NETWORK FLOWS WITH CONFLICT CONSTRAINTS

by

Zeynep Şuvak

B.S., Industrial Engineering, Boğaziçi University, 2010

M.S., Industrial Engineering, Boğaziçi University, 2013

Submitted to the Institute for Graduate Studies in
Science and Engineering in partial fulfillment of
the requirements for the degree of
Doctor of Philosophy

Graduate Program in Industrial Engineering
Boğaziçi University
2019

NETWORK FLOWS WITH CONFLICT CONSTRAINTS

APPROVED BY:

Prof. İ. Kuban Altınel (Thesis Supervisor)	....
Prof. Necati Aras (Thesis Co-supervisor)	....
Prof. Ümit Bilge	....
Prof. Temel Öncan	....
Prof. Z. Caner Taşkın	....
Assist. Prof. Hande Küçükaydın	....

DATE OF APPROVAL: 21.06.2019

ACKNOWLEDGEMENTS

Firstly, I would like to express my sincere gratitude to my advisor Prof İ. Kuban Altınel and co-advisor Prof Necati Aras for their invaluable guidance and motivation throughout my graduate study. Their inspiring suggestions have always illuminated my way and this work would not be completed without their significant contributions. Beyond working together, I feel really fortunate to know them because I see them as two role models with immense academic knowledge and supportive attitude as mentors.

I am very grateful to Prof. Temel Öncan and Prof Z. Caner Taşkın for the contributions they made in the progress of my research and the hard questions which incented me to widen my research from various perspectives. Besides, I would like to thank to the rest of my thesis committee: Prof. Ümit Bilge and Assist. Prof. Hande Küçükaydın for their insightful comments and encouragement.

I also gratefully acknowledge the partial support of TÜBİTAK (The Scientific and Technological Research Council of Turkey) under the grant no: 213M441 for the completion of this thesis. Moreover, I would like to acknowledge Düzce University for the concerned and understanding attitude all this time and the support of ÖYP (Teaching Staff Training Program) conducted by YÖK (The Council of Higher Education).

I am also very thankful to my friends and work fellows for the great times we had together during my graduate study. No words can be found to express my feelings for my family. My mother and my father have always been by my side whenever I need their affection and support. I feel as the luckiest person to have my husband, Vedat. You have always been considerate and encouraging in all circumstances. My lovely daughter Mehtap, I hope you will always be proud of your mother because this is the strongest motivation I have for the times we are apart. Your presence, as a whole, made it possible, thank you.

ABSTRACT

NETWORK FLOWS WITH CONFLICT CONSTRAINTS

It is a commonly used approach to model real life problems as network flow problems and they appear in a wide range of areas including telecommunication, wireless networks, transportation, healthcare and scheduling. Our focus in this thesis is on an extension of network flow problems with conflict constraints that prevent the simultaneous usage of some arc pairs to send flow. We particularly concentrate on four of them: the minimum cost noncrossing flow problem on layered networks, the minimum cost flow problem with conflicts, the maximum flow problem with conflicts and the assignment problem with conflicts. The minimum cost noncrossing flow problem on layered networks, which emerges from the quay crane scheduling problem in container terminals, is proven to be NP-hard. Further complexity results including the strong NP-hardness and the non-existence of polynomial time approximation algorithm for the the minimum cost flow problem with conflicts on general networks are also provided. Moreover, polynomially solvable special cases for the minimum cost noncrossing flow problem on layered networks and the assignment problem with conflicts, which is known to be NP-hard, are explored. Similarly, the conditions which limit the number of feasible solutions with a polynomial number are indicated for the minimum cost flow problem with conflicts and the maximum flow problem with conflicts taking advantage of the conflict graph representation. Alternative mathematical representations for these problems are developed. Pre-optimization procedures to reduce the problem size and to find an initial feasible solution are defined. Exact solution algorithms including a branch-and-bound algorithm enriched with the subroutines that exploit the special structure of the considered problem, an improved Russian doll search algorithm and a Benders decomposition with strengthened cuts are proposed. The methods are tested on a large set of test instances and they are shown to be superior than solving the underlying mathematical formulations with a commercial optimization solver.

ÖZET

ÇATIŞMA KISITLI AĞ AKIŞLARI

Telekomünikasyon, kablosuz ağlar, taşımacılık, tıp ve sağlık hizmetleri, çizelgeleme gibi alanlarda karşımıza çıkan birçok problemi ağ akış problemleri olarak modellemek yaygın bir yaklaşımdır. Bu tez bağlamında odaklanılan ise, ağ akış problemlerinin belirli okların aynı anda akış taşımasını engelleyen çatışma kısıtlı uzantısıdır. Ele alınan problemlerin bazıları daha önceden yazında var olan, bazıları da ilk defa bu tezde çalışılmış katmanlı ağlarda en küçük maliyetli kesişmeyen akış problemi, çatışma kısıtlı en küçük maliyetli akış problemi, çatışma kısıtlı en büyük akış problemi ve çatışma kısıtlı atama problemidir. Konteyner limanlarında kıyı vinci çizelgeleme problemini modellemek için ortaya çıkan katmanlı ağlarda en küçük maliyetli kesişmeyen akış probleminin NP-zor olduğu kanıtlanmıştır. Çatışma kısıtlı en küçük maliyetli akış probleminin güçlü NP-zorluğu ve polinom zamanlı yakınlaştırma algoritmasının bulunmazlığı gibi karmaşıklık çözümleme sonuçlarına da ulaşılmıştır. Ayrıca, katmanlı ağlarda en küçük maliyetli kesişmeyen akış problemi ve çatışma kısıtlı atama problemi için polinom zamanda çözülebilen özel durumlar açıklanmıştır. Benzer şekilde, çatışma kısıtlı en küçük maliyetli akış problemi ve çatışma kısıtlı en büyük akış problemi için olurlu çözüm sayısını polinom bir sayıyla sınırlayan durumlar çatışma çizgesinden faydalanılarak işaret edilmiştir. Tüm problemler için çeşitli matematiksel gösterimler elde edilmiştir. Hızlı bir biçimde problem boyutunu küçülten ve olurlu bir çözüm bulan eniyileme öncesi işlemler önerilmiştir. Eldeki problemin özel yapısını kullanan etkili alt yordamlarla zenginleştirilmiş dal-sınır algoritması, yenilikçi yaklaşımlarla iyileştirilmiş matruşka araması ve güçlü kesiler kullanan Benders ayrıştırması gibi kesin çözüm yöntemleri geliştirlmiştir. Algoritmalar geniş bir örnek problem kümesi üzerinde denenmiş ve başarımlarının matematiksel gösterimlerini bir ticari eniyileme yazılımı ile çözmekten daha iyi olduğu sonucuna varılmıştır.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS

$A(N)$	Arc set of network N
$A_{l(l+1)}(N)$	The set of arcs (i,j) having i in layer l and j in layer $l+1$
APC_K	The clique formulation of APC
APC_S	The strong formulation of APC
APC_W	The weak formulation of APC
$\mathrm{APC}^{(t)}$	The subproblem of APC at node t of the BB tree
$\mathrm{AP}^{(t)}$	The AP relaxation at node t of the BB tree
b_i	Supply/demand of vertex i
C	The conflict graph
c_{ij}	Cost of sending one unit of flow along arc (i,j)
$\overline{c}_{ij}$	The reduced cost of arc (i,j) or edge $\{i,j\}$
C_l	The candidate set at level l of RDS
$d_G(v)$	Degree of vertex v of graph G
$E(G)$	Edge set of graph G
F_l	The free vertex set at level l of RDS
f_{ij}	Flow amount on arc (i,j)
G	Graph
$\overline{G}$	Complement of graph G
$G[W]$	The subgraph of G induced by vertex set $W \subset V(G)$
$\mathcal{I}$	The partial independent set kept during RDS
$\mathcal{K}$	The family of maximal cliques of the conflict graph C
l_{ij}	Lower bound on the flow along arc (i,j)
$\mathcal{MC}$	A minimum capacity cut
MCFPC_K	The clique formulation of MCFPC
MCFPC_S	The strong formulation of MCFPC
MCFPC_W	The weak formulation of MCFPC
$\mathrm{MCFPC}^{(t)}$	The subproblem of MCFPC at node t of the BB tree
$\mathrm{MCFP}^{(t)}$	The MCFP relaxation at node t of the BB tree

MCNFP$_K$	The clique formulation of MCNFP
MCNFP$_S$	The strong formulation of MCNFP
MCNFP$_W$	The weak formulation of MCNFP
MFPC$_K$	The clique formulation of MFPC
MFPC$_S$	The strong formulation of MFPC
MFPC$_W$	The weak formulation of MFPC
MFPC$^{(t)}$	The subproblem of MFPC at node t of the BB tree
MFP$^{(t)}$	The MFP relaxation at node t of the BB tree
m	Number of arcs in network N
n	Number of vertices in network N
n_l	The cardinality of vertex set in layer l of a layered network
N	Network
$N_G(v)$	The set of adjacent vertices of vertex $v \in V(G)$
$\overline{N}_G(v)$	The subset $V(G)\backslash(N_G(v) \cup \{v\})$
pn_{ij}	The penalty of deleting an arc (i,j) or edge $\{i,j\}$
S_{ij}	The arcs cross with arc (i,j)
S_l	A maximal stable set which is a subset of F_l
s	Source vertex
t	Sink vertex
U	The summation of flow upper bounds over $(i,j) \in A(N)$
u_{ij}	Flow upper bound on arc (i,j)
$V(G)$	Vertex set of graph G
$V(N)$	Vertex set of network N
$V^+(N)$	Subset of transshipment supply vertices
$V^-(N)$	Subset of transshipment demand vertices
$V^\pm(N)$	Subset of pure transshipment vertices
$V_l(N)$	The set of vertices in layer l of a layered network
v	Value of flow sent form the source vertex to the sink vertex
$w(W)$	The sum of the weights of the vertices in subset W
x_{ij}	Binary variable associated to arc (i,j) or edge $\{i,j\}$ which is set to one if it is selected, and zero otherwise

$\overline{z}$	A global upper bound for the optimal objective value
$\underline{z}$	A global lower bound for the optimal objective value
$\overline{z}^{(t)}$	The upper bound obtained from the relaxed problem at node t of the BB tree
$\underline{z}^{(t)}$	The lower bound obtained from the relaxed problem at node t of the BB tree
$z(.)$	The objective value of a feasible solution
$\delta_C(i,j)$	The conflict list of arc (i,j) or edge $\{i,j\}$
$\delta_G(v)$	The subset of edges incident with vertex $v \in V(G)$
μ_{ij}	The dual variable assigned to flow capacity constraint of arc (i,j)
π_i	The dual variable assigned to flow balance constraint of vertex i

LIST OF ACRONYMS/ABBREVIATIONS

AP	Assignment Problem
APC	Assignment Problem with Conflicts
BAP	Berth Allocation Problem
BB	Branch-and-Bound
BD	Benders Decomposition
BFS	Breadth First Search
BIP	Binary Integer Programming Problem
BPPC	Bin Packing Problem with Conflicts
DFS	Depth First Search
DSP	Dual of the Subproblem of Benders Decomposition
KPC	Knapsack Problem with Conflicts
LLB	Largest Lower Bound
LP	Linear Programming Problem
MCFP	Minimum Cost Flow Problem
MCFPC	Minimum Cost Flow Problem with Conflicts
MCNFP	Minimum Cost Noncrossing Flow Problem on Layered Networks
MCNFP-RC	Minimum Cost Noncrossing Flow Problem on Layered Networks with Restricted Cost
MCP	Minimum Capacity Cut Problem
MCPMPC	Minimum Cost Perfect Matching Problem with Conflicts
MFP	Maximum Flow Problem
MFPC	Maximum Flow Problem with Conflicts
MILP	Mixed Integer Linear Programming Problem
MKP	Maximum Clique Problem
MMPC	Maximum Matching Problem with Conflicts
MP	Master Problem of Benders Decomposition
MSP	Maximum Stable Set Problem
MSTPC	Minimum Spanning Tree Problem with Conflicts

MWKP	Maximum Weight Clique Problem
MWSP	Maximum Weight Stable Set Problem
QCAP	Quay Crane Assignment Problem
QCSP	Quay Crane Scheduling Problem
RDS	Russian Doll Search
SLB	Smallest Lower Bound
SP	Subproblem of Benders Decomposition
SPP	Set Partitioning Problem
SPPC	Shortest Path Problem with Conflicts
TPC	Transportation Problem with Conflicts

1. INTRODUCTION

Many systems or structures encountered in daily life can be represented as networks of interconnecting objects. Power networks to distribute electricity, telecommunication/internet networks to ensure data stream, highway or rail networks to provide means of transportation, supply chain or distribution networks in manufacturing, social networks to model the acquaintance between people or biological networks to examine the interaction across species in terms of food-webs or disease spread are some examples.

Network flow problems involve transmitting goods, materials, data, messages, money, passengers, etc. across a network. Flow is the general name for all items transferred along the links between the nodes of the network because their properties are similar to those of water flowing through pipes. Although various cases in many disciplines can be modeled as network flow problems, some are self-evident which are called physical networks while some are less obvious. The problems of second type, as we observe in scheduling applications, usually do not seem to involve flows at first glance.

As examples of the classical network flow problems, the minimum cost flow problem and its variant, the fixed charge network flow problem in which a fixed set-up cost is incurred whenever a flow is sent along an arc, the maximum flow problem and the shortest path problem, which can be considered as complementary since the first one excludes arc capacities and the other ignores unit costs of sending flow, the transportation and the assignment problems, which are special cases of the minimum cost flow problem can be listed.

In addition to the original network flow problems, their extensions are also studied before. The classical network flow problems mentioned above except the fixed charge network flow problem can be solved in a time polynomially bounded in the size of problem parameters but the addition of complicating side constraints make these problems

NP-hard. Multi-commodity flow problems where the cost, demand and supply parameters are commodity dependent but the flow capacities are the common resources utilized by all commodities are the most known extensions [2, 3]. Other examples are the *time-constrained flow*, which is limited by not only arc capacities but also the time windows needed to traverse that arc [4], and the *budget-constrained flow* where the total cost cannot exceed a given budget [5]. Similarly, we can talk about *generalized flows* which are eroded to some extent at the visited vertices [6, 7] or *path-constrained flow* in which the flow is either split at most k paths or not split at all [8, 9]. In this thesis, we study another extension of network flow problems where some arcs are incompatible with each other. In other words, we analyze the case where at most one of the incompatible arcs is allowed to carry flow in a feasible solution.

1.1. Motivation

Network flows subject to conflict constraints preventing simultaneous usage of some arc pairs is first appeared in container terminals [10, 11]. A container terminal is a facility where the containerized cargo is loaded to vessels, unloaded from them, transferred to trucks or trains to be delivered to inland destinations and stored until the execution time of the next scheduled operation arrives. Due to the increased volume of containerized transshipment in recent years, the optimization problems arising within a container port are extensively studied in the operations research literature. Since investing to resources such as berths, i.e. quay spaces where vessels are loaded and unloaded, cranes, transportation vehicles and handling equipment is very costly; the efficient utilization of the available instruments gains importance to meet the growing demand. One of the most popular problems observed in the quayside of the terminal is the berth allocation problem (BAP) in which we allocate a berth segment and a service time interval to each arriving vessel. Containers are unloaded from vessels by quay cranes and placed onto trucks, while containers are loaded to vessels following the reverse process. Quay cranes can move along the quayside usually on a rail system. The quay crane assignment problem (QCAP) deals with assigning the number of quay cranes to berthed vessels and the quay crane scheduling problem (QCSP) determines particular cranes which serve them along a planning horizon. Each quay crane is

Figure 1.1. The Port of Colombo, Sri Lanka [1]

either assigned to a vessel or waits in the spaces between them. The position of cranes changes whenever a vessel arrival or departure occurs, they remain unchanged otherwise. Given the assigned berths and number of assigned cranes to each arriving vessel in the planning horizon, scheduling of specific quay cranes can be modeled as the minimum cost flow problem in which the aim is to minimize the total set-up and operation cost that stems from crane displacements. Since, quay cranes cannot cross over each other, a schedule that requires crossing crane movements are unacceptable. Therefore, the minimum cost noncrossing flow problem on layered networks (MCNFP) where the flow represents the crane movements across different service locations is born [12].

In the course of time, the physical meaning of crossing is abstracted to the notion of *conflict* and the general networks are considered instead of special layered structures. After gaining insights about the nature and the exact solution methods for the minimum cost flow problem with conflicts (MCFPC) [13], we also study the maximum flow problem with conflicts (MFPC) [14] and the assignment problem with conflicts (APC) [15], which can be considered as the special forms of MCFPC.

1.2. Organization and Contributions

As mentioned before, network flow problems arise in both physical networks where the flow of items are evident and in scheduling applications such as the quay crane scheduling problem (QCSP) described earlier. Depending on the context of the problem, extra restrictions might be imposed in addition to the existing flow conservation constraints. When QCSP is modeled as a network flow problem, it is observed that the usage of some pairs of arcs, namely the crossing ones, leads to an infeasible schedule violating the physical constraint that a quay crane cannot cross over another. Although the concept of conflicting arcs appeared previously in different problems, network flow problems with conflicting arc pairs are not studied thoroughly. This thesis aims to fill this gap and provide a comprehensive analysis of network flow problems in the existence of incompatible, conflicting arcs. Some of the studied problems in this thesis are newly defined while the others are introduced previously.

The notations and definitions that seem necessary for a better understanding of the remaining part of the thesis are given in Chapter 2. The mathematical representations of the ordinary flow problems and the relation between these formulations are also presented. A formal definition of flow and an introductory section about conflict notion, which is the main concern of this study, can be found there. In Chapter 3, a broad review of the literature about the problems with conflicting variables, which we consider as the most related works to ours are given. The review is summarized under four main subtitles. The works examining the complexity and polynomially solvable special cases, and the ones proposing exact solution algorithms and heuristics are grouped separately. We give brief explanations about these theoretical and algorithmic approaches and we present the examples of real-life applications of the problems with conflict constraints, as well.

First of all, the minimum cost noncrossing flow problem on layered networks (MCNFP) which is not studied before is introduced. This problem is defined on a layered network in which the arcs exist between two consecutive layers only in forward direction. The flows that cross each other with respect to a special embedding of vertices within each layer are not allowed and the least cost distribution of the available supply is searched so that the demand requirements are satisfied subject to finite arc flow bounds. We first prove that this problem is NP-hard by reducing from the set partitioning problem (SPP) to an intermediate problem that is followed by another instance reduction from the introduced intermediate problem to MCNFP. The conditions which make the problem polynomially solvable are described in detail and a procedure for obtaining a noncrossing flow from a given crossing flow is found under these conditions. In addition to the mixed integer linear programming formulations (MILP), a preprocessing method for eliminating the nonpromising arcs is proposed. This method primarily relies on the special structure of the layered network.

Secondly, the minimum cost flow problem with conflicts (MCFPC) is considered. It is a generalized version of MCNFP where we work with arbitrary graphs instead of layered networks and conflicting arcs instead of crossing ones. The conflicting arcs do not necessarily represent physical incompatibility with respect to a particular embedding which is the case for MCNFP. Every arc of the network is defined with a list of conflicting arcs with itself. Some arcs might have an empty list. The reason behind a conflict might change depending on the context of the problem. Here, *conflict* is considered as an abstract notion. That is exactly the reason why the developed solution methods to solve MCFPC is valid for MCNFP. After showing that MCFPC is strongly NP-hard by reducing from the maximum flow problem with conflicts (MFPC), which is known to be strongly NP-hard [16], further complexity analysis is carried out regarding the existence of polynomial time approximation algorithms for a class of instances with certain property. MCFPC is formulated using alternative MILPs - weak, strong and clique versions - and modeled as a combinatorial optimization problem benefiting from the conflict graph representation. Pre-optimization procedures that enable us to find a quick feasible solution and reduce the problem size by deleting the arcs causing infeasibility when they have positive flow on it, are described. Besides, two exact solution algorithms, branch-and-bound (BB) and Russian doll search (RDS) algorithms, are developed to solve middle sized instances. Penalty calculation techniques exploiting the spanning tree structures corresponding basic feasible solutions and pegging operation are introduced for strong branching strategies. As opposed to the ordinary

RDS algorithm, dynamic candidate set approach is proposed in order to make it faster. The success of the algorithms stems from the use of sophisticated subroutines as well as the polynomial time algorithms to solve the emerging subproblems. In chapter 10, the results of the computational experiments are analyzed in detail to understand the effect of each pre-optimization procedure.

