodity flows formulation: Model 2

In this section, we propose a multi-commodity flow formulation on the corresponding directed graph $G = (V_G, A_G, c_G)$. In the literature, many problems have been modeled using multi-commodity flows [49, 30, 25, 54]. However, this problem is slightly more complex, as we do not know which multicast nodes are spanned.

This model introduces variables $y^k_{ij} \in \mathbb{R}^+$ measuring the flow, through arc $(i,j) \in A_G$, from the root node r to a node $k \in V_G \setminus \{r\}$ and the binary variables x_{ij} stating whether arc (i,j) is in the solution tree T.

In this model (Figure 4.2), the constraints (3.3g) ensure that a flow

$$Minimize \sum_{(i,j)\in A_G} c_G(i,j)x_{ij} \tag{3.4a}$$

$$x(\delta_G^-(r)) = 0 \tag{3.4b}$$

$$x(\delta_G^+(r)) \geq 1 \tag{3.4c}$$

$$x(\delta_G^+(W)) \geq x(\delta_G^+(v)), \forall W \subset V_G, 2 \leq |W| \leq |V_G|-1, \forall v \in W, r \notin W \tag{3.4d}$$

$$x(\delta_G^-(j)) \leq 1, \forall j \in V_G \tag{3.4e}$$

$$x(\delta_G^-(S \setminus \{r\})) = q - 1 \tag{3.4f}$$

$$x_{ij} \in \{0,1\}, \forall (i,j) \in A_G \tag{3.4g}$$

Figure 3.3: Arborescence tree formulation

is sent along an arc only if the arc is traversed. The constraints (3.3c), (3.3e) and (3.3f) ensure that there exists a flow from the root r to q nodes in the set of multicast nodes S (note that we assumed that r is a multicast node). The constraints (3.3b), (3.3d) and (3.3f) are flow conserving constraints.

In this model, there are $|A_G| \times |S|$ variables and a polynomial number of constraints.

3.2.3 Arborescence tree formulation: Model 3

In this model, we propose a formulation, called "arborescence tree formulation", on the corresponding directed graph $G = (V_G, A_G, c_G)$. In the literature, many similar formulations have been proposed for problems related to finding a spanning tree [43, 66, 69, 85].

This model introduces variables the binary variables x_{ij} stating whether arc (i, j) is in the solution tree T.

In this model (Figure 4.3), the constraints (3.4b) indicate that there are no arcs arriving at r. The constraint (3.4c) ensures that there exists at least one arc leaving r. The constraints (3.4d) are connectivity constraints (see [33, 24, 62, 66] for more details about the connectivity constraints). The constraint (3.4f) ensures that the optimal tree T includes exactly q multicast nodes.

In this model, there are $|A_G|$ variables corresponding to the number of arcs in the corresponding directed graph G, and an exponential number of constraints in (3.4d).

3.2.4 Miller–Tucker–Zemlin formulation: Model 4

In this model, we propose a formulation using Miller–Tucker–Zemlin constraints as connectivity constraints on the corresponding directed graph $G = (V_G, A_G, c_G)$ [75, 64].

This model introduces variables t_i constrained by $t_i \leq t_j$ if arc $(i, j) \in T$ (these constraints prevent subtours in the solution); the variables p_i state whether or not node i is in T; and the binary variables x_{ij} stating whether arc (i, j) is in the solution tree T.

In this model (Figure 3.4), the constraints (3.5g) present the relative position of the nodes in the tree. They state that if $x_{ij} = 1$, then $t_i < t_j$. This prevents the solution from containing subtours. The constraints (3.5d) ensure that the root node r must connect to other nodes in the arborescence tree.

In this model, there are $(|A_G| + 2 \times |V_G|)$ variables and a polynomial number of constraints.

3.3 Solving QRP as mixed integer programs

In this section, we propose four different approaches, based on the above models, to solve QRP. Model 2 and Model 4 have a polynomial numbers of variables and constraints. They can be directly used in a MIP solver (CPLEX). These approaches will be denoted *Mod2_B&B* and *Mod4_B&B*. Model 1 and Model 3 have an exponential number of constraints. The constraints are relaxed, and B&C algorithms are employed.

$$minimize \sum_{(i,j)\in A_G} c_G(i,j)x_{ij} \tag{3.5a}$$

$$p_r = 1 \tag{3.5b}$$

$$x(\delta_G^-(r)) = 0 \tag{3.5c}$$

$$x(\delta_G^+(r)) \geq 1 \tag{3.5d}$$

$$x(\delta_G^-(v)) = p_v, \forall v \in V_G \setminus \{r\} \tag{3.5e}$$

$$x_{ij} \leq p_i, \forall (i,j) \in A_G \tag{3.5f}$$

$$|V|x_{ij} + t_i + 1 \leq t_j + |V_G|, \forall (i,j) \in A_G \tag{3.5g}$$

$$1 + x(A_G) = \sum_{v\in V_G} p_v \tag{3.5h}$$

$$\sum_{i\in S} p_i = q \tag{3.5i}$$

$$x_{ij} \in \{0,1\}, \forall (i,j) \in A_G \tag{3.5j}$$

$$p_i \in \{0,1\}, \forall i \in V_G \tag{3.5k}$$

$$t_i \in \{1 \ldots |V|\}, \forall i \in V_G \tag{3.5l}$$

Figure 3.4: Miller–Tucker–Zemlin formulation

3.3.1 Lazy constraint approach

This approach is applicable to Model 1 and Model 3, where connectivity constraints (3.2b) and (3.4d) are considered as lazy constraints. The linear programming relaxation of the initial model without lazy constraints is solved. All isolated components are identified for every feasible integral solution that is not yet feasible. If a solution to a linear programming relaxation is feasible, then there is no isolated component. To check for isolated components, we use the union-find data structure [28, 60]. If there is only one component, then the solution is feasible. Otherwise, there is a cycle in some components, a lazy constraint is then added for each component as follows. For Model 3, C is the set of all nodes of the component and i is a random node in C; for Model 1, C is the set of all nodes of the component. All these lazy constraints are then added directly to the model, and the linear programming relaxation of the current model is reoptimized. This procedure is repeatedly executed until an optimal solution has been found. The two corresponding approaches will be denoted *Mod1_B&C_lazy* and *Mod3_B&C_lazy*.

3.3.2 Dynamic constraint separation approach

This approach can be applied to Model 3, where connectivity constraints (3.4d) are dynamically separated [33]. This approach, denoted by *Mod3_B&C_dyn*, finds violated connectivity constraints on the support graph. Given a solution x^* to a LPRP (containing all connectivity constraints separated so far), the support graph D^* of x^* has all nodes V_{D^*} and edges $\{(i,j) \in E_{D^*} : c_{D^*}(i,j) = x^*_{ij} + x^*_{ji} > 0\}$[7].

This approach consists of two stages. First, the support graph is checked for isolated components not connected to the root node. For each isolated component, only constraint (3.4d) is added to the current model, where C is the set of all nodes in the isolated component and i is a random node in C. Second, if the support graph has only one component that includes the root node r, a maximum $r-v-$flow/minimum $r-v-$cut problem is solved for each node $v \neq r$ in the component. To solve maximum-flow/minimum-cut problems, we use code written by Skorobohatyj [95]. A maximum flow that is less than the absolute inflow to v indicates a violated connectivity constraint (3.4d), in which

C are all nodes on the same side of the $r - v$ cut as v (of course node i in the constraint is the node v).

3.3.3 Preprocessing

Different reduction checks have been proposed for the minimum Steiner tree problem and others [66, 62, 68, 102, 66, 29]. In the preprocessing of QRP, the following *check for useless nodes* is useful. It is performed on the undirected graph D. If a node is not a multicast node and its degree is only one, then this node (and its edge) can be removed from the graph D. If a node v is not a multicast node with exactly two neighbors u and w, then the node v and the edges (u, v) and (v, w) can be removed. If there exists an edge (u, w) with cost $c_D(u, w)$, this cost is updated to $min(c_D(u, w), c_D(u, v)+c_D(v, w))$. Otherwise, an edge (u, w) is added with cost $c_D(u, v) + c_D(v, w)$. These checks can be applied iteratively until the graph remains unchanged. In practice, we limit ourselves to three iterations. Other reductions were considered, but they had only very marginal impact. This preprocessing is carried out in all algorithms in this paper.

3.4 Computational experiments

In [85], CP approach was tested on 960 random instances with the largest graph having 60 nodes. It was shown that the CP approach was better than the existing state-of-the-art complete approaches. We reuse these instances. We collect these instances into a class called **C1**.

We also collect 2500 instances in a class **C2**, generated from 100 undirected graphs of 160 nodes and 25 couples $\langle q, |S|\rangle$, ranging from $\{\langle 3, 20\rangle$ to $\langle 119, 140\rangle\ \}$. These 100 undirected graphs were extracted from 100 minimum Steiner tree instances of test set **I160** in the library SteinLib [63]. The multicast nodes were randomly chosen.

All B&C algorithms were implemented in C++, using IBM Ilog Cplex Concert Technology, version 12.4. The standard Cplex cuts were automatically added and the default setting of Cplex is used. The CP approach in [85] was implemented in Comet [1]. Finally, all experiments were performed on XEN virtual machines with 1 core of a CPU

Class	Approach	$\%opt$	$\overline{I}$	$\overline{N}$	$\overline{C}$	$\overline{T}$
C1	*CP*	97.1	na.	na.	na.	80.47
	Mod1_B&C_lazy	100	441.7	50.43	6.58	0.57
	Mod2_B&B	100	2159	1.38	na.	1.06
	Mod3_B&C_lazy	100	398.8	77.51	81.42	0.35
	Mod3_B&C_dyn	100	308.2	3.69	158.1	1.94
	Mod4_B&B	100	506.2	55.9	na.	0.64
C2	*CP*	0.08	na.	na.	na.	9.74
	Mod1_B&C_lazy	78.6	92804	8507	1110	150.9
	Mod2_B&B	60.2	66716	7.05	na.	318.6
	Mod3_B&C_lazy	94.4	30938	2312	1077	97.6
	Mod3_B&C_dyn	77.2	8408	54.64	6089	153.0
	Mod4_B&B	95.2	50327	6138	na.	92.8

Figure 3.5: A summary of computational results for two classes of instances

Intel Core2 Quad Q6600 @2.40GHz and 1GB of RAM. The time limit for each execution of an algorithm was 30 minutes.

3.4.1 Comparing the approaches

We first compare the MIP approaches as well as the CP approach from [85]. Figure 3.5 gives a summary of the experimental results. The columns have the following meanings: $\%opt$ is the percentage of instances solved to optimality within the time limit of 30 minutes; $\overline{I}$ is the average number of iterations; $\overline{N}$ is the average of the number of nodes in the search tree; $\overline{C}$ is the average of the number of separated constraints; $\overline{T}$ is the average computational time in seconds (on the solved instances). Figure 3.6 shows the evolution of the percentage of solved instances in **C2** with respect to the time limit.

It is clear that all the MIP approaches significantly outperform the CP approach. In the class **C1**, the MIP approaches are two orders of magnitude faster than CP. In the class **C2**, CP only solved two instances out of 2500, while *Mod4_B&B* solved 95.2% of the instances. In Figures 3.5 and 3.6, there is no major difference between *Mod3_B&C_lazy* and *Mod4_B&B*, nor between *Mod1_B&C_lazy* and *Mod3_B&C_dyn*.

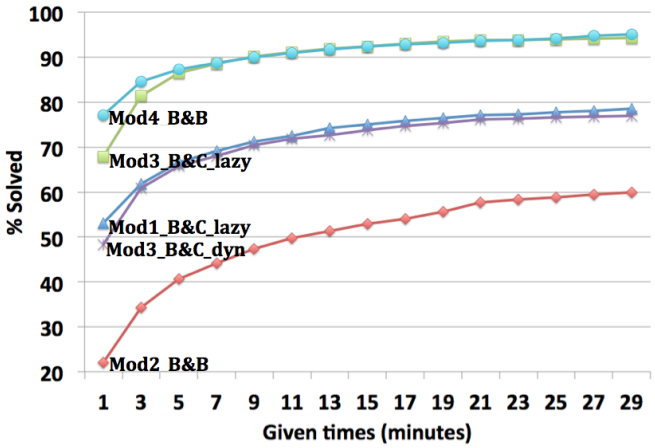

Figure 3.6: Percentage of solved instances in **C2** in given times

It is clear that *Mod3_B&C_lazy* and *Mod4_B&B* are the best two approaches, both in the percentage of solved instances and in the execution time. Normally, the use of Miller-Tucker-Zemlin subtour constraints leads to a large amount of computation time and a large search tree; however for this problem it works well. This may be explained by the small-size solution tree and the fact that the path from the root to each node in the solution tree is quite short in term of the number of nodes in the path. The approach *Mod2_B&B* is the worst among the MIP approaches, although it develops few nodes.The low number of nodes results from the fact that the integer relaxation version of this model is quite close to the optimal solution [54]. It is worth noting that *Mod3_B&C_dyn* has the smallest number of iterations. This mainly comes from the dynamic constraint separation approach, which produces smaller and thinner B&B trees. However, the number of added constraints is much larger.

Although not reported here for reasons of space, the experimental results also showed that Property (2) of the QRP solutions (Section 3.2) is very useful in all the models proposed in this chapter. For example, this property helps this model to solve 6.2% more instances of class **C2**.

3.4.2 Effect of the values of q and $|S|$ on the performance of the approaches

We analyze the sensitivity of the performance to the value of q and the size of the multicast node set. We divided the instances of class **C2** into two sets of groups. In the first set, a group contains instances of the same size of multicast node set. The groups *G20* (resp. *G50, G80, G110* and *G140*) consist of all instances with $|S| = 20$ (resp. $50, 80, 110$ and 140). In the second set, a group contains 500 instances with similar $\frac{q}{|S|}$. The groups *G1* through *G5* split the instances from the smallest to the largest values of $\frac{q}{|S|}$.

The experimental results for each group are given in Figure 3.7. First, the different approaches have differing sensitivities to q. The instances in group *G5* are more difficult to solve than those in the other groups, except for the algorithm *Mod2_B&B*. When considering the ratio $\frac{q}{|S|}$, *Mod3_B&_lazy* and *Mod3_B&_dyn* are better for high value of this ratio, while the other approaches are worse. These results also confirm that *Mod3_B&C_lazy* and *Mod4_B&B* are the two best approaches. However, there is a significant difference between these two approaches when the number of multicast nodes varies. A portfolio approach could be used to select *Mod4_B&B* for low values of $|S|$, and *Mod3_B&C_lazy* for high values.

3.5 Conclusion

This chapter solved the quorumcast routing problem to optimality as a mixed integer program. In this chapter, we proposed four mathematical formulations for QRP. We then solved QRP to optimality as a mixed integer program, introducing two constraint relaxations. The computational results showed that the MIP approaches are much more efficient than the state of the art approach (which is based on Constraint Programming). In addition, we showed that two of the approaches, *Mod3_B&C_lazy* and *Mod4_B&B*, are the two best ones. Finally, experimental results pointed out that the different approaches have different sensitivities to the parameters q and the size of the multicast node set. As future research, new separation constraints could be investigated.

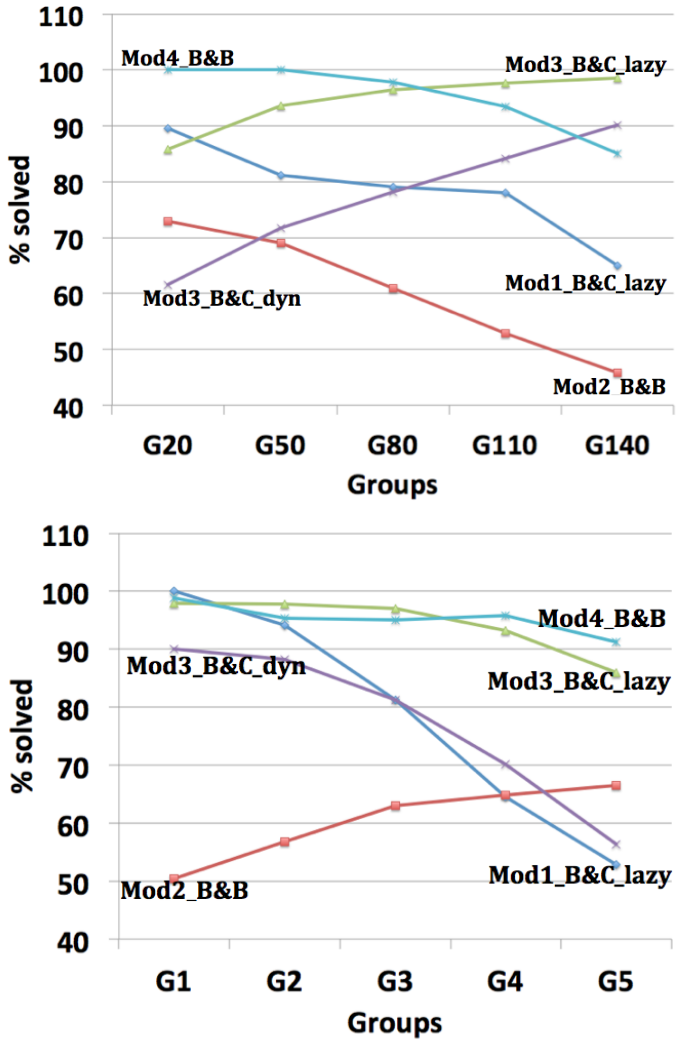

Figure 3.7: Comparing MIP approaches in solving groups of instances with respect to the percentage of solved instances

4

ELEMENTARY SHORTEST AND LONGEST PATH PROBLEMS

This chapter deals with ESPP and ELPP. These problems are NP-hard. We propose complete algorithms based on MIP for their solution, employing different valid inequalities. Moreover, we propose decomposition techniques which are very efficient for cases with special structure. Experimental results show the efficiency of our algorithms compared with the-state-of-the-art-exact algorithms.

4.1 Equivalence between the problems

Different cases of these problems depending on the cardinality of S and T have been considered in the literature. For instance, when $S = \{s\}$ and $T = \{t\}$ with $s \neq t$, ESPP is the elementary shortest path problem on a digraph from a specified source node to a specified destination node [54, 33]; when $S = T = \{d\}$ (d is a node in G), ESPP becomes the profitable tour problem [31, 14, 44, 77]; when $S = \{s\}$ and $T = \{t\}$ with $s \neq t$, ELPP is the elementary longest path problem on a directed graph from a specified source node to a specified destination node [84]; and when $S = T = V_G$, ELPP becomes the longest elementary path problem [106, 61, 86, 84].

All these problems are NP-hard. Note that when G does not con-

tain negative-cost cycles (resp. positive-cost cycles), then ESPP (resp. ELPP) is polynomially solvable, for example, by the well-known Bellman–Ford algorithm.

We show in this section some equivalences between these problems in the sense that the problems can be polynomially transformed into each other.

4.1.1 Equivalence between ESPP and ELPP

Suppose given a graph $G = (V_G, A_G, c_G)$, $S, T \subseteq V_G$. Construct the graph $G' = (V_{G'}, A_{G'}, c_{G'})$ where $V_{G'} = V_G$, $A_{G'} = A_G$, $c'_G(i,j) = -c_G(i,j), \forall (i,j) \in A_{G'}$. Clearly, if p is an optimal solution to ESPP(S, T, G), then p is also an optimal solution to ELPP(S, T, G'), and vice versa.

4.1.2 Equivalence between ESPP(S, T, G) and ESPP(s, t, G)

Given a graph $G = (V_G, A_G, c_G)$, and $S, T \subseteq V_G$, construct the graph $G' = (V_{G'}, A_{G'}, c_{G'})$ where $V_{G'} = V_G \cup \{s, t\}$ ($s, t \notin V_G$ are artificial nodes), $A_{G'} = A_G \cup \{(s,v) \mid v \in S\} \cup \{(v,t) \mid v \in T\}$, $c_{G'}(u,v) = c_G(u,v), \forall (u,v) \in A_G$ and $c_{G'}(s,v) = 0, \forall v \in S, c_{G'}(v,t) = 0, \forall v \in T$. It is evident that if $p = \langle s, v_1, \ldots, v_k, t \rangle$ is an optimal solution to ESPP(s, t, G'), then $p' = \langle v_1, \ldots, v_k \rangle$ is an optimal solution to ESPP(S, T, G), and vice versa.

4.1.3 Equivalence between ESPP(S, T, G) and PTP(d, G)

When $S = T = \{d\}$ for d a node in G, ESPP(S, T, G) becomes PTP(d, G).

Given a graph $G = (V_G, A_G, c_G)$, construct the graph $G' = (V_{G'}, A_{G'}, c_{G'})$ where $V_{G'} = V_G \cup \{d\}$ ($d \notin V_G$ is an artificial node), $A_{G'} = A_G \cup \{(d,v) \mid v \in S\} \cup \{(v,d) \mid v \in T\}$, $c_{G'}(u,v) = c_G(u,v), \forall (u,v) \in A_G$ and $c_{G'}(d,v) = 0, \forall v \in S, c_{G'}(v,d) = 0, \forall v \in T$. It is evident that if the tour $\mathcal{T} = \langle d, v_1, \ldots, v_k, d \rangle$ is an optimal solution to PTP(d, G), then $p' = \langle v_1, \ldots, v_k \rangle$ is an optimal solution to ESPP(S, T, G), and vice versa.

4.2 Related works

ESPP(s, t, G) is a special case of a well-studied problem: the elementary shortest path problem with resource constraints (ESPPRC), in which all resource constraints are removed. ESPPRC appears as a sub-problem in column-generation solution approaches for solving vehicle-routing problems (VRP) [100]. Dror in [70] proved that the ESPPRC is NP-hard. There are well-known labelling algorithms for solving the ESPPRC, based on dynamic programming, for example, [96]. Several techniques have been proposed for improving these labelling algorithms, such as dominance rules to prevent the search from considering paths that are shorter than the others already found [37], state space relaxation [15, 91], and bounding [12]. In order to use labelling algorithms to solve ESPP(s, t, G), Drexl and Irnich [33] added a virtual resource as a limitation on the number of visited nodes, but this addition could not prevent the labelling algorithm from considering all subsets of nodes. In their chapter, the authors stated that dynamic programming exact algorithms for ESPP(s, t, G) are not appropriate for complete graphs containing more than 20 nodes. This might be the case for problems without additional constraints. Using the bi-directional method of Gighini and Salani in [90], it can solve slightly larger cases. In short, labelling algorithms are not very good at solving ESPP(s, t, G).

Ibrahim et al. in [54] presented two mixed integer programming formulations for ESPP(s, t, G) and compared them in terms of the strength of their respective linear relaxations. These formulations have been tested on small graphs (containing no more than 25 nodes). The results obtained show that the commodity-flow formulation is stronger than the arc-flow formulation. Most recently, in [33], Drexl et al. have compared the integer versions of the formulations in terms of their computation times, memory requirements, and have assessed the quality of the lower bounds provided by an integer relaxation of the commodity-flow formulation. Some constraint relaxation techniques have been proposed for solving ESPP(s, t, G) to optimality. The approach of dynamic separation of sub-tour elimination constraints (SECs) applied to the arc-flow formulation outperforms the others[1]. A formulation applying tra-

[1]This arc-flow formulation is presented in Section 4.3.1.

ditional dynamic SEC separation schemata in [33] will be presented in detail later in this chapter.

PTP(d, G) was first presented by Dell'Amico et al. [31], as a variant of the STSP with profits. In the STSP with profits, each node is associated with a certain quantity of profit, and the overall goal is to find a sub-tour that maximizes the profits collected from the visited nodes but that simultaneously minimizes the travel costs. The STSP with profits becomes PTP(d, G) when the two objectives are combined into one (minimizing the travel costs less the profits). Volgenan and Jonker in [104] showed that PTP(d, G) can be polynomially reduced to the much more studied NP-complete ATSP. When the cost matrix is symmetric and satisfies the triangle inequality, three approximation algorithms were developed in [14, 44, 77] for PTP(d, G) with approximation factors of $\frac{5}{2}, 2 - \frac{1}{n-1}$ and $1 + log(n)$ (n is the number of nodes). In the [36], Feillet et al. reviewed other heuristic and exact approaches for solving the STSP with profits.

Wong et al. [106] mentioned ELPP(G) on graphs in the context of peer-to-peer information retrieval networks where weights are associated with nodes. They also proposed a genetic algorithm for solving this problem. ELPP(G) has also been addressed in [94] for evaluating the worst-case packet delay of Switched Ethernet. The given Switched Ethernet is transformed into a delay computation model represented by a directed graph. On this particular directed acyclic graph, the longest path can be computed using dynamic programming. ELPP(G) also appears in the domain of high-performance printed circuit board design in which one needs to find the longest path between two specified nodes on a rectangle grid routing [101]. In [87], ELPP(G) was described in the context of multi-robot patrolling and [86] proposed a genetic algorithm for solving ELPP(G).

In [61], approximating algorithms for ELPP(G) (unweighted graph) were considered, and it was shown that no polynomial-time algorithm can find a constant factor approximation for the longest path problem unless P=NP. In [46], a heuristic algorithm was proposed for ELPP(G), based on the strongly connected components of the graph.

In our previous work in [84], we considered ELPP(G) on arbitrary undirected graphs and proposed two constraint-based techniques for its solution: an exact algorithm based on constraint programming (CP) and

a constraint-based local search algorithm. To solve ELPP(G) on sparse undirected graphs with many bridges, we also proposed an efficient algorithm combining constraint-based techniques and dynamic programming.

4.3 MIP formulations for ESPP(s, t, G)

Given a directed graph $G = (V_G, A_G, c_G)$ with set of nodes V_G, set of arcs A_G, each arc $(i, j) \in A_G$ being associated with a cost $c_G(i, j) \in \mathbb{R}$, suppose that $s, t \in V_G$ are the source node and the destination node. An elementary path from s to t in G is a sequence of nodes $p = \langle v_1, \ldots v_n \rangle$, in which $v_1 = s$, $v_n = t$, $(v_k, v_{k+1}) \in A_G, \forall k \in \{1, \ldots, n-1\}$, $v_i \neq v_j, \forall i, j \in \{1, \ldots, n\}, i \neq j$, and the cost of the path is $c_G(p) = \sum_{i=1}^{n-1} c_G(v_i, v_{i+1})$.

4.3.1 Arc-flow formulation: Model 1

This formulation defines only one type of binary variable x_{ij}, which represents whether or not the arc (i, j) is included in the solution. [54, 58, 56, 33].

In this formulation (Figure 4.1), (4.1a), (4.1b), (4.1d) and (4.1e) represent, respectively, the objective function, the constraints of flow conservation, the SECs, and the integer constraints. In this formulation, there are $|V_G|^2$ variables. However, (4.1d) has an exponential number of constraints.

4.3.2 Commodity-flow formulation: Model 2

The following formulation has been studied in [54, 33] for ESPP(s, t, G). This formulation has three types of variables: binary variables x_{ij} equal to 1 iff arc (i, j) is used in the shortest path; binary variables y_i equal to 1 iff node i is traversed by the shortest path; and binary variables z_k equal to 1 iff a flow from the source node s travels to node $k \in V_G \setminus \{s\}$ using the arc (i, j). The following formulation (Figure 4.2) is presented in [33].

$$Minimize \sum_{(i,j)\in A_G} c_G(i,j)x_{ij} \tag{4.1a}$$

subject to

$$x(\delta_G^+(i)) - x(\delta_G^-(i)) = \begin{cases} 1, & i = s \\ -1, & i = t \\ 0, & i \in V_G \setminus \{s,t\} \end{cases} \tag{4.1b}$$

$$x(\delta_G^-(s)) + x(\delta_G^+(t)) = 0 \tag{4.1c}$$

$$x(\delta_G^+(S)) - x(\delta_G^+(i)) \geq 0, \forall S \subseteq V_G, |S| \geq 2, t \notin S, \forall i \in S \tag{4.1d}$$

$$x_{ij} \in \{0,1\}, \forall (i,j) \in A_G \tag{4.1e}$$

Figure 4.1: Arc-flow formulation

The justification of the constraints in this formulation is given in [54, 33]. This formulation contains $O(|V_G|^3)$ variables and constraints. A weakness of this formulation is the large number of variables.

4.3.3 Miller–Tucker–Zemlin formulation: Model 3

This formulation uses another type of SEC, introduced in [75, 64] introduced for the STSP. The SECs are in (4.3d), in which the constant number M is large enough (e.g., $M = |V_G|$), the binary variables x_{ij} represent whether the arc (i,j) is visited, and each integer variable p_i denotes the relative position of the visited node. In this formulation (Figure 4.3), the constraints (4.3d), the SECs, state that $p_i < p_j$ if $x_{ij} = 1$.

