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) SoICIT, HUST, Vietnam
- Pr. Laurence Wolsey (Examineur) CORE, UCL, Belgium
- Pr. Pierre Schaus (Examineur) ICTEAM, UCL, Belgium
- Pr. Peter Van Roy (Président) ICTEAM, UCL, Belgium
- Pr. Bernard Fortz (Examineur) 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
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 $S$
- $\delta_G(S) = \delta_G^+(S) \cup \delta_G^-(S)$

v
• 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 $X \subset A_G$, $c_G(X) = \sum_{(i,j) \in W} c_G(i,j)$
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 high-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.
viii
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 lifestyle.

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 développement).
# Table of Contents

<table>
<thead>
<tr>
<th>Section Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table of Contents</td>
<td>xi</td>
</tr>
<tr>
<td>List of Figures</td>
<td>xv</td>
</tr>
<tr>
<td>List of Tables</td>
<td>xix</td>
</tr>
<tr>
<td><strong>1 Introduction</strong></td>
<td></td>
</tr>
<tr>
<td>1.1 Combinatorial Optimization Problem</td>
<td>1</td>
</tr>
<tr>
<td>1.2 Applications</td>
<td>2</td>
</tr>
<tr>
<td>1.2.1 Quorumcast Routing Problem</td>
<td>2</td>
</tr>
<tr>
<td>1.2.2 Elementary Shortest and Longest Path Problems</td>
<td>3</td>
</tr>
<tr>
<td>1.2.3 Agricultural Land Allocation Problem</td>
<td>3</td>
</tr>
<tr>
<td>1.3 Contributions</td>
<td>5</td>
</tr>
<tr>
<td>1.4 Publications</td>
<td>7</td>
</tr>
<tr>
<td>1.5 Outline</td>
<td>8</td>
</tr>
<tr>
<td><strong>2 Optimization techniques</strong></td>
<td></td>
</tr>
<tr>
<td>2.1 Complete search methods</td>
<td>9</td>
</tr>
<tr>
<td>2.1.1 Branching search method</td>
<td>10</td>
</tr>
<tr>
<td>2.1.2 Branch-and-infer search method</td>
<td>11</td>
</tr>
<tr>
<td>2.1.3 Branch-and-cut search method</td>
<td>13</td>
</tr>
<tr>
<td>2.1.4 Branch-and-price search method</td>
<td>19</td>
</tr>
<tr>
<td>2.2 Local search</td>
<td>25</td>
</tr>
<tr>
<td>2.2.1 Local search algorithm</td>
<td>26</td>
</tr>
<tr>
<td>2.2.2 Constraint-based local search</td>
<td>28</td>
</tr>
<tr>
<td>2.2.3 CP-Large neighborhood search</td>
<td>28</td>
</tr>
<tr>
<td>2.2.4 Metaheuristics</td>
<td>28</td>
</tr>
</tbody>
</table>
### 3 Quorumcast routing problem

3.1 Related works ........................................ 31
3.2 Mathematical models ................................. 33
   3.2.1 Edge-based formulation: Model 1 ............. 33
   3.2.2 Multi-commodity flows formulation: Model 2 . 35
   3.2.3 Arborescence tree formulation: Model 3 .... 36
   3.2.4 Miller–Tucker–Zemlin formulation: Model 4 . 37
3.3 Solving QRP as mixed integer programs .......... 37
   3.3.1 Lazy constraint approach .................... 39
   3.3.2 Dynamic constraint separation approach ...... 39
   3.3.3 Preprocessing ................................ 40
3.4 Computational experiments ......................... 40
   3.4.1 Comparing the approaches .................. 41
   3.4.2 Effect of the values of $q$ and $|S|$ on the performance of the approaches .......... 43
3.5 Conclusion .......................................... 43

### 4 Elementary Shortest and Longest Path Problems

4.1 Equivalence between the problems ................. 45
   4.1.1 Equivalence between ESPP and ELPP .......... 46
   4.1.2 Equivalence between ESPP($S, T, G$) and ESPP($s, t, G$) 46
   4.1.3 Equivalence between ESPP($S, T, G$) and PTP($d, G$) 46
4.2 Related works ....................................... 47
4.3 MIP formulations for ESPP($s, t, G$) .............. 49
   4.3.1 Arc-flow formulation: Model 1 ............. 49
   4.3.2 Commodity-flow formulation: Model 2 ....... 49
   4.3.3 Miller–Tucker–Zemlin formulation: Model 3 . 50
4.4 Valid inequalities for ESPP($s, t, G$) .............. 52
   4.4.1 $D_k$-inequalities ............................ 55
   4.4.2 $T_k$-inequalities .............................. 55
   4.4.3 2-Matching inequalities ...................... 56
   4.4.4 Maximum outflow inequalities ............... 56
4.5 Solving ESPP($s, t, G$) to optimality .......... 56
   4.5.1 B&C algorithm schema of ESP_FltC .......... 57
   4.5.2 Preprocessing ................................ 59
   4.5.3 Node selection ................................ 60
   4.5.4 Branching variable selection ............... 60
# TABLE OF CONTENTS

4.5.5  Inequality generation .............................. 61
4.5.6  Inequality selection .............................. 64

4.6  Decompositions ....................................... 65
4.6.1  Computing bridge-blocks ......................... 65
4.6.2  Computing strongly connected components ...... 67
4.6.3  ELPP($G$) on sparse directed graphs with many bridges ......................... 68
4.6.4  ESPP($s,t,G$) on sparse directed graphs with many bridge-blocks .............. 72
4.6.5  ESPP($s,t,G$) on sparse directed graphs with strongly connected components .............................................. 73

4.7  Experiments ........................................... 73
4.7.1  Instances ........................................... 74
4.7.2  Settings ............................................ 77
4.7.3  Analysis of classes of valid inequalities ........ 78
4.7.4  Reduction of ESPP to STSP ......................... 79
4.7.5  Solving ESPP($s,t,G$) ............................ 79
4.7.6  Solving ELPP($G$) .................................. 87

4.8  Decomposition for the support graph ................. 91
4.8.1  Properties of the support graph ................ 92
4.8.2  Promising separation algorithm for SECs ....... 94

4.9  Conclusion ........................................... 96

5  Agricultural Land Allocation Problem .................. 105
5.1  Background ........................................... 105
5.1.1  Problem formulation ................................ 106
5.1.2  The government’s approach to ALAP ............... 107
5.1.3  Setting for experiments ......................... 110
5.2  Decomposing ALAP .................................. 110
5.2.1  PArea ............................................ 111
5.2.2  PPos1 ............................................ 112
5.2.3  PPos2 ............................................ 113
5.3  Solving PArea ....................................... 113
5.3.1  Two properties of optimal solutions of PArea .......... 114
5.3.2  Solving PArea by a B&C algorithm ................. 115
5.3.3  Solving PArea by a CBLS algorithm ............... 115
### TABLE OF CONTENTS

<table>
<thead>
<tr>
<th>Section</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>5.3.4</td>
<td>Solving $\text{PArea}$ by a large neighborhood search algorithm based on CP</td>
<td>118</td>
</tr>
<tr>
<td>5.3.5</td>
<td>Experiments</td>
<td>119</td>
</tr>
<tr>
<td>5.4</td>
<td>Solving $\text{PPos1}$</td>
<td>120</td>
</tr>
<tr>
<td>5.4.1</td>
<td>Solving $\text{PPos1}$ by a B&amp;I algorithm</td>
<td>120</td>
</tr>
<tr>
<td>5.4.2</td>
<td>Solving $\text{PPos1}$ by a B&amp;C algorithm</td>
<td>121</td>
</tr>
<tr>
<td>5.4.3</td>
<td>Solving $\text{PPos1}$ by a CBLS algorithm</td>
<td>121</td>
</tr>
<tr>
<td>5.4.4</td>
<td>Experiments</td>
<td>123</td>
</tr>
<tr>
<td>5.5</td>
<td>Solving $\text{PPos2}$</td>
<td>124</td>
</tr>
<tr>
<td>5.5.1</td>
<td>Solving $\text{PPos2}$ by a B&amp;I algorithm</td>
<td>124</td>
</tr>
<tr>
<td>5.5.2</td>
<td>Solving $\text{PPos2}$ by a B&amp;C algorithm</td>
<td>125</td>
</tr>
<tr>
<td>5.5.3</td>
<td>Solving $\text{PPos2}$ by B&amp;P algorithms</td>
<td>126</td>
</tr>
<tr>
<td>5.5.4</td>
<td>Solving $\text{PPos2}$ by a CBLS algorithm</td>
<td>128</td>
</tr>
<tr>
<td>5.5.5</td>
<td>Solving $\text{PPos2}$ by a hybrid incomplete algorithm</td>
<td>129</td>
</tr>
<tr>
<td>5.5.6</td>
<td>Experiments</td>
<td>130</td>
</tr>
<tr>
<td>5.6</td>
<td>Extension to fields of arbitrary shape</td>
<td>132</td>
</tr>
<tr>
<td>5.7</td>
<td>Comparison between the government’s approach and the proposed approach</td>
<td>132</td>
</tr>
<tr>
<td>5.8</td>
<td>Conclusion</td>
<td>134</td>
</tr>
<tr>
<td>6</td>
<td>Conclusion</td>
<td>143</td>
</tr>
<tr>
<td>6.1</td>
<td>Results</td>
<td>143</td>
</tr>
<tr>
<td>6.2</td>
<td>Future work</td>
<td>144</td>
</tr>
</tbody>
</table>

**Bibliography**

147
# List of Figures

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. 

2.1 The graph of the instance  

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

2.3 A two dimensional integer programming problem  

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

2.5 Iterated Local Search  

3.1 Edge-based formulation  

3.2 Multi-commodity flows formulation  

3.3 Arborescence tree formulation  

3.4 Miller–Tucker–Zemlin formulation  

3.5 A summary of computational results for two classes of instances  

3.6 Percentage of solved instances in C2 in given times  

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

4.1 Arc-flow formulation  

4.2 Commodity-flow formulation  

4.3 Miller–Tucker–Zemlin formulation
<table>
<thead>
<tr>
<th>Figure</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>4.4</td>
<td>Transformation from ESPP($s, t, G$) to ATSP</td>
<td>53</td>
</tr>
<tr>
<td>4.5</td>
<td>B&amp;C schema of $ESP_{FltC}$</td>
<td>58</td>
</tr>
<tr>
<td>4.6</td>
<td>In this example, the traditional separation algorithm for SECs cannot find any SECs.</td>
<td>62</td>
</tr>
<tr>
<td>4.7</td>
<td>The auxiliary graph $D$</td>
<td>67</td>
</tr>
<tr>
<td>4.8</td>
<td>The rooted spanning tree $T$ (solid arcs are in $T$, bold arcs are bridges)</td>
<td>67</td>
</tr>
<tr>
<td>4.9</td>
<td>The graph $G$</td>
<td>98</td>
</tr>
<tr>
<td>4.10</td>
<td>The graph $G$ after the preprocessing step</td>
<td>98</td>
</tr>
<tr>
<td>4.11</td>
<td>ESPP($s, t, G$) on an undirected graph with many bridge-blocks, $s = 1, t = 20$</td>
<td>99</td>
</tr>
<tr>
<td>4.12</td>
<td>List of algorithms compared</td>
<td>99</td>
</tr>
<tr>
<td>4.13</td>
<td>Parameter space for $irace$</td>
<td>100</td>
</tr>
<tr>
<td>4.14</td>
<td>Comparing two algorithms in terms of the number of solved instances of the group $P_{first_100}$ in given times</td>
<td>100</td>
</tr>
<tr>
<td>4.15</td>
<td>Comparing two algorithms in terms of the number of solved instances of the group $P_{last_but_one_100}$ in given times</td>
<td>101</td>
</tr>
<tr>
<td>4.16</td>
<td>Comparing $ESP_{Drexl}$ and $ESP_{FltC}$ in terms of the number of solved instances of the group $P_{last_100}$ in given times</td>
<td>101</td>
</tr>
<tr>
<td>4.17</td>
<td>Graph $D'$ constructed from the directed graph $D$ in Figure 4.17 and the strongly connected component containing ${1, 6, 7, 8, 9, 10}$</td>
<td>102</td>
</tr>
<tr>
<td>5.1</td>
<td>A solution guided by the instructions of the government</td>
<td>109</td>
</tr>
<tr>
<td>5.2</td>
<td>An illustration of $PPos1$</td>
<td>112</td>
</tr>
<tr>
<td>5.3</td>
<td>IP model for $PArea$</td>
<td>116</td>
</tr>
<tr>
<td>5.4</td>
<td>CP model for $PArea$</td>
<td>118</td>
</tr>
<tr>
<td>5.5</td>
<td>CP model for $PPos1$</td>
<td>120</td>
</tr>
<tr>
<td>5.6</td>
<td>IP model for $PPos1$</td>
<td>122</td>
</tr>
<tr>
<td>5.7</td>
<td>CP model for $PPos2$</td>
<td>124</td>
</tr>
<tr>
<td>5.8</td>
<td>MIP model for $PPos2$</td>
<td>137</td>
</tr>
<tr>
<td>5.9</td>
<td>Master problem</td>
<td>138</td>
</tr>
</tbody>
</table>
5.10 MIP model for the subproblem . . . . . . . . . . . . . . . . . 138
5.11 Adaptation to the algorithms for PPpos1 . . . . . . . . . . . 140
5.12 Adaptation to the algorithms for PPpos2 . . . . . . . . . . . 140
# List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2.1</td>
<td>Domain of the variables at the nodes of the part of the search tree in Figure 2.2.</td>
</tr>
<tr>
<td>2.2</td>
<td>Profit matrix</td>
</tr>
<tr>
<td>2.3</td>
<td>The capacity of the machines</td>
</tr>
<tr>
<td>2.4</td>
<td>The matrix of the capacity of machine by job</td>
</tr>
<tr>
<td>4.1</td>
<td>420 directed graphs used to create 15,000 instances of the class ( SI_1 )</td>
</tr>
<tr>
<td>4.2</td>
<td>15 directed graphs with many strongly connected component were used to generate 3,000 instances of the class ( SI_3 )</td>
</tr>
<tr>
<td>4.3</td>
<td>Effect of the different classes of valid inequalities</td>
</tr>
<tr>
<td>4.4</td>
<td>Comparing ( ESP_{FltC} ) to Concorde for solving ESPP instances</td>
</tr>
<tr>
<td>4.5</td>
<td>Computational results for random sparse instances</td>
</tr>
<tr>
<td>4.6</td>
<td>Computational results for random dense instances</td>
</tr>
<tr>
<td>4.7</td>
<td>Computational results for pricing instances</td>
</tr>
<tr>
<td>4.8</td>
<td>Computational results for pricing instances (continued)</td>
</tr>
<tr>
<td>4.9</td>
<td>Computational results for pricing instances (continued)</td>
</tr>
<tr>
<td>4.10</td>
<td>Computational results over all 15,000 instances</td>
</tr>
<tr>
<td>4.11</td>
<td>Computational results for instances of the class ( SI_2 )</td>
</tr>
<tr>
<td>4.12</td>
<td>Reducing the graph size of the instances in the class ( SI_2 ) by using the preprocessing by the algorithm ( ESP_{FltC} )</td>
</tr>
<tr>
<td>4.13</td>
<td>Computational results for instances of the class ( SI_3 )</td>
</tr>
<tr>
<td>4.14</td>
<td>Reducing the size of graphs of the instances in the class ( SI_3 ) by the use of preprocessing by the algorithm ( ESP_{FltC} )</td>
</tr>
<tr>
<td>Table</td>
<td>Description</td>
</tr>
<tr>
<td>-------</td>
<td>-------------</td>
</tr>
<tr>
<td>4.15</td>
<td>Computation times in seconds of three algorithms in solving 9 instances of the class LI1</td>
</tr>
<tr>
<td>4.16</td>
<td>Computation times in seconds of 5 algorithms at solving 10 instances of the class LI2</td>
</tr>
<tr>
<td>4.17</td>
<td>Comparing three algorithms in solving 10 instances of the class LI3 in terms of computation time</td>
</tr>
<tr>
<td>4.18</td>
<td>Comparison result of four separation algorithms</td>
</tr>
<tr>
<td>5.1</td>
<td>Comparing the proposed algorithms for PArea with respect to the quality of found solution and the computational time</td>
</tr>
<tr>
<td>5.2</td>
<td>Comparing the proposed algorithms for PPos1 with respect to the quality of found solution and the computational time</td>
</tr>
<tr>
<td>5.3</td>
<td>Comparing the proposed algorithms for PPos2 with respect to the quality of found solution and the computational time</td>
</tr>
<tr>
<td>5.4</td>
<td>Comparison between the existent solution conducted by the government’s approach and the solutions computed by our incomplete algorithms</td>
</tr>
</tbody>
</table>
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.

\[
\text{COP} = \langle X, D, C, f \rangle
\]

- \( X = \{x_1, ..., x_n\} \) 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
Chapter 1. Introduction

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

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 to find an optimal solution with the proof of 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 the 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].
1.2. Applications

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.