Next, we focus on MFPC, a particular version of MCFPC where there is no cost associated with flows on arcs, but another auxiliary variable which is the value of flow from a source node to a sink node is introduced. The objective is to maximize this value without considering costs. This problem is proven to be strongly NP-hard in [16]. The previously proposed BB and RDS algorithms to solve MCFPC are modified and similar algorithms are developed to solve MFPC. Moreover, MFPC has an appropriate structure for obtaining a successful Benders decomposition implementation. First, the dual problem of the Benders subproblem has always a finite optimum and this prevents addition of Benders feasibility cuts. Working with only optimality cuts shortens the expected solution times. Furthermore, we introduce strengthened Benders cuts and connectivity cuts which prevent producing disconnected subnetworks. Also, we add valid inequalities and try Benders decomposition with three different MILP formulations.

Finally, the assignment problem with conflicts (APC) is studied. This problem is a special version of MCFPC defined on a graph with equally sized bipartition subsets of the vertices and the flow is bounded by one unit. It is also shown to be NP-hard [17]. We describe a polynomially solvable special case for APC, and propose a BB algorithm with five different branching rules, and an RDS algorithm for its solution. Since this is a particular version of MCFPC, penalties, pegged variables and the reduced costs of arcs with respect to an optimal solution of a relaxed subproblem are calculated from the optimal solution of the ordinary assignment problem. A local search based heuristic is employed to start with a finite upper bound and a probing scheme to eliminate arcs is presented.

In Chapter 4, definitions and all mathematical formulations are given for the problems. Chapter 5 includes the complexity analysis of MCNFP and MCFPC, and describes the polynomially solvable special cases for MCNFP and APC. The preprocessing methods, the heuristics which find initial feasible solutions for the considered problems are provided in Chapter 6. In Chapter 7, the Benders decomposition algorithm developed for MFPC is given. Chapters 8 and 9 include BB and RDS algorithms, respectively, which are proposed to solve MCFPC, MFPC and APC. The results of the carried out experiments are summarized in Chapter 10. Finally, the concluding remarks are provided in Chapter 11.

2. BASIC CONCEPTS AND NOTATION

We begin by introducing the basic terminology and the conventional notation used in graph and network flow theory and continue with the definitions which are essential for full comprehension of this thesis. One can refer to [6, 18, 19] for further information.

2.1. Graphs

We let $G = (V(G), E(G))$ denote a graph with vertex set $V(G)$ and edge set $E(G)$. We assume that all graphs in this work are finite and simple (i.e. no loop, no parallel edges). The *complement* of a graph G is the graph $\overline{G} = (V(\overline{G}), E(\overline{G}))$, where $V(\overline{G}) = V(G)$ and $E(\overline{G}) = \{\{u, v\} \notin E(G) : u, v \in V(G),\ u \neq v\}$. For $W \subseteq V(G)$, the set $\delta_G(W) \subseteq E(G)$ and $\gamma_G(W) \subseteq E(G)$ are respectively the sets of edges with one endpoint in W and both endpoints in W respectively. Hence, $\delta_G(\{v\})$ for $v \in V(G)$, which we simply denote as $\delta_G(v)$, is the subset of edges incident with vertex $v \in V(G)$. Clearly, the *degree* of vertex v of graph G is $d_G(v) = |\delta_G(v)|$. For $W \subseteq V(G)$, the subgraph of G induced by W is $G[W] = (W, \{W \times W\} \cap E(G))$ (i.e. $G[W]$ has W as its vertex set and two vertices are adjacent in $G[W]$ if and only if they are adjacent in G). The set of all neighbors of a vertex $v \in V(G)$, excluding v itself, is denoted as $N_G(v)$, and its set of non-neighbors (i.e. the subset $V(G) \backslash (N_G(v) \cup \{v\})$) is denoted as $\overline{N}_G(v)$. The *edge density* of a graph is the number of edges divided by the maximum possible number of edges.

A *stable set* (*vertex packing*, *independent set*) of a graph is any subset $S \subseteq V(G)$ such that all vertices in S are mutually nonadjacent. A stable set is maximal if it is not a proper subset of any other stable set. The *maximum stable set problem* (MSP) is to find a stable set of maximum cardinality. The maximum value is the stability number of G denoted as $\alpha(G)$. We may assign weights to the vertices of a graph G; for our purpose the weights are assumed to be nonnegative rational numbers. For every $W \subseteq V(G)$, the weight of $w(W)$ is the sum of the weights of its vertices, $w(\emptyset) = 0$.

The *maximum weight stable set problem* (MWSP) is to find a stable set S of G with weight $w(S)$ maximum among all stable sets of G. The maximum value is denoted as $\alpha_w(G)$. Both MSP and MWSP are NP-Complete problems [20].

A *clique* $K = (V(K), E(K))$ is a complete subgraph of G. A clique is maximal if it is not a proper subset of any other clique. Cliques of G are stable sets of $\overline{G}$. So, the problems MSP and MWSP on G are equivalent to the *maximum clique problem* (MKP) and *maximum weight clique problem* (MWKP) on $\overline{G}$ and vice versa. The size of a maximum clique and total weight of a maximum weight clique in G are denoted as $\omega(G)$ and $\omega_w(G)$. A *clique cover* of a graph G is a collection of cliques K_j $j = 1, 2, \ldots, k$ such that $V(G) = \cup_{j=1}^{k} V(K_j)$. The smallest value of k for G is denoted as $\phi(G)$. Clique covers of G provide an upper bound on its stability number, i. e. $\alpha(G) \leq \phi(G) \leq k$. A *split* graph is a graph in which the vertices can be split into a clique and a stable set.

A vertex subset $S \subseteq V(G)$ is a *dominating set* of G if $S \cup N_G(S) = V(G)$, where $N_G(S)$ consists of the neighbors of the vertices in S, i.e. $N_G(S) = (\cup_{v \in S} N_G(v)) \backslash S$. A dominating set S is minimal if no proper subset of S is a dominating set. As can be observed easily, in any graph G, every maximal stable set is a minimal dominating set; but the reverse is not always true: one can find a minimal dominating set which is not a maximal stable set. In Figure 2.1, the vertex set $\{4, 6\}$ is a minimal dominating set since the subset obtained by deleting neither 4 nor 6 gives a dominating set. It does not form a stable set, clearly.

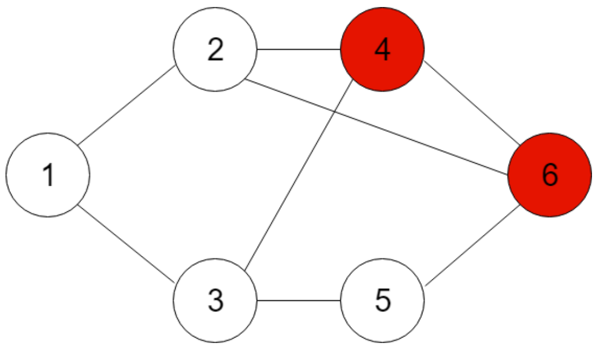


Figure 2.1. A minimal dominating set which is not a maximal stable set.

A *tree* is a connected graph that contains no cycle. If a tree contains exactly $n - 1$ arcs, then it is called *spanning tree* because the tree touches all vertices of the given graph G. Every two nodes of a tree are connected through a unique path.

A *matching* in G is a set of selected edges satisfying that every vertex in $V(G)$ is incident to at most one edge in this set. A matching is maximal if the addition of one more edge violates the matching property. If each vertex is incident to exactly one edge in a matching, then it is called a *perfect matching*. This means that every vertex is matched with another one.

A given graph G can be represented as an *intersection* graph of axis-parallel boxes where each vertex is represented with a box and the boxes intersect if and only if there exists an edge in G connecting the corresponding vertices. The *boxicity* of a graph G is the minimum dimension that allows representing G as an intersection graph of these boxes. In other words, G cannot be represented as the intersection of boxes with a lower dimension than its *boxicity*.

A subset $W \subseteq V(G)$ is said to be *s-plex* if the degree of each vertex in $G[W]$ is at least $|W| - s$. Notice that, the subset with s-plex property is equivalent to a clique, when $s = 1$. Another relevant property is *s-defective clique* defined on a simple graph G. For a given integer s, a subset $W \subseteq V(G)$ is called an *s-defective clique* if it contains at least $\binom{|W|}{2} - s$ edges. In other words, at most s edges are missing from a clique in an *s-defective clique*. An s-plex or s-defective clique is maximal if it is not strictly contained in any other s-plex or s-defective clique. On a given graph, the *maximum s-plex* and the *maximum s-defective clique* problems aim to find an s-plex and an s-defective clique with the maximum cardinality, respectively.

Given a subset $S \subseteq V(G)$, suppose that the induced subpgraph $G[S]$ has a graph property η. η is said to be *hereditary* if the deletion of any subset of vertices from S does not violate η on the remaining graph [21]. Stable set, clique, s-plex and s-defective properties can be mentioned as the examples of hereditary graph property. For example, any subgraph of a clique of a graph G is another clique of G.

A *chordal* graph is a simple graph whose cycles with length of at least four contains a chord. Since the maximum length of a cycle in a *chordal* graph is equal to three, it is also named as *triangulated* graph The *treewidth* of a graph G is one less than the size of the largest clique in the chordal graph containing G with the smallest clique number, i.e. the cardinality of the maximum clique. If the treewidth of a graph is bounded by some number k, then it is also called *partial k-tree.* An *interval* graph is an undirected graph constituted from intervals represented by vertices which are connected by an edge if and only if the intervals intersect. If it is possible to obtain an embedding of the vertices of a graph on a plane so that an edge intersect with another edge only at their endpoints, then this graph is a *planar* graph. A *cograph* is characterized by the property that two vertices of any induced subgraph of a *cograph* is connected by a path of length at most three. Given a graph G with vertex weights, G is called *threshold* graph if, for every edge, the sum of the weights of its endpoints is as large as some given threshold number. A *chain* graph is a type of graph where both directed arcs and undirected edges are allowed without no directed cycle.

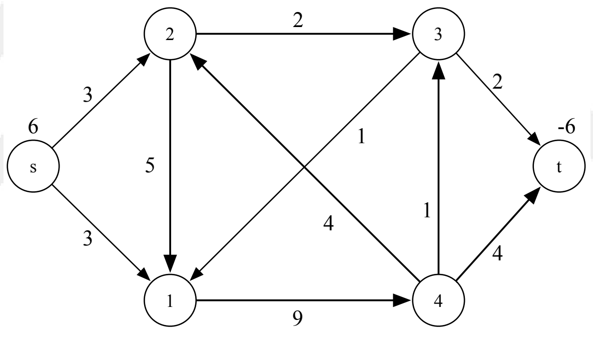
2.2. Networks and Flows

Networks are directed graphs on which flow problems are defined. This section is devoted to describe the necessary parameters to solve the considered network flow problems, give the useful notations and show the possible ways of representing flow on both general and layered networks.

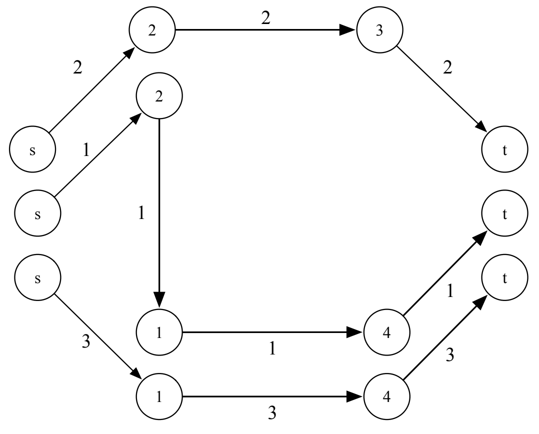
2.2.1. General Networks

Let $N = (V(N), A(N))$ be a network consisting of vertices $V(N)$ and arcs $A(N)$ with $|V(N)| = n$ and $|A(N)| = m$. Two of the vertices are different from the others: the source s has only outarcs and the sink t has only inarcs. The source and the sink either exist naturally in the original network or can be added as a supersource by aggregating the vertices with supplies and a supersink by combining the vertices with demands. Each arc $(i, j) \in A(N)$ has an associated unit flow cost c_{ij}, a capacity u_{ij} that denotes the maximum amount of flow allowed on the arc and a lower bound l_{ij}

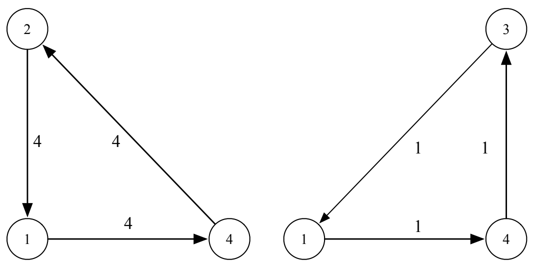
denoting the minimum amount of flow an arc must have. Each vertex $i \in V(N)$ has a number b_i representing its supply/demand. If $b_i > 0$, vertex i is a transshipment supply vertex, if $b_i < 0$ vertex i is a transshipment demand vertex with a demand of $-b_i$, and if $b_i = 0$, vertex i is a pure transshipment vertex. In other words, vertex set $V(N)$ can be expressed as $V(N) = \{s,t\} \cup V^+(N) \cup V^-(N) \cup V^\pm(N)$, where $V^+(N)$, $V^-(N)$ and $V^\pm(N)$ are respectively the subsets of transshipment supply, transshipment demand, and pure transshipment vertices. We assume that $\sum_{i=1}^{n} b_i = 0$, and $0 = l_{ij} \leq u_{ij}$ for all $(i,j) \in A(N)$ and satisfy sufficient conditions for the existence of a feasible flow [6]. The function $\mathbf{f} : A(N) \to \mathbb{R}$ is the flow function and associates the variable f_{ij} with arc (i,j). Note that a flow $\mathbf{f}$ is feasible if it satisfies the flow balance equalities at the vertices, lower bounds and capacity restrictions on the arcs.



(a) Arc flow



(b) Flow paths of an equivalent path and circuit flow



(c) Flow circuits of an equivalent path and circuit flow

Figure 2.2. Two ways to express a flow in a network

Traditionally, network flow problems can be formulated by either defining flows on arcs (i.e. arc flow), or directed paths and circuits (i.e. path and circuit flow). This is a consequence of the flow decomposition theorem [19], which eventually enables the (unique) representation of a path and circuit flow as nonnegative arc flow, and (not necessarily unique) representation of a nonnegative arc flow as a path and circuit flow. An example is provided in Figure 2.2. In Figure 2.2(a), we observe that a flow of 6 units is sent from s to t and the flow values carried by each arc are shown. Alternatively, this flow from s to t can be represented as the union of 3 different path flows and 2 circuit flows, as shown in Figure 2.2(b) and Figure 2.2(c). Although a different composition of directed paths and circuits can be obtained from the given arc flow, notice that there is always a flow path connecting a source vertex to a sink vertex.

2.2.2. Layered Networks

Layered graphs and networks provide effective modeling tools for the solution of some difficult combinatorial optimization problems, as recently detailed and classified in [22]. They are often encountered in container terminals, especially when the temporal allocation of the quay cranes to load/unload the berthed vessels according to their technical properties [11, 23], and when the scheduling of trains through the stations on the same railroad lines is targeted.

One of the studied problems in this thesis, namely the minimum cost noncrossing flow problem is defined on layered networks. A new parameter set that incorporates the layer information and other problem specific features are introduced. Let $N = (V(N), A(N))$ be a layered network consisting of L layers defined by the sets $V(N)$ of vertices and $A(N)$ of arcs. We define $V_l(N)$ as the set of vertices of layer l, and n_l its cardinality (i.e. $n_l = |V_l(N)|$, $l = 1, 2, \ldots, L$), and assume that $V_1(N) = \{s\}$, $V_L(N) = \{t\}$, $V(N) = \cup_{l=1}^{L} V_l(N)$, $n_1 = n_L = 1$ and $n = |V(N)| = \sum_{l=1}^{L} n_l$. s has only outarcs and t has only inarcs. Any arc $(i, j) \in A(N)$ of the network is forward (i.e. tail is closer to s in the number of arcs). There are neither backward arcs, nor arcs connecting two vertices at the same layer. If we let $A_{l(l+1)}(N)$ be the set of arcs (i, j) having $i \in V_l(N)$ and $j \in V_{l+1}(N)$, then $A(N) = \cup_{l=1}^{L-1} A_{l(l+1)}(N)$. We also assume that $A_{l(l+1)}(N)$

consists of all possible arcs with tail in $V_l(N)$ and heads in $V_{l+1}(N)$; i.e. $A_{l(l+1)}(N) = \{(i,j) : i \in V_l(N), j \in V_{l+1}(N)\}$. We consider a particular embedding of the network for vertex labeling: the vertices are located on the intersection points of a grid where the vertical lines represent the layers and numbered from 1 to n_l at layer l starting from the bottom to the top. The described layered network structure is illustrated in Figure 2.3. Recall that the vertex set $V(N)$ can be expressed as $V(N) = \{s,t\} \cup V^+(N) \cup$

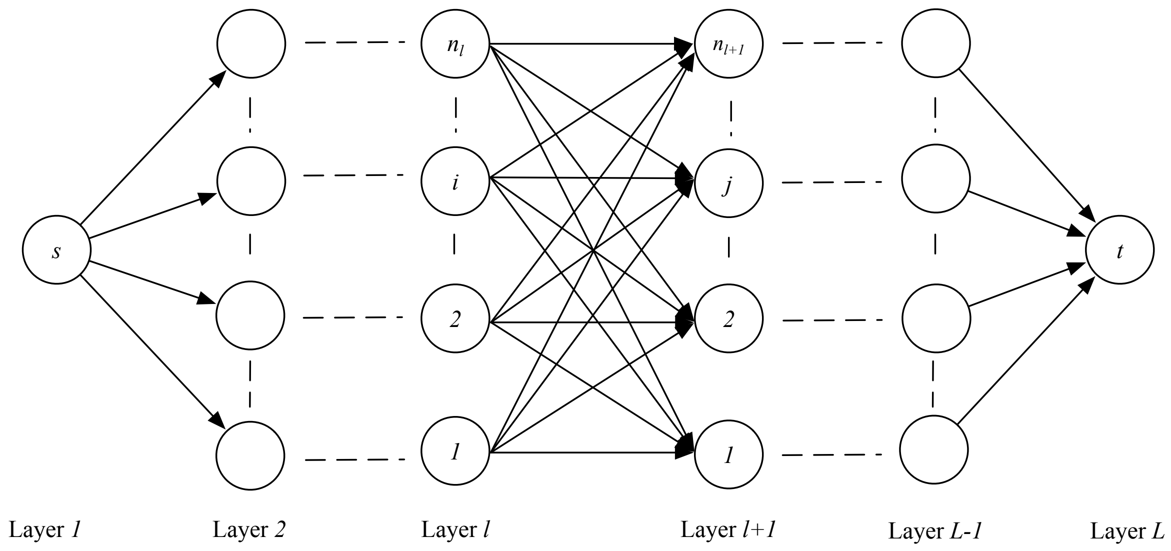
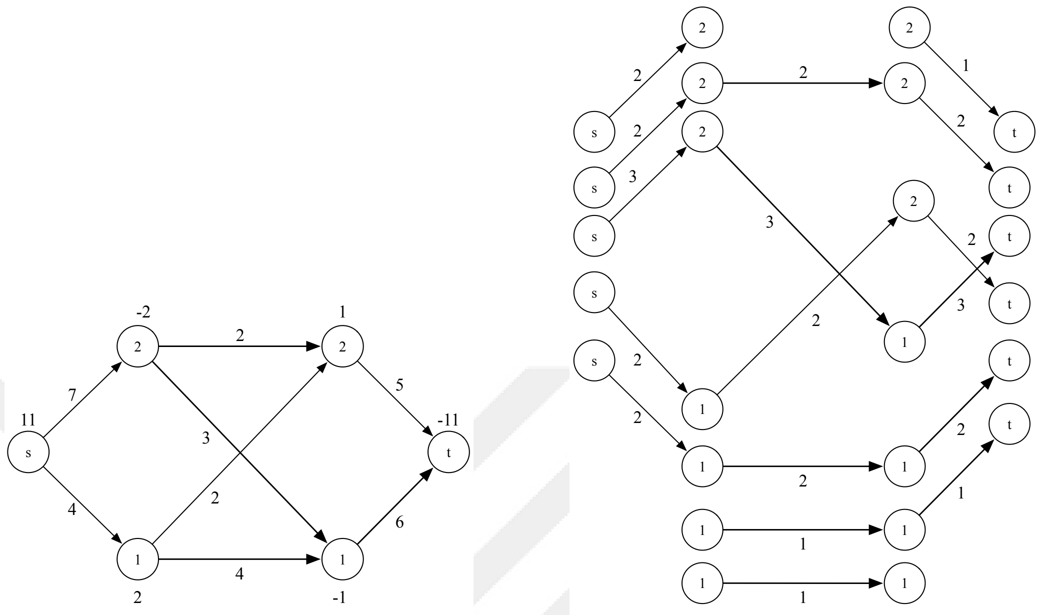


Figure 2.3. A layered network with L layers

$V^-(N) \cup V^{\pm}(N)$, where $V^+(N)$, $V^-(N)$ and $V^{\pm}(N)$ are respectively the subsets of transshipment supply, transshipment demand, and pure transshipment vertices at layer l. Similarly, for every level l, $V_l(N) = V_l^-(N) \cup V_l^+(N) \cup V_l^{\pm}(N)$, where $V_l^+(N)$, $V_l^-(N)$ and $V_l^{\pm}(N)$ denoting the subsets of transshipment supply, transshipment demand and pure transshipment vertices of layer l. Clearly, $V_1^-(N) = V_1^{\pm}(N) = \emptyset$, $V_1^+(N) = V_1(N) = \{s\}$, and $V_L^+(N) = V_L^{\pm}(N) = \emptyset$, $V_L^-(N) = V_L(N) = \{t\}$. We assume that $\sum_{l=1}^{L} \sum_{i \in V_l(N)} b_i = 0$, and $l_{ij} = 0 \leq u_{ij}$ for all $(i,j) \in A(N)$ and they satisfy sufficient conditions for the existence of a feasible flow [6].