In [33], Model 2 was shown to be less efficient than Model 1 when solving the ESPP(s,t,G). In addition to the variables of Model 1, Model 3 has $|V_G|$ more variables ($p_i, i \in A_G$). In this chapter, we investigate Model 1, exploiting various valid inequalities that have not been considered so far and a new separation strategy. We will show that our proposed algorithm is better than the state of the art algorithm of [33] in

$$Minimize \sum_{(i,j)\in A_G} c_G(i,j)x_{ij} \tag{4.2a}$$

subject to

$$z_{ij}^k \leq x_{ij}, \forall k \in V_G \setminus \{s,t\},\ (i,j) \in A_G, i \neq k, s \neq j, j \neq t \tag{4.2b}$$

$$\sum_{i\in V_G^+(s)} z_{si}^k = y_k, \forall k \in V_G \setminus \{s,t\} \tag{4.2c}$$

$$\sum_{j\in V_G^+(v)} z_{vj}^k - \sum_{i\in V_G^-(v)} z_{iv}^k = 0, \forall k \in V_G \setminus \{s,t\}, v \in V_G \setminus \{s,k,t\} \tag{4.2d}$$

$$\sum_{i\in V_G^-(k)} z_{ik}^k = y_k, \forall k \in V_G \setminus \{s,t\} \tag{4.2e}$$

$$x(\delta_G^+(i)) = y_i, \forall i \in V_G \setminus \{t\} \tag{4.2f}$$

$$x(\delta_G^-(i)) = y_i, \forall i \in V_G \setminus \{s\} \tag{4.2g}$$

$$x(\delta_G^+(s)) = 1 \tag{4.2h}$$

$$x(\delta_G^-(t)) = 1 \tag{4.2i}$$

$$x(\delta_G^-(s)) + x(\delta_G^+(t)) = 0 \tag{4.2j}$$

$$x_{ij} \in \{0,1\}, \forall (i,j) \in A_G \tag{4.2k}$$

$$y_i \in \{0,1\}, \forall i \in V_G \tag{4.2l}$$

$$z_{ij}^k \geq 0, \forall k \in V_G \setminus \{s,t\}, (i,j) \in A_G, i \neq k, s \neq j, j \neq t \tag{4.2m}$$

Figure 4.2: Commodity-flow formulation

$$minimize \sum_{(i,j)\in A_G} c_G(i,j)x_{ij} \tag{4.3a}$$

subject to

$$x(\delta_G^+(i)) - x(\delta_G^-(i)) = \begin{cases} 1, & i = s \\ -1, & i = t \\ 0, & i \in V_G \setminus \{s,t\} \end{cases} \tag{4.3b}$$

$$x(\delta_G^-(s)) + x(\delta_G^+(t)) = 0 \tag{4.3c}$$

$$p_i + Mx_{ij} + 1 \leq p_j + M, \forall (i,j) \in A_G \tag{4.3d}$$

$$x_{ij} \in \{0,1\}, \forall (i,j) \in A_G \tag{4.3e}$$

$$p_i \in \{1,\ldots,|V_G|\}, \forall i \in V_G \tag{4.3f}$$

Figure 4.3: Miller–Tucker–Zemlin formulation

most cases. Although Model 3 has a polynomial number of constraints, we show, in the experimental section, that that approach performs very poorly.

4.4 Valid inequalities for ESPP(s, t, G)

In [104], Volgenant and Jonker showed that PTP(d, G) can be polynomially reduced to ATSP. Based on this, we will show that ESPP(s, t, G) can be polynomially reduced to ATSP. Hence, some valid inequalities of ATSP can be applied to ESPP(s, t, G).

First of all, we construct a new directed graph $G' = (V_{G'}, A_{G'}, c_{G'})$ (called the transformed graph) from the directed graph $G = (V_G, A_G, c_G)$ (called the original graph) with $|V_G| - 1$ new nodes. That is, assuming that $V_G = \{s, v_1, \ldots v_n, t\}$, we put $G' = (V_{G'}, A_{G'}, c_{G'})$, with $V_{G'} = V_G \cup NV$ and $NV = \{v'_1, \ldots, v'_n, v'_{n+1}\}$. We also put $A_{G'} = A_G \cup NA$, where NA consists of $3n-1$ arcs: $(v'_1, v'_2), (v'_1, v_1), (v_1, v'_2), \ldots, (v'_{n-1}, v'_n)$, (v'_{n-1}, v_{n-1}), (v_{n-1}, v'_n) and $(t, v'_1), (v'_{n+1}, s)$. In addition, $c_{G'}(i,j) = c_G(i,j)\ \forall (i,j) \in G$ and $c_{G'}(i,j) = 0\ \forall (i,j) \in NA$.

Figure 4.4 gives a simple example of the transformation: the origi-

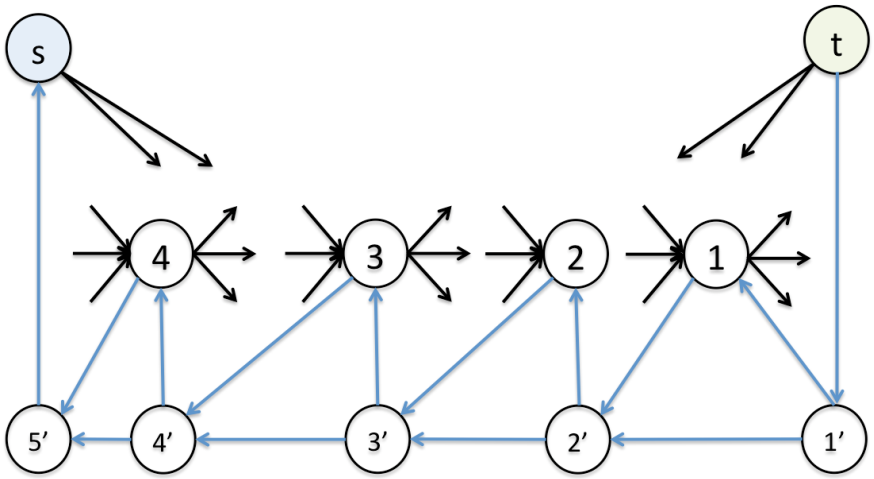

Figure 4.4: Transformation from ESPP(s, t, G) to ATSP

nal graph of 6 nodes is transformed to a graph of 11 nodes, which transformed graph has 5 new nodes $1', 2', 3', 4', 5'$ and 14 new arcs $(t, 1')$, $(1', 1)$, $(1, 2')$, $(1', 2')$, $\ldots$ $(5', s)$.

The ATSP(G') problem can be modeled by a set of binary variables $X' = \{x_{ij} | (i, j) \in A_{G'}\}$ indicating whether the arc (i, j) is included in the solution [39]:

$$minimize \sum_{(i,j)\in A_{G'}} c_{G'}(i, j)x_{ij} \tag{4.4a}$$

subject to

$$x(\delta^+_{G'}(i)) = x(\delta^-_{G'}(i)) = 1, \ \ \forall i \in X' \tag{4.4b}$$

$$x(\delta^+_{G'}(S)) \geq 1, \ \ S \subset V_{G'}, S \neq \emptyset \tag{4.4c}$$

$$x_{ij} \in \{0, 1\}, \ \forall (i, j) \in A_{G'}. \tag{4.4d}$$

We put $X = \{x_{ij} \in X' | (i, j) \in A_G\}$.

Theorem 1 *If $X'^* = \{x^*_{ij} \mid (i, j) \in A_{G'}\}$ is an optimal solution to ATSP(G'), then $X^* = \{x^*_{ij} \mid (i, j) \in A_G\}$ is an optimal solution to ESPP(s, t, G)*

Proof Suppose that X'^* corresponds to the shortest Hamiltonian cycle $C = \langle (s, u_1), (u_1, u_2), \ldots, (u_p, t), (t, u_{p+1}), \ldots, (u_{2n+1}, s) \rangle$ (starting and terminating at s) on the transformed graph G'. We will show that the elementary path $P = \langle (s, u_1), (u_1, u_2), \ldots, (u_p, t) \rangle$ corresponds to X^* and is an optimal solution to ESPP(s, t, G). We have:

- Due to v'_j is in C, so either (v'_j, v'_{j+1}) or (v'_j, v_j) is in C.
- Due to v'_{j+1} is in C, so either (v'_j, v'_{j+1}) or (v_j, v'_{j+1}) is in C.

It follows that we have that node v'_{j+1} always stands after node v'_j in C (the cycle starting and terminating at s). In addition, node v'_1 is always after node t in the cycle (there is only one arc (t, v'_1) coming to node v'_1). We have that all nodes in *NA* stand after the node t in C. Hence, all nodes of P belong to G. Thus, X^* models P and the cost of P is equal to the cost of C. We now show that P has minimal cost.

Suppose that the cost of P is not minimal, and $P' = \langle (s, x_1), (x_1, x_2), \ldots, (x_p, t) \rangle$ is an elementary path in G whose cost is smaller than that of P. We extend P' to establish a Hamiltonian cycle C' in Algorithm 5.

Algorithm 5: Extend(P', G')

```
C' ← P';
Add arc (t, v'_1) to C';
foreach j ∈ 1..n + 1 do
    if v_j ∈ P' then
        Add the arc (v'_j, v'_{j+1}) to C'
    else
        Add (v'_j, v_j) and (v_j, v'_{j+1}) to C';
return C';
```

Clearly, the cost of C' is equal to that of P' and is thus smaller than the cost of C (a contradiction, as C is an optimal solution to ASTP(G')).

So, P is the shortest elementary path in G. ∎

As a result of Theorem 1, we can tackle ESPP(s, t, G) on the graph G by solving ATSP in G'. However, in this approach, the input size of the problem is increased by a factor of two (the number of nodes of G' is about twice as large as that of G). We therefore do not follow this

approach. Instead, we exploit different valid inequalities of the model of ATSP(G') and integrate them into the model of ESPP(s, t, G). It is clear that if $\mathbb{I}(Y')$ is a valid inequality of Model 4.3.1, with $Y' \subseteq X'$, then $\mathbb{I}(Y)$ is a valid inequality of (1a)–(1e) with $Y \subseteq X$.

D_k, 2-Matching, and T_k are classes of valid inequalities for ASTP in variables in X' (see [47, 39, 9, 38, 35, 81, 11] for more details about these valid inequalities). From D_k and T_k, we can extract inequalities in variables in X and apply these inequalities to the solution of ESPP(s, t, G). In addition, we adapt the 2-Matching valid inequalities to establish valid inequalities for ESPP(s, t, G). As far as we know, these valid inequalities have not yet been exploited for solving ESPP(s, t, G). Moreover, we propose a simple class of valid inequalities which will be experimentally shown to be effective (see Section 4.4.4).

4.4.1 D_k-inequalities

For each cycle $\langle (i_1, i_2), \ldots, (i_k, i_1) \rangle$ of G ($k \in \{3, |V_G| - 1\}$), the two following inequalities are valid for ESPP(s, t, G):

D_k^--inequalities:

$$\sum_{j=1}^{k-1} x_{i_j i_{j+1}} + x_{i_k i_1} + 2 \sum_{j=3}^{k} x_{i_1 i_j} + \sum_{j=4}^{k} \sum_{h=3}^{j-1} x_{i_j i_h} \leq k - 1 \qquad (4.5)$$

and the D_k^+-inequalities:

$$\sum_{j=1}^{k-1} x_{i_j i_{j+1}} + x_{i_k i_1} + 2 \sum_{j=3}^{k} x_{i_j i_1} + \sum_{j=3}^{k-1} \sum_{h=2}^{j-1} x_{i_j i_h} \leq k - 1. \qquad (4.6)$$

4.4.2 T_k-inequalities

The T_k-inequalities that are valid for ESPP(s, t, G) on the original graph G are as follow.

For any $W \subset V_G \setminus \{s, t\}, 2 \leq |W| \leq |V_G| - 4, w \in W, p, q \in V_G \setminus W$, we have

$$x_{pw} + x_{pq} + x_{wq} + x(A_G(W)) \leq |W|. \qquad (4.7)$$

4.4.3 2-Matching inequalities

The 2-Matching inequalities discovered by Edmonds [35] are widely used in B&C algorithms for STSP and its variations [47, 81, 9]. We adapt the 2-Matching to be valid inequalities for ESPP(s, t, G) as follows:

$$x(\delta_G(S)) \geq 2x(F)+1-|F|, \forall S \subset V_G, s \notin S, t \notin S, F \subset \delta_G(S), |F| \text{ odd}. \tag{4.8}$$

Theorem 2 *Inequalities (4.8) are valid for ESPP(s, t, G).*

Proof Let P be the set of nodes of an elementary shortest path on the original graph G. We have $x(\delta_G(S)) \geq x(F)$ as $F \subseteq \delta_G(S)$ and $x(F) \leq |F|$. Hence $x(\delta_G(S)) - x(F) + |F| - x(F) \geq 0$.

In addition, because neither s and t are in S, we have $x(\delta_G(S)) = 2|P \cap S|$, which is even. So we have $x(\delta_G(S)) - x(F) + |F| - x(F) \geq 1$, as $|F|$ is odd. Then inequalities (4.8) are true.

Finally, we conclude that inequalities (4.8) are valid for ESPP(s, t, G).

■

4.4.4 Maximum outflow inequalities

We propose using the class of so-called maximum outflow inequalities:

$$x(\delta_G^+(v)) \leq 1, \forall v \in V_G \setminus \{t\}. \tag{4.9}$$

This class imposes a constraint on the maximum outflow at each node of the graph as in each solution to ESPP(s, t, G), there is at most one arc leaving each node of G.

4.5 Solving ESPP(s, t, G) to optimality

Models 2 and 3 described in Section 4.3 have a polynomial number of constraints. It is thus possible to solve the problem by a MIP solver with these models. We will assess the performance of these B&C algorithms (two algorithms correspond to two models) in the experiments

section. We denote these two B&C algorithms by *ESP_ComFlow* and *ESP_MTZ*.

In this section, we concentrate on proposing a B&C algorithm for solving ESPP(s, t, G) with Model 1. For the sake of readability, we denote that algorithm by *ESP_FltC*.

4.5.1 B&C algorithm schema of *ESP_FltC*

We present here a generic B&C procedure of *ESP_FltC*, (see [80, 10, 53] for more information on B&C procedures).

At a generic step of the B&C procedure, the original linear programming relaxation, $0 \leq x_{ij} \leq 1$, is enriched by additional valid inequalities for ESPP(s, t, G), and some of the binary variables are fixed either at their upper or lower bound. We denote by $\mathcal{C}$ the current family of valid inequalities for ESPP(s, t, G) and we assume that the linear system $Ax \geq b$ defining $\mathcal{C}$ contains at least all the inequalities in (4.1b) and (4.1c) of Model 1.

We denote by $\mathcal{F}_0$ and $\mathcal{F}_1$ the sets of variables that have been fixed at 0 and 1, respectively.

Let $\mathcal{K}(\mathcal{C}, \mathcal{F}_0, \mathcal{F}_1) = \{x : Ax \geq b$
$x_{ij} = 0$ for $x_{ij} \in \mathcal{F}_0$
$x_{ij} = 1$ for $x_{ij} \in \mathcal{F}_1\}$
and let $LP(\mathcal{C}, \mathcal{F}_0, \mathcal{F}_1)$ denote the linear program

$$Min\ cx : x \in \mathcal{K}(\mathcal{C}, \mathcal{F}_0, \mathcal{F}_1)$$

which is assumed to be feasible, with a finite minimum.

The active nodes of the enumeration tree are represented by a list $\mathcal{NL}$ of ordered pairs $(\mathcal{F}_0, \mathcal{F}_1)$. Let x^* be the best known solution to ESPP(s, t, G) and *UB* stand for the current upper bound (the value of the best known solution x^*).

During the solution process, the B&C procedure maintains an inequality pool $\mathcal{IP}$ containing the inequalities generated. This procedure is depicted in Figures 4.5. Its different steps are developed in the following sections.

1. **Preprocessing:** Removing useless nodes and arcs from the graph G.

2. **Initialization:** Let $\mathcal{C}$ be the set of constraints in the linear programming relaxation of ESPP(s, t, G) (containing only the inequalities in (4.1b) and (4.1c) of Model 1). Set $\mathcal{IP} = \emptyset$, $\mathcal{F}_0 = \mathcal{F}_1 = \emptyset$, $x^* = NULL$, $UB = +\infty$, and two temporary inequality sets $\mathcal{SI} = \mathcal{GI} = \emptyset$.

3. **Lower bounding:** Solve the linear problem $LP(\mathcal{C} \cup \mathcal{SI}, \mathcal{F}_0, \mathcal{F}_1)$. Clear the set of separated inequalities $\mathcal{SI}$. If there exists an optimal solution to the linear problem, denote it by $\overline{x}$. If the solution $\overline{x}$ is an elementary path, update the best known solution $x^* = \overline{x}$ and the upper bound value $UB = c\overline{x}$. If the solution $\overline{x}$ is not an elementary path, go to Step 5 (Inequality generation).

4. **Node selection:** If $\mathcal{NL} = \emptyset$, stop; otherwise, choose an ordered pair $(\mathcal{F}_0, \mathcal{F}_1)$ and remove it from $\mathcal{NL}$. Go to Step 3 (Lower bounding).

5. **Inequality generation:** Clear the set of generated inequalities $\mathcal{GI}$. Generate inequalities valid for ESPP(s, t, G) but violated by $\overline{x}$ and add them to $\mathcal{GI}$. If $\overline{x}$ is an integer-feasible solution, do $\mathcal{C} = \mathcal{C} \cup \mathcal{GI}$ and go to Step 3 (Lower bounding).

6. **Inequality selection:** Select some most violated inequalities from $\mathcal{GI}$ and then add them to $\mathcal{IP}$.

7. **Inequality detection:** All inequalities in $\mathcal{IP}$ that are violated by the solution $\overline{x}$ are inserted into $\mathcal{SI}$.

8. **Branching/Cutting decision:** If $\mathcal{SI}$ is not empty, go to Step 3 (Lower bounding).

9. **Branching:** Select an appropriate variable x_{ij} such that $\overline{x_{ij}}$ is fractional. Generate two subproblems corresponding to $(\mathcal{F}_0 \cup \{x_{ij}\}, \mathcal{F}_1)$ and $(\mathcal{F}_0, \mathcal{F}_1 \cup \{x_{ij}\})$, insert them into $\mathcal{NL}$. Go to Step 4 (Node selection).

Figure 4.5: B&C schema of *ESP_FltC*

4.5.2 Preprocessing

Preprocessing is a very important algorithmic tool for solving large scale combinatorial problems. The main idea is to detect unnecessary information in the problem and to reduce the size of the problem by logical implications. In *ESP_FltC*, we apply some of following simple reduction methods [66, 62, 68, 102, 66, 29].

Removing useless nodes

We can erase nodes that cannot be in any path from the source node to destination node. Such nodes can be detected by a simple search, for example, depth-first search.

- A forward search is used to find nodes connected from the source node, these nodes are collected into a set denoted by S_1.
- A backward search is used to find nodes connected to the destination node, these nodes are collected into a set denoted by S_2.

We can remove nodes in $V_G \setminus \{S_1 \cap S_2\}$ because they can not be in a path from the source node to the destination node.

Removing nodes of degree 1

A node with only one flow that arrives or leaves it is useless and can be removed.

1. Let v be a node such that $\delta_G^+(v) = \{u\}$ (there is only one flow going out from v). Then each arc $(i, v) \in \delta_G^-(v)$ can be replaced by an arc (i, u) of cost $c_G(i, v) + c_G(v, u)$.
2. Let v be a node such that $\delta_G^-(v) = \{u\}$ (there is only one flow arriving to v). Then each arc $(v, i) \in \delta_G^+(v)$ can be replaced by an arc (u, i) of cost $c_G(u, v) + c_G(v, i)$.

4.5.3 Node selection

The objective of the node selection step (Step 4) is to choose the next node to expand. A naive node selection strategy might lead to a huge search tree. By contrast, an intelligent node selection strategy leads quickly to a good feasible solution that sharply reduces the gap between the upper bound and lower bound and proves the optimality of the current incumbent. There exist some well-known node-selection strategies, such as depth-first-search, breadth-first-search, and best-bound search.

In order to minimize the size of the search tree, in *ESP_FltC*, the *best-bound search* strategy, which selects the node with the best lower bound, is carried out. Its advantage is that, for a fixed sequence of branching decisions, it minimizes the number of nodes that are explored, because all nodes that are explored would have been explored independently of the upper bound [13].

4.5.4 Branching variable selection

In the branching step (Step 9), a fractional variable is chosen and two new subproblems are generated by setting in turn the value of the variable to 0, and to 1. Selecting the right branching variable is an important component of a B&C algorithm. Ideally, we would like to choose the branching variables that minimize the size of the search tree. A simple branching strategy applied is to select the variable with the largest integer violation: this is known as *maximum fraction branching* [62, 9]. In practice, this rule is not efficient: it performs about as well as randomly selecting a branching variable [4].

The most successful branching strategies estimate the change in the lower bound after branching. Because we prune a node of the branch-and-bound tree whenever the lower bound of the node is greater than the current upper bound, we want to increase the lower bound as much as possible. In [4], Achterberg et al. also experimented with strong branching rules, pseudocost branching, strong branching, a hybrid of strong/pseudocost branching, pseudo cost branching with strong branching initialization, and reliability branching.

ESP_FltC is implemented in C++, using IBM Ilog Cplex Concert Technology, version 12.2. The default variable selection strategy of

CPLEX is a variant of hybrid branching [4, 3]. Based on the experiments in [4], we decided that *ESP_FltC* should carry out that default strategy.

4.5.5 Inequality generation

Given a solution $\overline{x}$ to $LP(\mathbb{C}, \mathcal{F}_0, \mathcal{F}_1)$ at a node of the branch-and-bound tree, we try to find valid inequalities that are not satisfied by $\overline{x}$, and add them to the model. Algorithms for finding these inequalities often rely on a support graph $\overline{G} = (V_{\overline{G}}, A_{\overline{G}}, c_{\overline{G}})$ with $V_{\overline{G}} = V$, $A_{\overline{G}} = \{(i,j)|\overline{x}_{ij} > 0\}$ and each arc $(i,j) \in A_{\overline{G}}$ is associated with a cost $c_{\overline{G}}(i,j) = \overline{x}_{ij}$.

Algorithms for separating valid inequalities D_k, T_k are presented in [39, 9]. For the generation of 2-Matching valid inequalities, we only separate inequalities with $|S| = 1$ in order to reduce the computational complexity [82, 81, 55].

Subtour elimination constraint generation

In model 1, the goal of generating an SEC that is violated by $\overline{x}$ (see Section 4.3.1) means finding a pair $\langle S, i\rangle$ in which $S \subseteq V_G \setminus \{t\}$ and $i \in S$ such that $\overline{x}(\delta_G^+(S)) < \overline{x}(\delta_G^+(i))$. The algorithm for generating SECs in [33] works on the undirected auxiliary support graph $\overline{D} = \{V_{\overline{D}}, E_{\overline{D}}, c_{\overline{D}}\}$ in which $V_{\overline{D}} = V$, $E_{\overline{D}} = \{(i,j)|\overline{x}_{ij} + \overline{x}_{ji} > 0\}$ with a cost $c_{\overline{D}}(i,j) = \overline{x}_{ij} + \overline{x}_{ji}$ associated to each edge $(i,j) \in E_{\overline{D}}$[2]. We call this algorithm the traditional separation algorithm for finding SECs.

The traditional separation algorithm for finding SECs works as follows. In the first step, it checks whether there exist any isolated components in $\overline{D}$ that are not connected to s and t. If such an isolated component is found, an SEC (4.1d) is generated in which S contains all nodes of the isolated component, and i is randomly selected from S. Otherwise (i.e., no isolated component is found), the algorithm performs the second step which solves, for each node $v \in V_G \setminus \{t\}$, the maximum $v-t$-flow/minimum $v-t$-cut problem on $\overline{D}$. If the maximum flow $v-t$ is less than the outflow from this node with respect to $\overline{x}$ (i.e., $\sum_{u \in V_G^+(v)} \overline{x}_{vu}$), then one SEC (4.1d) is generated; in this SEC

[2]We would like to thank Michael Drexl, the author of [33], for this information.

$\langle S, i \rangle$, S is the set of nodes that are on the same side of the $v - t$-cut than v and i is the node v.

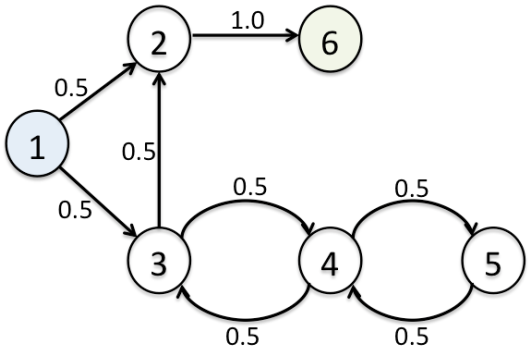

Figure 4.6: In this example, the traditional separation algorithm for SECs cannot find any SECs.

In some cases, the traditional algorithm cannot detect any SECs: this is illustrated in Figure 4.6: there are violated SECs (4.1d) in this support graph that cannot be detected by the traditional algorithm, for example, with $S = \{3, 4, 5\}$ and i being the node 3 or 4.

As the traditional algorithm cannot always find SECs even if they exist, we propose in this section another heuristic algorithm for finding SECs which is simple but efficient. Our proposed algorithm extends from a promising node (node i in (4.1d)) to obtain the set of nodes (set S in (4.1d)) until an SEC is found or until the current set of nodes cannot be extended anymore. In our proposed algorithm, a promising node is an endpoint of an arc (u, v) such that $\overline{x}(\delta^+_{\overline{G}}(u)) \geq 2$. For example, in Figure 4.6, nodes 1, 2, 3, 4, and 5 are promising nodes.

The first step of our heuristic separation algorithm for SECs is similar to the traditional separation algorithm, which checks for isolated components that are not connected to the source and the destination nodes. The second step of our heuristic algorithm tries to find an SEC by extending a promising node (as described in Algorithm 6) instead of finding the max flow from every node $v \in V_{\overline{G}}$ to t in $\overline{G}$.

In Algorithm 6, the procedure extends from a promising node i. At each iteration, a new node is added to the set S (lines 7–8), the outflow from S is firstly recomputed (lines 9–10), then it checks for a violation

of the SECs (lines 11–12). If a violated SEC is found, the procedure immediately stops. The procedure also stops immediately when S can not be extended (lines 13–15).

Algorithm 6: FindSEC($\overline{G} = (V_{\overline{G}}, A_{\overline{G}}, c_{\overline{G}}), i, t$)

Input: Support graph $\overline{G} = (V_{\overline{G}}, A_{\overline{G}}, c_{\overline{G}})$, the destination node t and a promising node i
Output: A pair $\langle S, i\rangle$ (if $S \neq \emptyset$ we have a violated SEC, otherwise, no violated SEC found corresponding to i)

1 Insert the promising node i into node set S and a queue Q;
2 $\Delta \leftarrow \sum_{v \in V^+_{\overline{G}}(i)} c_{\overline{G}}(i, v)$;
3 $OutFlowS \leftarrow \Delta$;
4 $Extend \leftarrow true$;
5 **while** $Extend$ **do**
6 Get and Remove top node u from the queue Q ;
7 **foreach** $v \in V^+_{\overline{G}}(u) : v \notin S, v \neq t, Extend = true$ **do**
8 Insert v into S and Q;
9 $OutFlowS \leftarrow OutFlowS + \sum_{u \in V^+_{\overline{G}}(v)} c_{\overline{G}}(v, u)$;
10 $OutFlowS \leftarrow OutFlowS - \sum_{u \in S}(c_{\overline{G}}(u, v) + c_{\overline{G}}(v, u))$;
11 **if** $OutFlowS < \Delta$ **then**
12 $Extend \leftarrow false$;
13 break ;
14 **if** Q *is empty* **then**
15 $Extend = false$;
16 $S \leftarrow \emptyset$;
17 **return** $\langle S, i\rangle$;

Clearly, our proposed heuristic separation algorithm for SECs can find two SECs from the support graph in the Figure 4.6: $\langle\{3, 4, 5\}, 3\rangle$ and $\langle\{4, 5\}, 4\rangle$.

Generation of maximum outflow inequality

It is clear that maximum outflow inequalities can easily be computed by an algorithm with complexity of $O(|V_{\overline{G}}|)$, checking for a violation of a

maximum outflow inequality at each node of the support graph $\overline{G}$.

Generating other inequalities

We also use other families of inequalities that are valid for a general mixed integer program. They are MIP mixed integer rounding inequalities, MIP zero-half inequalities, MIP Gomory fractional inequalities, etc.. Our algorithm *ESP_FltC* is implemented in CPLEX solver that provides an option for either generating these inequalities or not.

4.5.6 Inequality selection

The execution of *ESP_FltC* maintains an *inequality pool* $\mathcal{IP}$ that collects the inequalities generated. At each node, each time the algorithm produces a solution, the inequalities in the pool are checked as to whether they are violated by the solution. If some inequalities are violated by the solution, the algorithm adds them to the model of the current node and then re-optimizes. This procedure is iteratively executed until no inequality in the pool is violated.

The pool $\mathcal{IP}$ has an important role in *ESP_FltC* that becomes much stronger if its pool contains just the most violated inequalities. To reject useless violated inequalities, *ESP_FltC* must answer the two following questions:

1. When there are various classes of valid inequalities, how to decide whether one class is better than another? Although many B&C solvers report computational experience on this issue, it mostly remains an open question. In general, all valid inequalities are separated.

2. In a class of inequalities, is one inequality better than another? This is also an open question. However, two widely used measures of the quality of an inequality in the same class are presented in [10]. Given an inequality $\alpha x \geq \beta$, its quality is computed by one of the two following measures:

   (a) The value $\beta - \alpha\overline{x}$ measures the traditional violation of the inequality.

(b) The geometric measure uses the Euclidean distance between $\overline{x}$ and the hyperplane $\alpha x = \beta$, namely the distance between $\overline{x}$ and its orthogonal projection onto this hyperplane.