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 forget 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.
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;
Chapter 1. Introduction

• 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,
1.4. Publications

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


Chapter 1. Introduction

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.
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 high-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.

\begin{verbatim}
Algorithm 1: A basic branching search algorithm for a minimum COP
1 Let $S \leftarrow \{P_0\}$ and $v_{UB} \leftarrow \infty$;
2 while $S$ is nonempty do
3     Select a subproblem $P \in S$ and remove $P$ from $S$;
4     if $P$ is too hard to solve then
5         Define subproblem $P_1, \ldots, P_m$ of $P$ and add them to $S$;
6     else
7         Let $v$ be the optimal value of $P$ and let $v_{UB} \leftarrow \min\{v, v_{UB}\}$;
8 The optimal value of $P_0$ is $v_{UB}$;
\end{verbatim}

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) \[51\].
2.1. Complete search methods

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 \( P_0 = \langle X, D_0, C_0, f \rangle \leftarrow \) 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 \( f \leftarrow \) objective function that is minimized;
3 \( S \leftarrow \{ P_0 \} \) and \( v_{UB} = +\infty \);
4 \textbf{while} \( S \) is nonempty \textbf{do}
5 \hspace{1em} Select a subproblem \( P = \langle X, D_p, C_p, f \rangle \in S \) and remove \( P \) from \( S \);
6 \hspace{1em} Perform the inference using \( C \) to infer a set of constraints \( C' \);
7 \hspace{1em} \( C_p \leftarrow C_p \cup C' \cup f \leq v_{UB} \);
8 \hspace{1em} \textbf{if} \( \forall x \in X, |D(x)| = 1 \) \textbf{then}
9 \hspace{2em} \text{Let} \( v \) be the solution value of \( P \);
10 \hspace{2em} \( v_{UB} \leftarrow \min\{v, v_{UB}\} \);
11 \hspace{1em} \textbf{else if} \( \forall x \in X, |D(x)| \geq 1 \) \textbf{and} \( \exists x \in X, |D(x)| > 1 \) \textbf{then}
12 \hspace{2em} \text{Define subproblem} \( P_1, \ldots, P_m \) \text{of} \( P \) \text{and add them to} \( S \); \text{for each} \( P_i \), \text{in addition to} \( C \), \text{it has own branching constraints;}
13 \hspace{1em} \textbf{end} \text{if}
14 \textbf{end while}
15 \text{The optimal value of} \( P_0 \) \text{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.

\[ \text{COP} = \langle X, D, C, f \rangle, \]

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.);
- \( C = \{c_1, \ldots, c_{11}\} \) 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](image)

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

<table>
<thead>
<tr>
<th>Nodes</th>
<th>$v_1$</th>
<th>$v_2$</th>
<th>$v_3$</th>
<th>$v_4$</th>
<th>$v_5$</th>
<th>$v_6$</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>r,g,b,y</td>
<td>r,g,b,y</td>
<td>r,g,b,y</td>
<td>r,g,b,y</td>
<td>r,g,b,y</td>
<td>r,g,b,y</td>
</tr>
<tr>
<td>B</td>
<td>r</td>
<td>g,b,y</td>
<td>g,b,y</td>
<td>r,g,b,y</td>
<td>r,g,b,y</td>
<td>r,g,b,y</td>
</tr>
<tr>
<td>C</td>
<td>r</td>
<td>g,b,y</td>
<td>g,b,y</td>
<td>r</td>
<td>g,b,y</td>
<td>g,b,y</td>
</tr>
<tr>
<td>D</td>
<td>r</td>
<td>b</td>
<td>g,y</td>
<td>r</td>
<td>g,y</td>
<td>g,y</td>
</tr>
<tr>
<td>E</td>
<td>r</td>
<td>b</td>
<td>g</td>
<td>r</td>
<td>y</td>
<td>y</td>
</tr>
<tr>
<td>F</td>
<td>r</td>
<td>b</td>
<td>y</td>
<td>r</td>
<td>g</td>
<td>g</td>
</tr>
<tr>
<td>G</td>
<td>r</td>
<td>g,y</td>
<td>g,b,y</td>
<td>b</td>
<td>r,g,y</td>
<td>r,g,y</td>
</tr>
<tr>
<td>H</td>
<td>r</td>
<td>g</td>
<td>b,y</td>
<td>b</td>
<td>r,y</td>
<td>r,y</td>
</tr>
<tr>
<td>I</td>
<td>r</td>
<td>g</td>
<td>b</td>
<td>b</td>
<td>r</td>
<td>y</td>
</tr>
</tbody>
</table>

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.
2.1. Complete search methods

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].
Chapter 2. Optimization techniques

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.

\[
\begin{align*}
\text{min} & \quad z = -6x_1 - 5x_2 \\
\text{s.t} & \quad 3x_1 + x_2 \leq 11 \\
& \quad -x_1 + 2x_2 \leq 5 \\
& \quad x_1, x_2 \geq 0, \text{integer}
\end{align*}
\] (2.1a)

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?
2.1. Complete search methods

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 $\bar{z} \leftarrow +\infty$. Set $z_l \leftarrow -\infty$ for the one problem $l \in L$.

2. **Step 2. Termination:** If $L = \emptyset$, then solution $x^*$ which yielded the incumbent objective value $\bar{z}$ is optimal. If no such $x^*$ exists (i.e., $\bar{z} = +\infty$) 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 $z_l \leftarrow +\infty$ and go to **Step 6**. Let $z_l$ denote the optimal objective value of the relaxation if it is finite and let $x^{lR}$ be an optimal solution; otherwise set $z_l = -\infty$.

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 $z_l \geq \bar{z}$ go to **Step 2**.

   (b) If $z_l < \bar{z}$ and $x^{lR}$ is integral feasible, update $\bar{z} = z_l$, delete from $L$ all problems with $z_j \geq \bar{z}$, and go to **Step 2**.

7. **Step 7. Partitioning:** Let $S_1^l, \ldots, S_k^l$ be a partition of the constraint set $S^l$ of problem $ILP^l$. Add $ILP^{l_1}, \ldots, ILP^{l_k}$ to $L$, where $ILP^{l_j}$ is $ILP^l$ with feasible region restricted to $S_j^l$ and $z_{lj}$ for $j = 1, \ldots, k$ is set to the value of $z_l$ for the parent problem $l$. Go to **Step 2**.

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

\[
\begin{align*}
\text{min} \quad z &= -6x_1 - 5x_2 \\
\text{s.t} \quad 3x_1 + x_2 &\leq 11 \\
-x_1 + 2x_2 &\leq 5 \\
x_1 &\geq 3 \\
x_1, x_2 &\geq 0, \text{ integer}
\end{align*}
\]
Chapter 2. Optimization techniques

Figure 2.3: A two dimensional integer programming problem

and

\[
\begin{align*}
\min \quad & z = -6x_1 - 5x_2 \\
\text{s.t} \quad & 3x_1 + x_2 \leq 11 \\
& -x_1 + 2x_2 \leq 5 \\
& x_1 \leq 2 \\
& x_1, x_2 \geq 0, \text{integer}
\end{align*}
\]  
(2.3a)

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

\[
\begin{align*}
\text{min } & z = -6x_1 - 5x_2 \\
\text{s.t } & 3x_1 + x_2 \leq 11 \\
& -x_1 + 2x_2 \leq 5 \\
& x_1 \leq 2 \\
& 2x_2 + x_2 \leq 7 \\
& x_1, x_2 \geq 0, \text{ integer}
\end{align*}
\]

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
Chapter 2. Optimization techniques

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.
2.1. Complete search methods

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 Assignment 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 $k_2 = \{(1, 0, 0), (0, 1, 0), (0, 0, 1), (1, 0, 1)\}$. As the results, the Master Problem is now formulated as follows:

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

s.t.

$$y_1^1 + y_1^2 + y_4^2 + y_6^2 = 1 \quad (u_1) \quad (2.5)$$

$$y_2^1 + y_2^2 + y_1^2 + y_6^2 = 1 \quad (u_2) \quad (2.6)$$

$$y_3^1 + y_4^2 + y_3^2 + y_5^2 = 1 \quad (u_3) \quad (2.7)$$
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, $\bar{z} \leftarrow +\infty$; for each $l \in L$, $z_l \leftarrow -\infty$.

2 **Step 2.** Termination: If $L = \emptyset$, then the current incumbent solution $x^*$ with the objective value $\bar{z}$ 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 $\bar{z}_l$ and $x^R_l$ 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 $x^R_l$ is an integral solution, go to Step 6. Otherwise, $x^R_l$ is not integral, go to Step 7.

6 **Step 6.** Fathoming and Pruning: If $\bar{z}_l \geq \bar{z}$ go to Step 2. If $\bar{z}_l \geq \bar{z}$ and $x^R_l$ is integral, update $\bar{z} \leftarrow \bar{z}_l$, delete from $L$ all problems with $\bar{z}_l \geq \bar{z}$, 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 $\bar{z}_{lj}$ for $j = 1, \ldots, k$ is set to the value of $\bar{z}_l$ for the parent problem $l$. Go to Step 2.

\[ y^1_1 + y^1_2 + y^1_3 + y^1_4 \leq 1 \quad (v_1) \]
2.1. Complete search methods

| Machine $i = 1$ | 5 | 7 | 3 |
| Machine $i = 2$ | 2 | 10 | 5 |

Table 2.2: Profit matrix

<table>
<thead>
<tr>
<th>$i$</th>
<th>$C_i$</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>2</td>
<td>8</td>
</tr>
</tbody>
</table>

Table 2.3: The capacity of the machines

| Machine $i = 1$ | 3 | 4 | 2 |
| Machine $i = 2$ | 4 | 3 | 4 |

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

$$y_1^2 + y_2^2 + y_3^2 + y_4^2 + y_5^2 + y_6^2 \leq 1 \quad (v_2)$$  \hspace{1cm} (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:

$$max \quad z = 5y_1^1 + 15y_5^2$$  \hspace{1cm} (2.11)

s.t.

$$y_1^1 + 0 = 1 \quad (u_1)$$  \hspace{1cm} (2.12)

$$0 + y_5^2 = 1 \quad (u_2)$$  \hspace{1cm} (2.13)
Chapter 2. Optimization techniques

\[ 0 + y_5^2 = 1 \quad (2.14) \]

\[ y_1^1 + 0 = 1 \quad (2.15) \]

\[ 0 + y_5^2 = 1 \quad (2.16) \]

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

\[
\text{max} \quad (5 - 5)x_1^1 + (7 - 15)x_2^2 + (3 - 0)x_3^3 \\
\text{s.t.} \quad 3x_1^1 + 4x_2^2 + 2x_3^3 \leq 5 
\]

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

Hence the reduced cost \( z(KP_1) - v_1 = 3 - 0 = 3 \)

Subproblem for Machine 2

\[
\text{max} \quad (2 - 5)x_1^2 + (10 - 15)x_2^2 + (3 - 0)x_3^3 \\
\text{s.t.} \quad 5x_1^2 + 3x_2^2 + 4x_3^3 \leq 8 
\]

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

Hence the reduced cost \( z(KP_2) - v_2 = 5 - 0 = 5 \) 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:

\[
\text{max} \quad z = 5y_1^1 + 15y_5^2 + 5y_3^2 \\
\text{s.t.} \\
\quad y_1^1 + 0 + 0 = 1 \quad (u_1) \quad \quad (2.18) \\
\quad 0 + y_5^2 + 0 = 1 \quad (u_2) \quad \quad (2.19) \\
\quad 0 + y_5^2 + y_3^2 = 1 \quad (u_3) \quad \quad (2.20) 
\]
2.2. Local search

\[ y_1^1 + 0 + 0 = 1 \]  \hspace{1cm} (2.21)

\[ 0 + y_3^2 y_5^2 = 1 \]  \hspace{1cm} (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
\[
\begin{align*}
\text{max} & \quad (5 - 5)x_1^1 + (7 - 15)x_2^1 + (3 - 5)x_3^1 \\
\text{s.t.} & \quad 3x_1^1 + 4x_2^1 + 2x_3^1 \leq 5 \\
\end{align*}
\]

Optimal solutions are (1,0,0) and (0,0,0) with \( z = 0 \)
Hence the reduced cost \( z(KP_1) - v_1 = 0 - 0 = 0 \)

Subproblem for Machine 2
\[
\begin{align*}
\text{max} & \quad (2 - 5)x_1^2 + (10 - 15)x_2^2 + (5 - 5)x_3^2 \\
\text{s.t.} & \quad 5x_1^2 + 3x_2^2 + 4x_3^2 \leq 8 \\
\end{align*}
\]

Optimal solutions are (0,0,1) and (0,0,0) with \( z = 0 \)
Hence the reduced cost \( z(KP_2) - v_2 = 0 - 0 = 0 \)
Therefore, the reduced costs for all columns are 0. Hence the solution \( y_1^1 = 1, y_2^2 = 1, y_3^2 = 0 \) 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 \cite{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 discuss 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
2.2. Local search

Constraints of the problem are many and complicated, random initial solutions may violate many constraints, as consequently we may lose 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 high-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 function, 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 high-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 of 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](image)

**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.
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 $v \notin T$.

\[
\begin{align*}
\text{minimize} & \quad \sum_{v \in V \setminus \{r\}} c_D(v, x(v)) \\
x(v) \neq \perp & \implies y(v) = y(x(v)) + 1, \forall v \in V_D \setminus \{r\} \\
x(u) = \perp & \implies x(v) \neq u, \forall u, v \in V_D \setminus \{r\} \\
\sum_{v \in S} (x(v) \neq \perp) & \geq q \\
y(r) & = 0 \\
x(r) & = r
\end{align*}
\]