Given a flow defined on the arcs of a layered network, flow decomposition does not result in any circuits because of the (directed) layered structure of the network. As backward arcs, i.e from layer $l+1$ to l, or arcs between the vertices of the same layer are not allowed, the layered networks are circuit-free. Therefore, the arc flow can be represented only as a path flow, and vice versa. Figure 2.4 provides an example for this particular situation.



(a) Arc flow (b) An equivalent path flow

Figure 2.4. Two ways to express a flow in a layered network

2.3. Classical Network Flow Problems

In this section, we present the original versions of the network flow problems whose extensions are studied throughout this thesis. Here, the goal is to show the existing mathematical models to represent them and make clear how they are related.

2.3.1. Minimum Cost Flow Problem

The minimum cost flow problem (MCFP) is well-known and has widespread applications. It emerges as a relaxed subproblem in solving many difficult combinatorial optimization problems [6]. Due to its special structure it can be solved efficiently by various solution algorithms that have been developed ever since the publication of Ford and Fulkerson's seminal monograph [19]. MCFP formulation is the most general representation of all network flow problems. The problem is to determine the shipment of goods through a network by transferring the available supplies from transshipment supply vertices to transshipment demand vertices in order to satisfy their demands. This shipment must have the minimum cost among all alternatives and must obey the flow bounds on arcs it traverses. The mathematical representation of MCFP is shown as follows:

$$\begin{aligned}
\text{MCFP:}\ \min \quad & \sum_{(i,j)\in A(N)} c_{ij} f_{ij} && && (2.1)\\
\text{s.t.} \quad & \sum_{(i,j)\in A(N)} f_{ij} - \sum_{(j,i)\in A(N)} f_{ji} = b_i && i \in V(N) && (2.2)\\
& l_{ij} \le f_{ij} \le u_{ij} && (i,j) \in A(N). && (2.3)
\end{aligned}$$

The objective function (2.1) minimizes the total cost of sending flow through the network. The constraints (2.2) ensures that the difference between the total outflow leaving a vertex and the total inflow entering it equals to the supply/demand, i.e. b_i value, of that vertex and they are referred as *flow balance constraints*. A feasible flow must also guarantee that each flow amount on an arc is between flow bounds associated with that arc and this condition is shown with constraints (2.3). Since we assume $l_{ij} = 0$ for our problems, they are called *flow capacity constraints*. We assume that all the cost, capacity and supply/demand value parameters are integral and this assumption is referred as *integrality assumption*. Under integrality assumption, MCFP formulation can be solved by any linear programming (LP) solver and an integer optimal solution is returned although $f_{ij}, (i,j) \in A(N)$ are defined as continuous variables [6].

2.3.2. Maximum Flow Problem

The classical maximum flow problem (MFP) is first formulated by T. E. Harris and F. S. Ross in order to find the minimum interdiction of the Soviet railway system in 1955 as indicated in the review paper [24]. A year later, Ford and Fulkerson demonstrated the well-known Max Flow-Min Cut theorem [25], and since then many polynomial-time algorithms are devised to solve MFP.

MFP is a special version of MCFP where all flow unit costs are zero. Moreover, the supply/demand of vertices except for the source, s, and the sink, t, equal to zero and the supply of the source and the demand of the sink are both introduced by the same auxiliary variable. This auxiliary variable, v, i.e. the value of the flow that can be sent from the source to the sink, is maximized in the following formulation of MFP:

$$\text{MFP: max} \quad v \tag{2.4}$$

$$\text{s.t.} \quad \sum_{(i,j)\in A(N)} f_{ij} - \sum_{(j,i)\in A(N)} f_{ji} = \begin{cases} v & i = s \\ 0 & i \in V(N)\backslash\{s,t\} \\ -v & i = t \end{cases} \tag{2.5}$$

$$0 \le f_{ij} \le u_{ij} \qquad (i,j) \in A(N). \tag{2.6}$$

MFP is modeled as a linear programming problem where all variables are allowed to take fractional values. However, the optimal solution of not only MFP but also its dual, i.e. the minimum capacity cut problem, is integer as long as the integrality assumption holds due to the total dual integrality property of MFP [26].

2.3.3. Minimum Cut Problem

Before introducing the dual problem of MFP, that is the minimum capacity cut problem or the minimum cut problem (MCP), we had better give the necessary definitions. Given a partition of the vertex set $V(N)$ into V^1 and V^2 where $V^2 = V(N)\setminus V^1$, the set of arcs that have one endpoint in V^1 and the other endpoint in V^2 is called a *cut* and represented by the notation $[V^1, V^2]$. In the context of our problem MFP, we are interested in an *s-t cut* which is defined with respect to the source s and the sink t, and is a cut satisfying the property that $s \in V^1$ and $t \in V^2$.

The capacity of a cut $[V^1, V^2]$ is defined as the total flow capacities of forward arcs, i.e. from V^1 to V^2. That is,

$$cap(V^1, V^2) = \sum_{i\in V^1, j\in V^2} u_{ij}. \tag{2.7}$$

For the sample network given in Figure 2.5, V^1 consists of vertices 1, 2 and 3 while V^2 includes the others. The arcs belonging to cut $[V^1, V^2]$ are (2,4), (2,6), (3,5), (4,3) and the capacity of this cut is equal to $u_{24} + u_{26} + u_{35}$. The *s*-*t* *cut* with minimum capacity

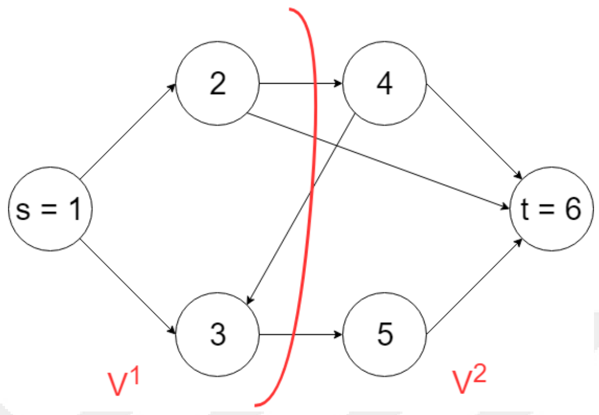


Figure 2.5. An *s*-*t* cut.

among all *s*-*t* *cuts* is called the minimum (capacity) cut and the Max Flow-Min Cut theorem states that the maximum flow that can be sent from s to t is equal to the capacity of minimum *s*-*t* cut [25]. Therefore, If we assign dual variables $\boldsymbol{\pi}$ to constraint set (2.5), and $\boldsymbol{\mu}$ to constraint set (2.6), the minimum cut problem formulation becomes

$$\text{MCP: } \min \sum_{(i,j)\in A(N)} \mu_{ij} u_{ij} \tag{2.8}$$

$$\text{s.t. } \pi_i - \pi_j + \mu_{ij} \geq 0 \qquad (i,j) \in A(N) \tag{2.9}$$

$$\pi_t - \pi_s = 1 \tag{2.10}$$

$$\pi_i \text{ unr }, \mu_{ij} \geq 0 \qquad i \in V(N), (i,j) \in A(N). \tag{2.11}$$

This representation always yields an integer optimal solution under the integrality assumption of the capacities u_{ij} although it is formulated as an LP problem. In fact, the arcs in the minimum cut, i.e. the arcs directed from the connected component containing s to the one with t have μ^*_{ij} values equal to one and the remaining ones have zero values. In other words, the arcs with optimal dual multipliers having value one form the minimum cut.

2.3.4. Assignment Problem

The assignment problem (AP) is well-known and has widespread applications in the areas of personnel scheduling, task assignment, job shop loading, facility location and workforce planning. It is also faced as a relaxed subproblem in solving many difficult combinatorial optimization problems. Due to its special structure, it can be solved efficiently and many polynomial time solution algorithms have been developed since the publication of the Hungarian algorithm [6, 27].

AP is defined on a bipartite graph G consisting of equally sized vertex sets $V_1(G)$ and $V_2(G)$, i.e. $V(G) = V_1(G) \cup V_2(G)$ and $|V_1(G)| = |V_2(G)|$. All the edges of graph G, denoted by $E(G)$ have one endpoint in $V_1(G)$ and the other in $V_2(G)$ as implied by the definition of bipartite graph. Besides, we assume a *complete bipartite* graph where $E(G)$ contains all the possible edges between $V_1(G)$ and $V_2(G)$ and there exists a cost c_{ij} associated with every edge $\{i, j\} \in E(G)$. In this problem, we wish to match the vertices of $V_1(G)$ and $V_2(G)$ at minimum cost where the most popular examples are to assign workers to jobs or jobs to machines.

AP can also be seen as a special type of MCFP on a network N consisting of four layers. The subsets $V_1(G)$ and $V_2(G)$ form two layers of N where the arc tails are in $V_1(G)$ and heads of arcs are in $V_2(G)$. $b_i = 1$ for all $i \in V_1(G)$ and $b_i = -1$ for all $i \in V_2(G)$. We also add a source, s, together with arcs $(s, i) \in A(N)$, $i \in V_1(G)$ and a sink, t, with arcs $(i, t) \in A(N)$, $i \in V_2(G)$. Addition of these nodes with $b_s = -b_t = |V_1(G)| = |V_2(G)|$ increments the number of layers by two. Moreover, $u_{ij} = 1\ \forall (i, j) \in A(N)$. If we solve MCFP on this network, we obtain the assignment with minimum cost by taking the arcs carrying unit flow from $V_1(G)$ to $V_2(G)$.

Although it is possible to transform AP into an MCFP instance, we prefer to continue with the original undirected graph representation where each feasible solution is a matching. We introduce binary decision variable x_{ij} for edge $\{i, j\} \in E(G)$, which is set to 1 if and only if edge $\{i, j\}$ is in the matching. Then, it can be formulated as a binary integer programming problem (BIP):

$$\text{AP: } \min \sum_{\{i,j\} \in E(G)} c_{ij} x_{ij} \tag{2.12}$$

$$\text{s.t.} \sum_{\{i,j\} \in E(G)} x_{ij} = 1 \qquad i \in V_1(G) \tag{2.13}$$

$$\sum_{\{i,j\} \in E(G)} x_{ij} = 1 \qquad j \in V_2(G) \tag{2.14}$$

$$x_{ij} \in \{0,1\} \qquad \{i,j\} \in E(G). \tag{2.15}$$

Although the flow capacity parameters $u_{ij} = 1$ for all $\{i,j\} \in E(G)$, there is no need to include flow capacity restrictions in this formulation due to the fact that flow balance constraints (2.13) and (2.14) imply these bounds. When the integrality constraints of x_{ij}, i.e. constraints (2.15), are replaced with $x_{ij} \geq 0$ $\{i,j\} \in E(G)$ and AP is solved as a linear programming problem, an integer optimal assignment is found since its constraint matrix is totally unimodular [26].

2.3.5. Other Problems

There are also other special versions of MCFP which are beyond the scope of this thesis such as the fixed charge network flow problem [28, 29], the shortest path problem [30], the transportation problem [31] and the circulation problem [32]. In the fixed charge network flow problem, there are fixed costs of utilizing arcs that are incurred once when positive flow sent through them. The shortest path problem and MFP can be considered as complementary models. The first one ignores the flow bounds and seeks the minimum cost s-t path whereas MFP searches for a maximum flow from s to t obeying the flow bounds but disregarding its cost. The transportation problem is a special instance of MCFP and a more general version of AP. This problem is defined on a bipartite graph where the two vertex subsets are not necessarily of equal cardinality and the flow upper bounds are free to take any value. Lastly, the circulation problem aims to find the minimum cost flow circulated in a network where $b_i = 0$ for all $i \in V(N)$.

2.4. Conflict Notion

Any pair of arcs are said to be conflicting when only one of them is allowed to have positive flow in a feasible solution. Any pair of paths with at least two conflicting arcs are said to be conflicting. Notice that conflicting paths are not necessarily arc or vertex disjoint. Any disjoint pair of paths can be conflicting, and any two compatible conflict-free paths can share arcs or vertices.

As a direct result of the flow decomposition theorem that we mention previously, it is possible to say that an (arc) flow on $N = (V(N), A(N))$ is conflict-free if and only if its equivalent path and circuit flow representations are conflict-free, since an arc (i, j) with positive flow (i.e. $f_{ij} > 0$) appears at least on one path or circuit with positive flow on it in a path and circuit flow representation, and if it conflicts another arc with positive flow, then there exists another path or circuit of the same representation having positive flow on it and conflicting with arc (i, j). For the network flow problems with conflicts, the directed paths from a source vertex to a sink vertex and circuits, namely flow paths and circuits, of a feasible path and circuit flow must be conflict-free. In other words, an optimal solution is a *conflict-free flow* with respect to an objective function, and feasible with respect to the balance, lower bound and capacity restrictions.

We consider a special form of conflict in the minimum cost noncrossing flow problem on layered networks (MCNFP). Here, any arc pair conflicts if and only if they cross each other with respect to the special embedding of the vertices in the layered network. As a consequence, if two arcs $(i_1, j_1), (i_2, j_2) \in A_{l(l+1)}(N)$ *cross*, then $i_1 > i_2$ and $j_1 < j_2$ and vice versa; they form a crossing. Observe that arcs $(s, j) \in A_{12}(N)$ as well as arcs $(i, t) \in A_{(L-1)L}(N)$ are noncrossing. Any pair of paths with at least two distinct arcs that cross each other is said to be crossing. Notice that crossing paths are not necessarily arc or vertex disjoint. Any disjoint pair of paths may cross, and any two noncrossing paths may share arcs or vertices.

Then it is possible to say that an (arc) flow on $N = (V(N), A(N))$ is *noncrossing* if and only if all paths of the equivalent path flows are noncrossing, since an arc (i, j)

with positive flow (i.e. $f_{ij} > 0$) appears on at least one path with positive flow on it, and if it is crossed by another arc, then there exists another path having positive flow on it with an arc crossing arc (i, j). For example, in Figure 2.4, the paths $s \to 2 \to 1 \to t$ and $s \to 1 \to 2 \to t$ are crossing since they both have positive flows (i.e. 3 and 2 units of flows respectively) and arcs $(2, 1)$ and $(1, 2)$ are crossing. The directed paths from a source vertex to a demand vertex with positive flow on its arcs, namely flow paths, must be noncrossing in addition to flow balance, lower and upper bound restrictions in order to be feasible. We say such flow is feasible and also noncrossing.

2.4.1. Conflict Graph

We can use a *conflict graph* $C = (V(C), E(C))$, as done in [17, 33], to represent conflict relations between the arcs of network $N = (V(N), A(N))$. Here, the vertex set $V(C)$ consists of the arcs $A(N)$ of the network, and the edge set $E(C)$ consists of the conflicting arc pairs. In other words, $(i, j), (k, l) \in A(N)$ are conflicting if and only if $\{(i, j), (k, l)\} \in E(C)$. Let $\delta_C(\{(i, j)\}) \subseteq E(C)$ be the set of edges incident with vertex (i, j) in the conflict graph and its size denotes the degree of vertex (i, j) in the conflict graph C, i.e. $d_C(i, j)$. For simplicity, we use the notation $\delta_C(i, j)$ instead of $\delta_C(\{(i, j)\})$. We also let $N_C(i, j)$ and $\overline{N}_C(i, j)$ denote the sets of neighbors and non-neighbors of (i, j) in the conflict graph C. Clearly, $|N_C(i, j)| = d_C(i, j)$.

Notice that every subset of conflict-free arcs of $A(N)$ forms a stable set of the conflict graph C. If the subset of conflict-free arcs cannot be expanded by adding one more arc without violating the conflict-free property, than it corresponds to a maximal stable set on C. Besides, cliques of the conflict graph C represent the family of the arc subsets of $A(N)$ where every arc conflicts with the remaining ones in the subset. A conflict-free subset of arcs of $A(N)$ can be constructed by picking at most one element from each clique of $V(C)$.

2.4.2. Conflict Formulations

We use conflicting arc lists in order not to allow positive flow values on conflicting arcs in a solution. For each arc $(i, j) \in A(N)$, a separate list $\delta_C(i, j)$ consisting of arcs conflicting with (i, j) is prepared. Recall that $\delta_C(i, j)$ is the set of neighbors of (i, j) in the conflict graph C. Obviously, $(k, l) \in \delta_C(i, j)$ if and only if $(i, j) \in \delta_C(k, l)$. In order to prevent the simultaneous selection of conflicting arcs, the binary variables x_{ij} $(i, j) \in A(N)$ are introduced. If $x_{ij} = 1$, then f_{ij} is allowed to take a positive value and f_{kl} for all $(k, l) \in \delta_C(i, j)$ are forced to be zero. Therefore, we introduce the conflict constraints

$$x_{ij} + x_{kl} \leq 1 \qquad (k, l) \in \delta_C(i, j);\ (i, j) \in A(N), \tag{2.16}$$

which are known as the packing constraints in the literature. Alternatively, conflict constraints (2.16) can be replaced with

$$\sum_{(k,l)\in\delta_C(i,j)} x_{kl} + |\delta_C(i, j)|x_{ij} \leq |\delta_C(i, j)| \qquad (i, j) \in A(N). \tag{2.17}$$

We refer to this new version as the weak (W) and the previous one as the strong (S) representation. Let P_S and P_W denote the polyhedrons representing the feasible solution sets of the LP relaxations of the strong and weak representations. In other words,

$$P_S = \{\boldsymbol{x} \in \mathbb{R}^m : x_{ij} + x_{kl} \leq 1\ (k, l) \in \delta_C(i, j);\ (i, j) \in A(N) \text{ and } x_{ij} \geq 0\ (i, j) \in A(N)\},$$

and

$$P_W = \{\boldsymbol{x} \in \mathbb{R}^m : \sum_{(k,l)\in\delta_C(i,j)} x_{kl} + |\delta_C(i, j)|x_{ij} \leq |\delta_C(i, j)| \text{ and } x_{ij} \geq 0\ (i, j) \in A(N)\}.$$

Then $P_S \subseteq P_W$ since inequalities (2.17) can be obtained by aggregating inequalities (2.16) for $(k,l) \in \delta_C(i,j)$.

If we let $\mathcal{K}$ denote the family of maximal cliques of C where every edge in $E(C)$ belongs to some $K \in \mathcal{K}$, a third set of constraints, which we call the *clique* (K) representation, can be obtained by replacing constraints (2.16) with the clique inequalities

$$\sum_{(i,j)\in V(K)} x_{ij} \leq 1 \qquad K \in \mathcal{K}. \tag{2.18}$$

P_K is the polyhedron corresponding to the LP relaxation of the clique representation of the conflicts represented by

$$P_K = \{\boldsymbol{x} \in \mathbb{R}^m : \sum_{(i,j)\in V(K)} x_{ij} \leq 1 \ K \in \mathcal{K} \text{ and } x_{ij} \geq 0 \ (i,j) \in A(N)\}.$$

Based on the previous works [34,35] we can say that $P_K \subseteq P_S$. Notice that, the number of maximal cliques can be exponential for a graph. Producing all maximal cliques is not realistic unless the conflict graph belongs to a special class on which the number of maximal cliques is limited by a number polynomial in the size of the graph. In this case, a sufficient number of cliques can be generated so that the edges of C are covered by those cliques. However, we cannot claim that a clique representation with a subset of $\mathcal{K}$ is stronger than the other two conflict formulations.

3. LITERATURE REVIEW

The conflict constraints are widely used in the literature with different names such as *negative disjunctive constraints, exclusionary side constraints* or *incompatibility constraints* within various contexts. For example, in the knapsack problem with conflicts (KPC) some pairs of items are not allowed to be packed together [33, 36–40]. Another extensively studied problem is the bin packing problem with conflicts (BPPC) where some pairs of objects are not allowed to be put into the same bin [41–48]. In the transportation problem with conflicts (TPC), certain pairs of supply nodes which are defined separately for each demand point can not send items to that demand point simultaneously [49–52].