In this chapter, we use the traditional measure to order the inequalities belonging to the same class. However, we must determine a good threshold for each class of inequalities to arrange that the pool $\mathcal{IP}$ contains only the most violated inequalities.

4.6 Decompositions

In [84], Pham et al. proposed an efficient dynamic algorithm to solve ELPP on undirected graphs of positive-cost edges, which have many bridge-blocks. The dynamic algorithm decomposes such graph into blocks; then uses CP approach to solve ELPP smaller-size instances on the blocks; lastly computes dynamically optimal solutions to the ELPP. In this thesis, we extend that decomposition to solve ESPP and ELLPP on directed graphs of positive-cost and negative-cost arcs, which consist of many bridges-blocks. Moreover, this thesis proposes a decomposition technique to solve ESPP and ELPP on directed graphs of positive-cost and negative-cost arcs, which contain many strongly connected components.

Before describing the decompositions, we give some definitions and notations. Given a directed graph $G = (V_G, A_G)$ and a set of nodes $S \subseteq V_G$, we denote by $G(S) = (V_{G(S)}, A_{G(S)})$ the sub-graph induced by S in which $V_{G(S)} = S$ and $A_{G(S)} = \{(u, v) \in A_G | u, v \in S\}$. An auxiliary graph of G is a graph $D = \{V_D, E_D\}$ where $E_D = \{(u, v) \mid (u, v) \in A_G \vee (v, u) \in A_G\}$. In a rooted tree, T, $T(v)$ denotes the set of descendant nodes of v in T, including v.

4.6.1 Computing bridge-blocks

Without loss of generality, we assume that the graph G is connected. The computation of bridges and bridge-blocks can be realized with a variant of the $O(|V| + |A|)$ depth-first search algorithm $DFS(r)$ of [97] on an auxiliary graph of G, denoted by D. The variant algorithm we de-

veloped requires fixing a root node r of V_G and produces the following results:

- The set $B(D)$ of bridges of D;
- A spanning tree T of D, rooted at r, in which $f(v)$ is the father of a node v (by convention $f(r) = r$); the spanning tree will be composed of k spanning subtrees (one per each bridge-block), connected by the bridges;
- The set $\{S_1, \ldots, S_k\}$, where S_i is the set of nodes of bridge-blocks of D;
- The set R of roots of the different subtrees $T(S_i)$, i.e., $R = \{r_i | r_i$ is the root of $T(S_i), 1 \leq i \leq k\}$;
- $Exit(r_i)$, the set of exit nodes of $T(S_i)$, which are nodes in S_i connected to (the root of) some other bridge-block in T plus the node r_i, i.e., $Exit(r_i) = \{v \in S_i | \exists r_j \in R : f(r_j) = v\} \cup \{r_i\}$;
- $ChRoot(v)$, the child nodes of v in T that are outside the bridge-blocks of v; by the definition of a bridge block, such children are necessarily members of R, i.e., $ChRoot(v) = \{r_i \in R | f(r_i) = v\}$;
- $NextRoot(r_i)$ is the set of root nodes of bridge-blocks connected to S_i, i.e., $NextRoot(r_i) = \cup_{v \in S_i} ChRoot(v)$;

Figures 4.7 and 4.8 illustrate the result of DFS; we give some of the results: $S_1 = \{1, 2, 3, 4, 5\}, S_2 = \{6, 7, 8, 9, 36\}, S_3 = \{17\}, S_4 = \{18, 19, 20, 21\}, S_5 = \{37\}, S_6 = \{12, 13, 14, 15, 16\}, S_7 = \{11\}, S_8 = \{10\}.R = \{1, 6, 17, 18, 37, 12, 11, 10\}, NextRoot(1) = \{6, 10\}, NextRoot(6) = \{11, 12, 17, 37\}, NextRoot(17) = \{18\}$. The other $NextRoot$ values are $\{\emptyset\}$. $B(G') = \{(6, 5), (10, 4), (11, 7), (12, 9), (17, 8), (18, 17), (37, 8)\}$. $ChRoot(1) = \{\emptyset\}, ChRoot(5) = \{6\}, ChRoot(8) = \{17, 37\}$ etc. $Exit(1) = \{1, 4, 5\}, Exit(6) = \{6, 7, 8, 9\}$, etc.

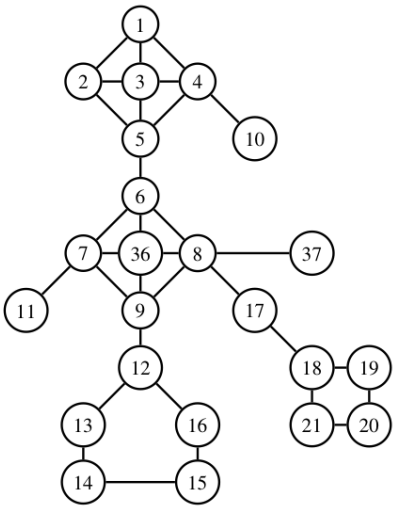

Figure 4.7: The auxiliary graph D

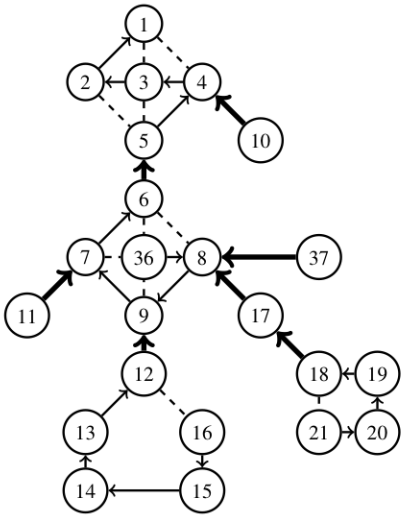

Figure 4.8: The rooted spanning tree T (solid arcs are in T, bold arcs are bridges)

4.6.2 Computing strongly connected components

A directed graph is *strongly connected* if there exists a path from each node in the graph to every other node. The *strongly connected components* (SCCs) of a directed graph are the maximal strongly connected subgraphs. It is evident that if each SCC is contracted to a single node, the contracted directed graph is acyclic. When solving ESPP(s, t, G), we could focus on the SCCs that are on paths in the contracted graph from the SCC containing s to the SCC containing t.

Given a directed graph $G = (V_G, A_G, c_G)$ together with a source node s and a destination node t, we introduce the following notation:

- $\mathcal{S}$ are the set of nodes connected from s by an elementary path and $\mathcal{T}$ is the set of nodes that connect to t by an elementary path.
- $SCC = \{c_1, \ldots, c_n\}$ is the set of SCCs of G with $c_i = (V_{c_i}, A_{c_i})$.
- We write $I(c_i) = \{v \in V_{c_i} \mid (u, v) \in A_G \wedge u \notin V_{c_i}\}$ and $O(c_i) = \{v \in V_{c_i} \mid (v, u) \in A_G \wedge u \notin V_{c_i}\}$. If c_i contains s, then s is included in $I(c_i)$. If c_i contains t, then t is included in $O(c_i)$.

- $Parent(c_i)$ is the set of SCCs connected to c_i by an arc, i.e.: $Parent(c_i) = \{c_j \in SCC | i \neq j \wedge \exists (u,v) \in V_G : u \in V_{c_j}, v \in V_{c_i}\}$.
- $Children(c_i)$ is set of SCCs connected from c_i by an arc, i.e.: $Children(c_i) = \{c_j \in SCC | i \neq j \wedge \exists (u,v) \in V_G : u \in V_{c_i}, v \in V_{c_j}\}$.

Before computing ESPP(s, t, G), the input graph is preprocessed in the two following steps:

- **Step 1** consists of removing nodes that cannot be a part of an elementary path from s to t from the graph. $\mathcal{S}$ and $\mathcal{T}$ are computed with a depth-first search algorithm [28]. Nodes in $V_G \setminus \{\mathcal{S} \cap \mathcal{T}\}$ are removed from G.
- **Step 2** consists of finding all SCCs in the graph. This computation can be realized with a variant of the $O(|V| + |A|)$ depth-first search algorithm in [28]. The variant of the depth-first search algorithm that we propose uses s as the root node. After this step, we have a set of SCCs: $SCC = \{c_1, \ldots, c_n\}$ and $I(c_i$, $O(c_i)$, $Parent(c_i)$, $Children(c_i)$ for each c_i.

Figures 4.9 and 4.10 illustrate the result of the preprocessing. The overall goal is to determine the elementary shortest path from node 1 to node 19. After Step 1, nodes 3, 11, 12, 13, 14, 15, 20 are removed. After Step 2, $SCC = \{c_1, c_2, c_3\}, V_{c_1} = \{1, 2, 4, 5\}, V_{c_2} = \{6, 7, 8, 9, 10\}, V_{c_3} = \{16, 17, 18, 19\}, I(c_1) = \{1\}, I(c_2) = \{6\}, I(c_3) = \{16, 17\}, O(c_1) = \{5\}, O(c_2) = \{8, 10\}, O(c_3) = \{19\}, Parent(c_1) = \emptyset, Parent(c_2) = \{c_1\}, Parent(c_3) = \{c_2\}$, $Children(c_1) = \{c_2\}$, $Children(c_2) = \{c_3\}$, and $Children(c_3) = \emptyset$.

4.6.3 ELPP(G) on sparse directed graphs with many bridges

In [84], Pham et al. proposed a dynamic programming approach for solving ELPP(G) on sparse undirected graphs with many bridges, but limited to positive arc costs. In the thesis, we extend the approach to directed graphs with positive and negative arc costs. This new algorithm

based on MIP is much faster than our previous algorithm in [84] that was based on Constraint Programming. We denote this new algorithm by *ELP_FltC_BB*.

The given directed graph is decomposed into bridge-blocks as described in Section 4.6.1. From the resulting data structures, we now introduce the following notations.

- $L(r_i)$: the cost of the elementary longest path in $G(T(r_i))$.
- $d(u, v)$: the cost of the elementary longest path from u to v in $G(S_i)$, with $u, v \in S_i$. By convention, $d(u, u) = 0$.
- $H_1(r_i)$: the cost of the elementary longest path starting from r_i in $G(T(r_i))$.
- $H_2(r_i)$: the cost of the elementary longest path ending at r_i in $G(T(r_i))$.
- $h_1(v)$: the cost of the elementary longest path in $G(T(v))$ starting from v, but without any arc in $G(S_i)$, with $v \in S_i$.
- $h_2(v)$: the cost of the elementary longest path in $G(T(v))$ ending at v, but without any arc in $G(S_i)$, with $v \in S_i$.

These last two quantities can be computed recursively as follows:

$$h_1(v) = max(max_{r_i \in ChRoot(v)} H_1(r_i) + c_G(v, r_i), 0)$$

$$h_2(v) = max(max_{r_i \in ChRoot(v)} H_2(r_i) + c_G(r_i, v), 0)$$

$$H_1(r_i) = max(max_{v \in S_i} d(r_i, v) + h_1(v), 0)$$

$$H_2(r_i) = max(h_2(v) + max_{v \in S_i} d(v, r_i), 0)$$

$h_1(v)$ and $h_2(v)$ are defined to be 0 when $ChRoot(v) = \emptyset$.

Using the above values, we can now compute $L(r_i)$, the cost of the elementary longest path in $G(T(r_i))$, with $r_i \in R$. We have $L(r_i) = max\{L_0(r_i), L_1(r_i), L_2(r_i)\}$, where $L_0(r_i)$ (resp. $L_1(r_i), L_2(r_i)$) is the cost of the elementary longest path in $G(T(r_i))$ containing no (resp. exactly one, at least 2) node of S_i. These values can be computed as follows.

- $L_0(r_i) = max_{r_j \in NextRoot(r_i)} L(r_j)$

- $L_1(r_i) = max_{v \in S_i} l(v)$ where

$$l(v) = \begin{cases} 0 \quad \text{if } ChRoot = \emptyset \\ max\{H_2(r_j) + c_{r_j v}, H_1(r_j) + c_G(v, r_j)\} \quad \text{if } ChRoot(v) = \{r_j\} \\ max_{r_k \neq r_j \in ChRoot(v)} H_2(r_k) + c_G(r_k, v) + c_G(v, r_j) + H_1(r_j) \quad \text{otherwise} \end{cases}$$

- $L_2(r_i) = max_{u \neq v \in S_i} h_2(u) + d(u, v) + h_1(v)$

The algorithm is composed of the sequential calls to the recursive methods ComputeFirstStep(r) and ComputeSecondStep(r) depicted in Algorithms 7 and 8, where r is the root node of the spanning tree T. Algorithm 7 computes the longest path in G containing some exit nodes and arc bridges. The variable f^* stores the cost of this path.

The algorithm 8 completes the search by considering pairs of nodes without arc bridges, using f^* as a lower bound. The search is performed only if the positive cost W (the sum of the positive arc costs) of the bridge-block $G(S_i)$ is greater than f^* (see lines 4–5) because the cost of the any longest path on S_i is always less than or equal to W. The method solveELPP($G(S_i)$, f^*) computes the cost of the elementary longest path in $G(S_i)$ so that the cost of the solution path is greater than f^*. That method is derived from any algorithm of ELPP with a lower bound constraint for the objective function. The data structure $L(r_i)$, $d(u, v)$, $H_1(r_i)$, $H_2(r_i)$, $h_1(v)$, $h_2(r_i)$ and f^* is global. At the end of the computation, f^* will be the cost of the optimal solution. The algorithm can be easily extended to also return the optimal path.

To compute ELPP(G) on an undirected graph, we replace each edge by two opposite arcs with the same cost as the edge. Our algorithm is slightly adapted to reduce the computation time. For a pair of nodes, it is not required to compute separately two elementary longest paths from one to the other, as the two paths have the same the length. This means that $H_1(r_i) = H_2(r_i)\ \forall$ root r_i and $h_1(v) = h_2(v)\ \forall v \in S$.

In [84], we used a Constraint Programming search method. Here, we use an MIP search method, adapted from *ESP_FltC* to solve ELPP(G). Furthermore, *ELP_FltC_BB* calls only once solveELPP($\{r_i\}, S_i \backslash Exit(r_i), G(S_i)$), and solveELPP($S_i \setminus Exit(r_i), \{r_i\}, G(S_i)$), replacing the various calls

Algorithm 7: ComputeFirstStep(r_i)

foreach $r_j \in \mathit{NextRoot}(r_i)$ **do**
  ComputeFirstStep(r_j);
foreach $u, v \in Exit(r_i) : u \neq v$ **do**
  $d(u, v, S_i) \leftarrow$ solveELPP($\{u\}, \{v\}, G(S_i)$); ;
$d_1(r_i) \leftarrow$ solveELPP($\{r_i\}, S_i \setminus Exit(r_i), G(S_i)$) ;
$d_2(r_i) \leftarrow$ solveELPP($S_i \setminus Exit(r_i), \{r_i\}, G(S_i)$) ;
foreach $v \in S_i$ **do**
  $h_1(v) \leftarrow max_{r_j \in \mathit{ChRoot}(v)} H_1(r_j) + c_G(v, r_j)$;
  $h_2(v) \leftarrow max_{r_j \in \mathit{ChRoot}(v)} H_2(r_j) + c_G(r_j, v)$;
  $h_1(v) \leftarrow max\{h_1(v), 0\}; h_2(v) \leftarrow max\{h_2(v), 0\}$;
$H_1(r_i) \leftarrow max_{v \in S_i \setminus Exit(r_i)} d(r_i, v) + h_1(v)$;
$H_2(r_i) \leftarrow max_{v \in S_i \setminus Exit(r_i)} d(v, r_i) + h_2(v)$;
$H_1(r_i) \leftarrow max\{H_1, d_1(r_i), 0\}$; $H_2(r_i) \leftarrow max\{H_2, d_2(r_i), 0\}$;
$L_0 = max_{r_j \in \mathit{NextRoot}(r_i)} L(r_j)$;
$L_1 \leftarrow 0$;
foreach $v \in S_i : \mathit{ChRoot}(v) \neq \emptyset$ **do**
  if $\mathit{ChRoot}(v) = \{r_j\}$ **then**
    $l_1 \leftarrow max\{H_2(r_j) + c_G(r_j, v), H_1(r_j) + c_G(v, r_j)\}$;
  else
    $l_1 \leftarrow$
    $max_{r_k \neq r_j \in \mathit{ChRoot}(v)} H_2(r_k) + c_G(r_k, v) + c_G(v, r_j) + H_1(r_j)$;
  $L_1 \leftarrow max\{L_1, l_1\}$;
$L_2 \leftarrow max_{u \neq v \in Exit(r_i)} h_2(u) + d(u, v) + h_1(v)$;
$L(r_i) \leftarrow max\{L_0, L_1, L_2, 0\}$;
$f^* \leftarrow max\{f^*, L(r_i)\}$;

of the method solveELPP($\{u\}, \{v\}, G(S_i)$).

Algorithm 8: ComputeSecondStep(r_i)

foreach $r_j \in \mathit{NextRoot}(r_i)$ **do**
  ComputeSecondStep(r_j);
$W \leftarrow$ positive cost of $G(S_i)$;
if $W > f^*$ **then**
  $d \leftarrow$ solveELPP($G(S_i), f^*$);
  $f^* \leftarrow max\{f^*, d\}$;

4.6.4 ESPP(s, t, G) on sparse directed graphs with many bridge-blocks

In this section, we apply the decomposition technique described in Section 4.6.1 to the resolution of ESPP(s, t, G). The proposed algorithm, denoted by *ESP_FltC_BB*, has the following successive steps:

1. Decompose the given graph into bridge-blocks as described in Section 4.6.1
2. Construct a contracted graph $G_c = (V_{G_c}, E_{G_c})$:
   - V_{G_c} is the set of contracted nodes: each contracted node corresponds to a set of nodes of a bridge-block. We denote by $C(S)$ the contracted node corresponding to the bridge-block S.
   - For each pair of bridge-blocks S_i and S_j, if there exist $u \in S_i$ and $v \in S_j$ such that (u, v) is an edge of the auxiliary graph D, then $(C(S_i), C(S_j))$ is an edge of G_c.
3. Denote by S^s and S^t respectively the bridge-blocks containing s and t.
4. Find the unique path from $C(S^s)$ to $C(S^t)$ in G_c (this path is unique because G_c is a tree). Suppose this path is $\langle C(S^s) = C(S_0), C(S_1), \ldots, C(S_k) = C(S^t)\rangle$.
5. Two consecutive bridge-blocks S_i and S_{i+1} are connected by one edge (u, v) in G'. We denote by *Out*(S_i) the node u and by *In*(S_{i+1}) the node v ($i = 0, 1, 2, \ldots, k-1$). We denote by *In*(S_0) the node s and by *Out*(S_k) the node t.
6. Apply the B&C algorithm proposed in Section 4.5 for finding the shortest elementary path from *In*(S_i) to *Out*(S_i) in S_i ($\forall i = 0, \ldots, k$).
7. Concatenate these paths, to establish the shortest elementary path from s to t in G. Note that if there is no arc connecting *Out*(S_i) to *In*(S_{i+1}) in G with $0 \leq i \leq k-1$, then there is no solution to the original problem.

To compute ESPP(s, t, G) on a sparse undirected graph, we replace each edge by two opposite arcs with the same cost as the edge.

Figure 4.11 illustrates the algorithm to compute the elementary shortest path from 1 to 20. After finding the unique path in the contracted graph, we have the sequence of bridge-blocks corresponding to this path: $S_0 = \{1, 2, 3, 4, 5\}, S_1 = \{6, 7, 8, 9, 36\}, S_2 = \{17\}, S_3 = \{18, 19, 20, 21\}$. The solution is the concatenation of the shortest elementary paths from 1 to 5 in S_0; from 6 to 8 in S_1; from 17 to 17 in S_2; and from 18 to 20 in S_3.

4.6.5 ESPP(s, t, G) on sparse directed graphs with strongly connected components

The algorithm *ESP_FltC_SCC* is depicted in detail in Algorithm 9. In lines 1–4, the variables $d(v)$ and $d(u, v)$ are initially assigned a very large value L (e.g., L is the sum of the positive arc costs in the graph). In lines 5–6, for each SCC c_i, the costs of the elementary shortest paths between a node in $I(c_i)$ and a node in $O(c_i)$ are computed by the algorithm *ESP_FltC*. Note that each SCC is contracted to a single node: the contracted directed graph is acyclic. Method SortTopologocialOrdering(SCC) (line 9) sorts SCC into the sequence $S[1..|SCC|]$ in a topological ordering so that if there exist $u \in S[i]$ and $\in S[j]$ and $(u, v) \in A_G$, then $i < j$. After the ordering, $S[1]$ contains the node s and $S[|SCC|]$ contains the node t. The main idea is based on the well-known algorithm for finding the shortest path on a directed acyclic graph. Each SCC $S[i]$ in the sorted list is considered at each iteration (line 10). Lines 11–20 compute the shortest elementary path from s to nodes of $I(S[i])$ and $O(S[i])$. We use the intermediate variables $d'(v)$ to store temporarily the value of $d(v)$ in order to avoid corruption in case the intersection of $I(S[i])$ and $O(S[i])$ is not empty.

4.7 Experiments

In this section, we compare, in terms of computation time, our algorithms introduced in this chapter to the state of the art algorithms in solving the two problems ESPP(s, t, G) and ELPP(G). The characteris-

Algorithm 9: *ESP_FltC_SCC(s,t,G)*

```
1  foreach u, v ∈ V_G do
2      d(u, v) ← L ;
3  foreach v ∈ V_G do
4      d(v) ← L; d'(v) ← L;
5  SCC ← compute the set of strongly connected components of G;
6  sz ← |SCC|;
7  foreach c_i ∈ SCC, u ∈ I(c_i), v ∈ O(c_i) do
8      d(u, v) ← solveESPP(u, v, c_i);
9  S[1..sz] ← SortTopologocialOrdering(SCC) ;
10 for i ← 1 to sz do
11     foreach v ∈ I(S[i]) do
12         foreach (u, v) ∈ A_G such that u and v are not in the
           same strongly connected component do
13             if d(v) > d(u) + c_G(u, v) then
14                 d(v) ← d(u) + c_G(u, v);
15     foreach v ∈ O(S[i]) do
16         foreach u ∈ I(S[i]) do
17             if d'(v) > d(u) + d(u, v) then
18                 d'(v) ← d(u) + d(u, v);
19     foreach v ∈ O(S[i]) do
20         d(v) ← d'(v);
21 return d(t);
```

tics of all the compared algorithms are summarized in Figure 4.12. The column "Reference" gives references providing experimental results.

4.7.1 Instances

We created a total of six classes of instances for the two problems to test the algorithms.

Instances for ESPP(s, t, G)

To test the algorithms in solving ESPP(s, t, G), we created the following three classes of instances:

- **SI1** consists of 15,000 instances, each instance was created from one of 420 directed graphs with one random pair of source and destination nodes. All the directed graphs were created in [33], in which 150 directed graphs were randomly generated and 270 directed graphs were extracted from the pricing sub-problems by a heuristic column generation algorithm for the asymmetric m-salesman travelling salesman problem [88]. All graphs contain at least one negative cycle. More details about the graphs are given in Table 4.1.

Graph group	Type	No. graphs	$\|V\|$	$\|A\|$	Arc cost range	Arc cost type
R_sparse_25	Random	20	26	300	$[-10; 10]$	Integer
R_sparse_50	Random	20	51	1225	$[-10; 10]$	Integer
R_sparse_100	Random	20	101	4950	$[-10; 10]$	Integer
R_dense_25	Random	30	26	553	$[-1000; 1000]$	Double
R_dense_50	Random	30	51	2353	$[-1000; 1000]$	Double
R_dense_100	Random	30	101	9703	$[-1000; 1000]$	Double
P_first_25	Pricing	30	28	651	$[-10^8; -9.48.10^7]$	Double
P_first_50	Pricing	30	53	2551	$[-10^8; -9.48.10^7]$	Double
P_first_100	Pricing	30	103	10101	$[-10^8; -9.48.10^7]$	Double
P_penultimate_25	Pricing	30	28	651	$[-10^7; 30000]$	Double
P_penultimate_50	Pricing	30	53	2551	$[-10^7; 30000]$	Double
P_penultimate_100	Pricing	30	103	10101	$[-10^7; 30000]$	Double
P_last_25	Pricing	30	28	651	$[-30000; 30000]$	Double
P_last_50	Pricing	30	53	2551	$[-30000; 30000]$	Double
P_last_100	Pricing	30	103	10101	$[-30000; 30000]$	Double

Table 4.1: 420 directed graphs used to create 15,000 instances of the class **SI1**

- **SI2** consists of 5,000 instances that were created from 10 random connected directed graphs with random pairs of source and destination nodes. These directed graphs have many bridge-blocks. In each digraph, the cost of its arcs was generated by an uniform

distribution with the range [-100; 100] and one-third of its arcs have a negative cost. In 5 out of 10 directed graphs, each bridge-block includes exactly 20 nodes. In the 5 other directed graphs, each of them has exactly 10 bridge-blocks.

- **SI3** consists of 3,000 instances that were created from 15 random connected directed graphs with many strongly connected components (50 nodes per one strongly connected component). In each digraph, the cost of the arcs was generated by an uniform distribution with the range $[-10.0^6, 10.0^6]$ and one-third of the arcs have a negative cost. More details about the directed graphs are given in Table 4.2.

Class	No. graphs	$\|V\|$	$\|A\|$	$\sharp SCC$	Arc cost range
g_scc_100	5	100	1903-2005	2	$[-10.0^6, 10.0^6]$
g_scc_500	5	500	10051-10068	10	$[-10.0^6, 10.0^6]$
g_scc_1000	5	1000	20130-20148	20	$[-10.0^6, 10.0^6]$

Table 4.2: 15 directed graphs with many strongly connected component were used to generate 3,000 instances of the class **SI3**

Instances for ELPP(G)

- **LI1** consists of 9 instances from 9 random sparse planar undirected graphs without negative-cost arcs. These instances are taken from [84].

- **LI2** consists of 10 instances from 10 random planar connected unidirected graphs without negative-cost arcs. These graphs are built to include many bridge-blocks (10 nodes per bridge-block) and are also taken from [84].

- **LI3** consists of 10 instances created from 10 connected directed graphs with bridges. In each digraph, the costs of the arcs were generated by a uniform distribution with range [-100; 100] and one-third of the arcs have a negative cost.

4.7.2 Settings

We implemented all the algorithms in C++. For the algorithms based on MIP, we used IBM Ilog Cplex Concert Technology version 12.4 with appropriate settings. For the algorithms based on constraint programming we use the Comet language [1]. The experiments were performed on XEN virtual machines with 1 core of a CPU Intel Core2 Quad Q6600 @2.40GHz and 1 GB of RAM running Linux. A time limit was set, of 20 minutes of CPU time, for each instance.

As discussion in subsection 4.5.6, the step of inequality selection keeps only really interesting violated inequalities in the inequality pool. To do that well, two difficulty questions are raised. In order to answer these questions in an empirical way, we use a tool for parameters tuning **irace** [89]. The **irace** package implements the iterated racing procedure; its main purpose is to automatically configure optimization algorithms by finding the most appropriate settings given a set of instances of an optimization problem. In order to achieve a fair comparison between algorithms, we generated randomly a set of training instances which are different from the test instances; these training instances were only used for **irace**. The task of **irace** is to determine

- among various classes of valid inequalities, which should be separated;
- if a class of valid inequalities should be separated, what is the threshold for this class of valid inequalities (only separated inequalities with violations at least equal to this threshold are kept into the inequality pool.);
- among two separation algorithms for SECs, which should be carried out.

Figure 4.13 is a configuration file for **irace**. It declares name of parameters, type of parameters, value space of parameters and condition defined on parameters. Meaning of this configuration is as follows.

- Three parameters “sec_ta”, “sec_na” and “sec_th” are associated with the class of SECs. Parameter “sec_ta” takes the value of 1 it means that the traditional separation algorithm for SECs is

carried out. Parameter "sec_na" takes the value of 1 if and only if the value of parameter "sec_ta" is assigned to 0. The value of "sec_na" is 1 it means that our heuristic separation algorithm for SECs is used. Parameter "sec_th" represents the violation threshold for SECs. Note that SECs are always be separated.

- Parameter "2mc" (resp. "mo", "dk" and "tk") represents whether or not 2-matching inequalities (resp. maximum out flow inequalities, D_k inequalities, T_k inequalities) are separated and if they are separated ("2mc") then parameter "2mc_th" will be tuned.

- Other classes of inequalities that are valid for a general MIPP are separated if and only if parameter "oi" takes the value of 1.

After using **irace**, we decide to the following parameter for *ESP_FltC* ("sec_ta" = 0, "sec_na" = 1, "sec_th" = 0.25, "2mc" = 1, "2mc_th" = 0.1, "dk" = 0, "tk" = 0, "mo" = 1, "mo_th" = 0.01, "io" = 1).

In the two algorithms *ESP_FltC_BB* and *ESP_FltC_SCC*, from Sections 4.6.4 and 4.6.5, the elementary shortest path between two specified nodes in each block and strongly connected component is computed by *ESP_FltC*.

In the algorithm *ELP_FltC_BB*, based on the equivalence between the problems in Section 4.1, we slightly adapted *ESP_FltC* to solve ELPP(S, T, G) by the following methods: solveELPP($\{u\}, \{v\}, G(S_i)$), solveELPP($\{r_i\}, S_i \setminus Exit(r_i), G(S_i)$), solveELPP($S_i \setminus Exit(r_i), \{r_i\}$, $G(S_i)$)), and solveELPP($G(S_i), f^*$)).