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.
Chapter 3. Quorumcast routing problem

Minimize \[ \sum_{e \in E} c_D(e)x_e \] \hspace{1cm} (3.2a)

\[ \sum_{e \in \delta_D(W)} x_e \leq |W| - 1, \forall W \subset V_D, 2 \leq |W| \leq |V_D| - 1 \] \hspace{1cm} (3.2b)

\[ \sum_{e \in \delta_D(i)} x_e \geq y_i, \forall i \in V_D \] \hspace{1cm} (3.2c)

\[ \sum_{e \in \delta_D(i)} x_e \leq (|V_G| - 1)y_i, \forall i \in V_D \] \hspace{1cm} (3.2d)

\[ 1 + \sum_{e \in E_D} x_e = \sum_{v \in V_D} y_v \] \hspace{1cm} (3.2e)

\[ y_r = 1 \] \hspace{1cm} (3.2f)

\[ \sum_{v \in S} y_v = q \] \hspace{1cm} (3.2g)

\[ x_e \in \{0, 1\}, \forall e \in E_D \] \hspace{1cm} (3.2h)

\[ y_i \in \{0, 1\}, \forall i \in V_D \] \hspace{1cm} (3.2i)

Figure 3.1: Edge-based formulation
3.2. Mathematical models

\[ \text{Minimize} \quad \sum_{(i,j) \in A_G} c_G(i,j)x_{ij} \tag{3.3a} \]

\[ \sum_{(r,i) \in A_G} (y_{ri}^{k} - y_{ir}^{k}) \leq 1, \forall k \in V_G \tag{3.3b} \]

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

\[ \sum_{(k,i) \in A_G} (y_{ki}^{k} - y_{ik}^{k}) \geq -1, \forall k \in V_G \tag{3.3d} \]

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

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

\[ y_{ij}^{k} \leq x_{ij}, \forall (i, j) \in A_G, \forall k \in V_G \cup \{r\} \tag{3.3g} \]

\[ y_{ij}^{k} \geq 0, \forall (i, j) \in A_G, \forall k \in V_G \cup \{r\} \tag{3.3h} \]

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

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 \( y_{ij}^{k} \in \mathbb{R}^+ \) 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
Chapter 3. Quorumcast routing problem

\[
\text{Minimize } \sum_{(i,j) \in A_G} c_G(i,j)x_{ij}
\]

\[(3.4a)\]

\[x(\delta_G^-(r)) = 0\]

\[(3.4b)\]

\[x(\delta_G^+(r)) \geq 1\]

\[(3.4c)\]

\[x(\delta_G^+(W)) \geq x(\delta_G^+(v)), \forall W \subset V_G, 2 \leq |W| \leq |V_G|-1, \forall v \in W, r \notin W\]

\[(3.4d)\]

\[x(\delta_G^-(j)) \leq 1, \forall j \in V_G\]

\[(3.4e)\]

\[x(\delta_G^+(S \setminus \{r\})) = q - 1\]

\[(3.4f)\]

\[x_{ij} \in \{0, 1\}, \forall (i,j) \in A_G\]

\[(3.4g)\]

is sent along an arc only if the arc is traversed. The constraints \[(3.3c), (3.3e)\) and \[(3.3i)\) 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 \[(3.3i)\) 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 \( \text{Mod2\_B&B} \) and \( \text{Mod4\_B&B} \). Model 1 and Model 3 have an exponential number of constraints. The constraints are relaxed, and B&C algorithms are employed.
\[ \text{minimize } \sum_{(i,j) \in A_G} c_G(i,j)x_{ij} \]  
(3.5a)

\[ p_r = 1 \]  
(3.5b)

\[ x(\delta^-_G(r)) = 0 \]  
(3.5c)

\[ x(\delta^+_G(r)) \geq 1 \]  
(3.5d)

\[ x(\delta^-_G(v)) = p_v, \forall v \in V_G \setminus \{r\} \]  
(3.5e)

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

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

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

\[ \sum_{i \in S} p_i = q \]  
(3.5i)

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

\[ p_i \in \{0, 1\}, \forall i \in V_G \]  
(3.5k)

\[ t_i \in \{1 \ldots |V|\}, \forall i \in V_G \]  
(3.5l)

Figure 3.4: Miller–Tucker–Zemlin formulation
3.3. Solving QRP as mixed integer programs

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 \( \text{Mod1}_B&C_{\text{lazy}} \) and \( \text{Mod3}_B&C_{\text{lazy}} \).

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 \( \text{Mod3}_B&C_{\text{dyn}} \), 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
Chapter 3. Quorumcast routing problem

\( C \) 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 \( C_1 \).

We also collect 2500 instances in a class \( C_2 \), generated from 100 undirected graphs of 160 nodes and 25 couples \( \langle q, |S| \rangle \), ranging from \( \{\langle 3, 20 \rangle \text{ to } \langle 119, 140 \rangle \} \). These 100 undirected graphs were extracted from 100 minimum Steiner tree instances of test set \( I_{160} \) 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 [11]. Finally, all experiments were performed on XEN virtual machines with 1 core of a CPU
3.4. Computational experiments

<table>
<thead>
<tr>
<th>Class</th>
<th>Approach</th>
<th>%opt</th>
<th>( T )</th>
<th>( N )</th>
<th>( C )</th>
<th>( T )</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>C1</strong></td>
<td>CP</td>
<td>97.1</td>
<td>na.</td>
<td>na.</td>
<td>na.</td>
<td>80.47</td>
</tr>
<tr>
<td></td>
<td>Mod1_B&amp;C_lazy</td>
<td>100</td>
<td>441.7</td>
<td>50.43</td>
<td>6.58</td>
<td>0.57</td>
</tr>
<tr>
<td></td>
<td>Mod2_B&amp;B</td>
<td>100</td>
<td>2159</td>
<td>1.38</td>
<td>na.</td>
<td>1.06</td>
</tr>
<tr>
<td></td>
<td>Mod3_B&amp;C_lazy</td>
<td>100</td>
<td>398.8</td>
<td>77.51</td>
<td>81.42</td>
<td>0.35</td>
</tr>
<tr>
<td></td>
<td>Mod3_B&amp;C_dyn</td>
<td>100</td>
<td>308.2</td>
<td>3.69</td>
<td>158.1</td>
<td>1.94</td>
</tr>
<tr>
<td></td>
<td>Mod4_B&amp;B</td>
<td>100</td>
<td>306.2</td>
<td>55.9</td>
<td>na.</td>
<td>0.64</td>
</tr>
<tr>
<td><strong>C2</strong></td>
<td>CP</td>
<td>0.08</td>
<td>na.</td>
<td>na.</td>
<td>na.</td>
<td>9.74</td>
</tr>
<tr>
<td></td>
<td>Mod1_B&amp;C_lazy</td>
<td>78.6</td>
<td>92804</td>
<td>8507</td>
<td>1110</td>
<td>150.9</td>
</tr>
<tr>
<td></td>
<td>Mod2_B&amp;B</td>
<td>60.2</td>
<td>66716</td>
<td>7.05</td>
<td>na.</td>
<td>318.6</td>
</tr>
<tr>
<td></td>
<td>Mod3_B&amp;C_lazy</td>
<td>94.4</td>
<td>30938</td>
<td>2312</td>
<td>1077</td>
<td>97.6</td>
</tr>
<tr>
<td></td>
<td>Mod3_B&amp;C_dyn</td>
<td>77.2</td>
<td>8408</td>
<td>54.64</td>
<td>6089</td>
<td>153.0</td>
</tr>
<tr>
<td></td>
<td>Mod4_B&amp;B</td>
<td>95.2</td>
<td>50327</td>
<td>6138</td>
<td>na.</td>
<td>92.8</td>
</tr>
</tbody>
</table>

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; \( T \) is the average number of iterations; \( N \) is the average of the number of nodes in the search tree; \( C \) is the average of the number of separated constraints; \( T \) 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.
Chapter 3. Quorumcast routing problem

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.
3.5. Conclusion

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
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 $T = \langle d, v_1, \ldots, v_k, d \rangle$ 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) \cite{100}. Dror in \cite{70} proved that the ESPPRC is NP-hard. There are well-known labelling algorithms for solving the ESPPRC, based on dynamic programming, for example, \cite{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 \cite{37}, state space relaxation \cite{15, 91}, and bounding \cite{12}. In order to use labelling algorithms to solve ESPP\((s, t, G)\), Drexl and Irnich \cite{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 \cite{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 \cite{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 \cite{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 vari-
ant 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 approxi-
mation 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 associ-
ated with nodes. They also proposed a genetic algorithm for solving
this problem. ELPP($G$) has also been addressed in [94] for evaluat-
ing 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 ap-
ppears 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].
Chapter 4. Elementary Shortest and Longest Path Problems

Minimize \( \sum_{(i,j) \in A_G} c_G(i,j) x_{ij} \)  

subject to

\[
x(\delta^+_G(i)) - x(\delta^-_G(i)) = \begin{cases} 
1, & i = s \\
-1, & i = t \\
0, & i \in V_G \setminus \{s, t\}
\end{cases}
\]  

\( x(\delta^-_G(s)) + x(\delta^+_G(t)) = 0 \)

\[
x(\delta^+_G(S)) - x(\delta^-_G(i)) \geq 0, \forall S \subseteq V_G, |S| \geq 2, t \notin S, \forall i \in S
\]

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

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

\[
\text{Minimize } \sum_{(i,j) \in A_G} c_{G}(i,j) x_{ij} \quad (4.2a)
\]

subject to

\[
z_{ij}^k \leq x_{ij}, \forall k \in V_{G \setminus \{s, t\}}, (i, j) \in A_G, i \neq k, s \neq j, j \neq t \quad (4.2b)
\]

\[
\sum_{i \in V_G^+(s)} z_{s i}^k = y_k, \forall k \in V_{G \setminus \{s, t\}} \quad (4.2c)
\]

\[
\sum_{j \in V_G^+(v)} z_{v j}^k - \sum_{i \in V_G^-(v)} z_{i v}^k = 0, \forall k \in V_{G \setminus \{s, t\}}, v \in V_{G \setminus \{s, k, t\}} \quad (4.2d)
\]

\[
\sum_{i \in V_G^-(k)} z_{i k}^k = y_k, \forall k \in V_{G \setminus \{s, t\}} \quad (4.2e)
\]

\[
x(\delta^+_G(i)) = y_i, \forall i \in V_{G \setminus \{t\}} \quad (4.2f)
\]

\[
x(\delta^-_G(i)) = y_i, \forall i \in V_{G \setminus \{s\}} \quad (4.2g)
\]

\[
x(\delta^+_G(s)) = 1 \quad (4.2h)
\]

\[
x(\delta^-_G(t)) = 1 \quad (4.2i)
\]

\[
x(\delta^-_G(s)) + x(\delta^+_G(t)) = 0 \quad (4.2j)
\]

\[
x_{ij} \in \{0, 1\}, \forall (i, j) \in A_G \quad (4.2k)
\]

\[
y_i \in \{0, 1\}, \forall i \in V_G \quad (4.2l)
\]

\[
z_{ij}^k \geq 0, \forall k \in V_{G \setminus \{s, t\}}, (i, j) \in A_G, i \neq k, s \neq j, j \neq t \quad (4.2m)
\]

Figure 4.2: Commodity-flow formulation
Chapter 4. Elementary Shortest and Longest Path Problems

\[ \text{minimize } \sum_{(i,j) \in A} c_G(i,j)x_{ij} \] (4.3a)

subject to

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

\[ x(\delta_G^-(s)) + x(\delta_G^+(t)) = 0 \] (4.3c)

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

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

\[ p_i \in \{1,\ldots,|V_G|\}, \forall i \in V_G \] (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-
4.4. Valid inequalities for ESPP($s, t, G$)

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

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]:

$$\text{minimize} \quad \sum_{(i,j) \in A_{G'}} c_{G'}(i,j)x_{ij}$$

subject to

$$x(\delta^+(i)) = x(\delta^-(i)) = 1, \quad \forall i \in X'$$

$$x(\delta^+(S)) \geq 1, \quad S \subset V_{G'}, \; S \neq \emptyset$$

$$x_{ij} \in \{0, 1\}, \quad \forall (i, j) \in A_{G'}.$$\\n
We put $X = \{x_{ij} \in X' | (i, j) \in A_{G}\}$.

**Theorem 1** If $X^*$ = $\{x_{ij}^* | (i, j) \in A_{G'}\}$ is an optimal solution to ATSP($G'$), then $X^* = \{x_{ij}^* | (i, j) \in A_{G}\}$ is an optimal solution to ESPP($s, t, G$).
Proof Suppose that $X^*$ corresponds to the shortest Hamiltonian cycle $C = \langle (s, u_1), (u_1, u_2), \ldots, (u_p, t), (t, u_{p+1}), \ldots, (u_{2m+1}, s) \rangle$ (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($P', G'$)

1. $C' \leftarrow P'$;
2. Add arc $(t, v'_1)$ to $C'$;
3. foreach $j \in 1..n + 1$ do
   4. if $v_j \in P'$ then
      5. Add the arc $(v'_j, v'_{j+1})$ to $C'$
   6. else
      7. Add $(v'_j, v_j)$ and $(v_j, v'_{j+1})$ to $C'$;
4. 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 ATSP($G'$)).

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

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 $I(Y')$ is a valid inequality of Model 4.3.1 with $Y' \subseteq X'$, then $I(Y)$ 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):

$$\sum_{j=1}^{k-1} x_{ij_{j+1}} + x_{i_k{i_1}} + 2 \sum_{j=3}^{k} x_{i_j{i_j}} + \sum_{j=4}^{k} \sum_{h=3}^{j-1} x_{i_j{i_h}} \leq k - 1$$

(4.5)

and the $D_k^+$-inequalities:

$$\sum_{j=1}^{k-1} x_{ij_{j+1}} + x_{i_k{i_1}} + 2 \sum_{j=3}^{k} x_{i_j{i_j}} + \sum_{j=3}^{k-1} \sum_{h=2}^{j-1} x_{i_j{i_h}} \leq k - 1.$$  

(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

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

(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:

\[ x(\delta_G(S)) \geq 2x(F) + 1 - |F|, \forall S \subseteq V_G, s \notin S, t \notin S, F \subseteq \delta_G(S), |F| \text{ odd}. \]  

(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:

\[ x(\delta^+_G(v)) \leq 1, \forall v \in V_G \setminus \{t\}. \]  

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

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 \(C\) the current family of valid inequalities for ESPP\((s, t, G)\) and we assume that the linear system \(Ax \geq b\) defining \(C\) contains at least all the inequalities in (4.1b) and (4.1c) of Model 1.

We denote by \(F_0\) and \(F_1\) the sets of variables that have been fixed at 0 and 1, respectively.

Let \(K(C, F_0, F_1) = \{ x : Ax \geq b \\ x_{ij} = 0 \text{ for } x_{ij} \in F_0 \\ x_{ij} = 1 \text{ for } x_{ij} \in F_1 \}\)

and let \(LP(C, F_0, F_1)\) denote the linear program

\[
\text{Min } cx : x \in K(C, F_0, F_1)
\]

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

The active nodes of the enumeration tree are represented by a list \(\mathcal{N} / \mathcal{L}\) 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 $SI = GI = \emptyset$.