Besides, we encounter variants of the well-known network optimization problems taking into account the incompatible pairs of edges/arcs. In all of these variants, we are given the sets of conflicting pairs of edges/arcs and at most one of them can be selected in a feasible solution. The first one of these problems is the minimum spanning tree problem with conflicts (MSTPC), which aims to find a tree touching all the vertices with minimum total edge cost. Conflict constraints prevent conflicting pairs of edges to be contained together in a feasible spanning tree [53–56]. The second one is the matching problem with conflicts (MPC). There exist several versions of MPC. Their objective is to find either a matching with maximum cardinality, i.e. the maximum (cardinality) matching problem with conflicts (MMPC) [17], or a matching with minimum total edge weights as in the minimum cost perfect matching problem with conflicts (MCPMPC) [57, 58]. Another variant is the shortest path problem with conflicts (SPPC) which can be treated as a network flow problem where all flows are equal to one [17]. The most related work which we consider in this thesis is the maximum flow problem with conflicts (MFPC) in which the goal is to send the maximum possible flow from the source to the sink by selecting at most one of the conflicting arc pairs [16].

3.1. Complexity Results

The difficulty of the mentioned problems is thoroughly studied in the literature. The authors mostly benefit from the *conflict graph* which represents the conflict relations between the variables of the concerned problem in order to carry out the complexity analysis and to determine the special conditions under which these problems are solvable in polynomial time [17, 33]. Recall that the vertices of the conflict graph correspond to the variables and there exists an edge between two vertices if they conflict with each other. If the problem in question can be represented on a conflict graph, it is simply reduced to finding an independent set that optimizes an objective function. The obtained results about the hardness of these problems show that the addition of the conflict constraints complicate the problem even though there exist polynomial or pseudo-polynomial time algorithms to solve their original versions (i.e. the versions without conflict constraints).

To begin with, the 0-1 knapsack problem with a polynomially bounded knapsack size can be optimally solved through a greedy heuristic or a dynamic programming approach. However, 0-1 KPC is proved to be strongly NP-hard in [33] by reminding that it can be seen as a maximum weight stable set problem (MWSP). This paper provides algorithms that run in pseudo-polynomial time if the conflict graph of the knapsack problem with conflicts belongs to the special graph classes, namely trees, graphs with bounded treewidth and chordal graphs. Furthermore, a dynamic programming algorithm is developed for this problem if its conflict graph is an interval graph in [47]. Bin packing problem is one of the classic NP-hard problems [20]. For BPPC, the authors in [41] propose approximation algorithms with a worst case bound if the corresponding conflict graph belongs to graph classes such as split graphs, cographs or partial k-trees on which coloring problem is solvable in polynomial time. Transportation problem is another problem for which there exist a number of polynomial time algorithms. But, the addition of conflict constraints makes it intractable as shown in [52]. Actually, finding a feasible solution for TPC is proven to be strongly NP-hard after applying a reduction from the graph 3-colorability problem, which is also strongly NP-hard [20]. On the other hand, if the number of demand nodes are limited to two and the conflicting supplier set is identical for all demand nodes or if the number of suppliers are fixed and not a part of the problem input, then it turns out to be weakly NP-hard and pseudo-polynomial time algorithms are proposed to solve these special instances in the same work.

We can also find complexity results about the conflict variants of network optimization problems. The minimum spanning tree problem can be solved in $O(|A(N)|$ $log|V(N)|)$ or $O(|A(N)| + |V(N)|$ $log|V(N)|)$ time with Kruskal's and Prim's algorithms, respectively [6]. On the contrary, MSTPC is strongly NP-hard [17]. If its conflict graph consists of disjoint maximal cliques, then it can be solved in strongly polynomial time [17]. The same study reveals that MMPC is strongly NP-hard and the SPPC belongs to APX-hard problems which constitute a subclass of NP-hard problems. For APX-hard problems, finding an approximation better than a constant c is as hard as finding the optimal solution indicating that APX-hardness is a stronger complexity result than being NP-hard. [54] and [57] provide polynomially solvable special cases of the MSTPC and MCPMPC, respectively. Finally, MFPC is shown to be strongly NP-hard by using a reduction from MWSP in [16].

3.2. Exact Solution Approaches

Exact solution algorithms have also been proposed for most of these problems. For KPC, an implicit enumeration scheme combined with an interval reduction method is proposed in [36]. A branch-and-bound (BB) algorithm together with apriori reduction techniques are proposed in [38]. The same study presents a dichotomous search procedure which makes use of dominating constraints and covering cuts to construct an alternative representation of the same problem. Another BB algorithm with improved upper bounding procedures and branching strategies is presented recently [40]. It is compared to other existing solution methods through experiments; the computational results are analyzed in-depth and reported in detail to assess the performance of every specific component of the algorithm. Three different branch-and-price algorithms for BPPC are introduced in [45], [46], and [47]. The one in [45] is based on the set covering formulation which can be used as an alternative representation of the considered problem. The second approach utilizes Lagrangean relaxation bounds within the branch-and-price scheme which allows branching not only on a single pattern but also on multiple patterns. The valid inequalities are also produced for the subproblems as the algorithm continues. The last one mostly relies on the way it solves the pricing problem and demonstrates an outstanding performance compared to the existing methods in the literature. MSTPC is solved exactly by a very simple branch-and-bound procedure, which makes use of the polynomially solvable cases in [54], and by a branch-and-cut method in [55]. For TPC, there exist two branch-and-bound schemes proposed in [10] and [50]. The transportation problem formulation in [10] includes nonlinear side constraints, each of which restricting the product of the associated integer flow variables to zero to prevent conflicting arcs from appearing in the solution at the same time. The proposed exact solution method is a BB algorithm that solves ordinary transportation problems obtained by relaxing the nonlinear equalities to compute bounds, and sets one of the two variables violating a nonlinear side constraint with their relaxed optimal values to zero to branch. [59] considers also the same formulation but they propose evolutionary algorithms for its solution. The primary difference of the works of [49] and [50] is in the considered TPC formulation; the nonlinear side constraints are linearized using additional binary variables, packing inequalities and big-M coefficients. As a solution method, BB with a combinatorial branching rule and sophisticated early pruning techniques using the spanning trees of network simplex algorithm are proposed. [60] is the first stone that paves the road towards developing a mature exact solution method to solve the assignment problem with conflicts (APC). A naïve Branch-and-Bound (BB) algorithm, which can be seen as a very preliminary version of the one presented in Chapter 8, is introduced. It is tested with only two branching rules, which are fairly straightforward. The computational experiments are also preliminary and performed on small sized instances. Still, the results are promising and encouraging.

3.3. Heuristics

In spite of the proposed efficient exact solution methods, large instances of the problems with conflict constraints cannot be solved in a reasonable amount of time due to their inherent difficulty. Hence, a number of heuristic methods are also presented in the literature. For KPC, a reactive local search method that uses a degrading process to escape from local maxima and to ensure diversification is described in [37]. Also a memory list is kept to avoid repetitive configurations. Other local search methods both naïve and enriched with diversification processes can be found in [36] and [39]. Several methods to solve BPPC including first fit decreasing packing heuristic, coloring and clique based heuristics and their hybrid forms are described in [43]. Different lower bounds [44] and an iterated local search with a few neighborhood structures that enhance the solution quality [48] are available in the literature. Naïve forms of a local search heuristic, a tabu search metaheuristic and a lagrangean heuristic for MSTPC are briefly presented in [54]. The authors in [57] focus on APC and develop tabu search methods improved with frequency matrices and enhanced by keeping the most recent local optimal solutions. Also, four different lower bounding schemes for this problem is given in the same study. [58] mainly focuses on simple iterated algorithms for APC. One of the iterated algorithms is based on a local search scheme and the other one employs a commercial mixed integer linear programming problem solver as a subroutine. For TPC, a tabu search metaheuristic which benefits from the linearized problem formulation is utilized in [49]. This metaheuristic takes advantage of the underlying spanning tree structure of a visited solution to estimate the best move in a neighborhood. Moreover, a sophisticated hybrid spanning tree based genetic algorithm for the same problem can be found in [51]. It is reported that it produces successful results in spite of prolonged computation time.

3.4. Applications

Several practical applications of BPPC appear in computer science. One of them aims, given a set of processes (e.g. multimedia streams), to determine the minimum number of processors where some pair of processes cannot be assigned to the same processor in order to increase fault tolerance by not allowing to process the two replicas in the same processor or increase efficiency by forcing the usage of two CPUs on different processors. Another problem arises in parallel computing of partial differential equations. If a result of subcomputation influences the other, they cannot be scheduled simultaneously and the goal is to minimize the total computation time [42]. Similarly, resource clustering problem in highly distributed parallel computing can be mentioned [61]. Other examples are the examination scheduling problem that minimizes the number of time periods where two exams cannot be held in the same period if they have common enrolled students and the delivery problem where flammable, explosive or toxic materials cannot be transported in the same vehicle [43, 62]. Those can be modeled as the BPPC and KPC emerges as a subproblem in this context.

The exclusionary flow restriction which mostly arises in the transportation problem can also be faced in real life. For example, containers arriving to a container terminal may not be positioned in the same row of the yard due to differences in size, ownership, or content [10]; a warehouse may not be allowed to receive goods simultaneously from some pairs of suppliers because of the safety requirements [49]; at most one supply node from each supplier may be allowed to deliver goods to a customer in the procurement problems under a total quantity discount policy [63]; messages may not be routed simultaneously in a sensor network because of the inference they cause in each others' communication channels [64]; some quay cranes may not be assigned to some pair of vessels at the same time because they have to move on a line and cannot cross each others [11, 65]; products may not be compatible to be carried on the same vehicle [66].

4. PROBLEM DEFINITIONS AND FORMULATIONS

In this chapter, the definitions of the problems we study within the scope of this thesis are given and the various mathematical models we develop to formulate them are presented.

4.1. Minimum Cost Noncrossing Flow Problem[1]

The minimum cost noncrossing flow problem (MCNFP) is an extension of the ordinary minimum cost flow problem (MCFP) on layered networks with additional compatibility constraints in conjunction with the flow balance, capacity, lower bound, and binary restrictions. In general, a layered network provides a graphical tool to model the scheduling of flow with spatial constraints.

4.1.1. Flow Scheduling with Spatial Constraints

Three important problems associated with the management of seaside operations at container terminals are the berth allocation problem (BAP), the quay crane assignment problem (QCAP), and the quay crane scheduling problem (QCSP). Excellent surveys of the related works with a classification according to some specific attributes are provided in [65, 67–70].

In general, BAP deals with the determination of optimal berthing times and positions of the vessels. It is possible to visualize a solution of BAP by means of a time-berth diagram where the y-axis represents the quay discretized in berth sections vessels can occupy and the x-axis represents time. A common assumption is that each berth section is just large enough to be occupied by only one quay crane. A sample time-berth diagram is given in Figure 4.1. There are five vessels and the rectangles represent the area they cover on the time-berth diagram. For example vessel 1 is berthed at berth sections 1 and 2, and stays berthed from time 0 to 3 (i.e. 3 time

[1] An earlier version of this section appears in [12] as a part of its content.

periods). QCAP finds the optimal number of cranes assigned to the vessels, and thus can be seen as a special form of the optimal crane splitting problem [70]. The numbers within the parenthesis in the rectangles are the crane numbers required per time period that guarantee the determined length of stay for the vessels. For example, vessel 2 stays berthed for 9 periods, and demands 3 cranes during periods 1 – 3 and 4 cranes during periods 4 – 9. QCSP focuses on assigning quay cranes to optimal work places in each interval, given the berth locations of the vessels along the quay and number of the cranes that should serve them (i.e. the information Figure 4.1 provides) with the objective of minimizing the total setup cost due to crane relocations on the berth over the planning horizon. An interval is the time that elapses between two sequential events. An event is a specific vessel activity capable to cause a change in the number of serving cranes; it is an arrival or departure. More than one event can occur at the same time. This is also illustrated in Figure 4.1. There are five intervals. For example, the first one starts with the arrivals of vessels 1, 2 and 3 at time 0, and ends with the departures of vessels 1 and 3, and arrival of vessel 4 at time 3, which is also the starting time of the second interval. Observe that the number of cranes in service during an interval cannot be larger than 7, which is the total number of cranes in the terminal. A layered network representation describing the sample situation explained above is

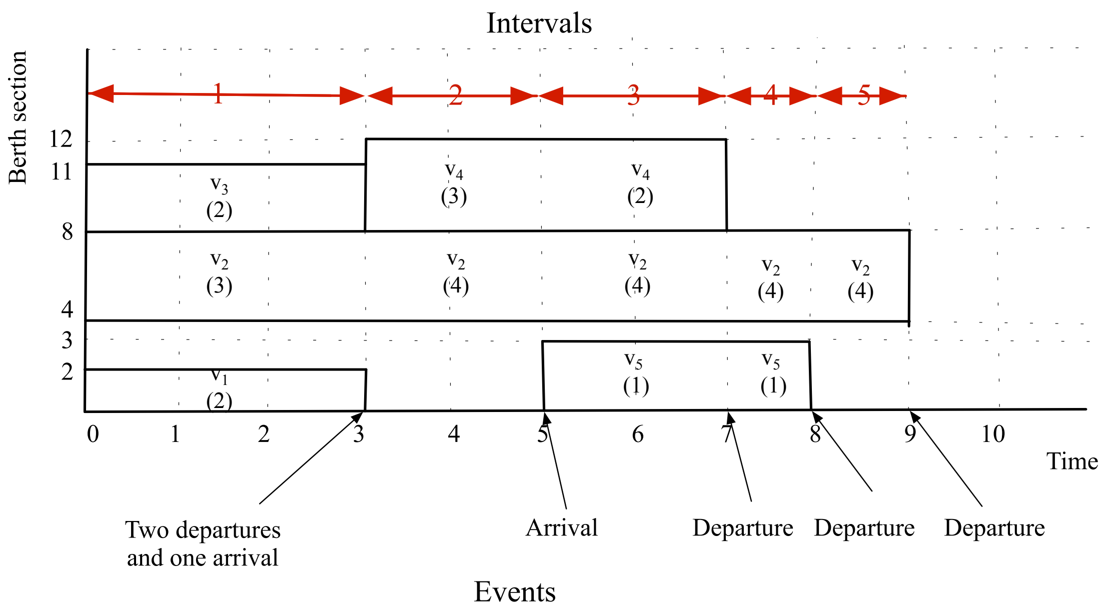


Figure 4.1. Sample berth allocations and quay crane assignments for four vessels

given in Figure 4.2. This is a directed, layered, single source, and single sink network. The only vertex of the first layer is the super source, which represents the terminal's resources, with supply equal to the total number of quay cranes. Similarly, the last layer consists of a single vertex as well. It is the super sink with demand equal to the total number of quay cranes; it also represents the terminal's resources. The remaining vertices belong to internal layers and represent the vessels demanding cranes at each time interval. Except the first and the last layers, each one of L layers represents a snapshot of the berth during a time interval. In other words, layer l exists in the network if a vessel berths or departs changing the current snapshot; it is then followed by a new one. Notice that the number of cranes assigned to the berthed vessels can change only between consecutive layers, since such a change can only be caused by either an arrival or departure. These events and the intervals they represent are also depicted in Figure 4.1.

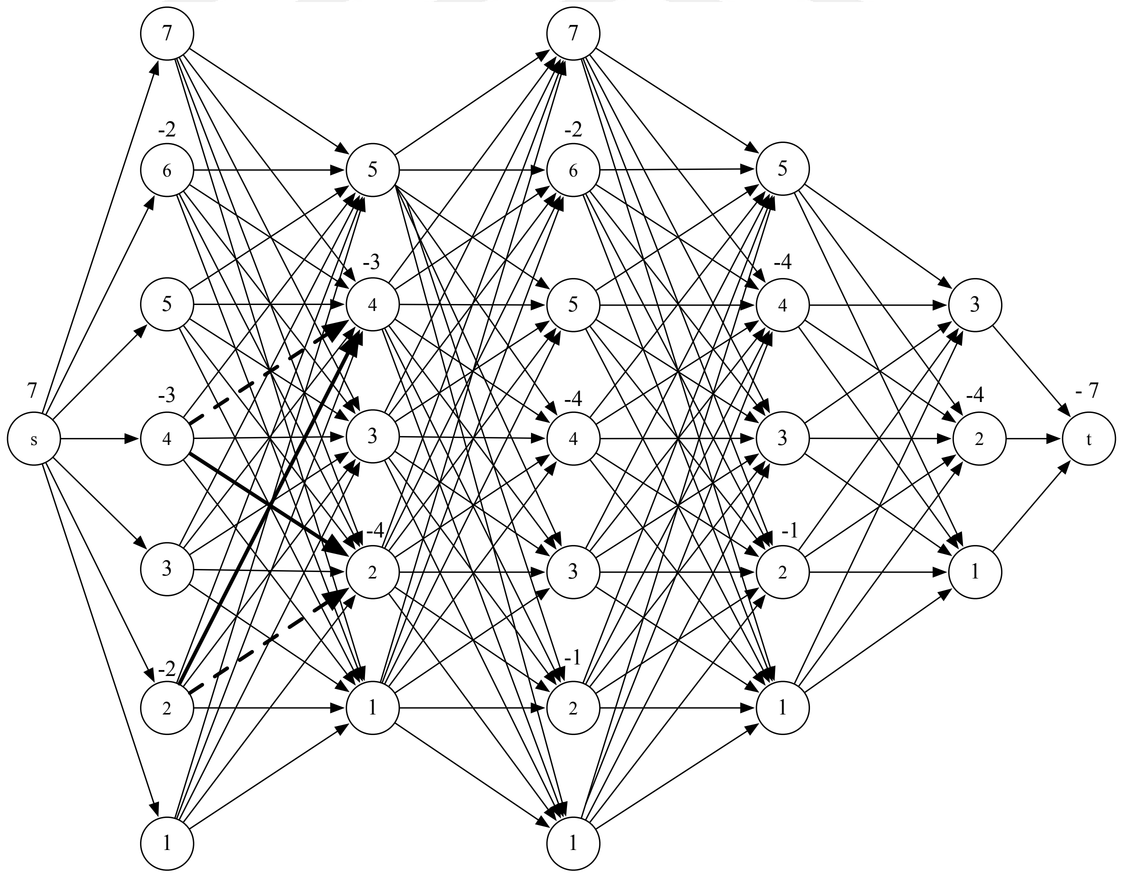


Figure 4.2. A layered network describing the situation given in Figure 4.1

At each layer l, a berthed vessel is represented by a vertex, whose demand is equal to the number of assigned cranes. These vertices are called vessel vertices, and they are ordered starting from the bottom of the layer to the top in accordance with their position in the berth from the beginning to the end. There is a second type of vertices below and above the vessel vertices. They are called parking vertices and represent the waiting area for the idle cranes. In any layer, the number of parking vertices is one more than the number of vessel vertices. To summarize, by letting n_l denote the number of berthed vessels at the quay during interval l, there are n_l vessel vertices and $n_l + 1$ parking vertices. Hence, the total number of vertices in layer l is $2n_l + 1$ and a vertex with an even index (i.e. $2, 4, \ldots, 2n_l$) corresponds to a vessel vertex, while those with an odd index (i.e. $1, 3, \ldots, 2n_l + 1$) represent parking vertices with finite capacities. The capacity of the first (last) parking vertex in each layer is equal to the number of available berth sections between the beginning (end) of the berth and the first (last) vessel, whereas the capacity of an intermediate parking vertex is equal to the number of available berth sections between two neighboring vessels. If there is not enough room in one of these locations, then the capacity of the corresponding vertex is set to zero, which is a consequence of the assumption that no more than one crane can physically fit into a berth section; this is the unit measure used for the discretization of the quay. The negative numbers on the even vertices are the crane demands of the vessels in Figure 4.2.

The network of Figure 4.2 is incomplete to be a base for the MCNFP formulation. As can be easily observed, the total demand is not equal to the total supply. Besides, the odd vertices are capacitated and they have to be appropriately presented. For this purpose, except the super source s and super sink t, we split the vertices by creating a clone for each one. At layer l, the original vertex i_l and its clone i'_l are connected by arc (i_l, i'_l). Since the demand of an original vessel vertex in layer l is equal to the number of assigned cranes, so is the demand of a clone vessel vertex. Therefore, both the lower and upper bounds on the flow of arc (i_l, i'_l) are set to the crane demand of the vessel, which implies that the flow on the arc (i_l, i'_l) for even i_l is forced to be equal to the demand value. Similar to the vessel vertices, parking vertices also have clones. An original parking vertex i_l and its clone i'_l for odd i_l are connected by an arc whose lower and upper bounds are set to zero and the capacity of the parking vertex i_l respectively. In short, every vertex has a weight. Vessel vertices have crane demands, and parking vertices have capacities. By using clones, vessel vertices are transformed from transshipment supply vertices into pure transshipment vertices, and parking vertices are transformed into uncapacitated, pure transshipment vertices.