4.7.3 Analysis of classes of valid inequalities

In order to analyze the effect of the different classes of valid inequalities, we compare the respective influence of the classes. We choose randomly 60 instances from the class of instances **SI1** as test instances in this comparison. Among these instances, 30 instances were created from 30 distinct directed graph "P_last_100" and 30 remaining instances were created from 30 distinct directed graph "P_penultimate_100". The comparison results are given in Table 4.3. This table reports the total computation times of *ESP_FltC* in the last column for solving all the 60 instances using 9 different parameter sets. In this table, columns "♯"

give the number of separated inequalities, the last column presents the total number of separated inequalities and the last row corresponds to the parameter set tuned by **irace**.

Taking individually inequalities actually increases the total number of separated inequalities. The setting obtained by **irace** produces the smallest number of separated inequalities and being the most efficient.

4.7.4 Reduction of ESPP to STSP

As showed in section 4.4, we can transform an ESPP instance of n nodes into an equivalent ATSP instance of $2n$ nodes. Moreover, Roy Jonker et al. proposed a procedure that transforms an ATSP instance of n nodes into an equivalent STSP instance of $2n$ nodes [59]. So, we can transform an ESPP instance of n nodes into a STSP instance of $4n$ nodes. In order to show the interest of our algorithm *ESP_FltC* for solving ESPP, we compare the total computation time needed by *ESP_FltC* for solving some ESPP instances to the total computation time needed by Concorde (the strongest solver for solving STSP [27]), for solving equivalent STSP instances that are obtained from the ESPP instances.

We select randomly, from the class of instances **SI1**, 30 instances of "R_dense_25", 30 instances of "R_dense_50", 30 instances of "R_dense_100" and 30 instances of "P_last_25" as experiment test for this comparison. The experimental results are given in Table 4.4. In this table, last two columns present the total computation time of *ESP_FltC* and the solver Concorde to solve instances. From the result in this table, we see that our algorithm *ESP_FltC* is better than a reduction to STSP. The difference is particularly significative for pricing instances.

4.7.5 Solving ESPP(s, t, G)

The state of the art algorithm for ESPP(s, t, G), proposed in [33] and using the Arc-flow formulation (Model 1), is denoted by *ESP_Drexl*. We re-implemented this algorithm to compare it with the algorithms proposed in this dissertation: *ESP_MTZ*, *ESP_ComFlow*, *ESP_FltC*, *ESP_FltC_BB*, and *ESP_FltC_SCC*. Note that in order to make a fairness between algorithms in this experiment, the prepossessing proposed

Table 4.3: Effect of the different classes of valid inequalities

SECs				2-matching			Maximum outflow			D_k			T_k			others				
sec_ta	sec_na	sec_th	♯	2mc	2mc_th	♯	mo	mo_th	♯	dk	dk_th	♯	tk	tk_th	♯	oi	♯	Time (s)		
1	0	10^{-5}	106205	0	-	-	0	-	-	0	-	-	0	-	-	0	-	2362.5	106205	
0	1	10^{-5}	103474	0	-	-	0	-	-	0	-	-	0	-	-	0	-	1945.1	103474	
0	1	10^{-5}	84900	1	10^{-5}	26180	0	-	-	0	-	-	0	-	-	0	-	1641.7	111080	
0	1	10^{-5}	69006	0	-	-	1	10^{-5}	315	0	-	-	0	-	-	0	-	2049.4	69321	
0	1	10^{-5}	78151	0	-	-	0	-	-	1	10^{-5}	84894	0	-	-	0	-	5729.7	163045	
0	1	10^{-5}	90807	0	-	-	0	-	-	0	-	-	1	10^{-5}	85695	0	-	1945.1	176502	
0	1	10^{-5}	92661	0	-	-	0	-	-	0	-	-	0	-	-	1	502	1455.7	93163	
0	1	10^{-5}	54836	1	10^{-5}	6179	1	10^{-5}	248	1	10^{-5}	8842	1	10^{-5}	194	1	0	1774.2	70105	
0	1	0.25	53359	1	0.1	5306	1	0.01	257	0	-	-	0	-	-	1	19	1111.2	58941	

Instance type	No. instances	*ESP_FltC* (s)	Concorde (s)
R_dense_25	30	0.74	7.24
R_dense_50	30	4.13	16.64
R_dense_100	30	20.03	51.78
P_last_25	30	1.37	5067.58

Table 4.4: Comparing *ESP_FltC* to Concorde for solving ESPP instances

in subsection 4.5.2 was integrated into all algorithms.

Solving 15,000 instances of the class SI1

All instances of the class **SI1** were tested to compare 4 algorithms *ESP_Drexl, ESP_ComFlow, ESP_MTZ* and *ESP_FltC* in terms of their computation time.

Instance group	Algorithm	% optimal	B&B nodes (min./avg./max.)	Computation time (min./avg./max.)
Random_sparse_25	*ESP_ComFlow*	100	0/4.71/69	0.15/2.96/13.98
	ESP_MTZ	0	-/-/-	-/-/-
	ESP_Drexl	100	0/2.14/55	0.03/0.08/0.72
	ESP_FltC	100	0/1.82/47	0.02/0.16/15.92
Random_sparse_50	*ESP_ComFlow*	93.6	0/9.13/136	54.18/375.77/1194.59
	ESP_MTZ	0	-/-/-	-/-/-
	ESP_Drexl	100	0/5.52/159	0.06/0.45/18.64
	ESP_FltC	100	0/4.78/123	0.04/0.66/33.78
Random_sparse_100	*ESP_ComFlow*	0	-/-/-	-/-/-
	ESP_MTZ	0	-/-/-	-/-/-
	ESP_Drexl	100	0/4.25/146	0.23/2.09/31.01
	ESP_FltC	100	0/3.71/283	0.07/2.12/31.31

Table 4.5: Computational results for random sparse instances

The computational results are shown in Tables 4.5, 4.6, 4.7, 4.8, and 4.9, and summarized in Table 4.10. The tables contains the class of test instances and the instances as described in Section 4.7.1 (*Instance class* and *Instances*); the algorithm used to solve the problem (*Algorithm*); The number of nodes ($|V|$), the number of arcs ($|A|$), the number of edges ($|E|$), the number of bridge-blocks ($|B|$), the number of strongly

Instance group	Algorithm	% optimal	B&B nodes (min./avg./max.)	Computation time (min./avg./max.)
Random_dense_25	*ESP_ComFlow*	100	0/5.19/75	1.35/7.66/36.76
	ESP_MTZ	0	-/-/-	-/-/-
	ESP_Drexl	100	0/3/64	0.03/0.12/2.66
	ESP_FltC	100	0/2.55/48	0.02/0.1/2.99
Random_dense_50	*ESP_ComFlow*	51.7	0/3.19/35	151.58/525.98/1199
	ESP_MTZ	0	-/-/-	-/-/-
	ESP_Drexl	100	0/10.39/154	0.08/1.25/19.56
	ESP_FltC	100	0/9.6/100	0.06/0.3/1.21
Random_dense_100	*ESP_ComFlow*	0	-/-/-	-/-/-
	ESP_MTZ	0	-/-/-	-/-/-
	ESP_Drexl	100	0/22.07/236	0.33/4.07/52.44
	ESP_FltC	100	0/19.81/223	0.32/2.94/34.97

Table 4.6: Computational results for random dense instances

Instance group	Algorithm	% optimal	B&B nodes (min./avg./max.)	Computation time (min./avg./max.)
P_first_25	*ESP_ComFlow*	100	0/9.16/404	1.92/9.5/63.97
	ESP_MTZ	0	-/-/-	-/-/-
	ESP_Drexl	100	0/7.65/222	0.03/0.19/14.08
	ESP_FltC	100	0/3.61/147	0.02/0.09/2.35
P_first_50	*ESP_ComFlow*	51.7	0/6.89/61	142.13/689.63/1798.69
	ESP_MTZ	0	-/-/-	-/-/-
	ESP_Drexl	100	0/35.03/1003	0.08/1.11/55.66
	ESP_FltC	100	0/24.51/1659	0.06/1.19/35.32
P_first_100	*ESP_ComFlow*	0	-/-/-	-/-/-
	ESP_MTZ	0	-/-/-	-/-/-
	ESP_Drexl	97.4	0/917.84/10845	0/52.54/1142.01
	ESP_FltC	99.2	0/871.63/15453	0.91/96.74/1181.18

Table 4.7: Computational results for pricing instances

connected components ($|SCC|$); the percentage of instances solved to optimality in a given limited computation time (*% Optimal*); the number of nodes in the branch-and-bound tree (*B & B nodes*); and the overall CPU time in seconds (*Computation time*). For the rightmost columns, we give the minimum, average, and maximum values (*min./avg./max.*).

First of all, from the computational results we conclude that the algorithms *ESP_ComFlow* and *ESP_MTZ* are not efficient in solving

Instance group	Algorithm	% optimal	B&B nodes (min./avg./max.)	Computation time (min./avg./max.)
P_penultimate_25	*ESP_ComFlow*	100	0/0.93/149	0.93/3.73/18.38
	ESP_MTZ	82.6	0/361033/2494670	0.06/176.94/1196.37
	ESP_Drexl	100	0/7.6/109	0.03/0.11/13
	ESP_FltC	100	0/1.04/62	0.02/0.09/3.57
P_penultimate_50	*ESP_ComFlow*	98.8	0/0.75/58	1.0/216.95/1182.66
	ESP_MTZ	19.4	0/339502/1262260	0.19/385.01/1144.67
	ESP_Drexl	100	0/15.89/252	0/1.16/53/04
	ESP_FltC	100	0/3.86/86	0/0.53/8.09
P_penultimate_100	*ESP_ComFlow*	0	-/-/-	-/-/-
	ESP_MTZ	0	-/-/-	-/-/-
	ESP_Drexl	97.5	9/1279.99/11565	1.42/108.71/1185.39
	ESP_FltC	99.9	0/398.44/9935	0/48.74/1114.69

Table 4.8: Computational results for pricing instances (continued)

Instance group	Algorithm	% optimal	B&B nodes (min./avg./max.)	Computation time (min./avg./max.)
P_last_25	*ESP_ComFlow*	100	0/1.04/74	1.45/4.74/26.52
	ESP_MTZ	0	-/-/-	-/-/-
	ESP_Drexl	100	0/13.95/285	0.04/0.13/5.02
	ESP_FltC	100	0/1.76/64	0.03/0.11/4.67
P_last_50	*ESP_ComFlow*	95.2	0/2.99/203	0.68/256.19/1198.29
	ESP_MTZ	3	0/0/0	0.36/0.44/0.57
	ESP_Drexl	100	0/70.13/1859	0/4.62/215.17
	ESP_FltC	100	0/15.72/600	0/1.13/33.24
P_last_100	*ESP_ComFlow*	0	-/-/-	-/-/-
	ESP_MTZ	0	-/-/-	-/-/-
	ESP_Drexl	95.6	9/1505.53/10256	0.88/114.62/1186.68
	ESP_FltC	99.8	0/512.88/9649	0.85/58.62/1040.2

Table 4.9: Computational results for pricing instances (continued)

ESPP(s, t, G) on medium and large-sized graphs compared with the two other algorithms *ESP_Drexl* and *ESP_FltC*. Within the time limit of 20 minutes, *ESP_MTZ* solved instances of 25 nodes with a large computation time and could not solve any instance of 50 and 100 nodes. A weakness of *ESP_MTZ* is the number of nodes in its branch-and-bound tree. While *ESP_ComFlow* can solve some instances of 50 nodes, it cannot solve any instance of 100 nodes because the number of variables

Algorithm	% optimal	B&B nodes (min./avg./max.)	Computation time (min./avg./max.)
ESP_ComFlow	62.62	0/4.25/404	0.93/183.23/1798.69
ESP_MTZ	7	0/38760.42/2494670	0.06/210.34/1196.37
ESP_Drexl	99.36	0/253.53/11565	0/18.93/1186.68
ESP_FltC	99.93	0/124.58/15453	0/14.18/1181.18

Table 4.10: Computational results over all 15,000 instances

in Model 2 (used in *ESP_ComFlow*) is too large.

Next, Tables (4.5–4.9) and Figures (4.14–4.16) show that *ESP_FltC* outperforms *ESP_Drexl* in solving 15,000 instances of the class **SI1**. Over all 15,000 instances, within a time limit of 20 minutes, *ESP_FltC* solved 99.93% of all instances, while *ESP_Drexl* solved 99.36 % of all instances. The difference is not significant. However, when solving the 3,000 difficult instances with 100 nodes of the three groups *P_first_100, P_last_but_one_100* and *P_last_100*, our algorithm is faster than the state of the art algorithm. For example, in 1,000 instances of the group *P_last_100*, *ESP_FltC* solved 998 instances with an average computation time of 58.62 seconds, while *ESP_Drexl* solved just 956 instances, with an average computation time of 114.62 seconds. Moreover, Table 4.14 shows that *ESP_FltC* is much better than *ESP_Drexl* when the time limit is short.

Solving 5000 instances of the class SI2

We compare the algorithms *ESP_Drexl*, *ESP_FltC* and *ESP_FltC_BB* in solving 5,000 instances of the class **SI2**. All instances were created from connected directed graphs with many bridges. The computational results are given in Table 4.11. Note that as *ESP_FltC_BB* computes many times the elementary shortest path in blocks, the number of nodes in the branch-and-bound tree is meaningless.

From the computational results we conclude that among the three algorithms, *ESP_FltC_BB* is the best and *ESP_Drexl* is the worst in solving 5,000 instances of the class **SI2** in terms of computation time. For instance,

Instance group	$\|V\|$	$\|A\|$	$\|B\|$	Algorithm	% optimal	B&B nodes (min./avg./max.)	Computation time (min./avg./max.)
g_bb_500	500	6274	25	*ESP_Drexl*	100	0/2.31/86	102.8/272.92/1140.66
				ESP_FltC	100	0/2.49/121	0.07/0.75/9.55
				ESP_FltC_BB	100	na/na/na	0.05/0.33/8.25
g_bb_1000	1000	12549	50	*ESP_Drexl*	65	0/5.98/21	71.42/534.76/1193.88
				ESP_FltC	100	0/9.26/242	0.14/1.29/20.68
				ESP_FltC_BB	100	na/na/na	0.09/0.52/3.56
g_bb_1500	1500	18824	75	*ESP_Drexl*	2	0/1/2	632.17/879.6/1127.02
				ESP_FltC	100	0/9.56/411	0.25/6.47/81.33
				ESP_FltC_BB	100	na/na/na	0.15/8.65/152.08
g_bb_2000	2000	25099	100	*ESP_Drexl*	0	na/na/na	na/na/na
				ESP_FltC	100	0/9.26/210	0.39/9.87/86.22
				ESP_FltC_BB	100	na/na/na	0.25/8.41/165.56
g_bb_2500	2500	31374	125	*ESP_Drexl*	0	na/na/na	na/na/na
				ESP_FltC	100	0/7.93/114	0.51/7.85/109.72
				ESP_FltC_BB	100	na/na/na	0.3/15.88/185.05
g_bb_200	200	2009	10	*ESP_Drexl*	100	0/2.55/39	2.57/8.37/41.4
				ESP_FltC	100	0/2.64/56	0.03/0.41/6.81
				ESP_FltC_BB	100	na/na/na	0.03/0.62/74.5
g_bb_400	400	4009	10	*ESP_Drexl*	100	0/27.42/426	5.15/144.82/1148.39
				ESP_FltC	100	0/13.24/447	0.06/1.97/53.13
				ESP_FltC_BB	100	na/na/na	0.06/1.26/14.08
g_bb_600	600	6009	10	*ESP_Drexl*	79.2	0/18.79/98	17.28/383.29/1191.95
				ESP_FltC	99.4	0/132.52/3147	0.12/32.83/1188.2
				ESP_FltC_BB	100	na/na/na	0.09/8.65/89.19
g_bb_800	800	8009	10	*ESP_Drexl*	64	0/9.55/35	38.66/554.99/1199.02
				ESP_FltC	99.6	0/42.33/555	0.18/22.09/1142.31
				ESP_FltC_BB	100	na/na/na	0.15/9.85/199.1
g_bb_1000	1000	10009	10	*ESP_Drexl*	19.8	0/6.15/21	72.3/504.25/1174.62
				ESP_FltC	85.8	0/115.64/1434	0.39/141.9/1189.99
				ESP_FltC_BB	100	na/na/na	0.45/49.12/664.65

Table 4.11: Computational results for instances of the class **SI2**

- As to the 500 instances of the group *g_bb_2500*, *ESP_Drexl* could not solve any instances because of an 'out of memory' error, while *ESP_FltC* and *ESP_FltC_BB* solved all 500 instances with average computation times of 7.85 seconds and 15.88 seconds, respectively.

- As to the 500 instances of the group *g_bb_2500*, *ESP_Drexl* solved 99 instances (19.8%) with an average computation time of 504.25 seconds, while *ESP_FltC* solved 429 instances with an average computation time of 141.9 seconds and *ESP_FltC_BB* solved all 500 instances, with an average computation time of 49.12 seconds.

The efficiency of *ESP_FltC_BB* is analyzed as follows. It is a dynamic programming algorithm that benefits from the properties of the instances where graphs have many bridges. Firstly it computes many elementary shortest paths in blocks. These computations are very fast because the number of nodes in each block is very small, for instance, 20 nodes per one block. Finally it merges appropriately computed paths

into a complete solution path.

The computation time for the instances in the group *g_bb_2000* (instances of 2,000 nodes) is much smaller than that of instances in the group *g_bb_1000* (instances of 1,000 nodes), 8.41 seconds compared to 49.12 seconds. The reason is that the number of nodes per block for the instances of 2,000 nodes is only 20, while there are 100 nodes in each block of one of the instances of 1,000 nodes.

In this experiment, *ESP_FltC* clearly outperforms *ESP_Drexl*. For example, *ESP_Drexl* cannot solve any instances of the groups *g_bb_2000* and *g_bb_2,500* while *ESP_FltC* solved all 1,000 instances in these two classes. This efficiency mainly comes from the preprocessing carried out in *ESP_FltC*. Table 4.12 shows the efficiency of this preprocessing. For some instances of the group *g_bb_2500*, a graph of 2,500 nodes can be reduced to a graph of 20 nodes, as both the source and destination nodes are in the same block of 20 nodes. For some other instances of the group *g_bb_1000*, the reduction does not reduce the number of nodes. On average, the size of the graphs is greatly reduced, as illustrated in Table 4.12 for the 500 instances of the group *g_bb_2500*.

	Before		After	
Instance group	$\|V\|$	$\|A\|$	$\|V\|$ (min./avg./max.)	$\|A\|$ (min./avg./max.)
g_bb_500	500	6274	20/58.35/400	219/706.5/4993
g_bb_1000	1000	12549	20/57.14/300	220/691.32/3739
g_bb_1500	1500	18824	20/75.81/380	220/925.71/4742
g_bb_2000	2000	25099	20/75.72/520	218/924.57/6499
g_bb_2500	2500	31374	20/86.82/380	220/1063.71/4745
g_bb_200	200	2009	20/51.9/200	172/500.35/1994
g_bb_400	400	4009	39/104.87/399	369/1032.39/3992
g_bb_600	600	6009	60/139.94/600	567/1380.37/5993
g_bb_800	800	8009	80/259.25/800	768/2574.26/7995
g_bb_1000	1000	10009	100/306.02/1000	967/3041.99/9993

Table 4.12: Reducing the graph size of the instances in the class **SI2** by using the preprocessing by the algorithm *ESP_FltC*

Solving 3,000 instances of the class SI3

In this section, we compare the algorithms *ESP_Drexl*, *ESP_FltC* and *ESP_FltC_SCC* in solving 3,000 instances of the class **SI3** that are directed graphs with many strongly connected components. The computational results are given in Table 4.13.

Instance class	$\|V\|$	Algorithm	% optimal	B&B nodes (min./avg./max.)	Computation time (min./avg./max.)
g_scc_100	100	*ESP_Drexl*	100	0/10.7/296	0.15/2.63/226.45
		ESP_FltC	100	0/14/287	0.05/0.84/18.15
		ESP_FltC_SCC	100	-/-/-	0.04/0.83/6.88
g_scc_500	500	*ESP_Drexl*	75.6	0/22.82/167	8.11/230.04/1195.56
		ESP_FltC	94.1	0/124.59/4258	0.1/55.68/1197.87
		ESP_FltC_SCC	99.3	-/-/-	0.06/27.58/1196.81
g_scc_1000	1000	*ESP_Drexl*	7.2	0/0.19/12	53.02/118.48/1040.27
		ESP_FltC	77.1	0/184.82/54438	0.16/96.16/1195.28
		ESP_FltC_SCC	91.5	-/-/-	0.09/84.21/1122.21

Table 4.13: Computational results for instances of the class **SI3**

From the computational results, we see that *ESP_FltC_SCC* is much more efficient than the other two algorithms in terms of computation time. To reach this efficiency, the dynamic programming algorithm *ESP_FltC_SCC* exploits the special property of instances where the graphs have many strongly connected components.

As for solving instances of the class **SI2**, we also see that *ESP_FltC* is much better than *ESP_Drexl* at solving the 3,000 instances of the class **SI3**. This mainly comes from the preprocessing by *ESP_FltC*, as shown in Table 4.14.

4.7.6 Solving ELPP(G)

We showed that the two problems ESPP(s, t, G) and ELPP(G) are equivalent to each other. Based on this equivalence, we slightly adapt the two algorithms *ESP_FltC* and *ESP_Drexl* to solve ELPP(G). We obtain two new algorithms for solving ELPP(G): *ELP_FltC* and *ELP_Drexl*.

In addition, we propose in this chapter the algorithm *ELP_FltC_BB* to solve ELPP(G) on directed graphs with many bridges. This algorithm uses some different MIP search methods (solveELPP($\{r_i\}, S_i \setminus$

Graph class	Before		After	
	$\|V\|$	$\|A\|$	$\|V\|$ (min./avg./max.)	$\|A\|$ (min./avg./max.)
g_scc_100	100	1903-2005	50/66.95/100	898/1276.77/1977
g_scc_500	500	10051-10068	50/218.445/500	948/4347.46/10035
g_scc_1000	1000	20130-20148	47/283.29/1000	846/5656.28/20109

Table 4.14: Reducing the size of graphs of the instances in the class **SI3** by the use of preprocessing by the algorithm *ESP_FltC*

$Exit(r_i), G(S_i))$, solveELPP($S_i \backslash Exit(r_i), \{r_i\}, G(S_i)$) and solveELPP($G(S_i), f^*$)). We replace the MIP search methods by CP search methods in *ELP_FltC_BB* to obtain a new algorithm denoted by *ELP_CP2_BB*.

In the literature, for solving ELPP(G), there exist two state of the art algorithms, both based on CP, which were proposed in our previous chapter, [84]. The one that solves ELPP(G) on general undirected graphs is denoted by *ELP_CP*. The algorithm that solves ELPP(G) on undirected graphs with many bridges is denoted by *ELP_CP_BB*. Both are implemented in the COMET programming language [1].

Solving 9 instances of the class LI1

We here compare three algorithms in solving 9 instances of the class **LI1**. Two of them are based on MIP: they are *ELP_FltC* and *ELP_Drexl*. The last one is based on CP: it is *ELP_CP*. The time limit has been set here to 30 minutes to meet the experimental settings of *ELP_CP* in [84]. The computation time is given in Table 4.15. The two B&C algorithms solved very successfully only the first instance. *ELP_FltC* solved 5 instances, while two others could not solve any other instance. Although the graph size of the instances is not too large, they are very difficult because the costs of the edges are very similar.

Solving 10 instances of the class LI2

In this section, we compare 5 algorithms at solving ELPP(G) on undirected graphs with many bridges. The results are shown in Table 4.16. The following analysis can be made.

Instances	$\|V\|$	$\|A\|$	$\|B\|$	*ELP_FltC*	*ELP_Drexl*	*ELP_CP*
planar-n100-m285	100	285	1	0.8	3.3	-
planar-n150-m432	150	432	1	1660.42	-	-
planar-n200-m583	200	583	1	1761.42	-	-
planar-n250-m731	250	731	1	1763.04	-	-
planar-n300-m880	300	880	1	1760.9	-	-
planar-n350-m1031	350	1031	1	-	-	-
planar-n400-m1182	400	1182	1	-	-	-
planar-n450-m1329	450	1329	1	-	-	-
planar-n500-m1477	500	1477	1	-	-	-

Table 4.15: Computation times in seconds of three algorithms in solving 9 instances of the class **LI1**

Instances	$\|V\|$	$\|A\|$	$\|B\|$	*ELP_CP_BB*	*ELP_CP2_BB*	*ELP_Drexl*	*ELP_FltC*	*ELP_FltC_BB*
planar-n100-m216	100	216	10	4.0	1.95	3.34	2.43	1.09
planar-n200-m434	200	434	20	21.32	10.17	12.73	28.23	5.21
planar-n300-m655	300	655	30	54.73	25.28	68.06	192.25	92.5
planar-n400-m870	400	870	40	121.15	59.76	95.65	790.52	1.12
planar-n500-m1089	500	1089	50	235.73	111.19	267.59	-	5.8
planar-n600-m1301	600	1301	60	364.77	169.95	626.89	-	93.85
planar-n700-m1526	700	1526	70	569.15	257.78	-	-	131.3
planar-n800-m1747	800	1747	80	825.12	421.87	-	-	131.88
planar-n900-m1959	900	1959	90	1161.53	554.71	-	-	15.7
planar-n1000-m2177	1000	2177	100	-	186.58	-	-	16.05

Table 4.16: Computation times in seconds of 5 algorithms at solving 10 instances of the class **LI2**

- *ELP_CP2_BB* solved the instances much more quickly than *ELP_CP_BB*, for example, to solve the instance *planar-n1000-m2177*, *ELP_CP_BB* needed 1649.42 seconds while *ELP_CP2_BB* spent only 186.58 seconds. Both algorithms are based on CP and were implemented in the COMET programming language. The only difference between the two algorithms is that ELP_CP_BB executes only once the methods solveELPP($\{r_i\}, S_i \backslash Exit(r_i), G(S_i)$), and solveELPP($S_i \backslash Exit(r_i), \{r_i\}, G(S_i)$), while the algorithm *ELP_CP_BB* must execute many times the method solveELPP($\{u\}, \{v\}, G(S_i)$). Therefore, we conclude that this speeds up computations by *ELP_CP2_BB*.

- *ELP_FltC_BB* is much faster than *ELP_CP2_BB*. For example, to solve the instance *planar-n900-m1959*, *ELP_CP2_BB* needed 554.71 seconds while *ELP_FltC_BB* needed only 15.7 seconds. Here, too, there is only one difference between the two: *ELP_FltC_BB* uses MIP search methods while *ELP_CP2_BB* uses CP search

methods. In short, the MIP approach is more efficient than the CP approach in this problem and in this experiment.

- When we compare three dynamic algorithms, *ELP_CP_BB, ELP_CP2 _BB*, and *ELP_FltC_BB*, with the two algorithms that do not exploit the special properties of the graph (*ELP_Drexl* and *ELP_FltC*), we conclude that our proposed dynamic algorithms for solving ELPP(G) on graphs with many bridges is very efficient.

Solving 10 instances of the class LI3

Three algorithms were tested in this experiment: *ELP_Drexl*, *ELP_FltC* and *ELP_FltC_BB*. The task is to solve ELPP(G) on a directed graph with many bridges. The results in computation time are given in Table 4.17.

Instances	$\|V\|$	$\|A\|$	$\|B\|$	*ELP_Drexl*	*ELP_FltC*	*ELP_FltC_BB*
g_bb_500	500	6274	25	302	131.57	10.53
g_bb_1000	1000	12549	50	-	-	24.13
g_bb_1500	1500	18824	75	-	-	25.76
g_bb_2000	2000	25099	100	-	-	33.74
g_bb_2500	2500	31374	125	-	-	44.11
g_bb_200	200	2009	10	99.21	7.22	4.56
g_bb_400	400	4009	10	-	220.59	14.04
g_bb_600	600	6009	10	-	-	53.15
g_bb_800	800	8009	80	-	-	80.6
g_bb_1000	1000	10009	100	-	-	86.19

Table 4.17: Comparing three algorithms in solving 10 instances of the class **LI3** in terms of computation time

It is easy to see that *ELP_FltC_BB* clearly dominates the two other algorithms *ELP_Drexl* and *ELP_FltC* as it solves all instances with short computation times while the two other algorithms only solve, respectively, two and three instances.

In short, our proposed algorithm *ELP_FltC_BB* solves efficiently ELPP(G) on directed graphs with many bridges.