3. **Lower bounding:** Solve the linear problem $LP(\mathcal{C} \cup SI, \mathcal{F}_0, \mathcal{F}_1)$. Clear the set of separated inequalities $SI$. If there exists an optimal solution to the linear problem, denote it by $\bar{x}$. If the solution $\bar{x}$ is an elementary path, update the best known solution $x^* = \bar{x}$ and the upper bound value $UB = c\bar{x}$. If the solution $\bar{x}$ 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 $GI$. Generate inequalities valid for ESPP($s, t, G$) but violated by $\bar{x}$ and add them to $GI$. If $\bar{x}$ is an integer-feasible solution, do $\mathcal{C} = \mathcal{C} \cup GI$ and go to Step 3 (Lower bounding).

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

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

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

9. **Branching:** Select an appropriate variable $x_{ij}$ such that $\bar{x}_{ij}$ 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. Solving ESPP($s, t, G$) to optimality

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

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 $\pi$ to $LP(C, F_0, F_1)$ at a node of the branch-and-bound tree, we try to find valid inequalities that are not satisfied by $\pi$, and add them to the model. Algorithms for finding these inequalities often rely on a support graph $\bar{G} = (V_{\bar{G}}, A_{\bar{G}}, c_{\bar{G}})$ with $V_{\bar{G}} = V$, $A_{\bar{G}} = \{(i, j) | \bar{x}_{ij} > 0\}$ and each arc $(i, j) \in A_{\bar{G}}$ is associated with a cost $c_{\bar{G}}(i, j) = \bar{x}_{ij}$.

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 $\pi$ (see Section 4.3.1) means finding a pair $(S, i)$ in which $S \subseteq V_G \setminus \{t\}$ and $i \in S$ such that $\pi(\delta_C^-(S)) < \pi(\delta_C^+(i))$. The algorithm for generating SECs in [33] works on the undirected auxiliary support graph $\bar{D} = (V_{\bar{D}}, E_{\bar{D}}, c_{\bar{D}})$ in which $V_{\bar{D}} = V$, $E_{\bar{D}} = \{(i, j) | \bar{x}_{ij} + \bar{x}_{ji} > 0\}$ with a cost $c_{\bar{D}}(i, j) = \bar{x}_{ij} + \bar{x}_{ji}$ associated to each edge $(i, j) \in E_{\bar{D}}$. 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 $\bar{D}$ 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 $\bar{D}$. If the maximum flow $v - t$ is less than the outflow from this node with respect to $\pi$ (i.e., $\sum_{u \in V_G^+(v)} \pi_{vu}$), then one SEC (4.1d) is generated; in this SEC

\footnote{We would like to thank Michael Drexl, the author of [33], for this information.}
Chapter 4. Elementary Shortest and Longest Path Problems

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

![Diagram of a graph with nodes and edges showing the separation algorithm.]

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 \( \pi(\delta^+_G(u)) \geq 2 \). 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 \( v \in V_G \) to \( t \) in \( G \).

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($\mathcal{G} = (V_G, A_G, c_G), i, t$)**

**Input:** Support graph $\mathcal{G} = (V_G, A_G, c_G)$, 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. $\Delta \leftarrow \sum_{v \in V^+_G(i)} c_G(i, v)$;
3. $OutFlowS \leftarrow \Delta$;
4. $Extend \leftarrow true$;
5. **while** $Extend$ **do**
6. **foreach** $v \in V^+_G(u) : v \notin S, v \neq t, Extend = true$ **do**
7. Insert $v$ into $S$ and $Q$;
8. $OutFlowS \leftarrow OutFlowS + \sum_{u \in V^+_G(v)} c_G(v, u)$;
9. $OutFlowS \leftarrow OutFlowS - \sum_{u \in S} (c_G(u, v) + c_G(v, u))$;
10. **if** $OutFlowS < \Delta$ **then**
11. $Extend \leftarrow false$;
12. **break**;
13. **if** $Q$ is empty **then**
14. $Extend = false$;
15. $S \leftarrow \emptyset$;
16. **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 $O(|V_G|)$, 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* $\mathcal{I} \mathcal{P}$ 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 $\mathcal{I} \mathcal{P}$ 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.
4.6. Decompositions

(b) The geometric measure uses the Euclidean distance between \( \pi \) and the hyperplane \( \alpha x = \beta \), namely the distance between \( \pi \) 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 \( IP \) 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 \( E_D = \{(u, v) | (u, v) \in A_G \lor (v, u) \in A_G \} \). 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-
Chapter 4. Elementary Shortest and Longest Path Problems

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.
4.6. Decompositions

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:

- $S$ are the set of nodes connected from s by an elementary path and $T$ 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 $I(c_i) = \{v \in V_{c_i} \mid (u, v) \in A_G \land u \notin V_{c_i}\}$ and $O(c_i) = \{v \in V_{c_i} \mid (v, u) \in A_G \land u \notin V_{c_i}\}$. If $c_i$ contains s, then s is included in $I(c_i)$. If $c_i$ contains t, then t is included in $O(c_i)$. 

Figure 4.7: The auxiliary graph $D$

Figure 4.8: The rooted spanning tree $T$ (solid arcs are in $T$, bold arcs are bridges)
Chapter 4. Elementary Shortest and Longest Path Problems

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

- **Children**($c_i$) is set of SCCs connected from $c_i$ by an arc, i.e.:
  \[ \text{Children}(c_i) = \{ c_j \in \text{SCC} | i \neq j \land \exists (u, v) \in V_G : u \in V_{c_i}, v \in V_{c_j} \} \]

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. $S$ and $T$ are computed with a depth-first search algorithm [28]. Nodes in $V_G \setminus (S \cap T)$ 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: $\text{SCC} = \{ c_1, \ldots, c_n \}$ and $I(c_i), O(c_i), \text{Parent}(c_i), \text{Children}(c_i)$ 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, $\text{SCC} = \{ c_1, c_2, c_3 \}$, $V_{c_1} = \{ 1, 2, 4, 5 \}$, $V_{c_2} = \{ 6, 7, 8, 9, 10 \}$, $V_{c_3} = \{ 16, 17, 18, 19 \}$, $I(c_1) = \{ 1 \}$, $I(c_2) = \{ 6 \}$, $I(c_3) = \{ 16, 17 \}$, $O(c_1) = \{ 5 \}$, $O(c_2) = \{ 8, 10 \}$, $O(c_3) = \{ 19 \}$, $\text{Parent}(c_1) = \emptyset$, $\text{Parent}(c_2) = \{ c_1 \}$, $\text{Parent}(c_3) = \emptyset$, $\text{Children}(c_1) = \{ c_2 \}$, $\text{Children}(c_2) = \{ c_3 \}$, and $\text{Children}(c_3) = \emptyset$.

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
4.6. Decompositions

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:

\[
\begin{align*}
    h_1(v) &= \max (\max_{r_i \in \text{ChRoot}(v)} H_1(r_i) + c_G(v, r_i), 0) \\
    h_2(v) &= \max (\max_{r_i \in \text{ChRoot}(v)} H_2(r_i) + c_G(r_i, v), 0) \\
    H_1(r_i) &= \max (\max_{v \in S_i} d(r_i, v) + h_1(v), 0) \\
    H_2(r_i) &= \max (h_2(v) + \max_{v \in S_i} d(v, r_i), 0)
\end{align*}
\]

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

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.
Chapter 4. Elementary Shortest and Longest Path Problems

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

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

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

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

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 \( \text{solveELPP}(G(S_i), f^*) \) 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 \( \text{solveELPP}\(\{r_i\}, S_i \setminus \text{Exit}(r_i), G(S_i)\)\), and \( \text{solveELPP}(S_i \setminus \text{Exit}(r_i), \{r_i\}, G(S_i)) \), replacing the various calls.
of the method solveELPP(\{u\}, \{v\}, G(S_i)).

Algorithm 7: ComputeFirstStep(r_i)

1. **foreach** r_j ∈ NextRoot(r_i) **do**
   
   2. ComputeFirstStep(r_j);

3. **foreach** u, v ∈ Exit(r_i) : u ≠ v **do**
   
   4. d(u, v, S_i) ← solveELPP(\{u\}, \{v\}, G(S_i)); ;

5. d_1(r_i) ← solveELPP(\{r_i\}, S_i \ Exit(r_i), G(S_i)) ;

6. d_2(r_i) ← solveELPP(S_i \ Exit(r_i), \{r_i\}, G(S_i)) ;

7. **foreach** v ∈ S_i **do**
   
   8. h_1(v) ← max_{r_j ∈ ChRoot(v)} H_1(r_j) + c_G(v, r_j);
   
   9. h_2(v) ← max_{r_j ∈ ChRoot(v)} H_2(r_j) + c_G(r_j, v);

10. h(v) ← max \{h_1(v), 0\}; h_2(v) ← max \{h_2(v), 0\};

11. H_1(r_i) ← max_{v ∈ S_i \ Exit(r_i)} d(r_i, v) + h_1(v);

12. H_2(r_i) ← max_{v ∈ S_i \ Exit(r_i)} d(v, r_i) + h_2(v);

13. H_1(r_i) ← max \{H_1, d_1(r_i), 0\}; H_2(r_i) ← max \{H_2, d_2(r_i), 0\};

14. L_0 = max_{r_j ∈ NextRoot(r_i)} L(r_j);

15. L_1 ← 0;

16. **foreach** v ∈ S_i : ChRoot(v) ≠ ∅ **do**

17. **if** ChRoot(v) = \{r_j\} **then**

18. l_1 ← max \{H_2(r_j) + c_G(r_j, v), H_1(r_j) + c_G(v, r_j)\};

19. **else**

20. l_1 ← max \{H_2(r_j) + c_G(r_j, v) + c_G(v, r_j) + H_1(r_j)\};

21. L_1 ← max \{L_1, l_1\};

22. L_2 ← max_{v ∈ Exit(r_i)} h_2(u) + d(u, v) + h_1(v);

23. L(r_i) ← max \{L_0, L_1, L_2, 0\};

24. f^* ← max \{f^*, L(r_i)\};

Algorithm 8: ComputeSecondStep(r_i)

1. **foreach** r_j ∈ NextRoot(r_i) **do**

2. ComputeSecondStep(r_j);

3. W ← positive cost of G(S_i);

4. **if** W > f^* **then**

5. d ← solveELPP(G(S_i), f^*);

6. f^* ← 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.
4.7. Experiments

To compute $\text{ESPP}(s, t, G)$ 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 $\text{ESP}_\text{FltC}_\text{SCC}$ 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 $\text{ESP}_\text{FltC}$. Note that each SCC is contracted to a single node: the contracted directed graph is acyclic. Method $\text{SortTopologocialOrdering}(\text{SCC})$ (line 9) sorts $\text{SCC}$ into the sequence $S[1..|\text{SCC}|]$ in a topological ordering so that if there exist $u \in S[i]$ and $v \in S[j]$ and $(u, v) \in A_G$, then $i < j$. After the ordering, $S[1]$ contains the node $s$ and $S[|\text{SCC}|]$ 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** \( u, v \in V_G \) do
   2. \( d(u, v) \leftarrow L; \)
3. **foreach** \( v \in V_G \) do
   4. \( d(v) \leftarrow L; d'(v) \leftarrow L; \)
5. \( SCC \leftarrow \text{compute the set of strongly connected components of } G; \)
6. \( sz \leftarrow |SCC|; \)
7. **foreach** \( c_i \in SCC, u \in I(c_i), v \in O(c_i) \) do
   8. \( d(u, v) \leftarrow \text{solveESP}(u, v, c_i); \)
9. \( S[1..sz] \leftarrow \text{SortTopologocialOrdering}(SCC); \)
10. **for** \( i \leftarrow 1 \) to \( sz \) do
11. **foreach** \( v \in I(S[i]) \) do
12.   **foreach** \( (u, v) \in A_G \) 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.     \( d(v) \leftarrow d(u) + c_G(u, v); \)
15. **foreach** \( v \in O(S[i]) \) do
16.   **foreach** \( u \in I(S[i]) \) do
17.     **if** \( d'(v) > d(u) + d(u, v) \) **then**
18.     \( d'(v) \leftarrow d(u) + d(u, v); \)
19. **foreach** \( v \in O(S[i]) \) do
20. **foreach** \( u \in I(S[i]) \) do
21. **return** \( d(t); \)

The characteristics 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.
4.7. Experiments

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   | \([-10^5; -9.48 \times 10^7]\) | Double       |
| P_first_50           | Pricing  | 30         | 53    | 2551  | \([-10^5; -9.48 \times 10^7]\) | Double       |
| P_first_100          | Pricing  | 30         | 103   | 10101 | \([-10^5; -9.48 \times 10^7]\) | Double       |
| P_penultimate_25     | Pricing  | 30         | 28    | 651   | \([-10^5; 30000]\) | Double       |
| P_penultimate_50     | Pricing  | 30         | 53    | 2551  | \([-30000; 30000]\) | Double       |
| P_penultimate_100    | Pricing  | 30         | 103   | 10101 | \([-30000; 30000]\) | 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 a 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|    | |♯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 undirected 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. Experiments

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 discussed 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); and
- 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 if 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 “$z$”
4.7. Experiments

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_DrexI. 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

<table>
<thead>
<tr>
<th>Class</th>
<th>SEC_ta</th>
<th>SEC_na</th>
<th>SEC_th</th>
<th>#2mc</th>
<th>#2mc_th</th>
<th>#mo</th>
<th>mo_th</th>
<th>#dk</th>
<th>dk_th</th>
<th>#tk</th>
<th>tk_th</th>
<th>♯oi</th>
<th>♯Time (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>100/s</td>
<td>0.25</td>
<td>0.1</td>
<td>0.01</td>
<td>0.05</td>
<td>0.05</td>
<td>1</td>
<td>0.1</td>
<td>1</td>
<td>0.01</td>
<td>1</td>
<td>0.01</td>
<td>1</td>
<td>0.01</td>
</tr>
<tr>
<td>50/s</td>
<td>0.25</td>
<td>0.1</td>
<td>0.01</td>
<td>0.05</td>
<td>0.05</td>
<td>1</td>
<td>0.1</td>
<td>1</td>
<td>0.01</td>
<td>1</td>
<td>0.01</td>
<td>1</td>
<td>0.01</td>
</tr>
<tr>
<td>25/s</td>
<td>0.25</td>
<td>0.1</td>
<td>0.01</td>
<td>0.05</td>
<td>0.05</td>
<td>1</td>
<td>0.1</td>
<td>1</td>
<td>0.01</td>
<td>1</td>
<td>0.01</td>
<td>1</td>
<td>0.01</td>
</tr>
<tr>
<td>10/s</td>
<td>0.25</td>
<td>0.1</td>
<td>0.01</td>
<td>0.05</td>
<td>0.05</td>
<td>1</td>
<td>0.1</td>
<td>1</td>
<td>0.01</td>
<td>1</td>
<td>0.01</td>
<td>1</td>
<td>0.01</td>
</tr>
<tr>
<td>5/s</td>
<td>0.25</td>
<td>0.1</td>
<td>0.01</td>
<td>0.05</td>
<td>0.05</td>
<td>1</td>
<td>0.1</td>
<td>1</td>
<td>0.01</td>
<td>1</td>
<td>0.01</td>
<td>1</td>
<td>0.01</td>
</tr>
</tbody>
</table>

**Chapter 4. Elementary Shortest and Longest Path Problems**
4.7. Experiments

<table>
<thead>
<tr>
<th>Instance type</th>
<th>No. instances</th>
<th>ESP_FltC (s)</th>
<th>Concorde (s)</th>
</tr>
</thead>
<tbody>
<tr>
<td>R_dense_25</td>
<td>30</td>
<td>0.74</td>
<td>7.24</td>
</tr>
<tr>
<td>R_dense_50</td>
<td>30</td>
<td>4.13</td>
<td>16.64</td>
</tr>
<tr>
<td>R_dense_100</td>
<td>30</td>
<td>20.03</td>
<td>51.78</td>
</tr>
<tr>
<td>P_last_25</td>
<td>30</td>
<td>1.37</td>
<td>5067.58</td>
</tr>
</tbody>
</table>

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.