The flows on the arcs of this network correspond to crane relocations or movements from parking areas to vessels, from vessels to vessels (this includes the case where a crane continues serving the same vessel or starts serving a new vessel without an idle period), from vessels to parking areas in each time interval. The costs associated with these relocations are defined as unit flow costs. They become setup costs for an arc connecting a clone vertex of layer $l-1$ to an original vertex of layer l. Some of them can be zero as well. For example, the ones for cranes that keep serving the same vessel between consecutive intervals, and the ones that keep waiting at the same berth section or, move from parking areas to different parking areas. This construction is illustrated in Figure 4.3 for the example of Figure 4.1 and Figure 4.2. It is limited to the first three layers (i.e. time intervals) for the sake of simplicity. The numbers on arcs $(2,2')$, $(4,4')$, and $(6,6')$ are the demands (i.e. both the lower and upper bound values of the flow) of vessel vertices 2, 4 and 6. The capacities for parking vertices are not shown; but they can be selected appropriately. Notice that, except s and t, the vertices are pure transshipment vertices. QCSP becomes an ordinary MCFP on the described layered network if crane crossing is allowed. Unfortunately, this is not possible in reality; quay cranes are restricted to move on a rail and thus the relocation paths cannot cross. In other words, it is not enough to solve the MCFP on the described layered network to determine an optimal crane schedule for the example of Figure 4.1, since, for example, an optimal solution can include flows on arcs $(2,4)$ and $(4,2)$ in Figure 4.2 between the second and third layers. They become arcs $(2',4)$ and $(4',2)$ in Figure 4.3. This is actually the end of the first interval when both vessels 1 and 2 depart, and vessel 4 arrives. Four cranes have to be relocated to satisfy the demand of vessel 4, which is three cranes, and the additional demand of vessel 2, which is one crane. If this is done by sending one crane from vessel 1 (i.e. vertex $2'$) to vessel 4 and (i.e. vertex 4) a crossing with arc $(4',2)$ occurs because this crane is blocked by the cranes serving

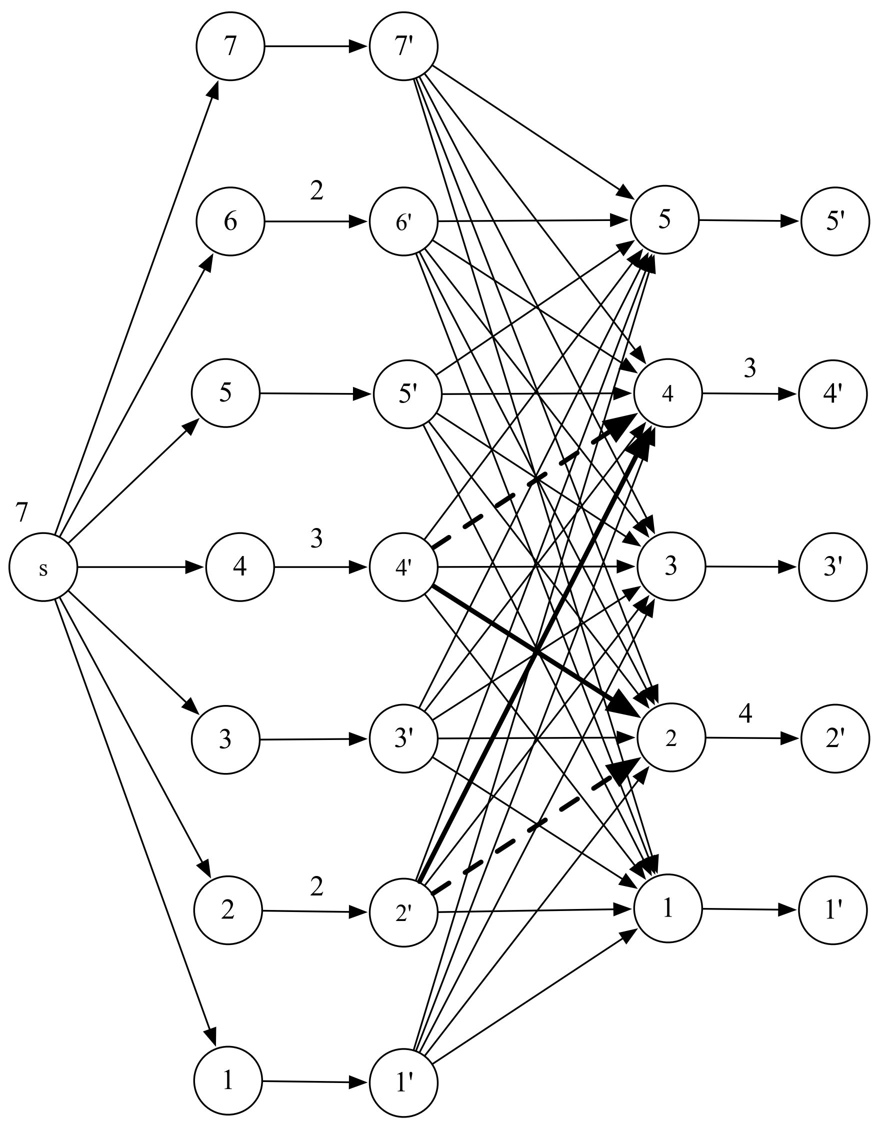


Figure 4.3. The layered network appropriate for a minimum cost network flow formulation of the example given in Figure 4.1 and Figure 4.2

vessel 2, which is shown as the crossing of the two bold solid arcs in Figure 4.2 and Figure 4.3. This can also be handled by sending one crane from vessel 2 to vessel 4 (i.e. on arc $(4', 4)$), and one crane from vessel 1 to vessel 2 (i.e. on arc $(2', 2)$), namely by sending flow on the bold dashed arcs of Figure 4.2 and Figure 4.3. As a consequence, it is possible to say that QCSP is in fact equivalent to a MCFP with additional spatial constraints allowing only noncrossing arcs to have positive flow values in an optimal solution, which makes it a particular subclass of MCNFP.

MCNFP is a generalization of the QCSP where we consider a flow problem with suppliers and customers located on a line. The commodity flow is realized by means of vehicles, which are restricted to move along a single track lane, and hence cannot pass each other as a spatial restriction. Besides, suppliers and customers have time varying operating characteristics: at a given time some of them can leave and/or new ones can arrive, and can change their supplies/demands. The purpose is to determine an optimal commodity flow schedule between them so that total distribution cost is minimized. We will consider this generalization in the rest of this thesis.

4.1.2. Problem Representation

It is possible to formulate MCNFP as a mixed-integer linear programming problem (MILP) by allowing only noncrossing arcs to have positive flows. In other words, for each arc $(i, j) \in A_{l(l+1)}(N)$, if there is a positive flow on (i, j), i.e. if $f_{ij} > 0$, then $f_{pq} = 0$ for all $(p, q) \in A_{l(l+1)}(N)$ with either $1 \leq p \leq i-1$ and $j+1 \leq q \leq n_{l+1}$, or $i+1 \leq p \leq n_l$ and $1 \leq q \leq j-1$. Obviously, $f_{ij} = 0$ if $f_{pq} > 0$ for one of such $(p, q) \in A_{l(l+1)}(N)$. In addition to the flow variables f_{ij}, we use the previously introduced binary design variables $x_{ij} \in A_{l(l+1)}(N)$ to model this. x_{ij} is set to 1 if $f_{ij} > 0$. Besides, if $x_{ij} = 1$ then $f_{pq} = 0$ for all (p, q) such that either $1 \leq p \leq i-1$ and $j+1 \leq q \leq n_{l+1}$, or $i+1 \leq p \leq n_l$ and $1 \leq q \leq j-1$. This allows us to define the list S_{ij} of arcs incompatible (i.e. crossing) with arc $(i, j) \in A_{l(l+1)}(N)$ as

$$\begin{aligned} S_{ij} = \{(p, q) \in A_{l(l+1)}(N) : 1 \leq p \leq i-1, j+1 \leq q \leq n_{l+1}; \\ i+1 \leq p \leq n_l, 1 \leq q \leq j-1\} \quad l = 1, 2, \ldots, L. \end{aligned} \tag{4.1}$$

Notice that S_{ij} is equivalent to the $\delta_C(i, j)$ which is the set of neighbors of (i, j) in the conflict graph C. But we prefer to use S_{ij} in the context of MCNFP rather than $\delta_C(i, j)$ in order to emphasize that arc incompatibility is caused by physical crossing of arcs. Then, we obtain the following MILP formulation for MCNFP.

$$\text{MCNFP:} \quad \min \sum_{(s,j)\in A_{12}(N)} c_{sj} f_{sj} + \sum_{l=2}^{L-2} \sum_{(i,j)\in A_{l(l+1)}(N)} c_{ij} f_{ij} + \sum_{(i,t)\in A_{L-1L}(N)} c_{it} f_{it} \tag{4.2}$$

$$\text{s.t.} \quad \sum_{(s,j)\in A_{12}(N)} f_{s,j} = b_s \tag{4.3}$$

$$\sum_{(i,j)\in A_{l(l+1)}(N)} f_{ij} - \sum_{(j,i)\in A_{(l-1)l}(N)} f_{ji} = b_i \qquad i \in V_l(N); \; l = 2,3,\ldots,L-1 \tag{4.4}$$

$$- \sum_{(i,t)\in A_{(L-1)L}(N)} f_{it} = b_t \tag{4.5}$$

$$0 \leq f_{ij} \leq u_{ij} x_{ij} \qquad (i,j) \in A_{l(l+1)}(N); \; l = 1,2,\ldots,L-1 \tag{4.6}$$

$$x_{pq} + x_{ij} \leq 1 \qquad (p,q) \in S_{ij}; \; (i,j) \in A_{l(l+1)}(N); \; l = 1,2,\ldots,L-1 \tag{4.7}$$

$$x_{ij} \in \{0,1\} \qquad (i,j) \in A_{l(l+1)}(N); \; l = 1,2,\ldots,L-1. \tag{4.8}$$

Without constraints (4.7) and (4.8), and with u_{ij} instead of $u_{ij}x_{ij}$ in constraints (4.6) the formulation is the one of ordinary MCFP on the layered network illustrated in Figure 2.3. The constraints (4.7) are equivalent to the conflict constraints (2.16) ensuring that flow can only be sent through only noncrossing arcs. This formulation is referred as the strong MCNFP formulation since the strong conflict representation is used to model the relation between the crossing arcs and it is denoted by MCNFP$_S$.

4.2. Minimum Cost Flow Problem with Conflicts[2]

The minimum cost flow problem with conflicts (MCFPC), in which the simultaneous shipment of goods through some pairs of arcs is prohibited, is obtained by generalizing the previous results obtained for MCNFP for general networks and dealing with a more abstract version of incompatibility constraints, namely conflicting arcs. As opposed to the *crossing arcs* encountered in MCNFP, conflicting arcs do not need to cross each other with respect to a particular embedding of vertices in the network. They do not have to share a common tail or head; they can be any pair of arcs in the network.

We can observe that MCFPC has an interesting recent application in online event-based social networks. The aim is to find conflict-aware event-participant arrangements given a set of users and a set of events. In this setting, every event allows limited number of participants and there is a maximum number of activities a user can join. A user cannot participate in certain pairs of events because either the events coincide or the time interval between two events is restrictive concerning the location of events. For each user-event pair an *interestingness* value is calculated based on the event history that the user attend previously. The higher this value is, the more the user enjoys that event. The objective is to find the user-event matching satisfying the capacity of users and events, and maximizing the total interestingness. Since this is equivalent to minimizing the total dissimilarity between the matched user-event pairs, this problem can be modeled as a MCFPC [71].

MCFPC can be formulated as an MILP by generalizing from the MCNFP formulation. We use the conflicting arc list of (i, j) which is previously defined and denoted by $\delta_C(i, j)$ in order not to allow positive flow values on conflicting arcs in an optimal solution. The previously introduced flow variables, $\boldsymbol{f}$, and binary design variables, $\boldsymbol{x}$, are used.

[2]An earlier version of this section appears in [13] as a part of its content.

If $f_{ij} > 0$, i.e. if there is a positive flow on (i,j), then $f_{kl} = 0$ for all $(k,l) \in \delta_C(i,j)$. If $f_{ij} > 0$, then $x_{ij} = 1$. Besides, if $f_{ij} > 0$, then $x_{kl} = 0$ and thus $f_{kl} = 0$ for all $(k,l) \in \delta_C(i,j)$. Then, MCFPC has the following MILP formulation:

$$\begin{aligned}
\text{MCFPC: min} \quad & \sum_{(i,j)\in A(N)} c_{ij} f_{ij} && && (4.9)\\
\text{s.t.} \quad & \sum_{(i,j)\in A(N)} f_{ij} - \sum_{(j,i)\in A(N)} f_{ji} = b_i && i \in V(N) && (4.10)\\
& x_{ij} + x_{kl} \leq 1 && (k,l) \in \delta_C(i,j);\ (i,j) \in A(N) && (4.11)\\
& 0 \leq f_{ij} \leq u_{ij} x_{ij} && (i,j) \in A(N) && (4.12)\\
& x_{ij} = 0, 1 && (i,j) \in A(N). && (4.13)
\end{aligned}$$

Without constraints (4.11) and (4.13) and with u_{ij} instead of $u_{ij}x_{ij}$ in constraints (4.12), the above MILP formulation becomes the LP formulation of the ordinary MCFP on a general network. It can be observed that MCFP is a relaxation of both MCFPC and MCNFP. MCFPC formulation ensures that any feasible flow cannot be sent through conflicting arcs by using the strong conflict constraints. Therefore, this model is referred as the strong formulation of MCFPC and denoted by MCFPC$_S$ in the sequel.

4.3. Maximum Flow Problem with Conflicts[3]

A variant of MFP obtained by adding the conflict constraints is examined. In a feasible solution of this extended problem, which we briefly call the maximum flow problem with conflicts (MFPC), at most one of the conflicting arcs is allowed to have positive flow on it. An interesting application of this problem arises in multi-hop wireless networks on which the performance of signal transmission degrades substantially with the increasing number of traversed hops. In a network of nodes with identical and omnidirectional radio ranges, going from a single hop to 2 hops halves the throughput of a flow. Hence interference dictates that only one of the links connecting these two

[3]An earlier version of this section appears in [14] as a part of its content.

hops can be active at a time where the aim is to maximize the flow throughput between the given source-destination pairs [72]. MFPC is known to be strongly NP-hard [16].

The goal is to determine the maximum flow from s to t obeying the flow bounds and conflict restrictions. We can formulate MFPC as an MILP since it is a special version of MCFPC where $b_i = 0$ for all vertices in $V(N)$ except the source and the sink. Hence, the following MILP is obtained:

$$\text{MFPC: } \max \quad v \tag{4.14}$$

$$\text{s.t.} \quad \sum_{(i,j)\in A(N)} f_{ij} - \sum_{(j,i)\in A(N)} f_{ji} = \begin{cases} v & i = s \\ 0 & i \in V(N)\backslash\{s,t\} \\ -v & i = t \end{cases} \tag{4.15}$$

$$x_{ij} + x_{kl} \leq 1 \qquad (k,l) \in \delta_C(i,j) : \ (i,j) \in A(N) \tag{4.16}$$

$$f_{ij} \leq u_{ij} x_{ij} \qquad (i,j) \in A(N) \tag{4.17}$$

$$x_{ij} = 0,1 \ , f_{ij} \geq 0 \qquad (i,j) \in A(N). \tag{4.18}$$

The flow-balance equalities (4.15) and the flow capacity constraints (4.17) without $\boldsymbol{x}$ variables are common with the original MFP formulation. The conflict constraints (4.16) prevent simultaneous flow on the conflicting arcs. Based on this fact, MFP is a relaxation of MFPC obtained by relaxing conflict constraints. This version is called the strong MFPC formulation and denoted as MFPC$_S$.

4.4. Assignment Problem with Conflicts[4]

Another considered problem is the assignment problem with conflicts (APC), which is an extension of AP with additional conflict constraints. In this extended problem, when an item (e.g. job) is assigned to a destination (e.g. machine) the assignment of a subset of items to a subset of destinations may be prohibited. In other words, APC is equivalent to finding a minimum weight perfect matching such that no more than one edge is selected from each conflicting edge pair. Although various

[4]An earlier version of this section appears in [15] as a part of its content.

additional restrictions have been previously studied, the consideration of conflicts in particular, has been recent.

APC is a close relative of graph and network based combinatorial optimization problems, which also include conflict constraints. Actually it is the special form of the minimum cost flow problem with conflicts where the underlying graph is undirected and bipartite and the flow upper bounds of all edges are set to one.

One of the important yard operations in container terminals is to find the best possible depositing position in the storage area for the arriving containers. The storage yard is divided into several rows and arriving containers are classified according to certain criteria (ownership, size, full/empty containers, preferences of the customers etc.). As a result, it is not possible to stack some of the containers in the same row at neighbor locations of the storage yard due to the incompatibilities between them. Then, the determination of an optimum assignment strategy that minimizes total cost of operations (searching and/or loading of containers) can be formulated as an APC.

A similar type of storage problem occurs in warehouses. Some of them may not be allowed to receive goods from some pairs of producers because of the incompatibilities between their product; they may cause deterioration or damage to each other or to the facility when they are stored at closely. For example food and toxic products cannot be stored together, they should be separated from each other and from other goods to eliminate damages.

Manpower planning is another area for possible applications of APC. Workers may be assigned to jobs according to interpersonal relations in addition to qualifications and requirements. In certain cases a pair of workers cannot be assigned to a pair of jobs. This may be because of their incapability of forming a team on these jobs. For example when some pilot is assigned to the crew of a flight, some of the cabin personnel should not be on the same crew due to some disfunctional conflict between them and the pilot.

In transportation, two or more products can be incompatible and are not allowed on the same vehicle. For example hazardous material may become unstable if mixed with other products, and food cannot be mixed with chemicals. The first approach is to use dedicated vehicles with homogenous loads. Unfortunately, this leads to a very large fleet size in case of a large number of incompatibilities between the products, namely to the waste of resources and money. The other approach is to use heterogeneous vehicles and avoid the loading of the incompatible products on the same vehicle, which can be handled by means of assignment constraints in conjunction with conflict inequalities.

We would like to finalize giving potential applications by mentioning that APC arises also as a subproblem when solving some of the difficult combinatorial optimization problems. One such problem is the quadratic bottleneck assignment problem [73], which is a generalization of the bandwidth assignment problem in matrices and graphs [74]. Another one is a potential extension of the ordinary asymmetric traveling salesman obtained after adding conflict inequalities to formulate the dependence between the exiting and entering arcs of the cities: some of the leaving arcs may not be selected if a particular entering arc is in the tour and a tour can consist of only compatible arcs. The relaxation obtained by disregarding the subtour elimination constraints is an APC.

It is possible to formulate APC as a binary integer programming (BIP) problem. To this end, we make use of a conflicting edge pair list. Let $G = (V_1(G) \cup V_2(G), E(G))$ be a complete bipartite graph, where $|V_1(G)| = |V_2(G)| = n$, $V_1(G) \cap V_2(G) = \emptyset$ and $E(G) = \{e_1, \ldots, e_m\}$. Note that, in the sequel, we will interchangeably denote an edge with $e = \{i, j\}$ such that $i \in V_1(G)$ and $j \in V_2(G)$. For each edge $\{i, j\} \in E(G)$, the previously defined conflicting edge list $\delta_C(i, j)$ is used. If $e \in E(M)$ where $E(M) \subseteq E(G)$ is the edge set of a perfect matching $M = (V(M), E(M))$, then the edges $f \in \delta_C(e)$ are not allowed to be in $E(M)$. Obviously, $f \in \delta_C(e)$ if and only if $e \in \delta_C(f)$, and $V(M) = V_1(G) \cup V_2(G)$ since M is perfect.

As a typical example, suppose there are n workers and n jobs. If worker i is assigned to job j the cost incurred will be c_{ij}. Assume worker i can work at job j and worker k can work at job l different from job j. In certain cases it may be required that workers i and k cannot be assigned to jobs j and l at the same time. This may be due to their inabilities to work together within the same team. In other words, the assignment of worker i to job j is in conflict with the assignment of worker k to job l, and thus they are the elements of each other's conflicting assignment list, i.e. $\{i, j\} \in \delta_C(k, l)$ and $\{k, l\} \in \delta_C(i, j)$. Our goal is to find a conflict-free assignment with minimum possible total cost.