4.8 Decomposition for the support graph

In this section, we propose a decomposition technique applied on support graphs to detect faster and more violated SECs (the definition of the support graph in subsection 4.5.5). Basically, *the support graph is decomposed into smaller-size graphs, each of them corresponds to a strongly connected component*. After that we can detect violated SECs on these smaller-size graphs. This decomposition might be very useful for large and sparse support graphs. For an easy of presentation, we introduce additionally following notations: Given a directed support graph $\overline{G} = (V_{\overline{G}}, A_{\overline{G}}, c_{\overline{G}})$,

- $mf(i, j, \overline{G})$ is the value of maximum flow from a source i to a sink j on $\overline{G}$;
- a minimum $i - j$ cut of $\overline{G}$ divides $V_{\overline{G}}$ into disjoint subsets $V^{(i)}$ and $V_{\overline{G}} \setminus V^{(i)}$ such that $i \in V^{(i)}, j \in V \setminus V^{(i)}$ and total crossing cost is minimized;
- $F^{+}_{\overline{G}}(S) = \{i | i \in S, j \in V_{\overline{G}} \setminus S, (i,j) \in \delta^{+}_{\overline{G}}(S)\}$ and $F^{-}_{\overline{G}}(S) = \{i | i \in S, j \in V_{\overline{G}} \setminus S, (j,i) \in \delta^{-}_{\overline{G}}(S)\}$;
- for each strongly connected component scc_i of the support graph $\overline{G}$ with the node set W, we define a directed graph called *extended graph* of this strongly connected component as follows: $G_i = (V_{G_i}, A_{G_i}, c_{G_i})$, in which $V_{G_i} = W \cup \{t_i\}$, $A_{G_i} = \{(u,v) \mid (u,v) \in A_{\overline{G}}, u, v \in W\} \cup \{(v, t_i) \mid v \in F^{+}_{\overline{G}}(W)\}$ and the cost function $c_{G_i}, \forall (u,v) \in A_{G_i}$,

$$c_{G_i}(u,v) = \begin{cases} c_s(u,v) & u, v \in W \\ \sum_{(u,v) \in \delta^{+}_{\overline{G}}(W)} c_{\overline{G}}(u,v) & v = t_i, u \in F^{+}_{\overline{G}}(W) \end{cases}$$

Illustratively, Figure 4.17 presents a directed support graphs, while Figure 4.18 presents three extended graphs corresponding to three strongly connected components of the support graph.

4.8.1 Properties of the support graph

In this subsection, two properties of $\overline{G}$ will be pointed out in two following theorems:

Theorem 3 *On $\overline{G}$, if a node i does not belong to any cycle then $mf(i,t,\overline{G}) = c_{\overline{G}}(\{i\})$.*

Proof Assuming that $i \in V_{\overline{G}}$ does not belong to any cycle on $\overline{G}$.

We denote $X \subseteq V_{\overline{G}}$ be the set of nodes that lie on a path traveling from i to t on $\overline{G}$ (conveniently, $i, t \in X$). A directed graph $\overline{G'} = (V_{\overline{G'}}, A_{\overline{G'}}, c_{\overline{G'}})$ is constructed for the set of nodes X as follows: the set of nodes $V_{\overline{G'}} = X \cup \{s'\}$, $A_{\overline{G'}} = \{(u,v)|(u,v) \in A_{\overline{G}}, u, v \in X\} \cup \left\{(s',v)|v \in F^-_{\overline{G}}(X)\right\}$; and the cost function $c_{\overline{G'}}$

$$c_{\overline{G'}}(u,v) = \begin{cases} c_{\overline{G}}(u,v) & u,v \in X, (u,v) \in A_{\overline{G}} \\ \sum_{(j,v)\in A_{\overline{G}}, j \notin X} c_{\overline{G}}(j,v) & u = s', v \in F^-_{\overline{G}}(X) \end{cases}$$

On the graph $\overline{G'}$, we have

$$\delta^-_{\overline{G'}}(s') = \{\emptyset\} \tag{4.10}$$

and

$$\delta^+_{\overline{G'}}(t) = \{\emptyset\} \tag{4.11}$$

We have

$$\forall v \in X \setminus \{s', t\},\ c_{\overline{G'}}(\delta^+_{\overline{G'}}(v)) = c_{\overline{G'}}(\delta^-_{\overline{G'}}(v)). \tag{4.12}$$

As results of (4.10), (4.11), (4.12) and the definition of the maximum flow value [5], we get

$$mf(s', t, G) = c_{\overline{G'}}(\delta^+_{\overline{G'}}(s')) \tag{4.13}$$

By assumption, node i does not belonging to any cycle on $\overline{G}$, therefore there exists only one way to walk from s' to i on $\overline{G'}$, that is to travel directly through arc (s', i), as a result of this and (4.12),

$$c_{\overline{G'}}(s', i) = c_{\overline{G'}}(\delta^-_{\overline{G'}}(i)) = c_{\overline{G'}}(\delta^+_{\overline{G'}}(i)) \tag{4.14}$$

From (4.13), (4.14), the procedure used to construct $\overline{G'}$ and the property that node i does not belonging to any cycle on $\overline{G}$, we have that there exists a flow of value of $\delta^+_{\overline{G'}}(i)$ passing from source i to sink t that uses only arcs $\overline{G}$.

From above result, the theorem holds.

Theorem 4 *Let scc_i be a strongly connected component of $\overline{G}$ and $G_i = (V_{G_i}, A_{G_i}, c_{G_i})$ be the extended graph of scc_i. For each node v in V_{G_i}, we have*

- $mf(v, t, \overline{G}) = mf(v, t_i, G_i)$;
- *and if $(V^{(v)}, V_{G_i} \setminus V^{(v)})$ is a v-t_i min-cut on G_i, then $(V^{(v)}, V_{\overline{G}} \setminus V^{(v)})$ is a v-t min-cut on $\overline{G}$.*

Proof We construct a graph $\overline{G'} = \{V_{\overline{G'}}, A_{\overline{G'}}, c_{\overline{G'}}\}$ as follows:

- $V_{\overline{G'}} = V_{\overline{G}} \cup \{t_i\}$;
- $A_{\overline{G'}} = A_{\overline{G}} \cup \mathcal{X} \setminus \delta^+_{\overline{G}}(V_{G_i})$, in which $\mathcal{X} = \{(u, t_i), (t_i, v) \mid (u, v) \in \delta^+_{\overline{G}}(V_{G_i})\}$
- and the cost function, $\forall (u, v) \in A_{\overline{G'}}$,

$$c_{\overline{G'}}(u, v) = \begin{cases} c_{\overline{G}}(u, v) & u, v \in V_{\overline{G}} \\ \sum_{(u,j) \in \delta^+_{\overline{G}}(V_{G_i})} c_{\overline{G}}(u, j) & u \in F^+_{\overline{G}}(V_{G_i}), v = t_i \\ \sum_{(j,v) \in \delta^+_{\overline{G}}(V_{scc_i})} c_{\overline{G}}(j, v) & u = t_i, (u, v) \in A_{\overline{G'}} \end{cases}$$

An example of this construction is given in Figure 4.19.
From the way used to construct $\overline{G'}$ and G_i, we have,

$$mf(v, t, \overline{G}) = mf(v, t, \overline{G'}) \tag{4.15}$$

and

$$mf(v, t_i, \overline{G'}) = mf(v, t_i, G_i) \tag{4.16}$$

On $\overline{G'}$, every flow passing from v to t must travel through t_i, so we get

$$\begin{aligned} mf(v,t,\overline{G'}) &= \min\left(mf(v,t_i,\overline{G'}), mf(t_i,t,\overline{G'})\right) \\ &= \min\left(mf(v,t_i,G_i), mf(t_i,t,\overline{G'})\right) \end{aligned} \tag{4.17}$$

Because t_i does not belong to any cycle of $\overline{G'}$, so applying Theorem 3 we have

$$mf(t_i,t,\overline{G'}) = c_{\overline{G'}}(\delta^+_{\overline{G'}}(t_i)) = c_{\overline{G'}}(\delta^-_{\overline{G'}}(t_i)) \tag{4.18}$$

Looking at the definition of the maximum flow in [5], we get

$$mf(v,t_i,G_i) \leq c_{G_i}(\delta^-_{G_i}(t_i)) \tag{4.19}$$

Finally, from (4.15), (4.16), (4.17), (4.18) and (4.19), we have

$$mf(v,t,\overline{G}) = mf(v,t_i,G_i)$$

Finally, of course, if $(V^{(v)}, V_{G_i} \setminus V^{(v)})$ is a v-t_i min-cut on G_i, then $(V^{(v)}, V_{\overline{G}} \setminus V^{(v)})$ is also a v-t min-cut on $\overline{G}$.

From above theorems, we have two following observations:

1. If a strongly connected component of $\overline{G}$ consists of only one node v, then every SEC $\langle S, v\rangle$ is not violated by $\overline{x}$.

2. We can detect SECs violated by $\overline{x}$ via solving max-flow/min-cut problems on the extended graphs of the strongly connected components of $\overline{G}$ and their auxiliary undirected versions instead of solving max-flow/min-cut problems on the directed support graph $\overline{G}$ and the auxiliary undirected support graph of $\overline{G}$, respectively.

4.8.2 Promising separation algorithm for SECs

We created four different separation algorithms for SECs and compare them with respect to the number of violated SECs found and the computation time.

- The first algorithm, denoted by **Algo1**, detects violated SECs via solving max-flow/min-cut problems on the directed support graph.
- The second algorithm, which is denoted by **Algo2**, detects violated SECs via solving max-flow/min-cut problems on the extended graphs corresponding to strongly connected components of the directed support graph.
- The third algorithm, which is denoted by **Algo3**, detects violated SECs via solving max-flow/min-cut problems on the auxiliary undirected graph of the directed support graph.
- The last algorithm, which is denoted by **Algo4**, detects violated SECs via solving max-flow/min-cut problems on the auxiliary undirected graphs of the extended graphs corresponding to strongly connected components of the directed support graph.

In order to compare 4 separation algorithms, we collected totally 4447 directed support graphs during solving 20 random ESPP instances of 500 nodes by a B&C algorithm. These directed support graphs are divided into 2 groups; the first one consists of 4306 graphs which contain only one strongly connected component of more than one node; the remaining graphs, which contain more than one strongly connected component of at least two nodes, are collected in the second group.

Algorithms	Second group		First group	
	time (s)	no. SECs	time (s)	no. SECs
Algo1	3379.94	13649	69077.2	164681
Algo2	2231.86	13649	60045.2	164681
Algo3	6.86	0	305.4	18126
Algo4	4.04	3916	348.4	54278

Table 4.18: Comparison result of four separation algorithms

All experimental-comparison result are given in Table 4.18. In this table, column "time (s)" presents the total computation time needed to find out the number of violated SECs given in column "no. SECs" from all instances of each group by the separation algorithm in the first column. Looking at the table, we see that

- In comparison to **Algo1**, **Algo2** detected the same number of violated SECs (as discussed in the subsection **??**) but it needed a smaller amount of computation time, a time decrease of 13.1% for the instances of the first group and a time decrease of 34% for the instances of the second group;

- For 141 instances of the second group, **Algo3** took 6.86 seconds but could not detect any violated SEC, whereas **Algo4** needed only 4.04 seconds (58.9 %) to detect 3916 violated SECs;

- For 4306 instances of the first group, **Algo4** took a slightly larger computation time than **Algo3** but the number of violated SECs detected by **Algo4** (54278) is up to three times of the number of violated SECs found by **Algo3** (18126).

In some separation instances, **Algo4** found out some violated SECs which could not found by **Algo3**. This interesting result comes from the fact that **Algo3** works directly on the auxiliary undirected support graphs, while **Algo4** works indirectly on the auxiliary undirected graphs of the extended graphs of strongly connected components of the directed support graphs.

As a message for concluding this large section, the proposed decomposition is very promising for solving large-size ESPP and ELPP instances.

4.9 Conclusion

In this chapter, we focus on solving two problems: the elementary shortest path problem between two specified nodes ESPP(s, t, G) and the elementary longest path problem ELPP(G). We also demonstrated some equivalences: one between these two problems and some between other relevant problems.

For ESPP(s, t, G), we presented three integer programming formulations. We slightly adapted a few classes of inequalities that are valid for the asymmetric traveling salesman problem and its variants, to obtain new classes of inequalities that are valid for ESPP(s, t, G). The central point of this chapter is that we proposed an exact algorithm based

on mixed integer programming. The salient fact about this algorithm is that it generates lots of inequalities of different classes but it uses only the most violated inequalities as cuts to tighten relaxed solutions until it obtains an optimal solution. Moreover, we proposed two dynamic programming algorithms that solve the problem on directed graphs with many bridges and many strongly connected components. The experiments showed that all our proposed algorithms are more interesting than the state of the art algorithms.

For ELPP(G), based on the equivalence between the two problems ESPP(s, t, G) and ELPP(G), we proposed algorithms for ELPP(G) by adapting an algorithm for ESPP(s, t, G). We also extended the dynamic programming algorithm from our previous chapter, which was for solving ELPP(G) on undirected graphs with many bridges, to solve ELPP(G) on directed graphs with many bridges. An experiment showed that all our proposed algorithms are more efficient than the state of the art algorithms.

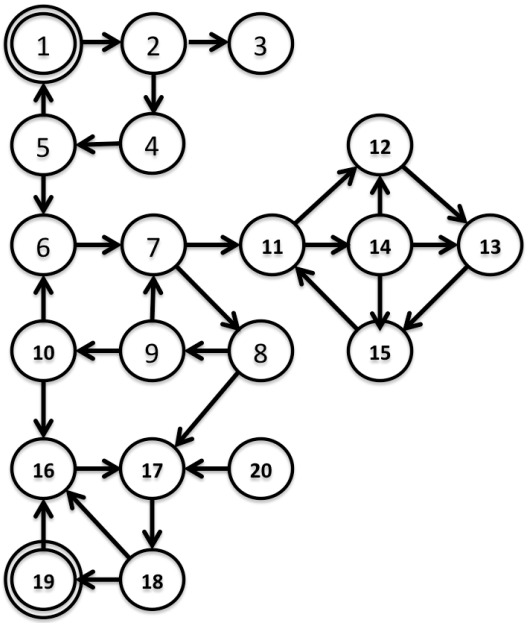

Figure 4.9: The graph G

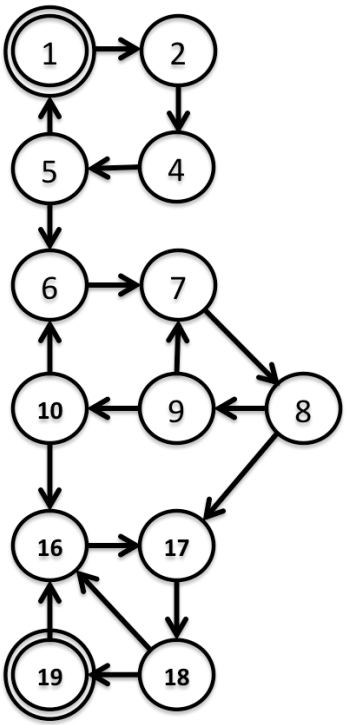

Figure 4.10: The graph G after the preprocessing step

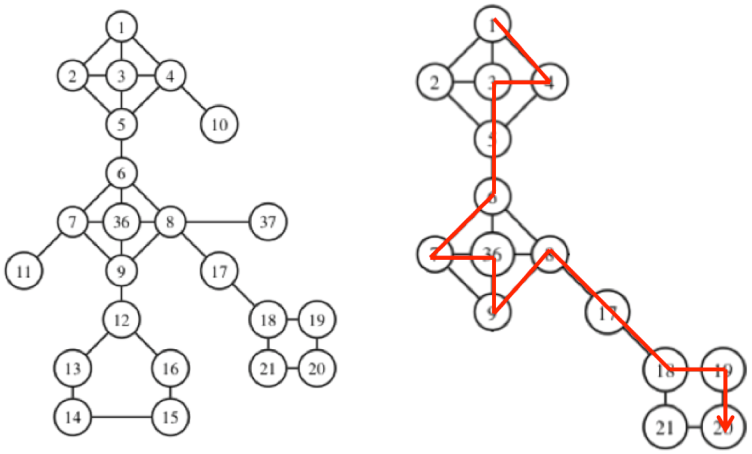

Figure 4.11: ESPP(s, t, G) on an undirected graph with many bridge-blocks, $s = 1$, $t = 20$

Name	Problem	Underlying Model	On Graph	Reference
ESP_Drexl	ESPP	Model 1 + B&C	Both	[33]
ESP_ComFlow	ESPP	Model 2 + B&C	Both	[this chapter]
ESP_MTZ	ESPP	Model 3 + B&C	Both	[this chapter]
ESP_FltC	ESPP	Model 1 + B&C	Both	[this chapter]
ESP_FltC_BB	ESPP with bridges	Model 1 + B&C	Both	[this chapter]
ESP_FltC_SCC	ESPP with SCC	Model 1 + B&C + Dyn	Undirected	[this chapter]
ELP_Drexl	ELPP	Model 1	Both	[33]
ELP_FltC	ELPP	Model 1 + B&C	Both	[this chapter]
ELP_FltC_BB	ELPP with bridges	Model 1 + B&C + Dyn	Both	[this chapter]
ELP_CP	ELPP	CP	Undirected	[84]
ELP_CP_BB	ELPP with bridges	CP + Dyn	Undirected	[84]
ELP_CP2_BB	ELPP	CP + Dyn	Both	[this chapter]

Figure 4.12: List of algorithms compared

```
### PARAMETER FILE FOR THE ACOTSP SOFTWARE

# name              switch               type       values  conditions (using R syntax)]

sec_ta              "--sec_ght_algo"        i       (0,1)
sec_na              "--sec_heu_algo"        i       (1,1) | sec_ta == 0
2mc                 "--2matching"           i       (0,1)
mo                  "--MaximumOutflow"      i       (0,1)
dk                  "--Dk"                  i       (0,1)
tk                  "--Tk"                  i       (0,1)
oi                  "--other_inequality"    i       (0,1)

sec_th              "--sec_threshold"       r       (0.00001,0.99)
2mc_th              "--2mc_threshold"       r       (1.0,1.0)           | 2mc == 1
mo_th               "--mo_threshold"        r       (0.00001,0.01)      | mo == 1
dk_th               "--dk_threshold"        r       (0.00001,0.99999)   | dk == 1
tk_th               "--tk_threshold"        r       (0.00001,0.99999)   | tk == 1
```

Figure 4.13: Parameter space for **irace**

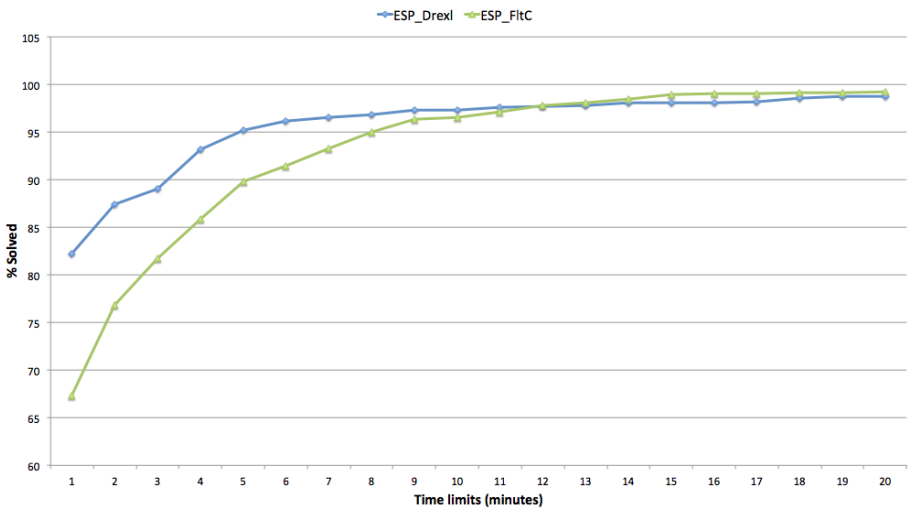

Figure 4.14: Comparing two algorithms in terms of the number of solved instances of the group *P_first_100* in given times

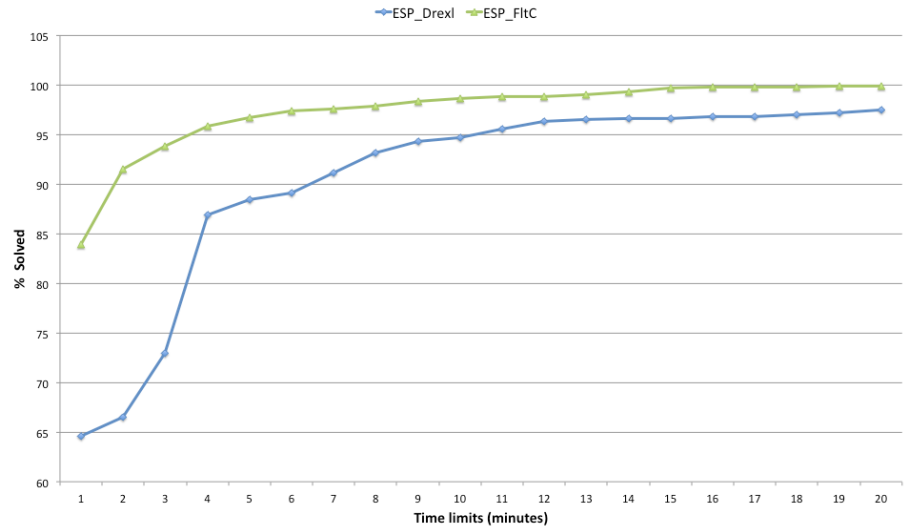

Figure 4.15: Comparing two algorithms in terms of the number of solved instances of the group *P_last_but_one_100* in given times

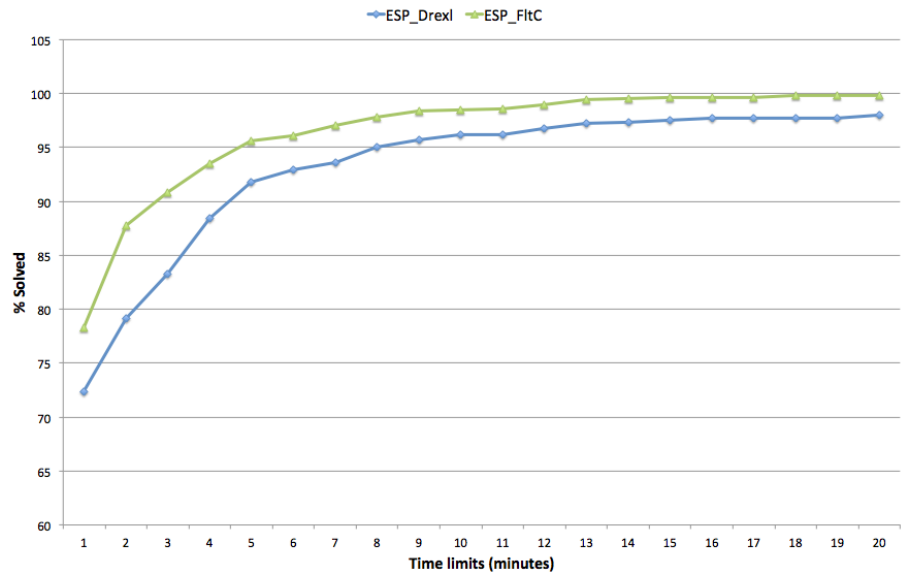

Figure 4.16: Comparing *ESP_Drexl* and *ESP_FltC* in terms of the number of solved instances of the group *P_last_100* in given times

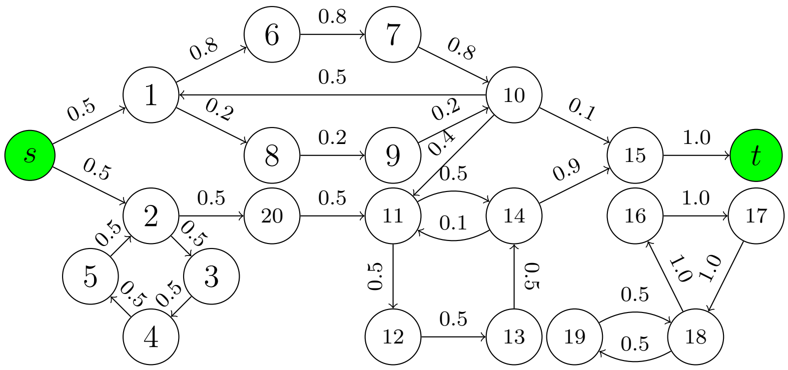

Figure 4.17: A directed support graph

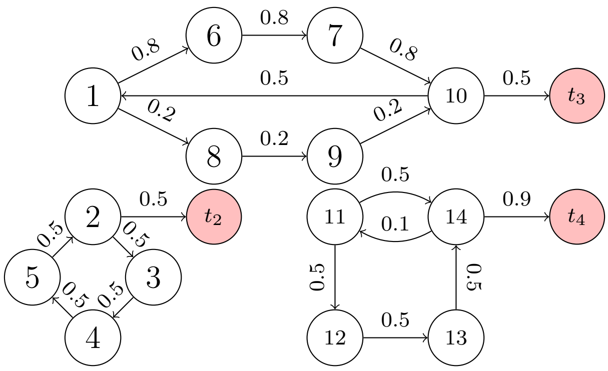

Figure 4.18: Three extended graphs corresponding to three strongly connected components of the directed support graph

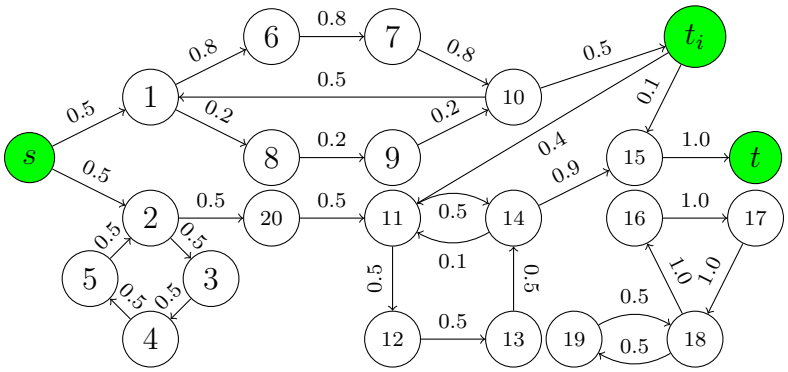

Figure 4.19: Graph $\mathcal{D}'$ is constructed from the directed graph $\mathcal{D}$ in Figure 4.17 and the strongly connected component containing $\{1, 6, 7, 8, 9, 10\}$

5

AGRICULTURAL LAND ALLOCATION PROBLEM

Agricultural land fragmentation, where a single field consists of a large number of small separate plots, is a common agricultural phenomenon in most provinces of Vietnam. The land fragmentation is considered to be a big problem to efficient crop production and agricultural modernization. In several provinces where the land allocation has been done, the results obtained are very promising. In this chapter, we first introduce and formulates the land allocation problem as an optimization problem. We then apply several operation research techniques to propose complete and incomplete algorithms for solving this problem. The experimental results show that solutions computed by our algorithms are much better than solutions conducted by the government's approach.

5.1 Background

The consolidation of land has occurred in many countries around the world, e.g., Germany, Turkey, Japan, China [8, 17, 18, 21, 22, 52, 57, 73, 93, 98]. However, each country has its own solutions for tackling this common problem, to meet particular specified constraints [16, 17, 18, 21, 22, 98].

After discussions with farmers and the leaders of some municipality,

the following farmers' expectations were formulated.

- The number of plots adjacent to water canals should be maximized. It is easy to understand this expectation as the irrigation system in Vietnam is mainly based on these water canals; plots next the water canals can easily be provided with water.
- The plots should have a shape close to a square. This reduces the tracks between plots to ease the access of machines to the culture.

5.1.1 Problem formulation

In this thesis, we consider only fields with rectangular shape as it simplifies the formulation of ALAP. Moreover, almost complete algorithms, which will be proposed here, just work on the rectangular-shape fields.

For an ease of the presentation, we introduce some notations. A *field* is a large agricultural land region which normally belongs to different households. Basically, a field can be divided into *zones* by parallel tracks which are also parallel to the sides of the field. A zone can then be partitioned into *plots* by parallel short tracks which are perpendicular to the tracks defining the zones. Illustratively, the field on the left hand side of Figure 1.1 is divided into 3 zones: 2 zones of 4 plots and 1 zone of 3 plots.

Additionally, we use two following notations: the *relative position* of a plot denotes the field where the plot locates on; and the *exact position* of a plot is the zone of the field where the plot locates on.

- **Input:**
  - a set of land categories C;
  - a set of rectangular-shape fields F, where each $f \in F$ is associated with a land category $c_f \in C$ and an area $fa_f \in \mathbb{R}^+$;
  - a set of households H, each pair of household $h \in H$ and land category $c \in C$ is associated with an area ha_h^c denoting the expected area of land category c of household h;
  - the input respect the area consistency property: for each $c \in C, \sum_{h \in H} ha_h^c = \sum_{f \in F, c_f = c} fa_f$.

- **Output:**
  - each field $f \in F$ is partitioned into a set of zones Z^f;
  - each zone $z \in Z^f$ of the field $f \in F$ is divided into a set of plots P^z;
  - each plot $p \in P^z$ located in zone $z \in Z^f$ of field $f \in F$ is associated with an area pa^p and a household h^p, owner of p.
- **Constraints:**
  - conservation of the fields' area: for each $f \in F$,
$$\sum_{z \in Z^f} \sum_{p \in P^z} pa^p = a_f$$
  - conservation of the households' expected area: for each household $h \in H$, for each land category $c \in C$,
$$ha_h^c = \sum_{f \in F, c_f = c} \sum_{z \in Z^f} \sum_{p \in P^z, h^p = h} pa^p$$
- **Objective:** Minimize the number of plots $\sum_{f \in F} \sum_{z \in Z^f} |P^z|$

In this formulation, the first constraint requires that the total area of the plots located in a field must equal the area of the field; while the second constraint states that for each land category, each household receives plots with a total area equal to his expected area.