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

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
### Table 4.6: Computational results for random dense instances

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

### Table 4.7: Computational results for pricing instances

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

First of all, from the computational results we conclude that the algorithms ESP_ComFlow and ESP_MTZ are not efficient in solving 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.).
### 4.7. Experiments

<table>
<thead>
<tr>
<th>Instance group</th>
<th>Algorithm</th>
<th>% optimal</th>
<th>B&amp;B nodes (min./avg./max.)</th>
<th>Computation time (min./avg./max.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>P_penultimate_25</td>
<td>ESP_ComFlow</td>
<td>100</td>
<td>0/0.93/149</td>
<td>0.93/3.73/18.38</td>
</tr>
<tr>
<td></td>
<td>ESP_MTZ</td>
<td>82.6</td>
<td>0/361033/2494970</td>
<td>0.06/176.94/1196.37</td>
</tr>
<tr>
<td></td>
<td>ESP_Drex1</td>
<td>100</td>
<td>0/7.6/109</td>
<td>0.03/0.11/13</td>
</tr>
<tr>
<td></td>
<td>ESP_FitC</td>
<td>100</td>
<td>0/1.04/62</td>
<td>0.02/0.09/3.57</td>
</tr>
<tr>
<td>P_penultimate_50</td>
<td>ESP_ComFlow</td>
<td>98.8</td>
<td>0/0.75/58</td>
<td>1.0/216.95/1182.66</td>
</tr>
<tr>
<td></td>
<td>ESP_MTZ</td>
<td>19.4</td>
<td>0/339502/126260</td>
<td>0.19/385.01/1144.67</td>
</tr>
<tr>
<td></td>
<td>ESP_Drex1</td>
<td>100</td>
<td>0/15.89/252</td>
<td>0/1.16/53/04</td>
</tr>
<tr>
<td></td>
<td>ESP_FitC</td>
<td>100</td>
<td>0/3.86/86</td>
<td>0/0.53/8.09</td>
</tr>
<tr>
<td>P_penultimate_100</td>
<td>ESP_ComFlow</td>
<td>0</td>
<td>-/-/-</td>
<td>-/-/-</td>
</tr>
<tr>
<td></td>
<td>ESP_MTZ</td>
<td>0</td>
<td>-/-/-</td>
<td>-/-/-</td>
</tr>
<tr>
<td></td>
<td>ESP_Drex1</td>
<td>100</td>
<td>9/1279.99/11565</td>
<td>1.42/108.71/1185.39</td>
</tr>
<tr>
<td></td>
<td>ESP_FitC</td>
<td>99.9</td>
<td>0/398.44/9935</td>
<td>0/48.74/1114.69</td>
</tr>
</tbody>
</table>

Table 4.8: Computational results for pricing instances (continued)

<table>
<thead>
<tr>
<th>Instance group</th>
<th>Algorithm</th>
<th>% optimal</th>
<th>B&amp;B nodes (min./avg./max.)</th>
<th>Computation time (min./avg./max.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>P_last_25</td>
<td>ESP_ComFlow</td>
<td>100</td>
<td>0/1.04/74</td>
<td>1.45/4.74/26.52</td>
</tr>
<tr>
<td></td>
<td>ESP_MTZ</td>
<td>0</td>
<td>-/-/-</td>
<td>-/-/-</td>
</tr>
<tr>
<td></td>
<td>ESP_Drex1</td>
<td>100</td>
<td>0/13.95/285</td>
<td>0.04/0.13/5.02</td>
</tr>
<tr>
<td></td>
<td>ESP_FitC</td>
<td>100</td>
<td>0/1.76/64</td>
<td>0.03/0.11/4.67</td>
</tr>
<tr>
<td>P_last_50</td>
<td>ESP_ComFlow</td>
<td>95.2</td>
<td>0/2.99/203</td>
<td>0.68/256.19/1198.29</td>
</tr>
<tr>
<td></td>
<td>ESP_MTZ</td>
<td>3</td>
<td>0/0/0</td>
<td>0.36/0.44/0.57</td>
</tr>
<tr>
<td></td>
<td>ESP_Drex1</td>
<td>100</td>
<td>0/70.13/1859</td>
<td>0/4.62/215.17</td>
</tr>
<tr>
<td></td>
<td>ESP_FitC</td>
<td>100</td>
<td>0/15.72/600</td>
<td>0/1.13/33.24</td>
</tr>
<tr>
<td>P_last_100</td>
<td>ESP_ComFlow</td>
<td>0</td>
<td>-/-/-</td>
<td>-/-/-</td>
</tr>
<tr>
<td></td>
<td>ESP_MTZ</td>
<td>0</td>
<td>-/-/-</td>
<td>-/-/-</td>
</tr>
<tr>
<td></td>
<td>ESP_Drex1</td>
<td>95.6</td>
<td>9/1505.53/10256</td>
<td>0.88/114.62/1186.68</td>
</tr>
<tr>
<td></td>
<td>ESP_FitC</td>
<td>99.8</td>
<td>0/512.88/9649</td>
<td>0.85/58.62/1040.2</td>
</tr>
</tbody>
</table>

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_Drex1 and ESP_FitC. 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
Chapter 4. Elementary Shortest and Longest Path Problems

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>% optimal</th>
<th>B&amp;B nodes (min./avg./max.)</th>
<th>Computation time (min./avg./max.)</th>
</tr>
</thead>
<tbody>
<tr>
<td>ESP_ComFlow</td>
<td>62.62</td>
<td>0/4.25/404</td>
<td>0.93/183.23/1798.69</td>
</tr>
<tr>
<td>ESP_MTZ</td>
<td>7</td>
<td>0/38760.42/2494670</td>
<td>0.06/210.34/1196.37</td>
</tr>
<tr>
<td>ESP_DrexI</td>
<td>99.36</td>
<td>0/253.53/11565</td>
<td>0/18.93/1186.68</td>
</tr>
<tr>
<td>ESP_FltC</td>
<td>99.93</td>
<td>0/124.58/15453</td>
<td>0/14.18/1181.18</td>
</tr>
</tbody>
</table>

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_DrexI 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_DrexI 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_DrexI 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_DrexI when the time limit is short.

Solving 5000 instances of the class SI2

We compare the algorithms ESP_DrexI, 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_DrexI is the worst in solving 5,000 instances of the class SI2 in terms of computation time. For instance,
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 $g_{bb\_2500}$.

| Instance group | Before $|V|$ | Before $|A|$ | After $|V|$ (min./avg./max.) | After $|A|$ (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 $ESP\_FltC$.
4.7. Experiments

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 | | Algorithm | % optimal | B&B nodes | Computation time |
|----------------|--------------------------|--------------------------|--------------------------|--------------------------|
| 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 | 0/22.8/167 | 8.11/230.04/1195.56 |
| | | ESP_FltC | 94.1 | 0/124.5/4258 | 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\(\{r_i\}, S_i \)

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

<table>
<thead>
<tr>
<th>Graph class</th>
<th>Before</th>
<th>After</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>$</td>
<td>V</td>
</tr>
<tr>
<td>g_scc_100</td>
<td>100</td>
<td>1903-2005</td>
</tr>
<tr>
<td>g_scc_500</td>
<td>500</td>
<td>10051-10068</td>
</tr>
<tr>
<td>g_scc_1000</td>
<td>1000</td>
<td>20130-20148</td>
</tr>
</tbody>
</table>

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.
4.7. Experiments

Table 4.15: Computation times in seconds of three algorithms in solving 9 instances of the class $LI_1$

| 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 4.16: Computation times in seconds of 5 algorithms at solving 10 instances of the class $LI_2$

| Instances        | $|V|$ | $|A|$ | $|B|$ | $ELP_{CP2_{BB}}$ | $ELP_{CP_{BB}}$ | $ELP_{Drexl}$ | $ELP_{FltC}$ | $ELP_{FltC_{BB}}$ |
|------------------|------|------|------|------------------|------------------|--------------|-------------|------------------|
| planar-n100-m16  | 100  | 16   | 10   | 4.05            | 1.95             | 3.34         | 2.43        | 1.09             |
| planar-n200-m54  | 200  | 54   | 20   | 21.32           | 10.17            | 12.73        | 28.23       | 5.21             |
| planar-n300-m85  | 300  | 85   | 30   | 54.73           | 25.28            | 60.88        | 132.95      | 92.31            |
| planar-n400-m100 | 400  | 100  | 40   | 121.15          | 59.15            | 95.05        | 267.59      | 112.18           |
| planar-n500-m139 | 500  | 139  | 50   | 235.73          | 111.19           | 267.59       | -           | 5.8              |
| planar-n600-m177 | 600  | 177  | 60   | 364.97          | 169.95           | 426.89       | -           | 93.85            |
| planar-n700-m226 | 700  | 226  | 70   | 569.15          | 257.78           | -            | -           | 131.3            |
| planar-n800-m147 | 800  | 147  | 80   | 925.12          | 421.87           | -            | -           | 131.88           |
| planar-n900-m195 | 900  | 195  | 90   | 1161.53         | 554.91           | -            | -           | 15.7             |
| planar-n1000-m217| 1000 | 217  | 100  | 1863.58         | -                | -            | -           | 16.09            |

• $ELP_{CP2_{BB}}$ 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 $solveELPP(\{r_i\}, S_i \setminus \text{Exit}(r_i), G(S_i))$, and $solveELPP(S_i \setminus \text{Exit}(r_i), \{r_i\}, G(S_i))$, 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, \textit{ELP\_CP\_BB}, \textit{ELP\_CP2\_BB}, and \textit{ELP\_FltC\_BB}, with the two algorithms that do not exploit the special properties of the graph (\textit{ELP\_Drexl} and \textit{ELP\_FltC}), we conclude that our proposed dynamic algorithms for solving \textit{ELPP}(G) on graphs with many bridges is very efficient.

**Solving 10 instances of the class LI3**

Three algorithms were tested in this experiment: \textit{ELP\_Drexl}, \textit{ELP\_FltC} and \textit{ELP\_FltC\_BB}. The task is to solve \textit{ELPP}(G) on a directed graph with many bridges. The results in computation time are given in Table 4.17.

| Instances | | | | \textit{ELP\_Drexl} & \textit{ELP\_FltC} & \textit{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 \textit{ELP\_FltC\_BB} clearly dominates the two other algorithms \textit{ELP\_Drexl} and \textit{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 \textit{ELP\_FltC\_BB} solves efficiently \textit{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 $G = (V_G, A_G, c_G)$,

- $mf(i,j,G)$ is the value of maximum flow from a source $i$ to a sink $j$ on $G$;

- a minimum $i-j$ cut of $G$ divides $V_G$ into disjoint subsets $V^{(i)}$ and $V_G \setminus V^{(i)}$ such that $i \in V^{(i)}, j \in V \setminus V^{(i)}$ and total crossing cost is minimized;

- $F^+_G(S) = \{i| \, i \in S, j \in V_G \setminus S, (i,j) \in \delta^+_G(S)\}$ and $F^-_G(S) = \{i| \, i \in S, j \in V_G \setminus S, (j,i) \in \delta^-_G(S)\}$;

- for each strongly connected component $\text{sec}_i$ of the support graph $G$ 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 $V_{G_i} = W \cup \{t_i\}$, $A_{G_i} = \{(u,v) \mid (u,v) \in A_G, u,v \in W\} \cup \{(v,t_i) \mid v \in F^+_G(W)\}$ and the cost function $c_{G_i}, \forall (u,v) \in A_{G_i},$

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

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.
Chapter 4. Elementary Shortest and Longest Path Problems

4.8.1 Properties of the support graph

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

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

**Proof** Assuming that $i \in V_G$ does not belong to any cycle on $G$.

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

$$c_{G'}(u, v) = \begin{cases} c_G(u, v) & u, v \in X, (u, v) \in A_G \\ \sum_{(j, v) \in A_G, j \notin X} c_G(j, v) & u = s', v \in F_G(X) \end{cases}$$

On the graph $G'$, we have

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

and

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

We have

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

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

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

By assumption, node $i$ does not belonging to any cycle on $G$, therefore there exists only one way to walk from $s'$ to $i$ on $G'$, that is to travel directly through arc $(s', i)$, as a result of this and (4.12),
4.8. Decomposition for the support graph

\[ c_{\overline{G}'}(s', i) = c_{\overline{G}'}(\delta_{\overline{G}'}^{-}(i)) = c_{\overline{G}'}(\delta_{\overline{G}'}^{+}(i)) \] (4.14)

From (4.13), (4.14), the procedure used to construct \( G' \) 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 \( \text{sc}_{i} \) 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 \( \text{sc}_{i} \). For each node \( v \) in \( V_{G_i} \), we have

- \( mf(v, t_i, G_i) = mf(v, t, G) \); (4.15)
- 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, \( \forall (u, v) \in A_{\overline{G}}, \)

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

An example of this construction is given in Figure 4.19.

From the way used to construct \( \overline{G} \) and \( G_i \), we have,

\[ mf(v, t, \overline{G}) = mf(v, t, G) \] (4.15)

and

\[ mf(v, t_i, \overline{G}) = mf(v, t_i, G_i) \] (4.16)
Chapter 4. Elementary Shortest and Longest Path Problems

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

$$mf(v, t, G') = \min\left(mf(v, t_i, G'), mf(t_i, t, G')\right)$$

(4.17)

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

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

(4.18)

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

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

(4.19)

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

$$mf(v, t, G) = mf(v, t_i, G_i)$$

Finally, of course, if $(V^{(v)}, V_{G} \setminus V^{(v)})$ 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 $G$.

From above theorems, we have two following observations:

1. If a strongly connected component of $G$ 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 $G'$ and their auxiliary undirected versions instead of solving max-flow/min-cut problems on the directed support graph $G$ and the auxiliary undirected support graph of $G'$, 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.
4.8. Decomposition for the support graph

- The first algorithm, denoted by Algo\textsubscript{1}, detects violated SECs via solving max-flow/min-cut problems on the directed support graph.

- The second algorithm, which is denoted by Algo\textsubscript{2}, 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 Algo\textsubscript{3}, 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 Algo\textsubscript{4}, 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.