Notice that the binary decision variables x_{ij} for $e = \{i, j\} \in E(G)$ which exist in the ordinary assignment problem (AP) formulation, are sufficient to model the conflict relations. No extra variables are needed. x_{ij} set to 1 if and only if edge $e = \{i, j\} \in E(M)$. Then, the following BIP formulation which is first proposed in [58] is obtained:

$$\text{APC: } \min \sum_{\{i,j\}\in E(G)} c_{ij}x_{ij} \tag{4.19}$$

$$\text{s.t.} \sum_{\{i,j\}\in E(G)} x_{ij} = 1 \qquad i \in V_1(G) \tag{4.20}$$

$$\sum_{\{i,j\}\in E(G)} x_{ij} = 1 \qquad j \in V_2(G) \tag{4.21}$$

$$x_{ij} + x_{kl} \leq 1 \qquad \{k, l\} \in \delta_C(i, j) : \ \{i, j\} \in E(G) \tag{4.22}$$

$$x_{ij} = 0, 1 \qquad \{i, j\} \in E(G). \tag{4.23}$$

Equations (4.20) and (4.21) are AP constraints and ensure the construction of a perfect matching in the bipartite graph G. Conflict constraints (4.22) guarantee that no edge pair in a feasible perfect matching can be conflicting. In other words, when $\{\{i, j\}, \{k, l\}\}$ is a conflicting pair then at most one of the edges, i.e. either $\{i, j\}$ or $\{k, l\}$, or neither can appear in a perfect matching. AP is a relaxation of APC, obtained by removing the conflict constraints and this version is called the strong mathematical representation of APC which we denote as APC$_S$ in the sequel.

4.5. Alternative Formulations

4.5.1. Weak Formulations

An equivalent formulation of MCFPC is obtained by replacing inequalities (4.11) with

$$\sum_{(p,q)\in\delta_C(i,j)} x_{pq} + |\delta_C(i,j)|x_{ij} \leq |\delta_C(i,j)| \qquad (i,j) \in A(N); \qquad (4.24)$$

Notice that, the constraint set (4.24) is the same with the weak conflict representation shown with constraints (2.17).

Weak conflict constraints (2.17) can be used to obtain the weak formulation of MCNFP. Similarly, constraints (4.21) of MFPC can be replaced with their aggregated versions given as constraint (2.17). Moreover, we can rewrite the conflict constraints (4.22) of APC as

$$\sum_{(k,l)\in\delta_C(i,j)} x_{kl} + |\delta_C(i,j)|x_{ij} \leq |\delta_C(i,j)| \qquad \{i,j\} \in E(G). \qquad (4.25)$$

As can be noticed, these formulations give weaker linear programming (LP) bounds since inequalities (2.17) and (4.25) are obtained by aggregating strong conflict constraints over the list $\delta_C(i,j)$ for each arc (i,j) and edge $\{i,j\}$. Hence, we refer to these versions as the weak formulations and we denote them as MCNFP$_W$, MCFPC$_W$, MFPC$_W$ and APC$_W$.

4.5.2. Clique Formulations

Given the family of maximal cliques of the conflict graph, denoted by $\mathcal{K}$, clique formulations of MCNFP, MCFPC, MFPC and APC can be attained by replacing constraints (4.7), (4.11), (4.21) and (4.22) with the clique inequalities (2.18). In the remaining chapters, those are referred as MCNFP$_K$, MCFPC$_K$, MFPC$_K$ and APC$_K$, respectively.

Since the number of maximal cliques can be exponential in the size of the graph, we generate a subset of $\mathcal{K}$ instead of generating all members of $\mathcal{K}$. This subset of clique inequalities are selected so that they cover all the edges of the conflict graph C i.e. all conflict relations are represented. When we use a subset of $\mathcal{K}$, we cannot claim that the LP bound of clique formulation is better than the one of the strong formulation but it still gives relaxation bound for the problem.

```
Algorithm 4.4: Greedy heuristic to find a subset of $\mathcal{K}$
Input: A conflict graph $C = (V(C), E(C))$;
Output: A subset of $\mathcal{K}$;
begin
 $l = 1$, $W = V(C)$;
 while $W \neq 0$ do
   Sort the vertices of $V(C)$ in nonincreasing $d_C(i,j)$ values with respect to $C[W]$
   for all arcs $i \in V(C)$ do
     Put $i$ into $U$ if it is adjacent to all vertices in $U$;
   end for
   $K_l \longleftarrow U$;
   $W \longleftarrow W \setminus K_l$;
   $l \longleftarrow l + 1$;
 end while
 Return $K_j,\ j = 1, 2, ..., l - 1$;
end
```

Figure 4.4. Greedy heuristic to find a subset of $\mathcal{K}$.

4.5.3. Combinatorial Optimization Problem Formulations

Recall that a subset S of conflict-free arcs form a stable set of the conflict graph C. As a consequence, MCFPC becomes the combinatorial optimization problem

$$\min\{z^1(S) : S \in \mathcal{S}(C)\}, \tag{4.26}$$

where $\mathcal{S}(C)$ is the family of the maximal stable sets of the conflict graph and the value of the objective function is calculated by solving a reduced MCFP on the subnetwork $(V(N), S)$ with vertices $V(N)$ and arcs S, of the original network N. In other words,

$$\begin{aligned}
\text{MCFP}(S)\colon\ z^1(S) = \min \quad & \sum_{(i,j)\in S} c_{ij} f_{ij} && && (4.27)\\
\text{s.t.} \quad & \sum_{(i,j)\in S} f_{ij} - \sum_{(j,i)\in S} f_{ji} = b_i && i \in V(N) && (4.28)\\
& 0 \le f_{ij} \le u_{ij} && (i,j) \in S. && (4.29)
\end{aligned}$$

In case MCFP(S) is not feasible for a given S, we set $z^1(S) = \infty$ for convenience.

Proposition 4.1. *Let S and S' be two stable sets of the conflict graph C. Then, $z^1(S') \le z^1(S)$ for $S \subseteq S'$.*

Proof. The problem MCFP(S') is a relaxation of MCFP(S) since its set of variables include the set of variables of MCFP(S) as a subset. □

Note that the given formulation is valid to model MCNFP if the vertex and arc indices are modified accordingly. Furthermore, since APC is a special case of MCPFC, the combinatorial optimization version of APC can be written similarly where S represents the edges of the original bipartite graph G. Suppose that the optimal objective value of AP on a stable set S, i.e. APC(S), is denoted by $z^2(S)$ and this value is equal to infinity if there is no feasible assignment on the given stable set. Then, finding the stable set with the minimum $z^2(S)$ value is equivalent to solving APC.

Therefore, we attain the combinatorial optimization problem, which is equivalent to APC,

$$\min\{z^2(S) : S \in \mathcal{S}(C)\}. \tag{4.30}$$

Corollary 4.1. *Let S and S' be two stable sets of the conflict graph C. Then, $z^2(S') \leq z^2(S)$ for $S \subseteq S'$.*

Proof. This is is a direct result of Proposition 4.1 as APC(S) is a particular version of MCFPC(S). □

A similar approach can be used to provide a combinatorial optimization formulation for MFPC. Since it is a maximization problem, MFPC can be reformulated using the stable sets of the conflict graph as

$$\max\{z^3(S) : S \in \mathcal{S}\}, \tag{4.31}$$

where the value $z^3(S)$ is calculated by solving a maximum flow problem on the subnetwork $(V(N), S)$ of the original network N. In other words,

$$\text{MFPC}(S)\text{: } z^3(S) = \max \quad v \tag{4.32}$$

$$\text{s.t.} \quad \sum_{(i,j)\in S} f_{ij} - \sum_{(j,i)\in S} f_{ji} = \begin{cases} v & i = s \\ 0 & i \in V(N)\backslash\{s,t\} \\ -v & i = t \end{cases} \tag{4.33}$$

$$0 \leq f_{ij} \leq u_{ij} \qquad (i,j) \in S. \tag{4.34}$$

Proposition 4.2. *Let S and S' be two stable sets of the conflict graph C such that $S \subseteq S'$. Then, $z^3(S') \geq z^3(S)$.*

Proof. The assertion trivially follows from the fact that the feasible solution space of MFPC(S) includes the feasible solutions of MFPC(S'). $\square$

As a direct consequence of these propositions, it is sufficient to consider only maximal stable sets of C. Besides, since the relaxation problems MCFP, AP and MFP are polynomially solvable, it is straightforward to say that for any conflict graph with a polynomial number of maximal stable sets MCFPC, APC and MFPC can be solved in polynomial time. At this point two questions one can ask are whether there exists graph classes with a number of stable sets bounded by a polynomial in the size of the graph (i.e. the number or vertices and/or edges), and whether there exist a polynomial time algorithm to generate them.

An answer for the first question can be provided by considering the cliques of the complement $\overline{C}$ of the conflict graph C, since stable sets of a graph are the cliques of its complement. For example chordal graphs have at most n maximal cliques [75], graphs of boxicity k have at most $(2n)^k$ maximal cliques [76], planar graphs have at most $7n/3-6$ maximal cliques [76], and $K_k - free$ graphs have at most $\max\{n, n\Delta^{k-2}/2^{k-2}\}$ maximal cliques [77], where n is the number of vertices and Δ is the maximum degree of any vertex. This is not true when $\overline{C}$ is a general graph since the maximum number of maximal cliques is $3^{\lceil n/3 \rceil}$ in a general graph [78]. Yet another answer can be given by checking the maximum number of the minimal dominating sets of the conflict graph itself since a maximal stable set is a minimal dominating set of a graph, i.e. the maximal stable sets of a graph form a subset of its minimal dominating sets. Lower and upper bounds on the maximum number of minimal dominating sets of certain graph classes are reported in [79]. These bounds are mostly exponential functions of the number of vertices. The two exceptions are the threshold and chain graphs. The maximum number of minimal dominating sets for the threshold graphs is equal to $\omega(G)$, which is the size of the largest clique of G. This number becomes $\lfloor n/2 \rfloor + m$ for chain graphs, where m is the number of edges. The lower and upper bounds respectively become $15^{n/6}$ and 1.7159^n for general graphs [80].

The second question is related to the generation of maximal stable sets and minimal dominating sets of C, and maximal cliques of $\overline{C}$. Most of the algorithms are enumerative and have running times within a polynomial factor of the proved upper bound for the graph class in question. For example, the one in [81] generates all maximal stable sets, the one in [82] generates all maximal cliques and the ones in [79] generate all minimal dominating sets.

In short, when the conflict graph belongs to one of the classes of threshold or chain graphs, or the complement of the conflict graph belongs to one of the classes of chordal graphs, graphs of boxicity k, planar graphs, and $K_k - free$ graphs, the maximum number of maximal stable sets are bounded by a polynomial in the number of vertices n and the number of edges m, and they can be generated in polynomial time. Hence, MCFPC, MFPC and APC can be solved in polynomial time for these particular cases by first generating the family $\mathcal{S}(C)$ of the maximal stable sets of C and then evaluating them by solving MCFPC(S), MFPC(S) or APC(S) for $S \in \mathcal{S}(C)$. Obviously, this can be done in time bounded by a polynomial in n and m in the worst case.

5. COMPLEXITY ANALYSIS

This chapter presents the complexity analysis results for the minimum cost noncrossing flow problem on layered networks (MCNFP) and the minimum cost flow problem with conflicts on general networks (MCFPC). Moreover, a particular condition which makes MCNFP and the assignment problem with conflicts (APC) polynomially solvable is explained.

5.1. The Difficulty of the Minimum Cost Noncrossing Flow Problem[5]

We first define the decision problems associated with the MCNFP and its variant with restricted total flow costs for the flow paths (MCNFP-RC) in the following.

MCNFP Instance: A layered network $N = (V(N), A(N))$ with $L \in \mathbb{Z}_+$ layers each of which having $n_l \in \mathbb{Z}_+$ vertices $l = 1, 2, \ldots, L$ except the first and last ones: they consist of single vertices, namely a source s for layer $l = 1$ and a sink t for layer $l = L$. There is a supply/demand $b_i \in \mathbb{Z}_+$ for every vertex $i \in V(N)$ satisfying $\sum_{i \in V^+(N)} b_i = \sum_{i \in V^-(N)} b_i$. For each arc $(i, j) \in A(N)$ there is a capacity $u_{ij} \in \mathbb{Z}_+$, and unit flow cost $c_{ij} \in \mathbb{Z}_+$. There is also a given number $C \in \mathbb{Z}_+$.

Question: Is there a noncrossing arc flow with total cost less than C?

MCNFP-RC Instance: The same as the MCNFP instance.

Question: Is there a noncrossing arc flow with total cost less than C for every flow path?

[5]An earlier version of this section appears in [12] as a part of its content.

Proposition 5.1. *MCNFP-RC is NP-complete for*

$$b_{i_l} = \begin{cases} B, & \text{if } l = 1 \text{ (i.e. } i_1 = s) \\ -B, & \text{if } l = L \text{ (i.e. } i_L = t) \\ 0, & \text{otherwise,} \end{cases}$$

with $B \in \mathbb{Z}_+$.

Proof. i. MCNFP-RC $\in NP$: If a flow is given, its feasibility can be checked in polynomial time. Checking the feasibility of the flow with respect to the flow balance constraints and bounds can be done in $O(|A(N)|)$ time. Checking whether there is a crossing requires at most $O(|V(N)|^2 L)$ time. As a consequence of the flow decomposition theorem [19] given a nonnegative arc flow it is possible to generate all flow paths in $O(|V(N)|+|A(N)|)$ time and the number of flow paths is $O(|V(N)|+|A(N)|)$ in the worst case. Thus, checking whether or not the total cost of each path is less than C takes $O((|V(N)| + |A(N)|)|A(N)|) = O(|A(N)|^2)$ time. Therefore, there is a polynomial time certificate checking algorithm and MCNFP-RC $\in NP$.

ii. MCNFP-RC is hard (Reduction from the set partitioning problem): The *Set Partitioning Problem* (SPP), which is known to be NP-complete [20], reduces polynomially to MCNFP-RC. The SPP deals with the following question: Given a set S of V elements with values $s_v \in \mathbb{Z}_+$, $v = 1, 2, \ldots, V$ and $\sum_{v\in S} s_v = D$, is there a subset $S' \subset S$ such that $\sum_{v\in S'} s_v = \sum_{v\in S\setminus S'} s_v = \frac{D}{2}$?

An instance of MCNFP-RC corresponding to an arbitrary SPP instance can be the complete layered network $N = (V(N), A(N))$ with

a. $L = V + 3$ layers,

b. $n_l = 2,\ l = 2, 3, \ldots, L-1;\ n_1 = n_L = 1$ vertices at each layer having a supply/demand

$$b_{i_l} = \begin{cases} 2, & \text{if } l = 1 \text{ (i.e. } i_1 = s) \\ -2, & \text{if } l = L \text{ (i.e. } i_L = t) \\ 0, & \text{otherwise,} \end{cases}$$

c. $C = D + (V+1)$,

d. unit flow cost

$$c_{i_l i_{l+1}} = \begin{cases} \frac{D}{2}, & \text{if } l = 1; i_{l+1} = 1, 2 \\ 1, & \text{if } l = L-1; i_{L-1} = 1, 2 \\ 1, & \text{if } l = 2, 3, \ldots, L-1; i_l = 1, i_{l+1} = 2 \\ 1, & \text{if } l = 2, 3, \ldots, L-1; i_l = 2, i_{l+1} = 1 \\ s_{l-1} + 1, & \text{if } l = 2, 3, \ldots, L-1; i_l = i_{l+1} = 1 \\ s_{l-1} + 1, & \text{if } l = 2, 3, \ldots, L-1; i_l = i_{l+1} = 2, \end{cases}$$

e. lower bounds $l_{i_l i_{l+1}} = 0$ and capacities

$$u_{i_l i_{l+1}} = \begin{cases} 1, & \text{if } l = 2, 3, \ldots, L-1; i_l = 1, i_{l+1} = 2 \\ 1, & \text{if } l = 2, 3, \ldots, L-1; i_l = 2, i_{l+1} = 1 \\ u_{i_l i_{l+1}} \in \mathbb{Z}_+, & \text{otherwise.} \end{cases}$$

Figure 5.1 illustrates the network obtained after this transformation for $S = \{s_1, s_2, s_3\}$, $V = 3$, $S' = \{s_1, s_3\}$, $L = 3 + 3 = 6$. The numbers on the arcs are the unit costs. For vertex numbering we use the previously mentioned convention. Two noncrossing flow paths satisfying the total flow cost restriction $C = D + 4$ are presented using dashed arcs. Observe that the paths have unit flow on them, satisfy balance equalities, lower bound and capacity restrictions, and cost restrictions. Furthermore, they are noncrossing. The first of the two paths presents subset S' (path $1 \to 2 \to 1 \to$

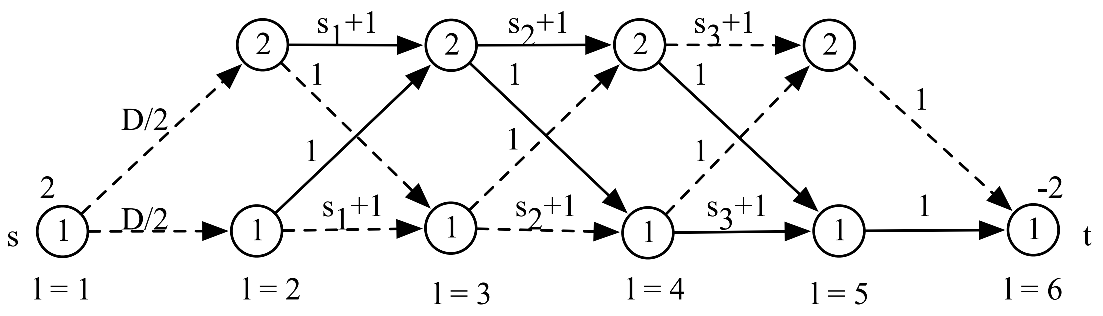


Figure 5.1. Noncrossing paths corresponding to an SPP instance

$1 \to 2 \to 1$) and the second one the subset $S \backslash S'$ (path $1 \to 1 \to 1 \to 2 \to 2 \to 1$). They are not necessarily disjoint; arc $(2, 1)$ between layers 5 and 6 is traversed by both paths. The noncrossing flow path representation of sets S' and S is not unique. They can be represented using two other paths as well. For example path $1 \to 2 \to 2 \to 1 \to 1 \to 1$ for S' and path $1 \to 1 \to 2 \to 2 \to 1 \to 1$ for $S \backslash S'$. Notice that, this time arc $(1, 1)$ between layers 5 and 6 is on both paths. What must be done now is to show that S has a subset S' such that $\sum_{v \in S'} s_v = \sum_{v \in S \backslash S'} s_v = \frac{D}{2}$ if and only if there is a feasible flow with noncrossing flow paths each having at most $C = D + (V + 1)$ total flow cost.

First, suppose that S has a subset S' such that $\sum_{v \in S'} s_v = \sum_{v \in S \backslash S'} s_v = \frac{D}{2}$. Then, it is possible to generalize the path structure of Figure 5.1 so that the first path includes the elements of S' and the second path the elements of $S \backslash S'$. As can be noticed, these flow paths are noncrossing, satisfy flow balance equalities, lower bounds and capacity restrictions, and each has a total flow cost $D + (V + 1)$ (i.e. $\frac{D}{2} + \sum_{v \in S'} s_v + (V + 1) = D + (V + 1) = \frac{D}{2} + \sum_{v \in S \backslash S'} s_v + (V + 1)$). Thus, if the set S has a subset S' such that $\sum_{v \in S'} s_v = \sum_{v \in S / S'} s_v = \frac{D}{2}$, it is possible to construct two noncrossing flow paths each with a total flow cost equal to $D + (V + 1)$.

Conversely, suppose that we are given a flow feasible with respect to flow balance equalities, lower bounds, capacity restrictions and having only noncrossing flow paths each with a cost less than $D + (V + 1)$. First of all for the described MCNFP-RC instance there can be at most two s-t flow paths since exactly two units of flow have to be sent out of source s. Single s-t flow path (with two units of flow on it) is not possible because the total cost of the one with the smallest total cost is $D + 2(V + 1)$, which is larger than the restriction $D+(V+1)$. Hence, two distinct flow paths have to start at source s. Besides, they must satisfy the cost restrictions (i.e. each has a total cost of at most $D+(V+1)$).