Note that with the above formulation, a solution provides the exact position of the plots and not the exact geographical position of the plots. This can easily be obtained by ordering the plots in a zone and ordering the zones in a field. This part is not considered as it is usually done by the farmers themselves.

5.1.2 The government's approach to ALAP

The Vietnamese government promulgated the following instructions to guide farmers for solving ALAP. The agricultural fields should be classified into several categories (1–4) and the farmers should use a system of coefficients that presents the equivalence between categories.

For example, a system of coefficients for three land categories could be $\langle 1.0, 1.2, 1.4 \rangle$, if 1 m^2 of the first land category is equivalent to 1.2 m^2 of the second land category and 1.4 m^2 of the third land category. Fields of each land category are considered in turn for division into plots, with the following rules:

step 1: The order of the fields is determined based on their land categories and geometrical positions and this is decided by the authorities.

step 2: The order of households is determined by lot and households are assigned plots with respect to this order. Suppose that this sorted list is $h_1, \ldots, h_n$.

step 3: Each field is divided into *zones* $z_1, \ldots, z_k$ of width 40–50 meters by lines which are parallel to one side of the field.

step 4: Each zone z_i is iteratively divided into plots corresponding to a sequence of households $h_j, h_{j+1}, \ldots, h_p$ by parallel lines that are perpendicular to the parallel lines already used to separate the field into the zones. The next zone z_{i+1} will then be partitioned into plots for households $h_{p+1}, \ldots$

step 5: At each step, suppose that household h_i is under consideration, the current zone is z_j, and the current land category is c. The remaining area of z_j is R m^2. If R is greater than or equal to the expected area of h_i, then the next plot of z_j will be allocated to h_i. Otherwise, h_i needs a supplementary area of S m^2. The following situations may occur:

- R is smaller than 100, in which case the previous household h_{i-1} will receive this remaining area of R m^2. An equivalent (to R m^2 of category c) area of the next category $c + 1$ will be subtracted from the expected area of h_{i-1} for the category $c + 1$. The next zone z_{j+1} will then be considered for allocation to h_i.

- Both R and S are greater than or equal to 100. The household h_i is allocated this remaining area of z_j (R m^2), and will then receive a plot of S m^2 in the next considered zone or field.

- R is greater than or equal to 100, but S is less than 100. The household h_i is allocated this remaining area of R m^2 and an equivalent (to S m^2 of category c) area of the next category $c + 1$ will be added to the expected area of h_i for the category $c + 1$.

Figure 5.1 illustrates the government's solution to ALAP with five households $H1$, $H2$, $H3$, $H4$, and $H5$, and two fields of two different land categories with coefficients $\langle 1, 1 \rangle$. The expected areas of the first land category of the five households are 300, 400, 250, 300, and 250 m^2 (left part of the figure) and 250, 260, 560, 250, and 180 m^2 of the second land category (right part of the figure). Applying the instructions of the government, each field is separated into three zones of width 50 m, and the allocated areas of the households are 300, 400, 300, 300, and 200 m^2 in the field on the left; and 250, 250, 500, 250, and 250 m^2 in the field on the right. In this solution, household $H2$ has three plots of two land categories and households $H3, H4, H5$ are allocated plots whose areas are different from their corresponding expected areas.

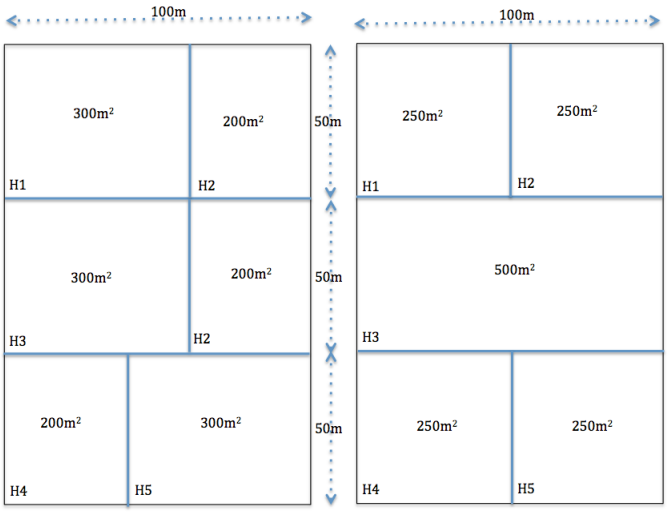

Figure 5.1: A solution guided by the instructions of the government

Limitations of the government's approach First, a lot of households have allocated areas which are different from their expected areas. The cause of this comes mainly from dividing a field into zones of fixed width, 40–50 m. Then, the threshold 100 m^2 may be not suitable for provinces where each household has thousands of square meters of agricultural land. In that case, there may be small plots (with areas slightly more than 100 m^2) next to large plots (with areas of some thousands of

square meters). Finally, the instructions of the government do not take into account other expectations of the farmers, such as optimizing the number of plots next to canals, minimizing the distance from the house to the plot, a nice form of the plots, etc.

5.1.3 Setting for experiments

All algorithms proposed in this chapter are implemented in the C++ programming language. The B&C algorithms are written with IBM Ilog Cplex Concert Technology, version 12.4; and the B&P algorithms are implemented with the framework [2] (using SoPlex as the linear programming solver). The large neighborhood search algorithm based on CP is implemented in OSCAR solver [79]. All experiments in this chapter were performed on a computer running Mac OS 10.9 with the configuration of 2.3 GHz Intel Core i7 CPU, 4 GB RAM. And the complete algorithms are restricted to a computational time limit of 5 hours.

5.2 Decomposing ALAP

In a given field, it is not always possible to find a subset of households such that the sum of their expected area is not strictly equals to the area of the field. To overcome this difficulty, we use the system of coefficients introduced in the government's solution. The coefficient state equivalences between land categories; allowing solutions where a household receive a plot with an area is different from his expected area. This system of coefficients brings another benefit: the ALAP problem can then be reduced to a much easier problem called **rALA**. In this reduced problem, we can focus on fields belonging to the same land category. In the rest of this thesis, we deal with **rALA** instead of ALAP.

Our approach prefers solutions to **rALA** such that each household receives a unique plot. There are however two other side objectives: the number of plots next to water canals should be maximized, but the shape of plots should be close a square. To obtain our solution, we divide **rALA** into three consecutive subproblems: **PArea**, **PPos1** and **PPos2**. The first subproblem, **PArea** computes a unique plot for each household by indicating its area and its relative position. The remain-

ing subproblems compute the exact position of the plots in each field: **PPos1** specifies the exact position of the plots beside water canals and **PPos2** specifies the exact position of the remaining plots. Each subproblem is now clearly formulated.

5.2.1 PArea

PArea is viewed as an assignment problem. Given a set of fields and a set of households, each household is associated with an expected area (the total expected area of households is equal to the total area of the fields). The objective of this problem is to assign each household a unique plot by indicating its relative position and its area. This assignment should minimize the total difference between the area of the fields and the sum of the expected area of the households assigned to those fields.

Ideally, we partition the set of households, one subset per field. Each subset is the assigned to a specific field such that the difference between the area of the field and the total expected area of households assigned to the field is minimized. From this assignment, it is then possible to distribute the area of a field among the assigned households. If there is a difference between the area (a_f) of a field and the sum of the expected areas (s) of the households assigned to this field, the plot assigned to a specific household with an expected area of a will have an area of $a \times \frac{s}{a_f}$. This shows that the area of the plots are derived from the assignment of households to fields.

Input: a set of fields of the same land category $\mathcal{F} = \{1, \ldots, m\}$, each field $i \in \mathcal{F}$ is associated with an area fa_i; and a set of households $\mathcal{H} = \{1, \ldots, n\}$, each household $j \in \mathcal{H}$ has an expected area ha_j such that $\sum_{i \in \mathcal{F}} fa_i = \sum_{j \in \mathcal{H}} ha_j$.

Output: the relative position of the plots assigned to the households $\mathcal{S} = (rp_1, \ldots, rp_n)$, where $rp_h \in \mathcal{F}, \forall h \in \mathcal{H}$.

Objective: $minimize \;\; O(\mathcal{S}) = \sum_{i \in \mathcal{F}} \left| fa_i - \sum_{j \in \mathcal{H}, rp_j = i} ha_j \right|$.

The objective function of this problem is to minimize the sum of differences between the area of the fields and the total expected area of households assigned to these fields.

Note that this formulation is independent from the shape of the

fields. All the proposed algorithms thus handle fields of arbitrary shape.

5.2.2 PPos1

This subproblem aims to meet the expectation of the farmers to maximize the number of plots next to water canals, but with an acceptable shape. Given a field with one side next to a water canal, the objective of this problem is thus to specify a zone next to that side and plots for some households located in that zone. Note that in a real-life case, e.g., for a field with several sides next to water canals, we solve this subproblem once for each side next to a canal (the consideration of the order of the sides next to canals is randomly determined). Figure 5.2 presents a solution of an instance of this problem.

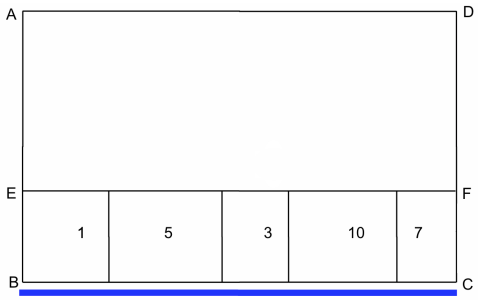

Figure 5.2: An illustration of **PPos1**

Input: a field of rectangular shape, the lengths of its sides are respectively W and L, one of its side of length W stays beside a water canal; a set of plots $\mathcal{P} = \{1, \ldots, n\}$, each plot $p \in \mathcal{P}$ has an area a_p such that $W \times L = \sum_{p \in \mathcal{P}} a_p$.

Output: a subset of plots $\mathcal{S} \subset \mathcal{P}$

Constraints: each plot $p \in \mathcal{S}$ has a rectangular shape, one of its sides of the length $l = \frac{\sum_{p \in \mathcal{S}} a_p}{W}$ and another side of the length $w_p = \frac{a_p}{l}$. So, for every $p \in \mathcal{S}$, the value $\frac{max\{l, w_p\}}{min\{l, w_p\}}$ is smaller or equal to a parameter α which is decided by the households.

Objective: Maximize the number of plots in the solution $|\mathcal{S}|$, that is the number of plots next to the canal.

Note that if the value of parameter α is too close to 1, the problem might be unfeasible on some instances. In such a case, the value of α must be increased. In practice, a value of $\alpha = 2$ is adequate.

5.2.3 PPos2

In this subproblem we now assume that none of the sides of the field are beside any canal. The objective is to specify the exact position of the plots of the households in the field. The field is decomposed into zones, each zone being decomposed into plots with an acceptable shape. Illustratively, the photo on the left hand side of Figure 1.1 presents a solution to an instance of this problem, where a field is divided into 3 zones: 2 zones of 4 plots and 1 zone of 3 plots.

This modeling of **rALA** ensures that each household receives a unique plot.

Input: a field of rectangular shape, its lengths are respectively W and L; a set of plots $\mathcal{P} = \{1, \ldots, n\}$, and each plot $p \in \mathcal{P}$ is associated with an area $a_p \in \mathbb{Z}^+$ such that $W \times L = \sum_{p \in \mathcal{P}} a_p$.

Output: a partition of $\mathcal{P}$ into a set of non-empty subsets, $\mathcal{S} = \{S_1, \ldots, S_m\}$.

Constraints: the subsets in the solution are disjoint $S_i \cap S_j = \{\emptyset\}, \forall i, j \in \{1, \ldots, m\}, i \neq j$, non-empty $S_i \neq \{\emptyset\}, \forall i \in \{1, \ldots, m\}$, and $S_1 \cup \cdots \cup S_m = \mathcal{P}$.

Objective: The shape of the plots should be close to square.

$$Minimize \quad \sum_{i=1}^{m} \sum_{p \in S_i} \left| \frac{\sum_{j \in S_i} a_j}{W} - \frac{a_p W}{\sum_{j \in S_i} a_j} \right|$$

The shape objective uses the length difference between two sides of the field. For a plot p in a zone S_i, this value is $\left| \frac{\sum_{j \in S_i} a_j}{W} - \frac{a_p W}{\sum_{j \in S_i} a_j} \right|$. In this formulation, each household receives a unique plot.

5.3 Solving PArea

We first state two important properties of optimal solutions to this subproblem. These properties will be used to optimize the complete algo-

rithms for this subproblem. In the last subsection, the proposed algorithms are compared on different test instances.

5.3.1 Two properties of optimal solutions of PArea

The two properties of the optimal solutions of **PArea** presented in this subsection will be converted to the appropriate constraints to be used in B&C algorithm for solving the problem.

Before presenting these two properties as two theorems, we introduce some notation: $maxha$ is the maximum expected area of the households, $maxha = max\{ha_i | i \in \mathcal{F}\}$ and $minha_j$ is the minimum expected area of the households assigned to field j, $minha_j = min\{ha_i | rp_i = j\}$.

Theorem 1 *If $\mathcal{S} = (rp_1, \ldots, rp_n)$ is an optimal solution to **PArea**, then*

$$\forall j \in \mathcal{F}, \sum_{i \in \mathcal{H}, rp_i = j} ha_i > fa_j - maxha. \tag{5.1}$$

Proof Suppose that in the optimal solution $\mathcal{S}$ there exists a field $j \in \mathcal{F}$ such that

$$\sum_{i \in \mathcal{H}, rp_i = j} ha_i = fa_j - maxha - \Delta, \Delta \geq 0. \tag{5.2}$$

From the property $\sum_{i \in \mathcal{F}} fa_i = \sum_{k \in \mathcal{H}} ha_k$, we have that there exists a field $k \in \mathcal{F}$ such that

$$\sum_{i \in \mathcal{H}, rp_i = k} ha_i = fa_k + \Delta', \Delta' > 0. \tag{5.3}$$

Let h be a household that is assigned to field k in the optimal solution: this means that $rp_h = k$. We produce a new solution $\mathcal{S}'$ from $\mathcal{S}$ by moving household h from field k to field j (this means resetting $rp_h = j$). Let us compare the objective values of these two solutions:

$$\begin{aligned} O(\mathcal{S}) - O(\mathcal{S}') &= |maxha + \Delta| + |\Delta'| - (|maxha - ha_h - \Delta| + |\Delta' - ha_h|) \\ &= ha_h + \Delta' - |\Delta' - ha_h| > 0. \end{aligned} \tag{5.4}$$

This is a clear contradiction of the hypothesis that $\mathcal{S}$ is an optimal solution.

Theorem 2 *If $\mathcal{S} = (rp_1, \ldots, rp_n)$ is an optimal solution to **PArea**, then*

$$\forall j \in \mathcal{F}, \sum_{i \in \mathcal{H}, rp_i = j} ha_i < fa_j + minha_j. \tag{5.5}$$

Proof We can apply the same procedure that was used to prove Theorem 1 to obtain the proof of this theorem.

5.3.2 Solving PArea by a B&C algorithm

In this section, we formulate **PArea** as an IPP. The proposed model (Figure 5.3) introduces, for each household i, a binary variable x_{ij} to indicate whether or not household i is assigned to field j and, for each field, a continuous variable δ_i to express the area difference between the total expected area of households assigned to field i and the area of field i.

In this IPP, inequalities (5.6c) ensure that every household must be assigned to only one field; inequalities (5.6d) and (5.6e) present the area difference between the total expected area of the households assigned to a field and the area of the field; inequalities (5.6f) express the property in Theorem 1; and we can not express exactly the property in Theorem 2 by linear inequalities, so a weaker variant of this property is represented in inequalities (5.6g).

5.3.3 Solving PArea by a CBLS algorithm

In this section, we present the CBLS algorithm for solving **PArea** that was already appeared in our paper [20]. We denote this algorithm by **PA_LS**. Given a solution $S = (rp_1, \ldots, rp_i, \ldots, rp_n)$ in which $f_i \in \mathcal{F}$, to **PArea**, the local search algorithm exploits the two following neighborhoods:

1. **Change-based** neighborhood

$$N_1(S) = \{(rp_1, \ldots, rp'_i, \ldots, rp_n) | rp'_i \neq rp_i, rp'_i \in \mathcal{F}\}$$

$$Minimize \sum_{i=1}^{m} \delta_i \tag{5.6a}$$

$$s.t. \tag{5.6b}$$

$$\sum_{j=1}^{m} x_{ij} = 1, \forall i \in \{1, \ldots, n\} \tag{5.6c}$$

$$\delta_j \geq fa_j - \sum_{i=1}^{n} ha_i x_{ij}, \forall j \in \{1, \ldots, m\} \tag{5.6d}$$

$$\delta_j \geq -fa_j + \sum_{i=1}^{n} ha_i x_{ij}, \forall j \in \{1, \ldots, m\} \tag{5.6e}$$

$$\sum_{i=1}^{n} ha_i x_{ij} > fa_j - maxha, \ \forall j \in \{1..m\} \tag{5.6f}$$

$$\sum_{i=1}^{n} ha_i x_{ij} < fa_j + maxha, \ \forall j \in \{1..m\} \tag{5.6g}$$

$$x_{ij} \in \{0, 1\}, \forall i \in \{1, \ldots, n\}, j \in \{1, \ldots, m\} \tag{5.6h}$$

$$\delta_i \geq 0, \forall i \in \{1, \ldots, m\} \tag{5.6i}$$

Figure 5.3: IP model for **PArea**

2. **Swap-based** neighborhood

$$N_2(S) = \{(rp_1, \ldots, rp_{i+j}, \ldots, rp_i, \ldots, rp_n) | rp_i \neq rp_{i+j}, 1 \leq i < i + j \leq n\}$$

The pseudo-code of **PA_LS** is depicted in Algorithm 10. In this algorithm, a basic search procedure is given in lines 4–12; an intensification component is given in lines 13–21, and a restarting component is given in the last lines 22–24. In the basic search procedure, we swap randomly two households in line 12 when the best neighbors in the neighborhoods are not better than the current solution; this increases the diversification of the algorithm. Note that this algorithm always returns the best solution which was exploited by the search.

Algorithm 10: **PA_LS**: A constraint-based local search algorithm for **PArea**

```
1  S ← A random solution to PArea; obj ← objective of S;
2  bestSol ← S; bestObj ← obj ;
3  while it < maxIt && obj > 0 do
4      S_1 ← the best neighbor of S in N_1(S) that is obtained by
       changing field of a household such that this household is not
       in tabu;
5      if S_1 is better than S then
6          S ← S_1;
7      else
8          S_2 ← the best neighbour of S in N_2(S) that is obtained
           by swapping the position of a pair of households such that
           this pair is not in tabu;
9          if S_2 is better than S then
10             S ← S_2;
11         else
12             Swap the position of two random households;
13     obj ← objective of S;
14     if obj < bestObj then
15         bestObj ← obj; bestSol ← S;
16         stable ← 0;
17     else
18         if stable = stableLimit then
19             S ← bestSol; stable ← 0;
20         else
21             stable ← stable + 1;
22     if it%restartFreq = 0 then
23         S ← a random solution; obj ← objective of S;
24         best ← obj;bestSol ← S;
25     it ← it + 1;
```

5.3.4 Solving PArea by a large neighborhood search algorithm based on CP

In this subsection, we propose a CP model and an efficient large neighborhood search procedure that are jointly used in an algorithm denoted by **PA_CPLNS** for solving **PArea**.

CP model

The CP model (Figure 5.4) introduces,

- for each household i, an integral-valued variable x_i to indicate a field where the plot of household i locates on; and
- for each field j, a real-valued variable l_j to represent the total areas of households whose plots are in field j.

$$Minimize \sum_{i \in \mathcal{F}} \left| fa_i - \sum_{j \in \mathcal{H}} (x_j = i) ha_j \right| \tag{5.7a}$$

$$s.t. \tag{5.7b}$$

$$binPacking(x_1, \ldots, x_n; ha_1, \ldots, ha_n; l_1, \ldots, l_m) \tag{5.7c}$$

$$l_i < fa_j + maxha \tag{5.7d}$$

$$l_i > fa_j - maxha \tag{5.7e}$$

$$x_i \in \{1, \ldots, m\} \; \forall i \in \{1, \ldots, n\} \tag{5.7f}$$

$$l_i \in \{0, \ldots, \sum_{j \in \mathcal{F}} fa_j\} \; \forall i \in \{1, \ldots, m\} \tag{5.7g}$$

Figure 5.4: CP model for **PArea**

In this model, constraint (5.7c) is the bin-packing constraint[1]; constraints (5.7d) express the property in Theorem 1; and constraints (5.7e) represents a weaker variant of the property in Theorem 2.

In this algorithm, we used the default search of OSCAR that is very strong for large neighborhood search algorithms.

[1]Many thanks to Pierre Schaus who suggested to formulate this problem with the Bin-Packing constraints.

5.3.5 Experiments

We begin this subsection by describing instances for the experiments. There are two classes of instances: the first one contains 6 random instances with $n = 30, 50$, m = 3, 4, 5, and expected areas in $[100, 700]$; and the second class consists of 3 real-life instances with n = 89, 100, m = 7, 8, 9, and expected areas in $[100, 1500]$ that appear in Dong Trung village, Thai Binh province, Vietnam [20].

We compare all algorithms for **PArea** in solving all 9 instances with respect to the quality of the computed solutions and the computational time. Due to this limitation, the complete algorithms might not return an optimal solution: in these cases they may output feasible solutions. The result of this experiment is shown in Table 5.1. In this table, columns "t(s)" present the computational times in seconds; columns "obj" present the objective value of the solutions found by the algorithms in the time limit (bold digits signify that they are the objective value of optimal solutions; otherwise they are the objective value of feasible solutions); the column "min_obj" (resp. avg_obj, max_obj) represents the minimum (resp. average, maximum) objective value of the best solutions found by the incomplete algorithm after 10 executions of the algorithm; and column "avg_t(s)" presents the computational time average of 10 executions of the incomplete algorithm.

Instances		**PA_CPLNS**				**PA_IP**		**PA_LS**			
n	m	min_obj	avg_obj	max_obj	avg_t(s)	t(s)	obj	min_obj	avg_obj	max_obj	avg_t(s)
30	3	**0**	**0**	**0**	0.11	0.08	**0**	**0**	**0**	**0**	0.02
30	4	**0**	**0**	**0**	0.11**0**	0.34	**0**	**0**	**0**	**0**	0.11
30	5	**0**	**0**	**0**	0.8	1.19	**0**	**0**	**0**	**0**	0.89
50	3	**0**	**0**	**0**	0.2	0.16	**0**	**0**	**0**	**0**	0.01
50	4	**0**	**0**	**0**	0.4	0.37	**0**	**0**	**0**	**0**	0.05
50	5	**0**	**0**	**0**	0.75	0.67	**0**	**0**	**0**	**0**	0.52
89	7	**0**	**0**	**0**	7.2	1.74	**0**	**0**	**0**	**0**	6.11
100	8	208	208	208	130.4	18000	212	208	210.2	212	114.22
100	9	**0**	**0**	**0**	31.6	18000	38	**0**	1	4	101.72

Table 5.1: Comparing the proposed algorithms for **PArea** with respect to the quality of found solution and the computational time

The result of this experiment shows that **PA_LS** and **PA_CPLNS**, the incomplete algorithms, yielded in a stable way very good solutions: for the first seven instances, it always output optimal solutions; even for last two instances, it needed a computational time of less than two

minutes to return solutions that are better than solutions computed by the exact algorithm in 5 hours.

5.4 Solving PPos1

We propose different algorithms for solving **PPos1**. All the algorithms are then compared on various test instances.

5.4.1 Solving PPos1 by a B&I algorithm

In this subsection, we propose a CP model along with an efficient search procedure that are jointly used in a B&I algorithm denoted by **P1_CP** for solving **PPos1** to optimality.

CP model

This model (Figure 5.5) introduces, for each plot, a binary variable x_p to represent whether or not plot p is in the solution $\mathcal{S}$.

$$Minimize \sum_{i=1}^{n} x_i \tag{5.8a}$$

$$s.t. \tag{5.8b}$$

$$W^2 a_k x_k \leq \alpha(\sum_{i=1}^{n} a_i x_i)^2, \forall k \in \{1, \ldots, n\} \tag{5.8c}$$

$$x_k(\sum_{i=1}^{n} a_i x_i)^2 \leq \alpha a_k W^2, \forall k \in \{1, \ldots, n\} \tag{5.8d}$$

$$x_i \in \{0, 1\}, \forall i \in \mathcal{P} \tag{5.8e}$$

Figure 5.5: CP model for **PPos1**

In this model, (5.8c) and (5.8d) present together the unique constraint of the problem:the ratio between two sides of a plot is always smaller than or equal to a threshold α.

Search procedure

It is true that, in general, a solution subset to **PPos1** with small-area plots is better than one with large-area plots. From this observation, we propose a search procedure for **P1_CP** as follows.

Algorithm 11: The proposed complete search for **P1_CP**

```
1 using{
2    forall(h in 1..n) by (-pa[h] ){
3       tryall<cp>(v in 0..1: x[h].memberOf(v)) by (v)
4          label(x[h],v);
5    }
6 }
```

This search tries to first select smaller-area plots for the solution. This leads quickly to a good solution for **PPos1**.

5.4.2 Solving PPos1 by a B&C algorithm

In this section, we propose a IP model that will be used to solve **PPos1** to optimality by a B&C algorithm denoted by **P1_IP**. The model (Figure 5.6) introduces, for each household, one binary variable x_i to indicate whether or not plot i is in the solution $\mathcal{S}$ and a continuous variable w_i that presents the length of one side of plot i when this plot is in the solution, otherwise the value of this variable is 0.

In this model, inequalities (5.9d) and (5.9e) present the relation between x_i and w_i (if x_i = 1 then w_i > 0, otherwise x_i = 0 and then w_i = 0); inequalities (5.9f) and (5.9g) ensure that if $x_i = 1$ and $x_j = 1$ then $\frac{a_i}{w_i} = \frac{a_j}{w_j}$; inequalities (5.9h) and (5.9i) express the constraint of the problem; and inequality (5.9c) with inequalities (5.9f) and (5.9g) ensure that w_i presents the length of one side of plot i if this plot is in the solution.

5.4.3 Solving PPos1 by a CBLS algorithm

In this section, for solving **PPos1** by an incomplete method, we modify slightly the bi-objective CBLS algorithm that was already introduced in

$$Maximize \sum_{i=1}^{n} x_i \tag{5.9a}$$

$$s.t. \tag{5.9b}$$

$$\sum_{i=1}^{n} w_i = W \tag{5.9c}$$

$$w_i \leq W x_i,\ \forall i \in \{1, \ldots, n\} \tag{5.9d}$$

$$a_i x_i \leq L w_i,\ \forall i \in \{1, \ldots, n\} \tag{5.9e}$$

$$a_j w_i - a_i w_j \leq a_j W(2 - x_i - x_j), \forall i, j \in \{1, \ldots, n\}, i \neq j \tag{5.9f}$$

$$a_i W(x_i + x_j - 2) \leq a_j w_i - a_i w_j, \forall i, j \in \{1, \ldots, n\}, i \neq j \tag{5.9g}$$

$$w_i \leq \alpha \frac{\sum_{j=1}^{n} a_j x_j}{W}, \forall i \in \{1, \ldots, n\} \tag{5.9h}$$

$$\frac{\sum_{j=1}^{n} a_j x_j}{W} \leq \alpha w_i + L(1 - x_i), \forall i \in \{1, \ldots, n\} \tag{5.9i}$$

$$x_i \in \{0, 1\}, \forall i \in \{1, \ldots, n\} \tag{5.9j}$$

$$w_i \geq 0, \forall i \in \{1, \ldots, n\} \tag{5.9k}$$

Figure 5.6: IP model for **PPos1**

our previous paper [19, 20]. We denote this algorithm by **P1_LS**.

Given a subset of plots $\mathcal{S}$ that is a solution to **PPos1**, the local search algorithm exploits the three following neighborhoods:

1. **Removal-based** neighborhood

$$N_1(\mathcal{S}) = \{\mathcal{S} \setminus \{h\} \mid h \in \mathcal{S}\}$$

2. **Insertion-based** neighborhood

$$N_2(\mathcal{S}) = \{\mathcal{S} \cup \{h\} \mid h \notin \mathcal{S}, h \in \mathcal{H}\}$$

3. **Swap-based** neighborhood

$$N_3(\mathcal{S}) = \{\mathcal{S} \setminus \{h_1\} \cup \{h_2\} \mid h_1 \in \mathcal{S}, h_2 \notin \mathcal{S}, h_2 \in \mathcal{H}\}$$

The pseudo-code of this algorithm is depicted in Algorithm 12. There are two steps in this algorithm:

The first step (lines 1–13) aims at finding a solution that respects the constraint (the ratio between the length of the two sides of a plot in the solution is bounded). In this step, the constraint violation of a solution is the number of plots that violate the constraints; in line 8, a solution $\mathcal{S}'$ is considered better than another solution $\mathcal{S}'$ if and only if the constraint violation of $\mathcal{S}'$ is strictly smaller than the constraint violation of $\mathcal{S}'$.