<table>
<thead>
<tr>
<th>Algorithms</th>
<th>Second group</th>
<th>First group</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>time (s)</td>
<td>no. SECs</td>
</tr>
<tr>
<td>Algo\textsubscript{1}</td>
<td>3379.94</td>
<td>13649</td>
</tr>
<tr>
<td>Algo\textsubscript{2}</td>
<td>2231.86</td>
<td>13649</td>
</tr>
<tr>
<td>Algo\textsubscript{3}</td>
<td>6.86</td>
<td>0</td>
</tr>
<tr>
<td>Algo\textsubscript{4}</td>
<td>4.04</td>
<td>3916</td>
</tr>
</tbody>
</table>

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 \textbf{Algo1}, \textbf{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, \textbf{Algo3} took 6.86 seconds but could not detect any violated SEC, whereas \textbf{Algo4} needed only 4.04 seconds (58.9 \%) to detect 3916 violated SECs;

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

In some separation instances, \textbf{Algo4} found out some violated SECs which could not found by \textbf{Algo3}. This interesting result comes from the fact that \textbf{Algo3} works directly on the auxiliary undirected support graphs, while \textbf{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.

\section{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
4.9. Conclusion

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
4.9. Conclusion

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

<table>
<thead>
<tr>
<th>Name</th>
<th>Problem</th>
<th>Underlying Model</th>
<th>On Graph</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>ESP_Drexl</strong></td>
<td>ESPP</td>
<td>Model 1 + B&amp;C</td>
<td>Both</td>
<td>[33]</td>
</tr>
<tr>
<td><strong>ESP_ComFlow</strong></td>
<td>ESPP</td>
<td>Model 2 + B&amp;C</td>
<td>Both</td>
<td>[this chapter]</td>
</tr>
<tr>
<td><strong>ESP_MTZ</strong></td>
<td>ESPP</td>
<td>Model 3 + B&amp;C</td>
<td>Both</td>
<td>[this chapter]</td>
</tr>
<tr>
<td><strong>ESP_FltC</strong></td>
<td>ESPP</td>
<td>Model 1 + B&amp;C</td>
<td>Both</td>
<td>[this chapter]</td>
</tr>
<tr>
<td><strong>ESP_FltC_BB</strong></td>
<td>ESPP with bridges</td>
<td>Model 1 + B&amp;C</td>
<td>Both</td>
<td>[this chapter]</td>
</tr>
<tr>
<td><strong>ESP_FltC_SCC</strong></td>
<td>ESPP with SCC</td>
<td>Model 1 + B&amp;C + Dyn</td>
<td>Undirected</td>
<td>[this chapter]</td>
</tr>
<tr>
<td><strong>ELP_Drexl</strong></td>
<td>ELPP</td>
<td>Model 1</td>
<td>Both</td>
<td>[33]</td>
</tr>
<tr>
<td><strong>ELP_FltC</strong></td>
<td>ELPP</td>
<td>Model 1 + B&amp;C</td>
<td>Both</td>
<td>[this chapter]</td>
</tr>
<tr>
<td><strong>ELP_FltC_BB</strong></td>
<td>ELPP with bridges</td>
<td>Model 1 + B&amp;C + Dyn</td>
<td>Both</td>
<td>[this chapter]</td>
</tr>
<tr>
<td><strong>ELP_CP</strong></td>
<td>ELPP</td>
<td>CP</td>
<td>Undirected</td>
<td>[53]</td>
</tr>
<tr>
<td><strong>ELP_CP_BB</strong></td>
<td>ELPP with bridges</td>
<td>CP + Dyn</td>
<td>Undirected</td>
<td>[54]</td>
</tr>
<tr>
<td><strong>ELP_CP2_BB</strong></td>
<td>ELPP</td>
<td>CP + Dyn</td>
<td>Both</td>
<td>[this chapter]</td>
</tr>
</tbody>
</table>

Figure 4.12: List of algorithms compared
Chapter 4. Elementary Shortest and Longest Path Problems

---

### PARAMETER FILE FOR THE ACOTSP SOFTWARE

<table>
<thead>
<tr>
<th># name</th>
<th>switch</th>
<th>type</th>
<th>values</th>
<th>conditions (using R syntax)</th>
</tr>
</thead>
<tbody>
<tr>
<td>sec_la</td>
<td>&quot;--sec_ght_algo&quot;</td>
<td>i</td>
<td>(0,1)</td>
<td></td>
</tr>
<tr>
<td>sec_na</td>
<td>&quot;--sec_heu_algo&quot;</td>
<td>i</td>
<td>(0,1)</td>
<td>sec_to = 0</td>
</tr>
<tr>
<td>2mc</td>
<td>&quot;--2matching&quot;</td>
<td>i</td>
<td>(0,1)</td>
<td></td>
</tr>
<tr>
<td>mo</td>
<td>&quot;--MaximumOutflow&quot;</td>
<td>i</td>
<td>(0,1)</td>
<td></td>
</tr>
<tr>
<td>dk</td>
<td>&quot;--DTime&quot;</td>
<td>i</td>
<td>(0,1)</td>
<td></td>
</tr>
<tr>
<td>tk</td>
<td>&quot;--Tk&quot;</td>
<td>i</td>
<td>(0,1)</td>
<td></td>
</tr>
<tr>
<td>ci</td>
<td>&quot;--other_inequality&quot;</td>
<td>i</td>
<td>(0,1)</td>
<td></td>
</tr>
<tr>
<td>sec_th</td>
<td>&quot;--sec_threshold&quot;</td>
<td>r</td>
<td>(0.00001, 0.99)</td>
<td></td>
</tr>
<tr>
<td>2mc_th</td>
<td>&quot;--2mc_threshold&quot;</td>
<td>r</td>
<td>(1.0, 1.0)</td>
<td>2mc = 1</td>
</tr>
<tr>
<td>mo_th</td>
<td>&quot;--mo_threshold&quot;</td>
<td>r</td>
<td>(0.00001, 0.9999)</td>
<td>mo = 1</td>
</tr>
<tr>
<td>dk_th</td>
<td>&quot;--Dk_threshold&quot;</td>
<td>r</td>
<td>(0.00001, 0.99999)</td>
<td>dk = 1</td>
</tr>
<tr>
<td>tk_th</td>
<td>&quot;--Tk_threshold&quot;</td>
<td>r</td>
<td>(0.00001, 0.99999)</td>
<td>tk = 1</td>
</tr>
</tbody>
</table>

---

Figure 4.13: Parameter space for irace

---

Figure 4.14: Comparing two algorithms in terms of the number of solved instances of the group \textit{P\_first\_100} in given times
4.9. Conclusion

Figure 4.15: Comparing two algorithms in terms of the number of solved instances of the group $P_{\text{last but one}_100}$ in given times

Figure 4.16: Comparing $ESP_{Drexl}$ and $ESP_{FltC}$ in terms of the number of solved instances of the group $P_{\text{last}_100}$ 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
4.9. Conclusion

Figure 4.19: Graph $D'$ is constructed from the directed graph $D$ in Figure 4.17 and the strongly connected component containing \{1, 6, 7, 8, 9, 10\}
Chapter 4. Elementary Shortest and Longest Path Problems
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 \( c \in C \), \( \sum_{h \in H} ha_h^c = \sum_{f \in F, c_f = c} fa_f \).
5.1. Background

- **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 \( c \in C \),
    \[
    ha^c_h = \sum_{f \in F, c_f = c} \sum_{z \in Z^f} \sum_{p \in P^z, h^p = h} pa^p
    \]

- **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.
Chapter 5. Agricultural Land Allocation Problem

For example, a system of coefficients for three land categories could be $(1.0, 1.2, 1.4)$, if $1 \text{ m}^2$ of the first land category is equivalent to $1.2 \text{ m}^2$ of the second land category and $1.4 \text{ m}^2$ 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 $R \text{ m}^2$. 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 $S \text{ m}^2$. 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 $R \text{ m}^2$. An equivalent (to $R \text{ m}^2$ 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 $z_j (R \text{ m}^2)$, and will then receive a plot of $S \text{ m}^2$ 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 $R \text{ m}^2$ and an equivalent (to $S \text{ m}^2$ 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 $H_1$, $H_2$, $H_3$, $H_4$, and $H_5$, 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 $H_2$ has three plots of two land categories and households $H_3$, $H_4$, $H_5$ are allocated plots whose areas are different from their corresponding expected areas.

![Figure 5.1: A solution guided by the instructions of the government](image)

**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-
5.2. Decomposing ALAP

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 \(F = \{1, \ldots, m\}\), each field \(i \in F\) is associated with an area \(f_a_i\); and a set of households \(H = \{1, \ldots, n\}\), each household \(j \in H\) has an expected area \(h_a_j\) such that \(\sum_{i \in F} f_a_i = \sum_{j \in H} h_a_j\).

**Output:** the relative position of the plots assigned to the households \(S = (r_p_1, \ldots, r_p_n)\), where \(r_p_h \in F, \forall h \in H\).

**Objective:** minimize \(O(S) = \sum_{i \in F} \left| f_a_i - \sum_{j \in H, r_p_j = i} h_a_j \right|\).

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](image)

**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 $\max\{l, w_p\} = \frac{\max\{l, w_p\}}{\min\{l, w_p\}}$ is smaller or equal to a parameter $\alpha$ 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 $\alpha$ is too close to 1, the problem might be unfeasible on some instances. In such a case, the value of $\alpha$ 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, $S = \{S_1, \ldots, S_m\}$.

**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.

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

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-
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: $\text{maxha}$ is the maximum expected area of the households, $\text{maxha} = \max\{ha_i | i \in \mathcal{F}\}$ and $\text{minha}_j$ is the minimum expected area of the households assigned to field $j$, $\text{minha}_j = \min\{ha_i | rp_i = j\}$.

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

\[ \forall j \in \mathcal{F}, \sum_{i \in \mathcal{H}, rp_i = j} ha_i > fa_j - \text{maxha}. \quad (5.1) \]

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

\[ \sum_{i \in \mathcal{H}, rp_i = j} ha_i = fa_j - \text{maxha} - \Delta, \Delta \geq 0. \quad (5.2) \]

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

\[ \sum_{i \in \mathcal{H}, rp_i = k} ha_i = fa_k + \Delta', \Delta' > 0. \quad (5.3) \]

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:

\[
\begin{align*}
O(\mathcal{S}) - O(\mathcal{S}') &= |\text{maxha} + \Delta| + |\Delta'| - (|\text{maxha} - ha_h - \Delta| + |\Delta' - ha_h|) \\
&= ha_h + \Delta' - |\Delta' - ha_h| > 0.
\end{align*}
\]  

(5.4)
This is a clear contradiction of the hypothesis that $S$ is an optimal solution.

**Theorem 2** If $S = (r_{p_1}, \ldots, r_{p_n})$ is an optimal solution to $\text{PArea}$, then

$$\forall j \in \mathcal{F}, \sum_{i \in \mathcal{H}, r_{p_i} = j} h_{a_i} < f_{a_j} + \min h_{a_j}. \quad (5.5)$$

**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 $\text{PArea}$ by a B&C algorithm

In this section, we formulate $\text{PArea}$ 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 $\delta_i$ 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 $\text{PArea}$ by a CBLS algorithm

In this section, we present the CBLS algorithm for solving $\text{PArea}$ that was already appeared in our paper [20]. We denote this algorithm by $\text{PA}_\text{LS}$. Given a solution $S = (r_{p_1}, \ldots, r_{p_i}, \ldots, r_{p_n})$ in which $f_i \in \mathcal{F}$, to $\text{PArea}$, the local search algorithm exploits the two following neighborhoods:

1. **Change-based** neighborhood

   $$N_1(S) = \{(r_{p_1}, \ldots, r'_{p_i}, \ldots, r_{p_n}) | r'_i \neq r_i, r'_i \in \mathcal{F}\}$$
Chapter 5. Agricultural Land Allocation Problem

\[
\text{Minimize } \sum_{i=1}^{m} \delta_i \quad (5.6a)
\]

s.t.

\[
\sum_{j=1}^{m} x_{ij} = 1, \forall i \in \{1, \ldots, n\} \quad (5.6b)
\]

\[
\delta_j \geq f_a_j - \sum_{i=1}^{n} h_a_i x_{ij}, \forall j \in \{1, \ldots, m\} \quad (5.6d)
\]

\[
\delta_j \geq -f_a_j + \sum_{i=1}^{n} h_a_i x_{ij}, \forall j \in \{1, \ldots, m\} \quad (5.6e)
\]

\[
\sum_{i=1}^{n} h_a_i x_{ij} > f_a_j - \max h_a, \forall j \in \{1..m\} \quad (5.6f)
\]

\[
\sum_{i=1}^{n} h_a_i x_{ij} < f_a_j + \max h_a, \forall j \in \{1..m\} \quad (5.6g)
\]

\[
x_{ij} \in \{0, 1\}, \forall i \in \{1, \ldots, n\}, j \in \{1, \ldots, m\} \quad (5.6h)
\]

\[
\delta_i \geq 0, \forall i \in \{1, \ldots, m\} \quad (5.6i)
\]

Figure 5.3: IP model for PArea

2. **Swap-based** neighborhood

\[
N_2(S) = \{(r_{p_i+1}, \ldots, r_{p_i+j}, \ldots, r_{p_i}, \ldots, r_{p_n}) | r_{p_i} \neq r_{p_i+j}, 1 \leq i < i + j \leq n\}
\]

The pseudo-code of \textbf{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. $S \leftarrow \text{A random solution to PArea}; \ obj \leftarrow \text{objective of } S$
2. $bestSol \leftarrow S; \ bestObj \leftarrow obj$
3. while $it < \text{maxIt} \& \& \ obj > 0$ do
4. 
   $S_1 \leftarrow \text{the best neighbor of } S \text{ in } N_1(S) \text{ that is obtained by changing field of a household such that this household is not in tabu;}
5. 
   \textbf{if } S_1 \text{ is better than } S \text{ then}
6. 
   $S \leftarrow S_1$
7. 
   \textbf{else}
8. 
   $S_2 \leftarrow \text{the best neighbor of } S \text{ in } N_2(S) \text{ that is obtained by swapping the position of a pair of households such that this pair is not in tabu;}
9. 
   \textbf{if } S_2 \text{ is better than } S \text{ then}
10. 
       $S \leftarrow S_2$
11. 
   \textbf{else}
12. 
       Swap the position of two random households;
13. 
   \textbf{end if}
14. 
   \textbf{end if}
15. \textbf{end while}
16. \textbf{if } obj < bestObj \textbf{ then}
17. 
   $bestObj \leftarrow \ obj; \ bestSol \leftarrow S$
18. 
   stable \leftarrow 0
19. \textbf{else}
20. 
   \textbf{if } stable = stableLimit \textbf{ then}
21. 
       $S \leftarrow bestSol; \ stable \leftarrow 0$
22. \textbf{else}
23. 
       $stable \leftarrow stable + 1$
24. \textbf{end if}
25. \textbf{if } it \% \text{restartFreq} = 0 \textbf{ then}
26. 
       $S \leftarrow \text{a random solution}; \ obj \leftarrow \text{objective of } S$
27. 
       best \leftarrow obj; bestSol \leftarrow S
28. 
       $it \leftarrow it + 1$
29. \textbf{end if}
Chapter 5. Agricultural Land Allocation Problem

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 \( j \).

\[
\text{Minimize } \sum_{i \in F} \left| f_a_i - \sum_{j \in H} (x_j = i) h_a_j \right| \quad (5.7a)
\]

\[\text{s.t.} \quad \text{binPacking}(x_1, \ldots, x_n; h_a_1, \ldots, h_a_n; l_1, \ldots, l_m) \quad (5.7b)\]

\[l_i < f_a_j + \max h_a \quad (5.7d)\]

\[l_i > f_a_j - \max h_a \quad (5.7e)\]

\[x_i \in \{1, \ldots, m\} \forall i \in \{1, \ldots, n\} \quad (5.7f)\]

\[l_i \in \{0, \ldots, \sum_{j \in F} f_a_j\} \forall i \in \{1, \ldots, m\} \quad (5.7g)\]

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. Solving PArea

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.