Consider the arcs (i_l, i_{l+1}) such that $i_l = i_{l+1} = 1, 2$ for $l = 2, 3, \dots, L-1$. This is the pair of arcs with costs s_v+1, $v \in S$. Then, at least one of these two arcs $i_l = i_{l+1} = 1, 2$ must appear on one of these two paths for each $l = 2, 3, \dots, L-1$. Otherwise, there is a crossing because of the network structure and unit upper bounds on arcs (i_l, i_{l+1}) with $i_l = 1$ and $i_{l+1} = 2$, and $i_l = 2$ and $i_{l+1} = 1$ for $l = 2, 3, \dots, L-1$ (i.e. in case there is one which is missing on both paths) or one of the paths has cost larger than $D+(V+1)$ (i.e. one of them can be traversed by both paths), which is a contradiction. In short, there are two flow paths each having unit flow on it and partitioning the arcs with costs s_v+1 $v \in S$ (i.e. these arcs appear exactly on one of them) and thus the sum of the total costs is equal to $2\frac{D}{2} + \sum_{v\in S} s_v + 2(V+1) = 2(D+(V+1))$. This implies that each flow path satisfies fully its total cost restriction $D+(V+1)$, since each has a total flow cost less than $D+(V+1)$ with grand total exactly equal to $2(D+(V+1))$. Finally, one of the flow paths cannot include all of them (i.e. the set S entirely) because this results in a total cost of $\frac{D}{2} + \sum_{v\in S} s_v + (V+1) = \frac{3}{2}D + (V+1) > D+(V+1)$. Let S be the set of these arcs and S' be its subset included in the first path. Then, other path would traverse the arcs in $S' \setminus S$. Recall that each one of these paths has total cost $D+(V+1)$. Therefore, $\frac{D}{2} + \sum_{v\in S'} s_v + (V+1) = D+(V+1)$ and $\frac{D}{2} + \sum_{v\in S\setminus S'} s_v + (V+1) = D+(V+1)$, which implies that $\sum_{v\in S'} s_v = \sum_{v\in S\setminus S'} s_v = D - \frac{D}{2} = \frac{D}{2}$. This transformation can be done in $O(V)$ time. $\square$

Proposition 5.2. *MCNFP is NP-complete when*

$$b_{i_l} = \begin{cases} B', & \text{if } l = 1 \text{ (i.e. } i_1 = s) \\ -B', & \text{if } l = L \text{ (i.e. } i_L = t) \\ 0, & \text{otherwise,} \end{cases}$$

with $B' \in \mathbb{Z}_+$.

Proof. i. MCNFP $\in NP$: First of all any certificate of MCNFP can be checked in polynomial time similar to MCNFP-RC. Hence, MCNFP $\in NP$.

ii. MCNFP is hard (Reduction from MCNFP-RC): Consider any arbitrary instance of MCNFP-RC with unit costs $c_{ij} \in \mathbb{Z}_+$, $(i,j) \in A(N)$, supply/demand $b_i \in \mathbb{Z}_+$, $i \in V(N)$ and capacities $u_{ij} \in \mathbb{Z}_+$, $(i,j) \in A(N)$ as assumed in the assertion, and cost restrictions $D \in \mathbb{Z}_+$ for the flow paths. To generate a particular MCNFP instance we keep the same layered network structure of a MCNFP-RC instance, but modify unit costs, arc capacities, supplies and demands.

We choose

$$b'_{i_l} = \begin{cases} B', & \text{if } l = 1 \text{ (i.e. } i_1 = s) \\ -B', & \text{if } l = L \text{ (i.e. } i_L = t) \\ 0, & \text{otherwise,} \end{cases}$$

with

$$B' = B + \left\lceil \frac{1}{(L-1)} \right\rceil = B + 1.$$

We set the unit costs all equal to λ', arc capacities to $u'_{ij} = u_{ij} + 1$, and the restriction $C = \lambda'(L-1)B'$. Here $\lambda' \in \mathbb{Z}_+$ is larger enough than BD.

First suppose that MCNFP-RC has a noncrossing flow $\mathbf{f} \in \mathbb{Q}_+^{|A(N)|}$ satisfying flow balance, arc capacity constraints and cost restrictions on the flow paths $P_k = (V(P_k), A(P_k))$ $k = 1, 2, \ldots, K$. Here, K is the number of noncrossing flow paths and $K \leq |V(N)| + |A(N)|$ as a consequence of the flow decomposition theorem [19]. However, for this particular case, due to the integrality of the flow and network structure $K \leq B$.

Since each flow path satisfies the cost restrictions,

$$\sum_{(i,j)\in A(P_k)} c_{ij} f_k \leq D \qquad k = 1, 2, \ldots, K,$$

and consequently

$$\sum_{k=1}^{K} \sum_{(i,j)\in A(P_k)} c_{ij} f_k \leq KD.$$

Here, f_k is the amount of positive flow sent through the kth flow path. Then,

$$\sum_{(i,j)\in A(N)} c_{ij} f_{ij} \leq KD \tag{5.1}$$

follows, since $f_{ij} = \sum_{k=1}^{K} \sum_{\{P_k:(i,j)\in A(P_k)\}} f_k$, where f_{ij} is the flow on arc (i, j).

At this point, we have to show the following claim.

Claim 5.1.

$$\sum_{k=1}^{K} \sum_{(i,j)\in A(P_k)} f_k = (L-1)B.$$

Proof. First of all, $\sum_{k=1}^{K} \sum_{(i,j)\in A(P_k)} f_k = \sum_{k=1}^{K} |A(P_k)| f_k$. Because of the special structure of the network $N = (V(N), A(N))$ (i.e. layered network with forward arcs) every feasible arc flow can be represented as a path flow having exactly $L - 1$ arcs. Besides, each flow path with positive flow on it connects a source vertex to a sink vertex [19]. Hence,

$$\sum_{k=1}^{K} \sum_{(i,j)\in A(P_k)} f_k = \sum_{k=1}^{K} |A(P_k)| f_k = (L-1) \sum_{k=1}^{K} f_k.$$

In addition, the sum of the flows over the flow paths is equal to the sum of supplies, which is equal to the negative of the sum of the demands, namely to B. In other words $\sum_{k=1}^{K} f_k = \sum_{i\in V^+(N)} b_i = -\sum_{i\in V^-(N)} b_i = B$, which completes the proof. $\square$

Then, for unit costs $c'_{ij} = c_{ij} + \lambda_{ij}$ with $\lambda_{ij} \in \mathbb{Z}_+$, $\overline{\lambda} = \max_{(i,j)\in A(N)}\{\lambda_{ij}\} = BD$ and $\underline{\lambda} = \min_{(i,j)\in A(N)}\{\lambda_{ij}\}$ the total flow cost becomes

$$\begin{aligned}
\sum_{(i,j)\in A(N)} c'_{ij} f_{ij} &= \sum_{(i,j)\in A(N)} c_{ij} f_{ij} + \sum_{(i,j)\in A(N)} \lambda_{ij} f_{ij} \\
&= \sum_{k=1}^{K} \sum_{(i,j)\in A(P_k)} c_{ij} f_k + \sum_{k=1}^{K} \sum_{(i,j)\in A(P_k)} \lambda_{ij} f_k \\
&\leq KD + \overline{\lambda} \sum_{k=1}^{K} \sum_{(i,j)\in A(P_k)} f_k \\
&= KD + \overline{\lambda}(L-1)B \\
&\leq BD + \overline{\lambda}(L-1)B \\
&= \overline{\lambda}[(L-1)B + 1] \\
&= \overline{\lambda}(L-1)\left(B + \frac{1}{L-1}\right) \\
&\leq \overline{\lambda}(L-1)(B+1) \\
&= \overline{\lambda}(L-1)B' \\
&\leq \lambda'(L-1)B'.
\end{aligned}$$

The second term of the fourth expression follows from the second term of the third expression as consequence of claim 5.1. The fifth expression follows from the fourth since $B \geq K$. We also use the definition $\overline{\lambda} = BD$ and the fact that $\lambda' \geq \overline{\lambda}$. Notice that,

$$C = \lambda'(L-1)B' = \lambda' \sum_{k=1}^{K} \sum_{(i,j)\in A(P_k)} f'_k.$$

In other words, this upper bound C is the total cost of an arc flow $\mathbf{f}'$, with path flow f'_k $k = 1, 2, \ldots, K$ on the flow paths P_k $k = 1, 2, \ldots, K$, for the same network structure with B' and u'_{ij} as defined previously, and unit flow costs set to λ'. Notice that it is possible to obtain $\mathbf{f}'$ by increasing the flow $\mathbf{f}$ on one of the flow paths P_k $k = 1, 2, \ldots, K$, say f_p on path P_p by one unit and keeping the remaining ones the same, i.e. by setting $f'_p = f_p + 1$, $f'_k = f_k$ for $k \neq p$. This is a feasible solution of the particular MCNFP instance we have created, i.e. a noncrossing flow feasible with respect to the flow balance equalities and capacity restrictions, with total cost equal to C.

Conversely, suppose that the particular MCNFP instance has a noncrossing flow with total cost not larger than C. Let $P'_k = (V(P'_k), A(P'_k))$, $k = 1, 2, \ldots, K'$ be the corresponding K' flow paths of a feasible flow $\mathbf{f}'$ of the particular MCNFP instance, which also satisfies total cost restriction. Let also f'_k $k = 1, 2, \ldots, K'$ be the path flow corresponding to these K' flow paths. Hence,

$$\begin{aligned}
C = \lambda'(L-1)B' &= \lambda' \sum_{k=1}^{K'} \sum_{(i,j)\in A(P'_k)} f'_k \\
&\geq \sum_{k=1}^{K'} \sum_{(i,j)\in A(P'_k)} c'_{ij} f'_k \\
&= \sum_{(i,j)\in A(N)} c'_{ij} f'_{ij} \\
&= \sum_{(i,j)\in A(N)} c_{ij} f'_{ij} + \sum_{(i,j)\in A(N)} \lambda_{ij} f'_{ij} \\
&= \sum_{k=1}^{K'} \sum_{(i,j)\in A(P'_k)} c_{ij} f'_k + \sum_{k=1}^{K'} \sum_{(i,j)\in A(P'_k)} \lambda_{ij} f'_k \\
&\geq \sum_{k=1}^{K'} \sum_{(i,j)\in A(P'_k)} c_{ij} f_k + \sum_{k=1}^{K'} \sum_{(i,j)\in A(P'_k)} \lambda_{ij} f'_k \\
&\geq \sum_{k=1}^{K'} \sum_{(i,j)\in A(P'_k)} c_{ij} f_k + \underline{\lambda} \sum_{k=1}^{K'} \sum_{(i,j)\in A(P'_k)} f'_k \\
&= \sum_{k=1}^{K'} \sum_{(i,j)\in A(P'_k)} c_{ij} f_k + \underline{\lambda}(L-1)B'.
\end{aligned}$$

The first inequality is a consequence of our selection of λ'. For example setting $\lambda' = \overline{c}'$ with $\overline{c}' = \max_{(i,j)\in A(N)}\{c'_{ij}\}$ is a possibility. The last equality is a consequence of claim 5.1, since it can be shown that $\sum_{k=1}^{K'}\sum_{(i,j)\in A(P'_k)} f'_k = (L-1)B'$ similarly. Hence, we can write

$$\sum_{k=1}^{K'}\sum_{(i,j)\in A(P'_k)} c_{ij}f_k \leq C - \underline{\lambda}(L-1)B' = \lambda'(L-1)B' - \underline{\lambda}(L-1)B' = (L-1)B'(\lambda' - \underline{\lambda}),$$

which becomes

$$\sum_{k=1}^{K'}\sum_{(i,j)\in A(P'_k)} c_{ij}f_k \leq D$$

after setting

$$\underline{\lambda} = \lambda' - \left\lfloor \frac{D}{(L-1)B'} \right\rfloor .$$

For example for $\lambda' = \overline{c}'$ and $\overline{\lambda} = BD$ it is possible to set

$$\underline{\lambda} = \overline{c}' - \left\lfloor \frac{D}{(L-1)B'} \right\rfloor ,$$

provided that

$$\overline{c}' \leq BD + \left\lfloor \frac{D}{(L-1)B'} \right\rfloor$$

in order to have $\overline{\lambda} \geq \underline{\lambda}$, which makes $C - \underline{\lambda}(L-1)B' \leq D$. Also for $\overline{c}' = \lceil \alpha\overline{c} + \overline{\lambda} \rceil$ with $\overline{c} = \max_{(i,j)\in A(N)}\{c_{ij}\}$,

$$\alpha = \frac{1}{(L-1)B'},$$

and $\overline{\lambda} = BD$ we have $\overline{c} \leq D$.

In short,

$$\sum_{(i,j)\in A(P'_k)} c_{ij} f_k \le D \qquad k = 1, 2, \ldots, K'$$

follows, since $c_{ij} \ge 0$ and $f'_k > 0$ $(i,j) \in A(P'_k)$, implying $\sum_{(i,j)\in A(P'_k)} c_{ij} f'_k \ge 0$, $k = 1, 2, \ldots, K'$. Therefore, it is possible to obtain a feasible solution $\mathbf{f}$ of MCNFP-RC using the flow paths of the noncrossing arc flow $\mathbf{f}'$ by simply decreasing the flow on one of the flow paths P'_k $k = 1, 2, \ldots, K'$, say f'_p on path P'_p by one unit and keeping the remaining ones the same, i.e. by setting $f_p = f'_p - 1$, $f_k = f'_k$ for $k \ne p$. Finally, this transformation can be done in $\sum_{l=1}^{L-1}(n_l n_{l+1}) + \sum_{l=1}^{L} n_l = O(|V(N)|^2 + |V(N)|)$, which is polynomial and the proof is complete. □

The next two propositions follow directly form Proposition 5.1 and Proposition 5.2.

Proposition 5.3. *MCNFP-RC is NP-complete for general demand supply/demand, i.e. $b_i \in \mathbb{Z}_+$ for every vertex $i \in V(N)$ satisfying $\sum_{i\in V^+(N)} b_i = \sum_{i\in V^-(N)} b_i$.*

Proof. In Proposition 5.1 we have shown that a restriction of MCNFP-RC is NP-complete. It is obtained by setting

$$b_{i_l} = \begin{cases} B, & \text{if } l = 1 \text{ (i.e. } i_1 = s) \\ -B, & \text{if } l = L \text{ (i.e. } i_L = t) \\ 0, & \text{otherwise,} \end{cases}$$

with $B \in \mathbb{Z}_+$. □

Proposition 5.4. *MCNFP is NP-complete for general demand supply/demand, i.e. $b'_i \in \mathbb{Z}_+$ for every vertex $i \in V(N)$ satisfying $\sum_{i\in V^+(N)} b'_i = \sum_{i\in V^-(N)} b'_i$.*

Proof. In Proposition 5.2 we have shown that a restriction of MCNFP is NP-complete. It is obtained by setting

$$b'_{i_l} = \begin{cases} B', & \text{if } l = 1 \text{ (i.e. } i_1 = s) \\ -B', & \text{if } l = L \text{ (i.e. } i_L = t) \\ 0, & \text{otherwise,} \end{cases}$$

with $B' \in \mathbb{Z}_+$. □

5.2. The Difficulty of the Minimum Cost Flow Problem with Conflicts[6]

This problem is not defined previously in the literature to the best of our knowledge but a close relative, the maximum flow problem with conflicts (MFPC) is introduced in [16] which includes complexity results on MFPC. The authors prove that MFPC is strongly *NP*-hard and polynomially inapproximable even for networks where the number of incoming arcs i.e. in-degree and the number of outgoing arcs i.e. out-degree of each vertex is bounded by two. We exploit these theoretical results to carry out the complexity analysis for MCFPC.

The same relation between the ordinary MCFP and the maximum flow problem (MFP) exists also between MCFPC and MFPC and this situation helps assess the complexity of MCFPC. We first define their associated decision problems.

MCFPC Instance: A network $N = (V(N), A(N))$ consisting of n vertices $V(N)$ and m arcs $A(N)$, and a list of conflicting arc pairs. Every vertex $i \in V(N)$ has a supply/demand $b_i \in \mathbb{Z}$ satisfying $\sum_{i \in V^+(N)} b_i = -\sum_{i \in V^-(N)} b_i$. For each arc $(i, j) \in A(N)$ there is a unit flow cost $c_{ij} \in \mathbb{Z}_+$, finite upper bound, i.e. capacity, $u_{ij} \in \mathbb{Z}_+$, and lower bound $l_{ij} = 0$.

[6]An earlier version of this section appears in [13] as a part of its content.

Question: Is there a conflict-free flow with total cost less than or equal to D for $D \in \mathbb{Z}_+$ given?

MFPC Instance: A network $N = (V(N), A(N))$ consisting of n vertices $V(N)$ and m arcs $A(N)$, and a list of conflicting arc pairs. There is no supply/demand, i.e. $b_i = 0$, for every vertex $i \in V(N)$. There is a finite capacity $u_{ij} \in \mathbb{Z}_+$ for each arc $(i, j) \in A(N)$.

Question: Is there a conflict-free flow with value larger than or equal to F for $F \in \mathbb{Z}_+$ given?

Proposition 5.5. *MCFPC is NP-complete for*

$$b_i = \begin{cases} B, & \text{if } i = s \\ -B, & \text{if } i = t \\ 0, & \text{otherwise,} \end{cases}$$

with $B \in \mathbb{Z}_+$.

Proof. i. MCFPC $\in NP$: The feasibility of a given flow can be checked in polynomial time in the worst case. First of all, it takes $O(|A(N)|)$ to check the feasibility of the flow balance equalities in the worst case. Besides, it takes at most $O(|A(N)|^2)$ to find out whether a conflict exists and $O(|A(N)|)$ to determine whether the total flow cost is not less than D. Therefore, there is a polynomial time certificate checking algorithm with a $O(|A(N)|^2)$ worst case time complexity and thus MCFPC $\in NP$.

ii. MCFPC is hard (MFPC is a restriction of MCFPC): It is possible to see that the question "Is there a conflict-free flow with value larger than or equal to F?" of the MFPC's decision question is equivalent to the MCFPC's, "Is there a conflict-free flow with total cost less than or equal to D?", after setting $D = 0$ and $B = F$. As a consequence, MFPC is a special class of MCFPC, which makes MCFPC strongly NP-hard since MFPC is strongly NP-hard as shown in [16]. □

We can also provide an even stronger complexity result for MCFPC using the definition of an approximation algorithm given in [83]. According to this work, given an optimization problem Π, an algorithm $\mathcal{A}$ is an approximation algorithm for Π if, for any given feasible instance, it returns an approximate solution, that is a feasible solution of Π.

Proposition 5.6. *There cannot exist any polynomial time approximation algorithm for the particular instances of MCFPC satisfying the supply/demand and cost conditions of Proposition 5.5 for networks having no vertices with in-degree and out-degree exceeding* 2 *and no circuits.*

Proof. Since even checking the existence of a conflict-free flow with value B is NP-complete for networks with the given characteristics [16], no polynomial time approximation algorithm can exist for MCFPC. □

5.3. The difficulty of the Maximum Flow Problem with Conflicts and the Assignment Problem with Conflicts

A detailed analysis about the difficulty of MFPC is provided in [16]. The ordinary MFP can be easily solved on a network consisting of disjoint s-t paths by summing up the minimum flow capacities on each path. Based on this simple network, MFPC is said to have a feasible solution if a conflict-free s-t path can be found on a given network N. Since this is equivalent to solving the problem of *path with forbidden pairs of edges* which is known to be NP-complete [20], MFPC is proved to be NP-hard, also. Moreover, there is no polynomial time approximation algorithm due to the fact that checking the existence of a feasible solution cannot be done in polynomial time. Another reduction from the maximum weight stable set problem (MWSP) is done by showing that a stable set on the conflict graph C with total weight W is equivalent to a conflict-free flow of W units. This reduction allows to explore a polynomially solvable instance of MFPC which is a network consisting of s-t disjoint paths of length at most two. The corresponding conflict graph of this network contains only disjoint paths and cycles and MWSP can be solved by dynamic programming on this class of graphs.

APC is proved to be strongly NP-complete in [17] by a reduction from a special subclass of 3-satisfiability (3-SAT) problem where each clause has size three and each literal occurs exactly twice. Satisfiability (SAT) problem is the first-known NP-complete problem which tries to find if it is possible to return a *true* answer by replacing a boolean formula with consistent true-false values [84]. Given an instance of 3-SAT problem with k clauses and n variables, finding whether the problem is satisfiable is equivalent to finding a conflict-free matching with weight $k + 2n$.

5.4. A Polynomially Solvable Special Case for the Minimum Cost Noncrossing Flow Problem[7]

Let us assume that the network $N = (V(N), A(N))$ is not only layered but also complete (i.e. all arcs between the vertices of layers l and $l + 1$ exist) and the unit costs are nonnegative, symmetric and additive for $i \neq j$. Namely,

$$c_{ij} \geq 0, \tag{5.2}$$

$$c_{ij} = c_{ji}, \tag{5.3}$$

$$c_{ij} = \sum_{k=i}^{j-1} c_{k(k+1)}. \tag{5.4}$$

Notice that (5.4) is valid if $i < j$. Otherwise we can interchange the limits of the summation and apply (5.3) as a consequence of symmetry.

Recall that for a pair of crossing arcs (i_1, j_1) and (i_2, j_2), $i_1 < i_2$ and $j_2 < j_1$. Besides, there are six possible orderings of these four vertices according to the convention we use for numbering the vertices (i.e. vertex labels denote their orders from bottom in their layers):

i. $i_1 < i_2 \leq j_2 < j_1$ **iii.** $i_1 \leq j_2 < j_1 \leq i_2$ **v.** $j_2 \leq i_1 < i_2 \leq j_1$

ii. $j_2 < j_1 \leq i_1 < i_2$ **iv.** $j_2 \leq i_1 < j_1 \leq i_2$ **vi.** $i_1 \leq j_2 < i_2 \leq j_1$.

[7]An earlier version of this section appears in [12] as a part of its content.