The second step (lines 14–29) aims at finding a solution whose constraint violation is less than or equal to the constraint violation of the solution computed in the first step, additionally the number of plots in the solution is maximized. In this step, the algorithm tries to insert more plots into the current solution in such a way that this insertion does not increase the constraint violation (lines 16–18); when an insertion like that does not exist, the algorithm tries to change the current solution by swapping randomly two plots in such a way that does not lead to an increase of the constraint violation; the random swap in lines 22–24 and the restart in line 25 help the algorithm to escape local optima. Note that this algorithm always outputs the best solution found during the search.

5.4.4 Experiments

We begin this last subsection by describing the instances for the experiments. We generated randomly 9 instances with n = 20, 30, 40, 50, 60, 70, 80, 90, 100, the area of the plots in $[100, 700]$, the fields of rectangular shape, and the parameter $\alpha = 2$.

We compare all algorithms for **PPos1** in solving all 9 instances with respect to the quality of the solution found and the computational time. Due to the time limitation of 5 hours, the complete algorithms might not return optimal solutions: in these cases they may output feasible solutions. The result of this experiment is reported in Table 5.2.

The result of this experiment shows that **P1_LS**, the incomplete algorithm produced in a stable way very good solutions that satisfy the constraint of the problem: for the first six instances, it always output optimal solutions; for the last instance, it needed a computational time average of 31.79 seconds to return solutions with very good quality, even 4 among 10 solutions have the same quality as the solution com-

puted by **P1_IP** in 5 hours.

5.5 Solving PPos2

We propose different complete and incomplete algorithms for solving **PPos2**. All the algorithms are then compared on various test instances.

5.5.1 Solving PPos2 by a B&I algorithm

We propose, in this subsection, a CP model for **PPos2** (Figure 5.7). This model introduces, for each plot $p \in \mathcal{P}$, the zone x_p that the plot is located in, and its domain is $D(x_p) = \{1, \ldots, n\}$.

$$Minimize \sum_{i=1}^{n} \left| \frac{\sum_{j=1}^{n} a_j(x_j = i)}{W} - \frac{W a_i}{\sum_{j=1}^{n} a_j(x_j = i)} \right| \quad (5.10a)$$

$$s.t. \quad (5.10b)$$

$$x_1 = 1 \quad (5.10c)$$

$$\sum_{j=1}^{n} a_j(x_j = i) >= \sum_{j=1}^{n} a_j(x_j = i+1), \forall i \in \{2, \ldots, n-1\} \quad (5.10d)$$

$$RDConstraint(x_1, \ldots, x_n) \quad (5.10e)$$

$$x_i \in \{1, \ldots, n\}, \forall i \in \{2, \ldots, n\} \quad (5.10f)$$

Figure 5.7: CP model for **PPos2**

In this model, the objective function (5.10a) is the total difference between the length of two sides of the plots; constraints (5.10c) and (5.10d) are used to break symmetry; the meaning of constraint $RDConstraint(x_1, \ldots, x_n)$ is as follows: the number of distinct values of $\{x_1, \ldots, x_n\}$ is equal to the maximum value $\{x_1, \ldots, x_n\}$. Due to the technical difficulty of the Comet language, we implement a specific propagator for that constraint. This propagator functions as follows: when two variables are already assigned to the same value (which means that there exist at least two plots assigned to the same zone) the constraint $RDConstraint(x_1, \ldots, x_n)$

is immediately invoked to remove the highest value (corresponding to a zone in which there does not exist any plot at that moment) from the domain of all variables that are not yet bounded.

5.5.2 Solving PPos2 by a B&C algorithm

In this section, we propose a MIP model that is used to solve **PPos2** to optimality by a B&C algorithm denoted by **P2_IP**. This model (Figure 5.8) introduces, for each plot j and zone i, three variables x_{ij}, w_{ij}, and δ_{ij}. The binary variable x_{ij} indicates whether or not plot j is in zone i, in case plot j is in zone i (which means that $x_{ij} = 1$), the continuous variable w_{ij} presents the length of one side of the plot and the continuous variable δ_{ij} presents the difference of the lengths of the two sides of the plot, otherwise (which means that $x_{ij} = 0$), the value of these two variables is set to 0. Moreover, the model also introduces n binary variables corresponding to the n zones: for each zone i, a binary variable y_i indicates whether or not the zone i contains at least one plot.

In this model, inequalities (5.11c), (5.11d) and (5.11e) are used to break symmetry; inequalities (5.11f) ensure that each plot has to be located in a zone; inequalities (5.11g) and (5.11h) present the relation between x_{ij} and y_i, which means that if there exists at least one plot in zone i, then y_i has to take the value of 1; inequalities (5.11i) say that if a zone i contains plots then the sum of the w_{ij} of these plots must be equal to W, otherwise all w_{ij} with $j \in \{1, \ldots, n\}$ must be set to 0; inequalities (5.11j) and (5.11k) indicate the relation between the variables x_{ij} and w_{ij}, namely, if $x_{ij} = 0$, then $w_{ij} = 0$ and if $x_{ij} = 1$, then $w_{ij} \neq 0$; inequalities (5.11l) and (5.11m) point out the fact that if two plots j and k are located in the same zone i, we must have $\frac{a_j}{w_{ij}} = \frac{a_k}{w_{ik}}$; inequalities (5.11n) and (5.11o) are used to express the fact that the value of δ_{ij} is always greater than or equal to the difference in the lengths of the two sides of plot j in zone i: these inequalities combine with the objective function (5.11a) to ensure that a solution of this model is truly optimal.

5.5.3 Solving PPos2 by B&P algorithms

This subsection is used to propose two B&P algorithms for solving **PPos2**, the difference between them is the method that is used to solve the subproblem of the B&P algorithms. First of all, we propose another IP model for **PPos2** that will be used as the master problem. Next, we formulate the subproblem as an IPP which can be directly and simply implemented in any MIP solver. After that, a CBLS algorithm will be proposed to solve the subproblem.

Master problem

The output of **PPos2** is a partition of $\mathcal{P}$ into a set of non-empty subsets, where each subset corresponds to a zone in the field. Ideally, if we have at hand the set of all subsets of $\mathcal{P}$, so solving this problem now is to select some elements from the set such that the selection satisfies some constraints and minimizes the objective function. This idea leads to solve this problem by a B&P algorithm, where each subset of $\mathcal{P}$ is considered as a "column".

Each subset of $\mathcal{P}$ is represented by a vector r of n elements, where each element v_i^r of r takes a binary value: if plot i is in the subset then v_i^r = 1, otherwise v_i^r = 0. The cost of r is denoted by c_r and is computed as follows: $c_r = \sum_{i \in r} \left| \frac{\sum_{j \in r} a_j}{W} - \frac{W a_i}{\sum_{j \in r} a_j} \right|$.

Let $\mathcal{Z}$ be the set of all vectors r that encode all distinct subsets of $\mathcal{P}$. The master problem (Figure 5.9) introduces, for each $r \in \mathcal{Z}$, a binary variable x_r that takes the value of 1 if the set r corresponding to a zone is in the optimal solution; otherwise it takes 0.

The master problem contains n constraints (5.12c) to ensure that every plot must appear in a zone of the optimal solution.

Subproblem

The objective of the subproblem of the B&P algorithms is to find a subset of $\mathcal{P}$ (a comlumn). Here, we propose an integer linear formulation for the subproblem that can be used directly to solve the subproblem as a mixed integer programming problem.

The integer linear formulation (Figure 5.10) introduces, for each plot $i \in \mathcal{P}$, three variables y_i, w_i, and δ_i. The binary variable y_i indicates

whether a plot i is in the solution or not; when the plot is in the solution (which means that $y_i = 1$), the continuous variable w_i presents the length of one side of the plot and the continuous variable δ_i presents the difference between the lengths of the two sides of the plot i, otherwise (which means that $y_i = 0$), the value of both continuous variables must be set to 0.

In this formulation, the parameters π_i with $i \in \{1, \ldots, n\}$ in the objective function (5.13a) are the values of the dual variables associated with the constraints (5.12c) of the master problem; inequalities (5.13d) and (5.13e) present the relation between w_i and y_i (if $y_i = 1$, then $w_i >$ 0, otherwise y_i = 0 and so then $w_i = 0$); inequalities (5.13f) and (5.13g) ensure that if $y_i = y_j = 1$, then $\frac{a_i}{w_i} = \frac{a_j}{w_j}$; inequality (5.13c) requires that the total sum of the w_i must be equal to W; inequalities (5.13h) and (5.13i) ensure that if plot i is in the solution, then δ_i represents the difference between the lengths of the two sides of the plot, otherwise $\delta_i = 0$.

Solving the subproblem by a CBLS algorithm

Solving the subproblem to optimality as a mixed integer programming problem may need a large amount of computational time. With the hope of solving the subproblem in an appropriate time, we propose here a CBLS algorithm for it.

Given a solution to the subproblem that is a subset of plots $\mathcal{S} \subset \mathcal{P}$, the CBLS algorithm for the subproblem exploits the three following neighborhoods:

- **Insertion-based** neighborhood

$$N_1(\mathcal{S}) = \{\mathcal{S} \cup \{p\} | p \in \mathcal{P} \setminus \mathcal{S}\}$$

- **Removal-based** neighborhood

$$N_2(\mathcal{S}) = \{\mathcal{S} \setminus \{p\} | p \in \mathcal{S}\}$$

- **Swap-based** neighborhood

$$N_3(\mathcal{S} = \{\mathcal{S} \setminus \{p_1\} \cup \{p_2\} | p_1 \in \mathcal{S}, p_2 \in \mathcal{P} \setminus \mathcal{S}\}$$

The pseudo-code of this algorithm is depicted in Algorithm 13. In this algorithm, a basic search procedure is given in lines 4–15; an intensification component is given in lines 17–24, and a restarting component is given in the last lines 25–27. In the basic search procedure, we swap the position of two random plots in lines 13–15 when the two best neighbors of the current solution in the three neighborhoods are not better than the current solution: this increases the diversification of the algorithm. Note that this algorithm always outputs the best solution found during the search.

Two B&P algorithms for PPos2

We set up two B&P algorithms for **PPos2** corresponding to the two algorithms for solving the subproblem.

- **P2_CGIP** is a B&P algorithm for **PPos2**, in which the subproblem is completely solved as a mixed integer programming problem (see Section 5.5.3).
- **P2_CGLI** is also a B&P algorithm for **PPos2**. In this algorithm, a subproblem is first solved by the CBLS algorithm described in the above subsection. If the reduced cost of the solution (or the objective value of the solution) computed by the CBLS algorithm is not negative, then the subproblem is resolved as a mixed integer programming problem. This procedure of solving the subproblem ensures that **P2_CGLI** is complete.

These B&P algorithms were implemented with SCIP solver of the default setting where Ryan-Foster branching rule is implemented and the pricer makes sure that the same column is added at most once in one pricing round.

5.5.4 Solving PPos2 by a CBLS algorithm

The CBLS algorithm for solving **PPos2** described in this subsection was already presented in our papers [19, 20]. This local search algorithm is denoted by **P2_LS**.

In **P2_LS**, there are two consecutive steps:

- **Step 1:** The objective of this step is to partition the whole set of plots into a set of subsets $\mathcal{S} = \{S_1, \ldots, S_m\}$. This algorithm repeats the execution of a procedure that is used to create subsets S_i with the smallest possible value of $\sum_{p \in S_i} \left| \frac{\sum_{j \in S_i} a_j}{W} - \frac{a_p W}{\sum_{j \in S_i} a_j} \right|$.
- **Step 2:** The solution computed in the first step is improved by changing the elements in the subsets.

Step 1: Finding a solution

At the end of this step, we obtain a solution $\mathcal{S} = \{S_1, \ldots, S_m\}$ to **PPos2**, in which each a subset S_i with $i \in \{1, \ldots, m\}$ is created in an iteration of the algorithm depicted in Algorithm 14 by executing the function $createOneSubset(\mathcal{P})$. The paradigm of this function is exactly the same as Algorithm 13. However, as compared with the CBLS algorithm for the subproblem of the B&P algorithm described in Algorithm 13, the objective function conducting the search in $createOneSubset(\mathcal{P})$ does not take into account the dual values π_i.

Step 2: Improving the solution computed in step 1

The objective of this step is to improve the solution computed in the previous step by changing plots between the subsets of the solution. In this step, **P2_LS** manipulates the two following neighborhoods of a solution $\mathcal{S} = \{S_1, \ldots, S_i, \ldots, S_j, \ldots, S_m\}$:

- **Change-based** neighborhood: $N_1(\mathcal{S}) = \{S_1, \ldots, S'_i, \ldots, S'_j, \ldots, S_m\} | S'_i = S_i \setminus \{p\}, S_j \cup \{p\}, i \neq j, p \in S_i\}$
- **Swap-base** neighborhood: $N_2(\mathcal{S}) = \{S_1, \ldots, S'_i, \ldots, S'_j, \ldots, S_m\} | S'_i = S_i \setminus \{p_1\} \cup \{p_2\}, S_j \setminus \{p_2\} \cup \{p_1\}, i \neq j, p_1 \in S_i, p_2 \in S_j\}$

Step 2 of **P2_LS** is clearly depicted in Algorithm 15.

5.5.5 Solving PPos2 by a hybrid incomplete algorithm

In this subsection, we propose a hybrid incomplete algorithm that combines a CBLS component, a column generation component, and an integer programming component. We denote this algorithm by **P2_LCI**.

The procedure of **P2_LCI** is as follows: first, the CBLS component executes **P2_LS** many times, to produce many solutions to **PPos2**; then the column generation component takes the subsets of the solutions generated by the CBLS component as the initial columns to compute more columns (subsets) by executing the algorithm that is used to solve the subproblem in **P2_CGLI**; lastly the integer programming component takes all subsets of plots that are already computed in the two other components as the input to compute a good solution to **PPos2** as an integer program that is similar to the program described in the subsection 5.5.3 with $\mathcal{Z}$ containing all subsets of plots that are already generated in the two other components.

5.5.6 Experiments

We begin this subsection by describing the instances that are used in the experiments. We generated in all 8 instances with the number of plots in [8, 30] and the area of the plots in [100, 700].

The configuration for the experiments is as follows. A time limit of 5 hours is set for all complete algorithms; **P2_LS** is executed 10 times; the 10 solutions computed by **P2_LS** are all used as initial solutions (columns) for **P2_LCI**, so the computational time reported for this algorithm does not take into account the computation time for computing the initial solutions. All results of the experiments are presented in Table 5.3.

Note that the amount of computational time needed for **P2_CGIP** and **P2_CGLI** for solving the last four instances is greater than 5 hours, because these algorithms could not terminate exactly at the deadline. But the amount of computational time used by **P2_IP** for solving the last three instances is less than 5 hours, because with the time limit of 5 hours the computer memory for stocking the search tree is not enough, for example in solving the instance of $n = 30$ in 1 hour, the memory used to stock nodes of the search tree is around 94 GB.

The result of this experiment yields the fact that.

- We know that **P2_LCI** takes 10 solutions computed by **P2_LS** as initial columns, but it could not improve these solutions (the solutions computed by **P2_LS** are always the best solutions computed

by **P2_LS** after 10 executions of the algorithm), which means that the solutions computed by the CBLS algorithm **P2_LS** have very good quality.

- In comparing the two B&P algorithms, **P2_CGLI** is slightly stronger than **P2_CGIP**: both of them solved the first four instances to optimality in the time limit, but **P2_CGLI** used much less computation time than the other for the remaining instances, the solutions outputted from **P2_CGLI** are better than or equal to the solutions computed by **P2_CGIP**.

- A look deep inside the B&P algorithms, almost all the computational time is used to solve the subproblem to optimality as mixed integer programming problems. For example, solving the instance with $n = 30$ with **P2_CGIP**, 99.98% of the computation time is used for solving in all 29 instances of the subproblem, and with **P2_CGLI**, this number is 81.79% for solving in all 1106 instances of the subproblem. The CBLS algorithm solved 8746 instances of the subproblem and needed 17.97% of the overall computational time. We conclude that the great amount of computation time spent for solving the subproblem to optimality as mixed integer programming problems is the weakness of the B&P algorithms.

- The efficiency of **P2_LS** is also confirmed when we compare it with the three complete algorithms; the quality of the solutions computed by **P2_LS** is always better than or equal to the quality of the solutions computed by the complete algorithms. Moreover the computation time of **P2_LS** is much smaller than the computation time of the complete algorithms (it is around two minutes).

As the main message of these experiments, among the proposed algorithms, the incomplete algorithm **P2_LS** is the most suitable for solving **PPos2**.

5.6 Extension to fields of arbitrary shape

The proposed algorithms are assuming a rectangular field. In this section, we extend the algorithm to handle fields with arbitrary polygonal shape. This is based on [20]. The **PArea** problem is totally independent from the shape of the field. The extension of **PPos1** and **PPos2** to a non rectangular shape needs some explanations.

Extension for PPos1 A canal is always a straight line. A field of arbitrary polygonal shape is first approximated as a field of rectangular shape. The solution produced by our algorithms for **PPos1** on this rectangular field is then adapted to fit the exact shape near the canal. This approximation might loose the exact property of the algorithms. This is illustrated in Figure 5.11 where hexagon (A, E, F, B, C, D) is approximated by a rectangle (A, B, C, D) with the same area. Then one of the algorithms proposed for **PPos1** is used to decide the plots next to the canal. Finally, these plots are adapted to obtain a real solution on the hexagon.

Extension for PPos2 An extension is only proposed for algorithms **P2_LS**. In **P2_LS**, the zones are computed sequencially. The idea is then to use the same idea as the extension for **PPos1** for each zone. This is illustrated in Figure 5.11, where quadrilateral (A, B, C, D) is approximated as the rectangle (A, B, G, H) for the first zone. Then quadrilateral (A', B', C, D) is approximated as the rectangle (A', B', X, Y) for the second zone.

5.7 Comparison between the government's approach and the proposed approach

This section compares solutions conducted by the government's approach (called government solution) and solutions computed by our incomplete algorithms (called our solution). In order to perform a meaningful comparison, we chose a real instance of ALAP from the 10^{th} hamlet of Dong Trung village, Tien Hai district, Thai Binh province,

Vietnam. In this hamlet, there are 103 households, 3 land categories $\{c_1, c_2, c_3\}$, and 24 fields summing to 204,744 m^2 (8 fields of $c1$, 7 fields of $c2$ and 9 fields of $c3$). This hamlet did already the land reform using the government's approach. The system of coefficients was set to $\langle 1.0, 1.0, 1.0 \rangle$ by the farmers.

Table 5.4 compares the government solution and our solution. As our solution is computed by an incomplete algorithm that could provide different solutions at each run, we made 20 runs and report the minimum, average and maximum value of the 20 solutions. The table reports the following values. The row $Max(c)$ (resp. $Avg(c)$ and $Var(c)$) is, for land category c, the greatest difference (resp. the average difference and the variance of differences) between the expected area and the resulting area of the households. The row Max (resp. Avg and Var) is, for all three land categories, the greatest difference (resp. the average difference and the variance of differences) between the expected area and the attributed area of the households. The row "Added plots" reports the number of added plots.

In the government solution, most households have three plots, each plot for a land category. However, some households (10) are allocated four plots (added plots in Table5.4). In our solution, all household have exactly one plot. Moreover, the values of all the other 12 criteria of our solutions are always much smaller than those for the government solution. Our solutions are thus clearly better than the government solution. Other experimental comparisons between the government approach and the incomplete approach proposed in this thesis can be found in our published papers [19, 20].

These experiments were presented to the Vice President of the Dong Trung village (Thai Binh province). This village is composed of 1460 households, and the land has already been reallocated using the the government's approach. The solutions produced by our algorithms were considered as a significant improvement over the traditional reallocation. The very low difference between the expected area and the allocated area for each household, the absence of added plots and the form of plots with a shape close to a square were particularly appreciated. His conclusion was that our tool should be used to the land reform of the remaining Vietnamese districts and provinces.

5.8 Conclusion

The agricultural land allocation problem has arisen in almost every province in Vietnam. Currently, both the Vietnamese government and the farmers are together solving this problem to increase the efficiency of the cultivation of agricultural land. In this paper, we formulated the agricultural land allocation problem and applied some optimization techniques to solve this problem with both complete and incomplete algorithms. All the new models, the new complete algorithms and the new incomplete algorithms, which appear in this paper, are the contribution of this paper. Specially, we showed that solutions computed by our algorithms are much better than solutions that is obtained by implementing the government's approach.

Algorithm 12: P1_LS: A constraint-based local search algorithm for **PPos1**

$\mathcal{S} \leftarrow$ a random subset of $\mathcal{P}$; $vio \leftarrow$ the constraint violation of $\mathcal{S}$; $it \leftarrow 0$;
while $it < maxIt_1$ && $vio > 0$ **do**
  $p \leftarrow$ the plot in $\mathcal{S}$ with the largest area that is not in tabu;
  $\mathcal{S}' \leftarrow$ the neighbor of $\mathcal{S}$ in $N_1(\mathcal{S})$ that is obtained by removing p from $\mathcal{S}$;
  $\mathcal{S}'' \leftarrow$ the best neighbor of $\mathcal{S}$ in $N_3(\mathcal{S})$ that is obtained by swapping a pair of plots such that this pair is not in tabu;
  if $\mathcal{S}'$ *is better than* $\mathcal{S}''$ **then**
    $\mathcal{S} \leftarrow \mathcal{S}'$;
  else
    $\mathcal{S} \leftarrow \mathcal{S}''$;
  $vio \leftarrow$ the constraint violation of $\mathcal{S}$;
  $it \leftarrow it + 1$;
$iniSol \leftarrow \mathcal{S}$; $bestSol \leftarrow \mathcal{S}$; $stable \leftarrow 0$;
while $it < maxIt_2$ **do**
  $\mathcal{S}' \leftarrow$ a neighbor of $\mathcal{S}$ in $N_2(\mathcal{S})$ such that the constraint violation of $\mathcal{S}'$ is less than or equal to the constraint violation of $\mathcal{S}$;
  if $\mathcal{S}' \neq NULL$ **then**
    $\mathcal{S} \leftarrow \mathcal{S}'$;
    if $|bestSol| < |\mathcal{S}|$ **then**
      $bestSol \leftarrow \mathcal{S}$;
  else
    $\mathcal{S}'' \leftarrow$ a neighbor of $\mathcal{S}$ in $N_3(\mathcal{S})$ such that the constraint violation of $\mathcal{S}''$ is less than or equal to the constraint violation of $\mathcal{S}$;
    if $\mathcal{S}'' \neq NULL$ && $stable < stableLimit$ **then**
      $\mathcal{S} \leftarrow \mathcal{S}''$;
      $stable \leftarrow stable + 1$;
    else
      $stable \leftarrow 0$;
      $\mathcal{S} \leftarrow iniSol$;
  $it \leftarrow it + 1$;

Instances	**P1_CP**		**P1_IP**		**P1_LS**			
n	t(s)	obj	t(s)	obj	min_obj	avg_obj	max_obj	avg_t(s)
20	0.52	**6**	0.68	**6**	6	6	6	0.28
30	31.35	**8**	4.67	**8**	8	8	8	1.41
40	445.58	**11**	12.17	**11**	11	11	11	2.36
50	11576.74	**12**	69.71	**12**	12	12	12	4.09
60	18000	13	679.33	**13**	13	13	13	6.5
70	18000	14	10888.16	**14**	14	14	14	9.3
80	1800	16	18000	16	15	15.4	16	31.79

Table 5.2: Comparing the proposed algorithms for **PPos1** with respect to the quality of found solution and the computational time

$$Minimize \sum_{i=1}^{n}\sum_{j=1}^{n}\delta_{ij} \tag{5.11a}$$

$$s.t. \tag{5.11b}$$

$$x_{11} = 1 \tag{5.11c}$$

$$y_1 = 1 \tag{5.11d}$$

$$\sum_{j=1}^{n} a_j x_{ij} \geq \sum_{j=1}^{n} a_j x_{(i+1)j}, \forall i \in \{2,\ldots,n-1\} \tag{5.11e}$$

$$\sum_{i=1}^{n} x_{ij} = 1, \forall j = \{2,\ldots,n\} \tag{5.11f}$$

$$y_i \leq \sum_{j=1}^{n} x_{ij}, \forall i \in \{2,\ldots,n\} \tag{5.11g}$$

$$ny_i \geq \sum_{j=1}^{n} x_{ij}, \forall i \in \{2,\ldots,n\} \tag{5.11h}$$

$$Wy_i = \sum_{j=1}^{n} w_{ij}, \forall i \in \{1,\ldots,n\} \tag{5.11i}$$

$$w_{ij} \leq W x_{ji}, \forall i,j \in \{1,\ldots,n\} \tag{5.11j}$$

$$Lw_{ij} \geq a_j x_{ij}, \forall i,j \in \{1,\ldots,n\} \tag{5.11k}$$

$$a_k W(2 - x_{ij} - x_{ik}) \geq a_k w_{ij} - a_j w_{ik}, \forall i,j,k \in \{1,\ldots,n\}, j \neq k \tag{5.11l}$$

$$a_j W(2 - x_{ij} - x_{ik}) \geq a_j w_{ik} - a_k w_{ij}, \forall i,j,k \in \{1,\ldots,n\}, j \neq k \tag{5.11m}$$

$$\delta_{ij} + W(1 - x_{ij}) \geq w_{ij} - \frac{\sum_{k=1}^{n} a_k x_{ik}}{W}, \forall i,j \in \{1,\ldots,n\} \tag{5.11n}$$

$$\delta_{ij} + L(1 - x_{ij}) \geq -w_{ij} + \frac{\sum_{k=1}^{n} a_k x_{ik}}{W}, \forall i,j \in \{1,\ldots,n\} \tag{5.11o}$$

$$x_{ij} \in \{0,1\}, \forall i,j \in \{1,\ldots,n\} \tag{5.11p}$$

$$w_{ij} \geq 0, \forall i,j \in \{1,\ldots,n\} \tag{5.11q}$$

$$\delta_{ij} \geq 0, \forall i,j \in \{1,\ldots,n\} \tag{5.11r}$$

$$y_i \in \{0,1\}, \forall i \in \{1,\ldots,n\} \tag{5.11s}$$

Figure 5.8: MIP model for **PPos2**

$$Minimize \sum_{r \in \mathcal{Z}} x_r c_r \tag{5.12a}$$

$$s.t. \tag{5.12b}$$

$$\sum_{r \in \mathcal{Z}} x_r v_i^r = 1, \forall i \in \{1, \ldots, n\} \tag{5.12c}$$

$$x_r \in \{0, 1\}, \forall r \in \mathcal{Z} \tag{5.12d}$$

Figure 5.9: Master problem

$$Minimize \sum_{i=1}^{n} \delta_i + \sum_{i=1}^{n} y_i \pi_i \tag{5.13a}$$

$$s.t. \tag{5.13b}$$

$$\sum_{i=1}^{n} w_i = W \tag{5.13c}$$

$$Lw_i \geq a_i y_i, \forall i \in \{1, \ldots, n\} \tag{5.13d}$$

$$w_i \leq W y_i, \forall i \in \{1, \ldots, n\} \tag{5.13e}$$

$$a_j W(2 - y_i - y_j) \geq a_j w_i - a_i w_j, \forall i, j \in \{1, \ldots, n\}, i \neq j \tag{5.13f}$$

$$a_i W(y_i + y_j - 2) \leq a_j w_i - a_i w_j, \forall i, j \in \{1, \ldots, n\}, i \neq j \tag{5.13g}$$

$$\delta_i + (1 - y_i)W \geq w_i - \frac{\sum_{j=1}^{n} a_j y_j}{W}, \forall i \in \{1, \ldots, n\} \tag{5.13h}$$

$$\delta_i + (1 - y_i)L \geq -w_i + \frac{\sum_{j=1}^{n} a_j y_j}{W}, \forall i \in \{1, \ldots, n\} \tag{5.13i}$$

$$y_i \in \{0, 1\}, \forall i \in \{1, \ldots, n\} \tag{5.13j}$$

$$w_i \geq 0, \forall i \in \{1, \ldots, n\} \tag{5.13k}$$

$$\delta_i \geq 0, \forall i \in \{1, \ldots, n\} \tag{5.13l}$$

$$\tag{5.13m}$$

Figure 5.10: MIP model for the subproblem

Algorithm 13: A constraint-based local search algorithm for the subproblem

```
1  S ← A random solution to the subproblem;
2  obj ← objective of S;
3  bestSol ← S; bestObj ← obj ;
4  while it < maxIt do
5      S_1 ← the best neighbor of S in N_1(S) and N_2S
       corresponding to a position change of a plot not in tabu;
6      if S_1 is better than S then
7          S ← S_1;
8      else
9          S_2 ← the best neighbour of S in N_3(S) corresponding to
           a position swap of a pair of plots not in tabu;
10         if S_2 is better than S then
11             S ← S_2;
12         else
13             p_1 ← a plot in S; p_2 ← a plot not in S;
14             if p_1 ≠ NULL && p_2 ≠ NULL then
15                 S ← S ∪ {p_2} \ {p1};
16     obj ← objective of S;
17     if obj < bestObj then
18         bestObj ← obj; bestSol ← S;
19         stable ← 0;
20     else
21         if stable = stableLimit then
22             S ← bestSol; stable ← 0;
23         else
24             stable ← stable + 1;
25     if it%restartFreq = 0 then
26         S ← a random solution; obj ← objective of S;
27         bestObj ← obj;bestSol ← S;
28     it ← it + 1;
```