<table>
<thead>
<tr>
<th>Instances</th>
<th>PA_CPLNS</th>
<th>PA_IP</th>
<th>PA_LS</th>
</tr>
</thead>
<tbody>
<tr>
<td>( n )</td>
<td>( m )</td>
<td>min_obj</td>
<td>avg_obj</td>
</tr>
<tr>
<td>30</td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>30</td>
<td>4</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>30</td>
<td>5</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>50</td>
<td>3</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>50</td>
<td>4</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>50</td>
<td>5</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>89</td>
<td>7</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>100</td>
<td>8</td>
<td>208</td>
<td>208</td>
</tr>
<tr>
<td>100</td>
<td>9</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

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 $S$.

\[
\text{Minimize } \sum_{i=1}^{n} x_i \quad (5.8a)
\]

\[
\text{s.t. } \quad \sum_{i=1}^{n} a_k x_k \leq \alpha (\sum_{i=1}^{n} a_i x_i)^2, \forall k \in \{1, \ldots, n\} \quad (5.8c)
\]

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

\[
x_i \in \{0, 1\}, \forall i \in P \quad (5.8e)
\]

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 $\alpha$. 

5.4. Solving PPos1

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

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

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 $S$ 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
Chapter 5. Agricultural Land Allocation Problem

\[ \text{Maximize } \sum_{i=1}^{n} x_i \]  \hspace{1cm} (5.9a)

\[ \text{s.t.} \]  \hspace{1cm} (5.9b)

\[ \sum_{i=1}^{n} w_i = W \]  \hspace{1cm} (5.9c)

\[ w_i \leq W x_i, \ \forall i \in \{1, \ldots, n\} \]  \hspace{1cm} (5.9d)

\[ a_i x_i \leq L w_i, \ \forall i \in \{1, \ldots, n\} \]  \hspace{1cm} (5.9e)

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

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

\[ w_i \leq \frac{\sum_{j=1}^{n} a_j x_j}{W}, \forall i \in \{1, \ldots, n\} \]  \hspace{1cm} (5.9h)

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

\[ x_i \in \{0, 1\}, \forall i \in \{1, \ldots, n\} \]  \hspace{1cm} (5.9j)

\[ w_i \geq 0, \forall i \in \{1, \ldots, n\} \]  \hspace{1cm} (5.9k)

Figure 5.6: IP model for PPpos1

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

Given a subset of plots \( S \) that is a solution to PPpos1, the local search algorithm exploits the three following neighborhoods:

1. **Removal-based** neighborhood

\[ N_1(S) = \{ S \setminus \{ h \} \mid h \in S \} \]

2. **Insertion-based** neighborhood

\[ N_2(S) = \{ S \cup \{ h \} \mid h \notin S, h \in H \} \]

3. **Swap-based** neighborhood

\[ N_3(S) = \{ S \setminus \{ h_1 \} \cup \{ h_2 \} \mid h_1 \in S, h_2 \notin S, h_2 \in H \} \]
5.4. Solving PPos1

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 $S'$ is considered better than another solution $S''$ if and only if the constraint violation of $S'$ is strictly smaller than the constraint violation of $S''$.

*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-
Chapter 5. Agricultural Land Allocation Problem

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 \( D(x_p) = \{1, \ldots, n\} \).

\[
\begin{align*}
\text{Minimize} & \quad \sum_{i=1}^{n} \frac{\sum_{j=1}^{n} a_j(x_j = i)}{W} - \frac{W a_i}{\sum_{j=1}^{n} a_j(x_j = i)} \\
\text{s.t.} & \quad x_1 = 1 \\
& \quad \sum_{j=1}^{n} a_j(x_j = i) > \sum_{j=1}^{n} a_j(x_j = i + 1), \forall i \in \{2, \ldots, n - 1\} \\
& \quad RDConstraint(x_1, \ldots, x_n) \\
& \quad x_i \in \{1, \ldots, n\}, \forall i \in \{2, \ldots, n\}
\end{align*}
\]

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 \( \delta_{ij} \). 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 \( \delta_{ij} \) 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 \( \delta_{ij} \) 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 $Z$ be the set of all vectors $r$ that encode all distinct subsets of $\mathcal{P}$. The master problem (Figure 5.9) introduces, for each $r \in Z$, 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 column). 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 $\delta_i$. 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 $\delta_i$ 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 $\pi_i$ 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 $\delta_i$ 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 $S \subset \mathcal{P}$, the CBLS algorithm for the subproblem exploits the three following neighborhoods:

- **Insertion-based** neighborhood
  
  \[ N_1(S) = \{S \cup \{p\}|p \in \mathcal{P} \setminus S\} \]

- **Removal-based** neighborhood
  
  \[ N_2(S) = \{S \setminus \{p\}|p \in S\} \]

- **Swap-based** neighborhood
  
  \[ N_3(S = \{S \setminus \{p_1\} \cup \{p_2\}|p_1, p_2 \in \mathcal{P} \setminus S\} \]
Chapter 5. Agricultural Land Allocation Problem

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:
5.5. Solving PPos2

- **Step 1:** The objective of this step is to partition the whole set of plots into a set of subsets $S = \{S_1, \ldots, S_m\}$. This algorithm repeats the execution of a procedure that is used to create subsets $S_i$ with the smallest possible value of $\sum_{p \in S_i} \left| \sum_{j \in S_i} a_j W - \sum_{j \in S_i} a_j \right|$.

- **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 $S = \{S_1, \ldots, S_m\}$ 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 $\pi_i$.

**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 $S = \{S_1, \ldots, S_i, \ldots, S_j, \ldots, S_m\}$:

- **Change-based** neighborhood: $N_1(S) = \{S_1, \ldots, S_i', \ldots, S_j', \ldots, S_m\}|S'_i = S_i \setminus \{p\}, S_j \cup \{p\}, i \neq j, p \in S_i$

- **Swap-base** neighborhood: $N_2(S) = \{S_1, \ldots, S_i', \ldots, S_j', \ldots, S_m\}|S'_i = S_i \setminus \{p_1\} \cup \{p_2\}, S_j \setminus \{p_2\} \cup \{p_1\}, i \neq j, p_1 \in S_i, p_2 \in S_j$

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 $P2_{LCI}$. 
Chapter 5. Agricultural Land Allocation Problem

The procedure of \textbf{P2\_LCI} is as follows: first, the CBLS component executes \textbf{P2\_LS} many times, to produce many solutions to \textbf{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 \textbf{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 \textbf{PPos2} as an integer program that is similar to the program described in the subsection 5.5.3 with \( Z \) 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; \textbf{P2\_LS} is executed 10 times; the 10 solutions computed by \textbf{P2\_LS} are all used as initial solutions (columns) for \textbf{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 \textbf{P2\_CGIP} and \textbf{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 \textbf{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 \textbf{P2\_LCI} takes 10 solutions computed by \textbf{P2\_LS} as initial columns, but it could not improve these solutions (the solutions computed by \textbf{P2\_LS} are always the best solutions computed
5.5. Solving PPos2

by \textbf{P2\_LS} after 10 executions of the algorithm), which means that
the solutions computed by the CBLS algorithm \textbf{P2\_LS} have very
good quality.

- In comparing the two B&P algorithms, \textbf{P2\_CGLI} is slightly stronger
than \textbf{P2\_CGIP}: both of them solved the first four instances to opti-
mality in the time limit, but \textbf{P2\_CGLI} used much less computa-
tion time than the other for the remaining instances, the solutions
outputted from \textbf{P2\_CGLI} are better than or equal to the solutions
computed by \textbf{P2\_CGIP}.

- A look deep inside the B&P algorithms, almost all the compu-
tational time is used to solve the subproblem to optimality as
mixed integer programming problems. For example, solving the
instance with \( n = 30 \) with \textbf{P2\_CGIP}, 99.98\% of the computation
time is used for solving in all 29 instances of the subproblem, and
with \textbf{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 com-
putation time spent for solving the subproblem to optimality as
mixed integer programming problems is the weakness of the B&P
algorithms.

- The efficiency of \textbf{P2\_LS} is also confirmed when we compare it
with the three complete algorithms; the quality of the solutions
computed by \textbf{P2\_LS} is always better than or equal to the quality
of the solutions computed by the complete algorithms. Moreover
the computation time of \textbf{P2\_LS} is much smaller than the computa-
tion time of the complete algorithms (it is around two minutes).

As the main message of these experiments, among the proposed al-
gorithms, the incomplete algorithm \textbf{P2\_LS} is the most suitable for solv-
ing \textbf{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 PPpos1 and PPpos2 to a non rectangular shape needs some explanations.

Extension for PPpos1 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 PPpos1 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 PPpos1 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 PPpos2 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 PPpos1 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 10th hamlet of Dong Trung village, Tien Hai district, Thai Binh province,
5.7. Comparison between the government’s approach and the proposed approach

Vietnam. In this hamlet, there are 103 households, 3 land categories \{c_1, c_2, c_3\}, and 24 fields summing to 204,744 m² (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 Table 5.4). 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

1. $S \leftarrow$ a random subset of $P$; $vio \leftarrow$ the constraint violation of $S$; $it \leftarrow 0$;
2. while $it < maxIt_1$ && $vio > 0$ do
   3. $p \leftarrow$ the plot in $S$ with the largest area that is not in tabu;
   4. $S' \leftarrow$ the neighbor of $S$ in $N_1(S)$ that is obtained by removing $p$ from $S$;
   5. $S'' \leftarrow$ the best neighbor of $S$ in $N_3(S)$ that is obtained by swapping a pair of plots such that this pair is not in tabu;
   6. if $S'$ is better than $S''$ then
      7. $S \leftarrow S'$;
   else
      8. $S \leftarrow S''$;
   9. $vio \leftarrow$ the constraint violation of $S$;
   10. $it \leftarrow it + 1$;
iniSol $\leftarrow S$; bestSol $\leftarrow S$; stable $\leftarrow 0$;
while $it < maxIt_2$ do
   14. $S' \leftarrow$ a neighbor of $S$ in $N_2(S)$ such that the constraint violation of $S'$ is less than or equal to the constraint violation of $S$;
   if $S' \neq NULL$ then
      15. $S \leftarrow S'$;
      16. if $|bestSol| < |S|$ then
         17. bestSol $\leftarrow S$;
   else
      19. $S'' \leftarrow$ a neighbor of $S$ in $N_3(S)$ such that the constraint violation of $S''$ is less than or equal to the constraint violation of $S$;
      if $S'' \neq NULL$ && stable $< stableLimit$ then
         21. $S \leftarrow S''$;
         22. stable $\leftarrow$ stable + 1;
      else
         25. stable $\leftarrow 0$;
         26. $S \leftarrow iniSol$;
   27. $it \leftarrow it + 1$;
Chapter 5. Agricultural Land Allocation Problem

Table 5.2: Comparing the proposed algorithms for PPos1 with respect to the quality of found solution and the computational time

<table>
<thead>
<tr>
<th>Instances</th>
<th>P1_CP</th>
<th></th>
<th>P1_IP</th>
<th></th>
<th>P1_LS</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>n</td>
<td>t(s)</td>
<td>obj</td>
<td>t(s)</td>
<td>obj</td>
<td>min_obj</td>
<td>avg_obj</td>
</tr>
<tr>
<td>20</td>
<td>0.52</td>
<td>6</td>
<td>0.68</td>
<td>6</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>30</td>
<td>31.35</td>
<td>8</td>
<td>4.67</td>
<td>8</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>40</td>
<td>445.58</td>
<td>11</td>
<td>12.17</td>
<td>11</td>
<td>11</td>
<td>11</td>
</tr>
<tr>
<td>50</td>
<td>11576.74</td>
<td>12</td>
<td>69.71</td>
<td>12</td>
<td>12</td>
<td>12</td>
</tr>
<tr>
<td>60</td>
<td>18000</td>
<td>13</td>
<td>679.33</td>
<td>13</td>
<td>13</td>
<td>13</td>
</tr>
<tr>
<td>70</td>
<td>18000</td>
<td>14</td>
<td>10888.16</td>
<td>14</td>
<td>14</td>
<td>14</td>
</tr>
<tr>
<td>80</td>
<td>1800</td>
<td>16</td>
<td>18000</td>
<td>16</td>
<td>15</td>
<td>15.4</td>
</tr>
</tbody>
</table>
5.8. Conclusion

\[ \text{Minimize} \quad \sum_{i=1}^{n} \sum_{j=1}^{n} \delta_{ij} \]  \hspace{1cm} (5.11a)

s.t.

\[ x_{11} = 1 \]  \hspace{1cm} (5.11b)

\[ y_1 = 1 \]  \hspace{1cm} (5.11c)

\[ \sum_{j=1}^{n} a_j x_{ij} \geq \sum_{j=1}^{n} a_j x_{i(j+1)}, \forall i \in \{2, \ldots, n-1\} \]  \hspace{1cm} (5.11e)

\[ \sum_{i=1}^{n} x_{ij} = 1, \forall j = \{2, \ldots, n\} \]  \hspace{1cm} (5.11f)

\[ y_i \leq \sum_{j=1}^{n} x_{ij}, \forall i \in \{2, \ldots, n\} \]  \hspace{1cm} (5.11g)

\[ n y_i \geq \sum_{j=1}^{n} x_{ij}, \forall i \in \{2, \ldots, n\} \]  \hspace{1cm} (5.11h)

\[ W y_i = \sum_{j=1}^{n} w_{ij}, \forall i \in \{1, \ldots, n\} \]  \hspace{1cm} (5.11i)

\[ w_{ij} \leq W x_{ji}, \forall i, j \in \{1, \ldots, n\} \]  \hspace{1cm} (5.11j)

\[ L w_{ij} \geq a_j x_{ij}, \forall i, j \in \{1, \ldots, n\} \]  \hspace{1cm} (5.11k)

\[ a_k W (2 - x_{ij} - x_{ik}) \geq a_k w_{ij} - a_j w_{ik}, \forall i, j, k \in \{1, \ldots, n\}, j \neq k \]  \hspace{1cm} (5.11l)

\[ a_j W (2 - x_{ij} - x_{ik}) \geq a_j w_{ik} - a_k w_{ij}, \forall i, j, k \in \{1, \ldots, n\}, j \neq k \]  \hspace{1cm} (5.11m)

\[ \delta_{ij} + W (1 - x_{ij}) \geq w_{ij} - \sum_{k=1}^{n} a_k x_{ik} W, \forall i, j \in \{1, \ldots, n\} \]  \hspace{1cm} (5.11n)

\[ \delta_{ij} + L (1 - x_{ij}) \geq -w_{ij} + \sum_{k=1}^{n} a_k x_{ik} W, \forall i, j \in \{1, \ldots, n\} \]  \hspace{1cm} (5.11o)

\[ x_{ij} \in \{0, 1\}, \forall i, j \in \{1, \ldots, n\} \]  \hspace{1cm} (5.11p)

\[ w_{ij} \geq 0, \forall i, j \in \{1, \ldots, n\} \]  \hspace{1cm} (5.11q)

\[ \delta_{ij} \geq 0, \forall i, j \in \{1, \ldots, n\} \]  \hspace{1cm} (5.11r)

\[ y_i \in \{0, 1\}, \forall i \in \{1, \ldots, n\} \]  \hspace{1cm} (5.11s)

Figure 5.8: MIP model for PPos2
Chapter 5. Agricultural Land Allocation Problem

Minimize \( \sum_{r \in \mathbb{Z}} x_r c_r \) \hspace{1cm} (5.12a)

s.t.

\[ \sum_{r \in \mathbb{Z}} x_r v_i^f = 1, \forall i \in \{1, \ldots, n\} \] \hspace{1cm} (5.12b)

\[ x_r \in \{0, 1\}, \forall r \in \mathbb{Z} \] \hspace{1cm} (5.12d)

Figure 5.9: Master problem

Minimize \( \sum_{i=1}^{n} \delta_i + \sum_{i=1}^{n} y_i \pi_i \) \hspace{1cm} (5.13a)

s.t.

\[ \sum_{i=1}^{n} w_i = W \] \hspace{1cm} (5.13c)

\[ L w_i \geq a_i y_i, \forall i \in \{1, \ldots, n\} \] \hspace{1cm} (5.13d)

\[ w_i \leq W y_i, \forall i \in \{1, \ldots, n\} \] \hspace{1cm} (5.13e)

\[ a_j W (2 - y_i - y_j) \geq a_j w_i - a_i w_j, \forall i, j \in \{1, \ldots, n\}, i \neq j \] \hspace{1cm} (5.13f)

\[ a_i W (y_i + y_j - 2) \leq a_j w_i - a_i w_j, \forall i, j \in \{1, \ldots, n\}, i \neq j \] \hspace{1cm} (5.13g)

\[ \delta_i + (1 - y_i) W \geq w_i - \sum_{j=1}^{n} a_{ij} y_j W, \forall i \in \{1, \ldots, n\} \] \hspace{1cm} (5.13h)

\[ \delta_i + (1 - y_i) L \geq -w_i + \sum_{j=1}^{n} a_{ij} y_j W, \forall i \in \{1, \ldots, n\} \] \hspace{1cm} (5.13i)

\[ y_i \in \{0, 1\}, \forall i \in \{1, \ldots, n\} \] \hspace{1cm} (5.13j)

\[ w_i \geq 0, \forall i \in \{1, \ldots, n\} \] \hspace{1cm} (5.13k)

\[ \delta_i \geq 0, \forall i \in \{1, \ldots, n\} \] \hspace{1cm} (5.13l)

(5.13m)

Figure 5.10: MIP model for the subproblem
Algorithm 13: A constraint-based local search algorithm for the subproblem

1. $S \leftarrow$ A random solution to the subproblem;
2. $obj \leftarrow$ objective of $S$;
3. $bestSol \leftarrow S; bestObj \leftarrow obj$;
4. while $it < maxIt$ do
5.   \( S_1 \leftarrow \) the best neighbor of $S$ in $N_1(S)$ and $N_2S$
6.   \( \) corresponding to a position change of a plot not in tabu;
7.   if $S_1$ is better than $S$ then
8.     $S \leftarrow S_1$;
9.   else
10.  \( S_2 \leftarrow \) the best neighbour of $S$ in $N_3(S)$ corresponding to
11.  \( \) a position swap of a pair of plots not in tabu;
12.  if $S_2$ is better than $S$ then
13.     $S \leftarrow S_2$;
14.  else
15.     $p_1 \leftarrow$ a plot in $S$; $p_2 \leftarrow$ a plot not in $S$;
16.     if $p_1 \neq $ NULL \&\& $p_2 \neq $ NULL then
17.         $S \leftarrow S \cup \{p_2\} \setminus \{p_1\}$;
18.  end if
19. end if
20. end if
21. $obj \leftarrow$ objective of $S$;
22. if $obj < bestObj$ then
23.     $bestObj \leftarrow obj; bestSol \leftarrow S$;
24.     stable $\leftarrow 0$;
25. else
26.     if $stable = stableLimit$ then
27.         $S \leftarrow bestSol; stable \leftarrow 0$;
28.     else
29.         stable $\leftarrow stable + 1$;
30. end if
31. end if
32. if $it \% restartFreq = 0$ then
33.     $S \leftarrow$ a random solution; $obj \leftarrow$ objective of $S$;
34.     $bestObj \leftarrow obj; bestSol \leftarrow S$;
35. end if
36. $it \leftarrow it + 1$;
Algorithm 14: Step 1 of P2_LS

1 \( i \leftarrow 0; \)
2 \textbf{while} \( \mathcal{P} \neq \text{NULL} \) \textbf{do}
3 \hspace{1em} \( i \leftarrow i + 1; \)
4 \hspace{1em} \( S_i \leftarrow \text{createOneSubset}(\mathcal{P}); \)
5 \hspace{1em} \( \mathcal{P} \leftarrow \mathcal{P} \setminus S_i; \)
6 \textbf{return} \( \{S_1, \ldots, S_i\}; \)

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. $stable \leftarrow 0; \ it \leftarrow 0$
2. $bestSol \leftarrow S$(solution computed in step 1);
3. **while** $it < maxIt$ **do**
   4. $p \leftarrow$ the plot that is not in tabu with the largest length difference between its two sides;
   5. $S' \leftarrow$ the best neighbor of $S$ in $N_1(S)$ that is obtained by changing the position of plot $p$;
   6. $S'' \leftarrow$ the best neighbor of $S$ in $N_2(S)$ 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. $S \leftarrow S'$
   **else**
      9. $S \leftarrow S''$
   10. **if** $S$ is better than $bestSol$ **then**
      11. $bestSol \leftarrow S; \ stable \leftarrow 0$
   **else**
      12. **if** $stable = stableLimit$ **then**
          13. $S \leftarrow bestSol; \ stable \leftarrow 0$
      **else**
          14. $stable \leftarrow stable + 1$
   15. **if** $it \% restartFreq = 0$ **then**
      16. $S \leftarrow solution$ computed in step 1;
          17. $bestSol \leftarrow S$
      18. $it \leftarrow it + 1$;
Chapter 5. Agricultural Land Allocation Problem

Table 5.3: Comparing the proposed algorithms for $\text{PPos2}$ with respect to the quality of found solution and the computational time

<table>
<thead>
<tr>
<th>Tno</th>
<th>P2 CF</th>
<th>P2 IF</th>
<th>P2 CGIF</th>
<th>P2 CGLI</th>
<th>P2 LS</th>
<th>P2 LCI</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>t(s)</td>
<td>t(s)</td>
<td>t(s)</td>
<td>t(s)</td>
<td>t(s)</td>
<td>t(s)</td>
</tr>
<tr>
<td>0</td>
<td>0.22</td>
<td>54.19</td>
<td>8.35</td>
<td>34.15</td>
<td>16.16</td>
<td>34.15</td>
</tr>
<tr>
<td>1</td>
<td>0.14</td>
<td>34.15</td>
<td>16.16</td>
<td>34.15</td>
<td>8.35</td>
<td>34.15</td>
</tr>
<tr>
<td>2</td>
<td>2.49</td>
<td>39.07</td>
<td>3.34</td>
<td>39.07</td>
<td>0.19</td>
<td>39.07</td>
</tr>
<tr>
<td>3</td>
<td>39.07</td>
<td>34.19</td>
<td>39.07</td>
<td>39.07</td>
<td>39.07</td>
<td>39.07</td>
</tr>
<tr>
<td>4</td>
<td>4.34</td>
<td>39.07</td>
<td>39.07</td>
<td>39.07</td>
<td>39.07</td>
<td>39.07</td>
</tr>
<tr>
<td>5</td>
<td>0.35</td>
<td>34.15</td>
<td>34.15</td>
<td>34.15</td>
<td>34.15</td>
<td>34.15</td>
</tr>
<tr>
<td>6</td>
<td>0.52</td>
<td>36.19</td>
<td>36.19</td>
<td>36.19</td>
<td>36.19</td>
<td>36.19</td>
</tr>
<tr>
<td>7</td>
<td>0.52</td>
<td>36.19</td>
<td>36.19</td>
<td>36.19</td>
<td>36.19</td>
<td>36.19</td>
</tr>
<tr>
<td>8</td>
<td>0.16</td>
<td>34.15</td>
<td>34.15</td>
<td>34.15</td>
<td>34.15</td>
<td>34.15</td>
</tr>
<tr>
<td>9</td>
<td>0.16</td>
<td>34.15</td>
<td>34.15</td>
<td>34.15</td>
<td>34.15</td>
<td>34.15</td>
</tr>
<tr>
<td>10</td>
<td>0.16</td>
<td>34.15</td>
<td>34.15</td>
<td>34.15</td>
<td>34.15</td>
<td>34.15</td>
</tr>
<tr>
<td>11</td>
<td>0.16</td>
<td>34.15</td>
<td>34.15</td>
<td>34.15</td>
<td>34.15</td>
<td>34.15</td>
</tr>
<tr>
<td>12</td>
<td>0.33</td>
<td>34.15</td>
<td>34.15</td>
<td>34.15</td>
<td>34.15</td>
<td>34.15</td>
</tr>
<tr>
<td>13</td>
<td>0.33</td>
<td>34.15</td>
<td>34.15</td>
<td>34.15</td>
<td>34.15</td>
<td>34.15</td>
</tr>
<tr>
<td>14</td>
<td>0.33</td>
<td>34.15</td>
<td>34.15</td>
<td>34.15</td>
<td>34.15</td>
<td>34.15</td>
</tr>
<tr>
<td>15</td>
<td>0.33</td>
<td>34.15</td>
<td>34.15</td>
<td>34.15</td>
<td>34.15</td>
<td>34.15</td>
</tr>
<tr>
<td>16</td>
<td>0.33</td>
<td>34.15</td>
<td>34.15</td>
<td>34.15</td>
<td>34.15</td>
<td>34.15</td>
</tr>
<tr>
<td>17</td>
<td>0.33</td>
<td>34.15</td>
<td>34.15</td>
<td>34.15</td>
<td>34.15</td>
<td>34.15</td>
</tr>
<tr>
<td>18</td>
<td>0.33</td>
<td>34.15</td>
<td>34.15</td>
<td>34.15</td>
<td>34.15</td>
<td>34.15</td>
</tr>
<tr>
<td>19</td>
<td>0.33</td>
<td>34.15</td>
<td>34.15</td>
<td>34.15</td>
<td>34.15</td>
<td>34.15</td>
</tr>
<tr>
<td>20</td>
<td>0.33</td>
<td>34.15</td>
<td>34.15</td>
<td>34.15</td>
<td>34.15</td>
<td>34.15</td>
</tr>
<tr>
<td>21</td>
<td>0.33</td>
<td>34.15</td>
<td>34.15</td>
<td>34.15</td>
<td>34.15</td>
<td>34.15</td>
</tr>
<tr>
<td>22</td>
<td>0.33</td>
<td>34.15</td>
<td>34.15</td>
<td>34.15</td>
<td>34.15</td>
<td>34.15</td>
</tr>
<tr>
<td>23</td>
<td>0.33</td>
<td>34.15</td>
<td>34.15</td>
<td>34.15</td>
<td>34.15</td>
<td>34.15</td>
</tr>
<tr>
<td>24</td>
<td>0.33</td>
<td>34.15</td>
<td>34.15</td>
<td>34.15</td>
<td>34.15</td>
<td>34.15</td>
</tr>
<tr>
<td>25</td>
<td>0.33</td>
<td>34.15</td>
<td>34.15</td>
<td>34.15</td>
<td>34.15</td>
<td>34.15</td>
</tr>
<tr>
<td>26</td>
<td>0.33</td>
<td>34.15</td>
<td>34.15</td>
<td>34.15</td>
<td>34.15</td>
<td>34.15</td>
</tr>
<tr>
<td>27</td>
<td>0.33</td>
<td>34.15</td>
<td>34.15</td>
<td>34.15</td>
<td>34.15</td>
<td>34.15</td>
</tr>
<tr>
<td>28</td>
<td>0.33</td>
<td>34.15</td>
<td>34.15</td>
<td>34.15</td>
<td>34.15</td>
<td>34.15</td>
</tr>
<tr>
<td>29</td>
<td>0.33</td>
<td>34.15</td>
<td>34.15</td>
<td>34.15</td>
<td>34.15</td>
<td>34.15</td>
</tr>
<tr>
<td>30</td>
<td>0.33</td>
<td>34.15</td>
<td>34.15</td>
<td>34.15</td>
<td>34.15</td>
<td>34.15</td>
</tr>
</tbody>
</table>

Table 5.4: Comparison between the existent solution conducted by the government’s approach and the solutions computed by our incomplete algorithms

<table>
<thead>
<tr>
<th>Criteria</th>
<th>government solution</th>
<th>20 solutions of our algorithms</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\text{Max}(c_1)$</td>
<td>88</td>
<td>5</td>
</tr>
<tr>
<td>$\text{Avg}(c_1)$</td>
<td>39.51</td>
<td>2.06</td>
</tr>
<tr>
<td>$\text{Var}(c_1)$</td>
<td>29.37</td>
<td>1.22</td>
</tr>
<tr>
<td>$\text{Max}(c_2)$</td>
<td>98</td>
<td>6</td>
</tr>
<tr>
<td>$\text{Avg}(c_2)$</td>
<td>32.93</td>
<td>1.95</td>
</tr>
<tr>
<td>$\text{Var}(c_2)$</td>
<td>25.31</td>
<td>1.29</td>
</tr>
<tr>
<td>$\text{Max}(c_3)$</td>
<td>89</td>
<td>1</td>
</tr>
<tr>
<td>$\text{Avg}(c_3)$</td>
<td>13.96</td>
<td>0.28</td>
</tr>
<tr>
<td>$\text{Var}(c_3)$</td>
<td>17.73</td>
<td>0.45</td>
</tr>
<tr>
<td>$\text{Max}$</td>
<td>89</td>
<td>1</td>
</tr>
<tr>
<td>$\text{Avg}$</td>
<td>16.92</td>
<td>0.04</td>
</tr>
<tr>
<td>$\text{Var}$</td>
<td>10</td>
<td>0</td>
</tr>
</tbody>
</table>

Table 5.5: Comparison between the existent solution conducted by the government’s approach and the solutions computed by our complete algorithms

18000 | 393.78 | 69.15 | 20707 | 129.23 | 103.23 | 69.15 | 18.71 |
| 25000 | 810.72 | 7491 | 20126 | 65.07 | 63.93 | 63.93 | 83.33 |
| 30000 | 1899.63 | 6086 | 19553 | 2031.73 | 19598 | 122.92 | 580.61 |
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 routing 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.
Chapter 6. Conclusion


1996. GLOBECOM '96. 'Communications: The Key to Global Prosperity, volume 2, pages 1069–1073, nov 1996. [2][31]


the third annual ACM-SIAM symposium on Discrete algorithms, pages 307–316, 1992. \[45\], \[48\]


[75] C. E. Miller, A. W. Tucker, and R. A. Zemlin. Integer pro-


[82] M. W. Padberg and M. R. Rao. Odd minimum cut-sets and b-