Algorithm 14: Step 1 of **P2_LS**

```
1 i ← 0;
2 while P ≠ NULL do
3     i ← i + 1;
4     S_i ← createOneSubset(P);
5     P ← P \ S_i;
6 return {S_1, ..., S_i};
```

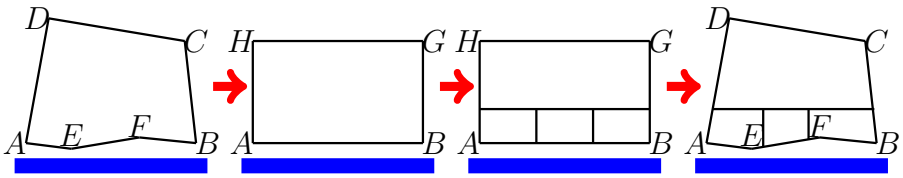

Figure 5.11: Adaptation to the algorithms for **PPos1**

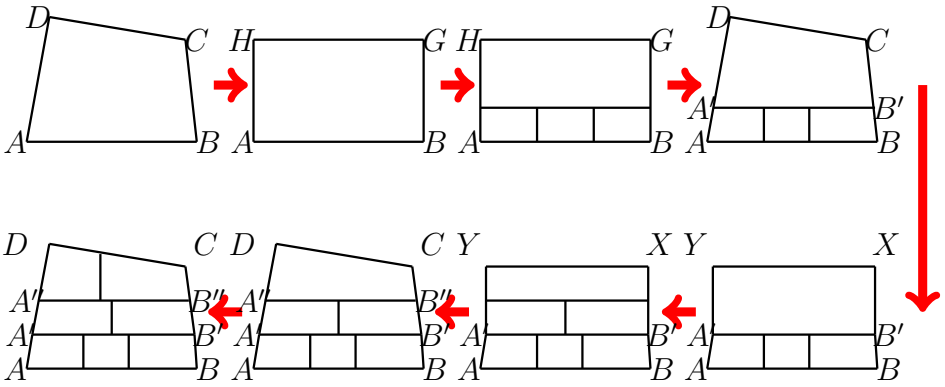

Figure 5.12: Adaptation to the algorithms for **PPos2**

Algorithm 15: Step 2 of **P2_LS**

```
1  stable ← 0; it ← 0;
2  bestSol ← S(solution computed in step 1);
3  while it < maxIt do
4      p ← the plot that is not in tabu with the largest length
       difference between its two sides;
5      S' ← the best neighbor of S in N1(S) that is obtained by
       changing the position of plot p;
6      S'' ← the best neighbor of S in N2(S) that is obtained by
       swapping the position of plot p and some plot that is also not
       in tabu;
7      if S' is better than S'' then
8          S ← S'
9      else
10         S ← S''
11     if S is better than bestSol then
12         bestSol ← S; stable ← 0;
13     else
14         if stable = stableLimit then
15             S ← bestSol; stable ← 0;
16         else
17             stable ← stable + 1;
18     if it%restartFreq = 0 then
19         S ← solution computed in step 1;
20         bestSol ← S;
21     it ← it + 1;
```

Ins	P2_CP		P2_IP		P2_CGIP		P2_CGLI		P2_LS				P2_LCI	
n	t(s)	obj	t(s)	obj	t(s)	obj	t(s)	obj	min_obj	avg_obj	max_obj	avg_t(s)	obj	t(s)
8	0.22	**34.19**	0.52	**34.19**	3.52	**34.19**	0.16	**34.19**	34.19	34.19	34.19	0.14	34.19	0.45
9	2.49	**39.87**	0.75	**39.87**	4.34	**39.87**	0.19	**39.87**	39.87	39.87	39.87	0.15	39.87	0.4
10	13.59	**43.61**	3.26	**43.61**	8.5	**43.61**	0.33	**43.61**	43.61	43.61	43.61	0.82	43.61	1.74
11	121.94	**42.58**	4.52	**42.58**	12.17	**42.58**	0.33	**42.58**	42.58	42.58	42.58	0.25	42.58	0.89
15	18000	77.51	8652	**47.98**	20667	48.96	18214	50.07	47.98	47.98	47.98	1.89	47.98	2.91
20	18000	393.78	14508	69.15	20707	129.23	19598	129.23	69.15	69.15	69.15	103.23	69.15	18.71
25	18000	810.72	7491	70.12	20126	65.07	20236	65.07	63.93	63.93	63.93	83.33	63.93	148.02
30	18000	1889.63	6086	131.88	19392	2031.73	19553	162.32	122.92	123.44	123.64	13.27	122.92	580.61

Table 5.3: Comparing the proposed algorithms for **PPos2** with respect to the quality of found solution and the computational time

Criteria	government solution	20 solutions of our algorithms		
		MIN	AVG	MAX
$Max(c_1)$	88	5	7.75	15
$Avg(c_1)$	39.51	2.06	2.39	3.21
$Var(c_1)$	29.37	1.22	1.87	3.27
$Max(c_2)$	98	6	8.45	17
$Avg(c_2)$	32.93	1.95	2.39	3.16
$Var(c_2)$	25.31	1.29	1.96	3.23
$Max(c_3)$	89	1	3.35	6
$Avg(c_3)$	13.96	0.28	0.85	1.68
$Var(c_3)$	17.73	0.45	0.90	1.43
Max	89	1	1.15	2
Avg	13.65	0.04	0.17	0.33
Var	16.92	0.19	0.36	0.51
Added plots	10	0	0	0

Table 5.4: Comparison between the existent solution conducted by the government's approach and the solutions computed by our incomplete algorithms

6

CONCLUSION

In this section, we summarize the main achievement of this thesis in Section 6.1 and the promising directions for the future work are indicated in Section 6.2.

6.1 Results

The main achievement of this thesis is MIP formulations and algorithms for solving four COPs (QRP, ESPP, ELPP, and ALAP) and additionally two separation algorithms for SECs.

- Chapter 3 focused on solving QRP by proposing five B&C algorithms using four distinct MIP formulations which were also proposed in this thesis. All new algorithms outperform the-state-of-the-art B&I algorithm. This chapter also showed the relationship between the values q and $|S|$ and the performance of the B&C algorithms by the experimental evaluations.
- Chapter 4 dedicated to solve two problems: ESPP and ELPP. This chapter first showed that the equivalence between few problems. Then few B&C algorithms were proposed for solving them. These algorithms exploited new valid inequalities for ESPP (some of which are derived from ASTP), implemented a SEC separation algorithm that was also proposed in this chapter and specially used a proposed inequality filter. These help the proposed

B&C algorithms solve the problems more quickly than the-state-of-the-art algorithms. In the case, the input graph consists of many bridge-blocks or many strongly connected components, we proposed and adapted decomposition techniques for solving the problems more efficiently.

- Chapter 5 presented ALAP which appears in Vietnam. A careful description of this problem was given in this chapter. In addition, we dived this problem into three subproblems and formulated mathematically these subproblems. For each subproblem, we proposed at least a B&I algorithm, a B&C algorithm, and a CBLS algorithm. In addition, we proposed two B&P algorithms for a subproblems.

- All the models, algorithms, data and experiments, which appeared in this thesis, are the open source.

6.2 Future work

The following directions would be interesting.

- **Facetial inequalities** We are going to study facetial inequalities for the elementary path polytope which can be used for solving ESPP and ELPP as integer programs. After this, we will study facetial inequalities for the quromcast rounting polytope which will be used for solving QRP as an integer program.

- **Adding SECs strategies** There are fews strategies for adding SECs in the separation step. Such as, only one violated SEC is added or all violated SECs found are added; the violated SECs are added as local cuts or global cuts. In the step after this thesis, we will try to compare these different strategies in an empirical way.

- **Quality of separated inequalities** There is not any research on measuring the quality of separated inequalities. It is an interesting research direction which focuses on measuring the quality and

the impact of separated inequalities on the performance of the branch-and-cut algorithm.

- **Inequality filter** Too many separated inequalities added in the separation step may reduce the performance of the branch-and-cut algorithm. So an efficient inequality filter, which keeps only very-interesting separated inequalities can accelerate deeply the speed of the B&C algorithm. Improving the simple inequality filter proposed in this thesis is one of our future works.

- **SEC separation** In addition to measuring the quality of separated inequalities, we also try to find out an efficient and fast separation algorithm for SECs that are used in the B&C algorithms for solving problems of finding a tree. Moreover, we continue to improve the decomposition the support graph (see the last section of chapter 4) to solve separation problems more efficiently.

- **ALAP** We will package all current algorithms proposed for solving ALAP into a complete solution software and will introduce it to the Vietnamese government in hope of applying it in Vietnam. However we continue to design new algorithms for covering more expectations of farmers. These new algorithms will be multi-objective algorithms or single objective algorithms.

- We will try to apply the separation algorithms proposed in Chapter 6 in branch-and-cut algorithms for solving a class of vehicle routing problems, for example, the Generalized Vehicle Routing Problem, Capacitated Vehicle Routing Problem (note that these problems can be reduced to problems of finding a set of elementary paths.).

- We will try to develop some branch-and-price algorithms for solving QRP. Then, we compare the branch-and-price algorithms to the branch-and-cut algorithms proposed in this thesis in term of the performance.

BIBLIOGRAPHY

[1] *Comet Tutorial*. Dynamic Decision Technologies, Inc, 2010. 40, 77, 88

[2] T. Achterberg. Scip: Solving constraint integer programs. *Mathematical Programming Computation*, 1(1):1–41, July 2009. `http://mpc.zib.de/index.php/MPC/article/view/4`. 110

[3] T. Achterberg and T. Berthold. Hybrid branching. In W. J. van Hoeve and J. N. Hooker, editors, *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, 6th International Conference, CPAIOR 2009*, volume 5547 of *Lecture Notes in Computer Science*, pages 309 – 311. Springer, 2009. 61

[4] T. Achterberg, T. Koch, and A. Martin. Branching rules revisited. *Operations Research Letters*, 33(1):42 – 54, 2005. 16, 60, 61

[5] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms, and Applications*. Prentice Hall, 1993. 33, 92, 94

[6] K. A. Andersen, K. JÃÂűrnsten, and M. Lind. On bicriterion minimal spanning trees: An approximation. *Computers and amp; Operations Research*, 23(12):1171 – 1182, 1996. 33

[7] D. L. Applegate, R. E. Bixby, V. Chvatal, and W. J. Cook. *The Traveling Salesman Problem: A Computational Study (Princeton Series in Applied Mathematics)*. Princeton University Press, Princeton, NJ, USA, 2007. 15, 39

[8] Y. Arimoto. Impact of land readjustment project on farmland use and structural adjustment: The case of niigata, japan. In *Agricultural and Applied Economics Association 2010 AAEA, CAES, WAEA Joint Annual Meeting,*, 2010. 105

[9] N. Ascheuer, M. Jünger, and G. Reinelt. A branch & cut algorithm for the asymmetric traveling salesman problem with precedence constraints. *Comput. Optim. Appl.*, 17(1):61–84, Oct. 2000. 16, 55, 56, 60, 61

[10] E. Balas, S. Ceria, and G. Cornuéjols. Mixed 0-1 programming by lift-and-project in a branch-and-cut framework. *Management Science*, 42(9):1229–1246, 1996. 57, 64

[11] E. Balas and M. Fischetti. A lifting procedure for the asymmetric traveling salesman polytope and a large new class of facets. *Mathematical Programming*, 58:325–352, 1993. 55

[12] R. Baldacci, A. Mingozzi, and R. Roberti. Recent exact algorithms for solving the vehicle routing problem under capacity and time window constraints. *European Journal of Operational Research*, 218(1):1 – 6, 2012. 47

[13] P. Belotti, C. Kirches, S. Leyffer, J. Linderoth, J. Luedtke, and A. Mahajan. Mixed-integer nonlinear optimization. *Acta Numerica*, 22:1–131, 4 2013. 15, 60

[14] D. Bienstock, M. X. Goemans, D. Simchi-Levi, and D. Williamson. A note on the prize collecting traveling salesman problem. *Mathematical Programming*, (59):413–420, 1993. 45, 48

[15] N. Boland, J. Dethridge, and I. Dumitrescu. Accelerated label setting algorithms for the elementary resource constrained shortest path problem. *Operations Research Letters*, 34(1):58 – 68, 2006. 47

[16] S. Borgwardt, A. Brieden, and P. Gritzmann. Constrained minimum-k-star clustering and its application to the consolidation of farmland. *Operational Research*, 11(1):1–17, 2011. 105

[17] S. Borgwardt, A. Brieden, and P. Gritzmann. Geometric clustering for the consolidation of farmland and woodland. *The Mathematical Intelligencer*, 36(2):37–44, 2014. 105

[18] A. Brieden and P. Gritzmann. A quadratic optimization model for the consolidation of farmland by means of lend-lease agreements. In D. Ahr, R. Fahrion, M. Oswald, and G. Reinelt, editors, *Operations Research Proceedings 2003*, volume 2003 of *Operations Research Proceedings*, pages 324–331. Springer Berlin Heidelberg, 2004. 105

[19] Q. T. Bui, Q.-D. Pham, and Y. Deville. Constraint-based local search for fields partitioning problem. In *Proceedings of the Second Symposium on Information and Communication Technology*, SoICT '11, pages 19–28, New York, NY, USA, 2011. ACM. 122, 128, 133

[20] Q. T. Bui, Q. D. Pham, and Y. Deville. Solving the agricultural land allocation problem by constraint-based local search. In C. Schulte, editor, *Principles and Practice of Constraint Programming*, volume 8124 of *Lecture Notes in Computer Science*, pages 749–757. Springer Berlin Heidelberg, 2013. 115, 119, 122, 128, 132, 133

[21] T. Cay, T. Ayten, and F. Iscan. Effects of different land reallocation models on the success of land consolidation projects: Social and economic approaches. *Land Use Policy*, 27(2):262 – 269, 2010. Forest transitions Wind power planning, landscapes and publics. 105

[22] T. Cay and M. Uyan. Evaluation of reallocation criteria in land consolidation studies using the analytic hierarchy process (ahp). *Land Use Policy*, 30(1):541 – 548, 2013. 105

[23] S. Cheung and A. Kumar. Efficient quorumcast routing algorithms. In *INFOCOM '94. Networking for Global Communications., 13th Proceedings IEEE*, volume 2, pages 840–847, jun 1994. 2, 31

[24] M. Chimani, M. Kandyba, I. Ljubić, and P. Mutzel. Obtaining optimal k-cardinality trees fast. *J. Exp. Algorithmics*, 14:5:2.5–5:2.23, Jan. 2010. 37

[25] S. Chopra and C. Tsai. Polyhedral approaches for the steiner tree problem on graphs. In D.-Z. D. X. Cheng, editor, *Steiner Trees in Industries*, volume 11, pages 175–202. Kluwer Academic Publishers, 2001. 35

[26] J. Clausen. Branch and bound algorithms - principles and examples, 2003. 15

[27] W. Cook. http://www.math.uwaterloo.ca/tsp/concorde/index.html. 79

[28] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms, Third Edition*. The MIT Press, 3rd edition, 2009. 39, 68

[29] A. Costa, J.-F. Cordeau, and G. Laporte. Steiner tree problems with profits. *INFOR*, 44:99–115, 2006. 40, 59

[30] A. C. Cristina Requejo, Agostinho Agra and E. Santos. Formulations for the weightâĂŘconstrained minimum spanning tree problem. In *AIP Conf. Proc. 1281*, pages 2166–2169, 2010. 35

[31] M. Dell'Amico, F. Maffioli, and P. VÃÂd'rbrand. On prize-collecting tours and the asymmetric travelling salesman problem. *International Transactions in Operational Research*, (38):1073 – 1079, 1995. 45, 48

[32] Q.-T. Do and L. Iyer. Land rights and economic development, evidence from vietnam. *The World Bank*, 2003. 4

[33] M. Drexl and S. Irnich. Solving elementary shortest-path problems as mixed-integer programs. *OR Spectrum*, pages 1–16, 2012. 6, 37, 39, 45, 47, 48, 49, 50, 61, 75, 79, 99

[34] B. Du, J. Gu, D. Tsang, and W. Wang. Quorumcast routing by multispace search. In *Global Telecommunications Conference,*

1996. GLOBECOM '96. 'Communications: The Key to Global Prosperity, volume 2, pages 1069 –1073, nov 1996. 2, 31

[35] J. Edmonds. Maximum matching and a polyhedron with 0, 1-vertices. *Journal of Research of the National Bureau of Standards B*, 69:125–130, 1965. 55, 56

[36] D. Feillet, P. Dejax, and M. Gendreau. Traveling salesman problems with profits. *Transportation Science*, 39:188–205, 2005. 48

[37] D. Feillet, P. Dejax, M. Gendreau, and C. Gueguen. An exact algorithm for the elementary shortest path problem with resource constraints: Application to some vehicle routing problems. *Networks*, pages 216–229, 2004. 47

[38] M. Fischetti. Facets of the asymmetric traveling salesman polytope. *Mathematics of Operations Research*, 16(1):pp. 42–56, 1991. 55

[39] M. Fischetti, A. Lodi, and P. Toth. Exact methods for the asymmetric traveling salesman problem. In G. Gutin and A. Punnen, editors, *The Traveling Salesman Problem and Its Variations*, volume 12 of *Combinatorial Optimization*, pages 169–205. Springer US, 2004. 53, 55, 61

[40] T. Fujie. The maximum-leaf spanning tree problem: Formulations and facets. *Networks*, 43(4):212–223, 2004. 33

[41] M. R. Garey and D. S. Johnson. *Computers and Intractability; A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., New York, NY, USA, 1990. 3, 31

[42] F. Glover and M. Laguna. *Tabu Search*. Kluwer Academic Publishers, Norwell, MA, USA, 1997. 29

[43] M. X. Goemans and Y.-S. Myung. A catalog of steiner tree formulations. *Networks*, 23(1):19–28, 1993. 36

[44] M. X. Goemans and D. P. Williamson. A general approximation technique for constrained forest problems. In *Proceedings of*

the third annual ACM-SIAM symposium on Discrete algorithms, pages 307–316, 1992. 45, 48

[45] R. E. Gomory. An algorithm for integer solutions to linear program. In R. L.Graves and P. Wolfe, editors, *Recent Advances in Mathematical Programming*. McGraw-Hill, New York, 1963. 15

[46] M. Gondran and M. Minoux. *Graphes et algorithmes, 3e édition revue et augmentée*. Eyrolles, 1995. 48

[47] M. Grötschel and M. Padberg. Polyhedral theory. In I. E. Lawler, J. Lenstra, A. R. Kan, and S. D.B., editors, *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*, pages 251–305. Wiley, 1985. 55, 56

[48] R. Heltberg. Rural market imperfections and the farm size-productivity relationship:evidence from pakistan. *World Development*, 1998. 4

[49] S. T. Henn. *Weight-Constrained Minimum Spanning Tree Problem*. PhD thesis, University of Kaiserslautern, 2007. 33, 35

[50] S. Hong. *A Linear Programming Approach for the Traveling Salesman Problem*. PhD thesis, Johns Hopkins University, Baltimore, Maryland, USA, 1972. 15

[51] J. Hooker. Search. In *Integrated Methods for Optimization*, volume 170 of *International Series in Operations Research & Management Science*, pages 161–222. Springer US, 2012. 10, 11

[52] Q. Huang, M. Li, Z. Chen, and F. Li. Land consolidation: An approach for sustainable development in rural china. *AMBIO*, 40(1):93–95, 2011. 105

[53] IBM. *IBM ILOG CPLEX V12.1 User's Manual for CPLEX*, 2009. 57

[54] M. M. Ibrahim M, Maculan N. A strong flow-based formulation for the shortest path problem in digraphs with negative cycles. *International transactions in operational research*, 16:361–369, 2009. 35, 42, 45, 47, 49, 50

[55] E. F. a. J. A á aoz and O. Meza. A simple exact separation algorithm for 2-matching in- equalities. Research Report DR-2007/13, EIO Departament, Technical University of Catalo- nia (Spain). http : //www.optimization âĹŠ online.org/DBH T M L/2007/11/1827.html2007., 2007. 61

[56] N. C. J. E. Beasley. An algorithm for the resource constrained shortest path problem. *Network*, 19:379–394, 1989. 49

[57] F. R. James RIDDELL. Farm land rationalisation and land consolidation: strategies for multifunctional use of rural space in eastern and central europe. In *International Symposium on Land Fragmentation and Land Consolidation in CEEC: A Gate Towards Sustainable Rural Development in the New Millennium*, 2002. 105

[58] S. S. Jepsen M, Petersen B. A branch-and-cut algorithm for the elementary shortest path problem with a capacity constraint. Technical report, Department of Computer Science, University of Copenhagen, 2008. 49

[59] R. Jonker and T. Volgenant. Transforming asymmetric into symmetric traveling salesman problems. *Operations Research Letters*, 2(4):161 – 163, 1983. 79

[60] W. K. Union-find algorithms. http://www.cs.princeton.edu/ſs/AlgsDS07/01UnionFind.pdf. 2008. 39

[61] D. Karger, R. Motwani, and G. Ramkumar. On approximating the longest path in a graph. *Algorithmica*, 18:82–98, 1997. 10.1007/BF02523689. 45, 48

[62] T. Koch and A. Martin. Solving steiner tree problems in graphs to optimality. *Networks*, 32:207–232, 1998. 16, 37, 40, 59, 60

[63] T. Koch, A. Martin, and S. Voß. SteinLib: An updated library on steiner tree problems in graphs. Technical Report ZIB-Report 00-37, Konrad-Zuse-Zentrum für Informationstechnik Berlin, Takustr. 7, Berlin, 2000. 40

[64] R. Kulkarni and P. Bhave. Integer programming formulations of vehicle routing problems. *European Journal of Operational Research*, 20(1):58 – 67, 1985. 37, 50

[65] L. M. Lan. Land fragmentation - a constrain for vietnam agriculture. *Vietnamś socioeconomic development*, 2001. 4

[66] I. Ljubić, R. Weiskircher, U. Pferschy, G. W. Klau, P. Mutzel, and M. Fischetti. An algorithmic framework for the exact solution of the prize-collecting steiner tree problem. *Mathematical Programming*, 105:427–449, 2006. 36, 37, 40, 59

[67] C. P. Low. A fast search algorithm for the quorumcast routing problem. *Inf. Process. Lett.*, 66(2):87–92, Apr. 1998. 2, 31, 33

[68] A. Lucena and J. E. Beasley. A branch and cut algorithm for the steiner problem in graphs. *Networks*, 31:39–59, 1998. 40, 59

[69] A. Lucena, N. Maculan, and L. Simonetti. Reformulations and solution algorithms for the maximum leaf spanning tree problem. *Computational Management Science*, 7:289–311, 2010. 33, 36

[70] D. M. Note on the complexity of the shortest path models for column generation in wrptw. *Operations Research*, 42(5):977–978, 1994. 47

[71] T. L. Magnanti and L. A. Wolsey. Chapter 9 optimal trees. In C. M. M.O. Ball, T.L. Magnanti and G. Nemhauser, editors, *Network Models*, volume 7 of *Handbooks in Operations Research and Management Science*, pages 503 – 615. Elsevier, 1995. 33

[72] S. P. March and T. G. MacAulay. Farm size and land use change in vietnam following land reforms. *The University of Sydney*, 2006. 4

[73] V. Ç. D. Mehmet GÜR and Z. DEMIREL. Abstract land consolidation as a tool of rural sustainable development. In *2nd FIG Regional Conference*, 2003. 105

[74] W. Michiels, E. Aarts, and J. Korst. *Theoretical Aspects of Local Search*. Springer, 2007. 25

[75] C. E. Miller, A. W. Tucker, and R. A. Zemlin. Integer programming formulation of traveling salesman problems. *J. ACM*, 7(4):326–329, Oct. 1960. 37, 50

[76] J. E. Mitchell. *Handbook of Applied Optimization*, chapter Branch-and-Cut Algorithms for Combinatorial Optimization Problems, pages 65–77. Oxford University Press, January 2002. 13, 15, 16

[77] V. H. Nguyen and T. T. T. Nguyen. Approximating the asymmetric profitable tour. *Electronic Notes in Discrete Mathematics*, 36:907 – 914, 2010. 45, 48

[78] G. S. Office. *Statistical Yearbook 2004*. Statistical Publishing House, 2004. 4

[79] OscaR Team. OscaR: Scala in OR, 2012. Available from `https://bitbucket.org/oscarlib/oscar`. 110

[80] M. Padberg and G. Rinaldi. Optimization of a 532-city symmetric traveling salesman problem by branch and cut. *Operations Research Letters*, 6(1):1 – 7, 1987. 15, 57

[81] M. Padberg and G. Rinaldi. Facet identification for the symmetric traveling salesman polytope. *Mathematical Programming*, 47:219–257, 1990. 55, 56, 61

[82] M. W. Padberg and M. R. Rao. Odd minimum cut-sets and b-matchings. *Mathematics of Operations Research*, 7:67–80, 1982. 61

[83] J. A. Patrik Sundqvist, Lisa Andersson and E. Jakobsson. *A study of the impacts of land fragmentation on agricultural productivity in Northern Vietnam*. DEPARTMENT OF ECONOMICS, Uppsala University, 2006. 4

[84] Q. D. Pham and Y. Deville. Solving the longest simple path problem with constraint-based techniques. In *Proceedings of the*

9th international conference on Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems, CPAIOR'12, pages 292–306, Berlin, Heidelberg, 2012. Springer-Verlag. 6, 45, 48, 65, 68, 69, 70, 76, 88, 99

[85] Q. D. Pham and Y. Deville. Solving the quorumcast routing problem by constraint programming. *Constraints*, 17:409–431, 2012. 2, 31, 32, 36, 40, 41

[86] D. Portugal, C. Antunes, and R. Rocha. A study of genetic algorithms for approximating the longest path in generic graphs. In *Systems Man and Cybernetics (SMC), 2010 IEEE International Conference on*, pages 2539 –2544, 2010. 45, 48

[87] D. Portugal and R. Rocha. Msp algorithm: multi-robot patrolling based on territory allocation using balanced graph partitioning. In *Proceedings of the 2010 ACM Symposium on Applied Computing*, SAC '10, pages 1271–1276, 2010. 48

[88] A. Punnen. The traveling salesman problem: Applications, formulations and variations. In G. Gutin and A. Punnen, editors, *The Traveling Salesman Problem and Its Variations*, volume 12 of *Combinatorial Optimization*, pages 1–28. Springer US, 2004. 75

[89] A. Radulescu, M. López-Ibáñez, and T. Stützle. Automatically improving the anytime behaviour of multiobjective evolutionary algorithms. Technical Report TR/IRIDIA/2012-019, IRIDIA, Université Libre de Bruxelles, Belgium, 2012. 77

[90] G. Righini and M. Salani. Symmetry helps: Bounded bi-directional dynamic programming for the elementary shortest path problem with resource constraints. *Discrete Optimization*, 3:255–273, 2006. 47

[91] G. Righini and M. Salani. New dynamic programming algorithms for the resource constrained elementary shortest path problem. *Network*, 51:155–170, 2008. 47

[92] F. Rossi, P. v. Beek, and T. Walsh. *Handbook of Constraint Programming (Foundations of Artificial Intelligence)*. Elsevier Science Inc., New York, NY, USA, 2006. 11

[93] M. Rusu. Agricultural land fragmentation and land consolidation rationality. In *13th International Farm Management Congress*, 2002. 105

[94] K. Schmidt and E. G. Schmidt. A longest-path problem for evaluating the worst-case packet delay of switched ethernet. In *SIES*, pages 205–208, 2010. 48

[95] G. Skorobohatyj. Finding a minimum cut between all pairs of nodes in an undirected graph, 2008. 39

[96] I. Stefan and G. Desaulniers. *Shortest Path Problems with Resource Constraints, in Column Generation*. Springer, 2005. 47

[97] R. Tarjan. Depth-frist search and linear graph algorithm. *SIAM J. Comput*, 1:146–160, 1972. 65

[98] T. A. Tayfun CAY and T. Fatih ISCAN. An investigation of reallocation model based on interview in land consolidation. In *Proceeding of the 23rd GIF Congress, Shaping the Change*, 2006. 105

[99] O. team. Large neighborhood search. https://www.info.ucl.ac.be/ pschaus/cp4impatient/lns.html. 28

[100] V. D. Toth P. *The vehicle routing problem*. SIAM Monographs on Discrete Mathematics and Applications, Philadelphia, 2002. 47

[101] I. Tseng, H. Chen, and L. C.I. Obstacle-aware longest-path routing with parallel milp solvers. In *Proceedings of WCECS-ICCS, Vol. 2. pp. 827–831*, 2010. 48

[102] E. Uchoa. Reduction tests for the prize-collecting steiner problem. *Operations Research Letters*, 34(4):437 – 444, 2006. 40, 59

[103] P. Van Hentenryck and L. Michel. *Constraint-based local search*. The MIT Press, London, England, 2005. 28, 29, 30

[104] T. Volgenant and R. Jonker. On some generalizations of the travelling-salesman problem. *Journal of the Operational Research Society*, (3):297 – 308, 1987. 48, 52

[105] B. Wang and J. Hou. An efficient QoS routing algorithm for quorumcast communication. *Computer Networks Journal*, 44(1):43–61, 2004. 2, 31

[106] W. Y. Wong. Information retrieval in p2p networks using genetic algorithm. In *In Proceedings of the 14th International World Wide Web Conference, Pages 922–923*, 2005. 45, 48