These six cases are illustrated in Figure 5.2 with six snapshots from two consecutive layers of a layered network. Horizontal lines represent the inequalities of the orderings. Strict inequalities are reflected with additional nodes below or underneath of the tail/head of the crossing arcs. Solid arcs represent the crossings, whereas dashed ones represent their compatible equivalents. Observe that, if the flow conservation is satisfied and there is one unit of flow on each one of the crossing (solid) arcs before the correction, there must be one unit of flow on the new (dashed) arcs and zero unit of flow on the crossing arcs in order to correct the crossing and guarantee flow balance equations at the same time. The next lemma shows such change does not increase total flow cost.

Proposition 5.7. *The unit correction cost is nonincreasing under assumptions* (5.2) – (5.4) *of the unit flow costs.*

Proof. We will evaluate the cost of one unit of flow on arcs (i_1, j_1) and (i_2, j_2) (i.e. $f_{i_1j_1} = f_{i_2j_2} = 1$ and $f_{i_2j_1} = f_{i_1j_2} = 0$) with the cost of one unit of flow on arcs (i_2, j_1) and (i_1, j_2) (i.e. $f_{i_1j_1} = f_{i_2j_2} = 0$ and $f_{i_2j_1} = f_{i_1j_2} = 1$), namely $c_{i_1j_1} + c_{i_2j_2}$ with $c_{i_2j_1} + c_{i_1j_2}$ for the six possible crossings.

$$\begin{array}{lllcl}
\textbf{i.} & i_1 < i_2 \le j_2 < j_1\text{:} & c_{i_1j_1} + c_{i_2j_2} & = & c_{i_1i_2} + c_{i_2j_2} + c_{j_2j_1} + c_{i_2j_2} \\
 & & & = & c_{i_2j_1} + c_{i_1j_2} \\
\textbf{ii.} & j_2 < j_1 \le i_1 < i_2\text{:} & c_{i_1j_1} + c_{i_2j_2} & = & c_{i_2i_1} + c_{i_1j_1} + c_{j_1j_2} + c_{i_1j_1} \\
 & & & = & c_{i_2j_1} + c_{i_1j_2} \\
\textbf{iii.} & i_1 \le j_2 < j_1 \le i_2\text{:} & c_{i_1j_1} + c_{i_2j_2} & = & c_{i_1j_2} + c_{j_2j_1} + c_{i_2j_1} + c_{j_1j_2} \\
 & & & = & c_{i_2j_1} + c_{i_1j_2} + 2c_{j_1j_2} \\
\textbf{iv.} & j_2 \le i_1 < j_1 \le i_2 : & c_{i_1j_1} + c_{i_2j_2} & = & c_{i_1j_2} + c_{j_2j_1} + c_{i_2j_1} + c_{j_1j_2} \\
 & & & = & c_{i_2j_1} + c_{i_1j_2} + 2c_{j_1j_2} \\
\textbf{v.} & j_2 \le i_1 < i_2 \le j_1 : & c_{i_1j_1} + c_{i_2j_2} & = & c_{i_1i_2} + c_{i_2j_1} + c_{i_2i_1} + c_{i_1j_2} \\
 & & & = & c_{i_2j_1} + c_{i_1j_2} + 2c_{i_1i_2} \\
\textbf{vi.} & i_1 \le j_2 < i_2 \le j_1\text{:} & c_{i_1j_1} + c_{i_2j_2} & = & c_{i_1j_2} + c_{j_2i_2} + c_{i_2j_1} + c_{i_2j_2} \\
 & & & = & c_{i_2j_1} + c_{i_1j_2} + 2c_{i_2j_2}.
\end{array}$$

□

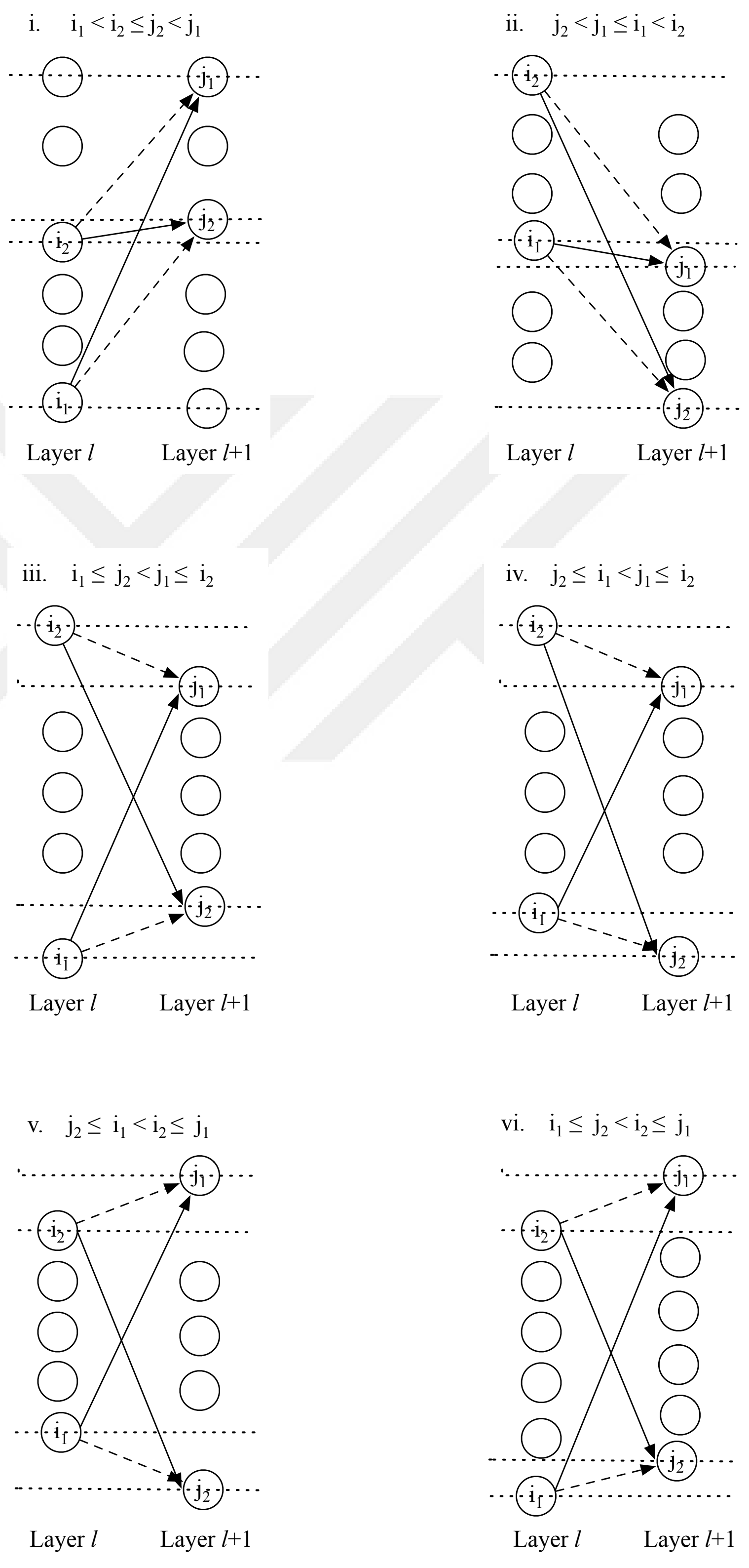

i. $i_1 < i_2 \le j_2 < j_1$
ii. $j_2 < j_1 \le i_1 < i_2$
iii. $i_1 \le j_2 < j_1 \le i_2$
iv. $j_2 \le i_1 < j_1 \le i_2$
v. $j_2 \le i_1 < i_2 \le j_1$
vi. $i_1 \le j_2 < i_2 \le j_1$
i_1
i_2
j_1
j_2
Layer l
Layer l+1

Figure 5.2. Six possible crossings

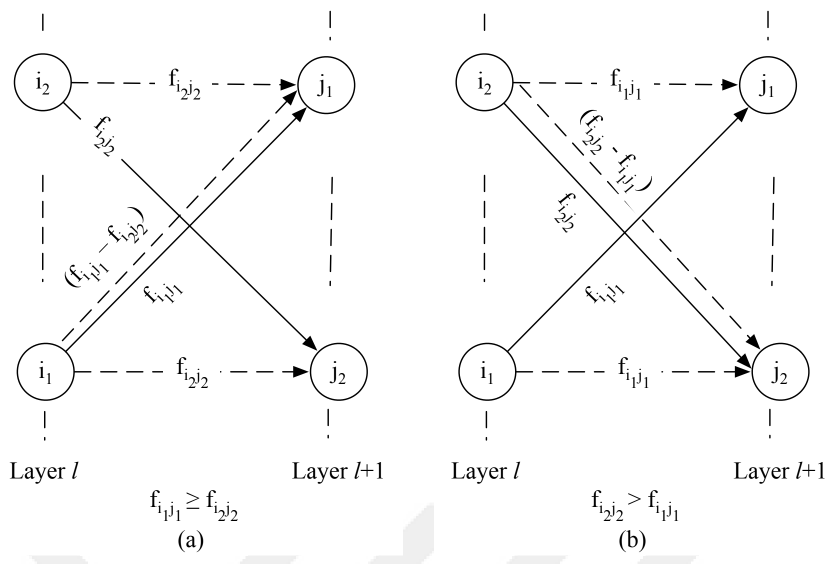
Then, as a consequence of Proposition 5.7, it is possible to show that correcting the crossings in an optimal alternate solution of MCFP results in an optimal noncrossing flow.

Proposition 5.8. *If the unit costs satisfy assumptions (5.2) – (5.4), then the MCFP has an optimal solution with no crossing arcs with positive flows.*

Proof. Consider an optimal solution $\mathbf{f}^*$ of the MCFP and crossing arcs (i_1, j_1) and (i_2, j_2), which means $f^*_{i_1,j_1} > 0$ and $f^*_{i_2,j_2} > 0$, and either $i_1 < i_2$ and $j_2 < j_1$ or $i_2 < i_1$ and $j_1 < j_2$. Without loss of generality we can assume that $i_1 < i_2$ and $j_2 < j_1$. It is possible to correct the crossing by a simple operation and adjust the flows on the corresponding arcs without harming its feasibility. If $f^*_{i_1j_1} \geq f^*_{i_2j_2} > 0$, then add new arcs (i_2, j_1) and (i_1, j_2) with flows $f^*_{i_2j_2}$, adjust the flow on arc (i_1, j_2) by subtracting $f^*_{i_1j_1}$, and finally delete arc (i_2, j_1). However, if $f^*_{i_2j_2} > f^*_{i_1j_1}$, then operate similarly by adding new arcs (i_2, j_1) and (i_1, j_2) with flows $f^*_{i_1,j_1}$, adjust the flow on arc (i_2, j_1) by subtracting $f_{i_1j_1}$, and finally delete arc (i_1, j_2).

These operations are illustrated in Figure 5.3. The crossing represented by solid arcs is corrected by replacing them with dashed arcs. Observe that flow balance is preserved at vertices i_1, i_2, j_1, j_2. Consequently, only cases (i) and (ii) or cases (iv)−−(vi) respectively with $c_{j_1j_2} = 0$, $c_{i_1i_2} = 0$ and $c_{i_2j_2} = 0$ can occur in an optimal solution of the MCFP, since otherwise it is possible to create a new feasible flow with one fewer crossing and smaller total flow cost after implementing the above operations, which contradicts the optimality of flow $\mathbf{f}^*$. Therefore, the elimination of the crossings in an optimal solution of the MCFP results in an alternative optimal solution with no crossings. □

Notice that Proposition 5.8 has an implicit assumption as well: the two correction operations are implementable, which may not be possible if (i_1, j_1) or (i_2, j_2) are missing in the network, and/or there is not enough residual capacity on both of them. However, in case the complete layered network is uncapacitated (i.e. $u_{ij} = \infty$, $(i, j) \in A(N)$) they can be applied to correct the crossings.



i_2
j_1
i_1
j_2
$f_{i_2j_2}$
$(f_{i_1j_1} - f_{i_2j_2})$
$f_{i_1j_1}$
Layer l
Layer l+1
$f_{i_1j_1} \geq f_{i_2j_2}$
(a)
$f_{i_1j_1}$
$(f_{i_2j_2} - f_{i_1j_1})$
$f_{i_2j_2}$
$f_{i_2j_2} > f_{i_1j_1}$
(b)

Figure 5.3. Two possible corrections

Propositon 5.7 and Proposition 5.8 have also some implications when arcs have finite capacities. This is stated with the following two corollaries.

Corollary 5.1. *For positive (i.e. $c_{ij} > 0$, $(i,j) \in A(N)$), symmetric and additive costs, and finite upper bounds, if an optimal solution of the MCFP has crossings of one of the types (iv)—–(vi), then $f^*_{i_2j_1} = u_{i_2j_1}$ and $f^*_{i_1j_2} = u_{i_1j_2}$.*

Proof. Assume that an optimal solution $\mathbf{f}^*$ of the MCFP has a crossing consisting of arcs (i_1, j_1) and (i_2, j_2). As a consequence of the positivity assumption of unit costs and unit cost comparisons given in Proposition 5.7, $c_{i_1j_1} + c_{i_2j_2} > c_{i_1j_2} + c_{i_2j_1}$, and as a consequence of correction operations given in Proposition 5.8, the new flow is still feasible since this operation conserves flow balance at vertices i_1, i_2, j_1, j_2 and has smaller total cost, which contradicts the optimality of $\mathbf{f}^*$. Hence, this operation must have been blocked, which is possible only if $f^*_{i_2j_1} = u_{i_2j_1}$ and $f^*_{i_1j_2} = u_{i_1j_2}$. □

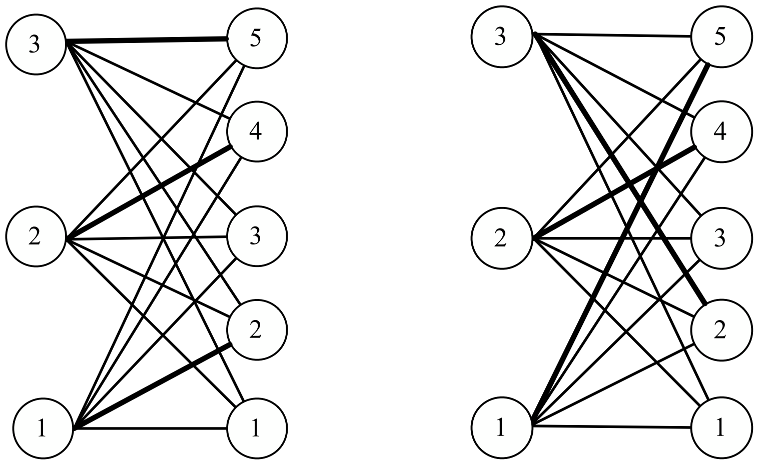
Corollary 5.2. *For positive (i.e. $c_{ij} > 0$, $(i,j) \in A(N)$), symmetric and additive costs, and finite upper bounds the crossing of arcs (i_1, j_1) and (i_2, j_2) can be corrected by one of the two operations given in Proposition 5.8 if $f^*_{i_2j_2} < \min\{u_{i_1j_2}, u_{i_2j_1}\}$ for $f^*_{i_1j_1} \geq f^*_{i_2j_2}$ or if $f^*_{i_1j_1} < \min\{u_{i_1j_2}, u_{i_2j_1}\}$ for $f^*_{i_2j_2} > f^*_{i_1j_1}$.*

Proof. Directly follows from the definition of the correction operations. □

5.5. A Polynomially Solvable Special Case: Bipartite Noncrossing Matching Problem[8]

We again consider the matching problem on a bipartite graph $G = (V_1(G) \cup V_2(G), E(G))$ with edges representing possible assignments; but there is a minor difference: $2 \leq |V_1(G)| \leq |V_2(G)|$ and the bipartite graph is complete, i.e. $E(G) = V_1(G) \times V_2(G)$. Hence, the size of a maximum cardinality matching can be at most $|V_1(G)|$, which means a maximum (cardinality) matching is $V_1(G)$-*perfect.* The main differences are in the conflict and cost structures.

A conflict is particularly defined on a crossing for a particular embedding of the vertices: given that the vertices in $V_1(G)$ and $V_2(G)$ are numbered as $1, 2, \ldots, |V_1(G)|$ and $1, 2, \ldots, |V_2(G)|$ from bottom to top, two edges $\{i_1, j_1\}$, $\{i_2, j_2\}$ are in conflict if $i_1 < i_2$ and $j_2 < j_1$, or $i_2 < i_1$ and $j_1 < j_2$. Hence, a $V_1(G)$-perfect matching is feasible if there is no such pair of the edges in the matching. This is illustrated with Figure 5.4. The first matching, which is shown with bold lines, does not include any conflict and thus it is feasible. However, the second one is infeasible since it includes at least one conflicting edge pair such as the edges $\{2, 4\}$ and $\{3, 2\}$.



(a) feasible (b) infeasible

Figure 5.4. Matchings

[8]An earlier version of this section appears in [15] as a part of its content.

Unit costs are nonnegative, symmetric and additive for $i \neq j$. Namely,

$$c_{ij} \geq 0 \tag{5.5}$$

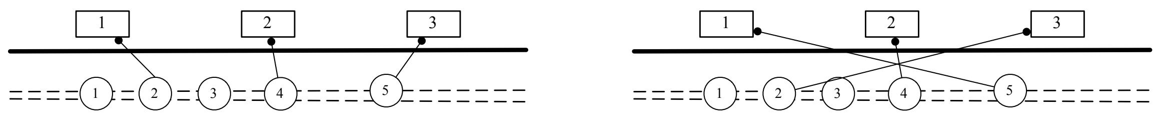
$$c_{ij} = c_{ji} \tag{5.6}$$

$$c_{ij} = \sum_{k=i}^{j-1} c_{k(k+1)}. \tag{5.7}$$

Notice that (5.7) is valid if $i < j$. Otherwise, we can interchange the limits of the summation and apply (5.6) as a consequence of symmetry.

There can be more than one feasible conflict-free $V_1(G)$-perfect matching. Then, the problem becomes the determination of a conflict-free minimum weight $V_1(G)$-perfect matching. Clearly, the BIP formulation of the problem is almost the same as (4.19) – (4.23). The only difference is that equalities (4.21) become less than or equal to type inequalities. Now, a feasible ordinary matching is $V_1(G)$-perfect, but not necessarily $V_2(G)$-perfect.

This special setting is quite realistic. Consider the situation where jobs are static and served by mobile servers moving on a line and assume that the number of servers is at least as the number of jobs, and machines can serve any one of the jobs. A typical example is frequently encountered in container terminals [67, 68]. Quay cranes, which are to be assigned to load/unload berthed vessels, are spatially restricted to move in line on a rail along the quay and cannot cross each others as a result. However, since they can serve any vessel and there are at least vessel many cranes, there is always a feasible solution, namely an assignment of the cranes to vessels. We illustrate the situation with Figure 5.5. There are three berthed vessels and five quay cranes, which are respectively represented by rectangles and circles. They are numbered starting from the beginning of the quay in order. Crane assignment of Figure 5.5(a) is feasible. However, the one of Figure 5.5(b) is infeasible. Crane 2 has to pass by cranes 3, 4, and 5 to stand alongside vessel 3. Similarly, crane 5 has to pass cranes 4, 3, and 2 to reach vessel 1.



(a) feasible (b) infeasible

Figure 5.5. Crane assignments

The problem can be represented as a bipartite graph $G = (V_1(G) \cup V_2(G), E(G))$, where $2 \leq |V_1(G)| \leq |V_2(G)|$ with edges representing possible assignments. For example $\{i, j\} \in E(G)$ exists if vessel i can be served by crane j with a nonnegative assignment (i.e. set-up) cost c_{ij}. When we adopt the numbering convention used in Figure 5.4(a) and Figure 5.4(b), namely jobs and servers are labelled from left to right starting from the beginning of the line, a feasible crane assignment becomes the determination of the subset of edges with spatial restrictions: no edge pair $\{i_1, j_1\}$, $\{i_2, j_2\}$ can be in a feasible solution if $i_1 < i_2$ and $j_2 < j_1$ or $i_1 > i_2$ and $j_1 < j_2$. Feasible and infeasible crane assignments become respectively the feasible and infeasible matchings of Figure 5.4(a) and Figure 5.4(b). A conflict is explicitly defined and has the described special property, and the edges of the conflict graph have this property with the mentioned vertex numbering convention. Moreover, it is very common to assume that the assignment cost is proportional with the travel distance, which makes the equivalent redefinition of the assignment cost c_{ij} as the cost of moving from the location of crane i to the location of vessel j, and thus the satisfaction of assumptions (5.5) – (5.7) becomes possible, since cranes and vessels (servers and jobs) can be assumed aligned on the same line.

Recall that for a pair of conflicting edges $\{i_1, j_1\}$ and $\{i_2, j_2\}$, $i_1 < i_2$ and $j_2 < j_1$. Besides, there are six possible orderings of these four vertices according to the convention we use for numbering (i.e. particular embedding of the vertices):

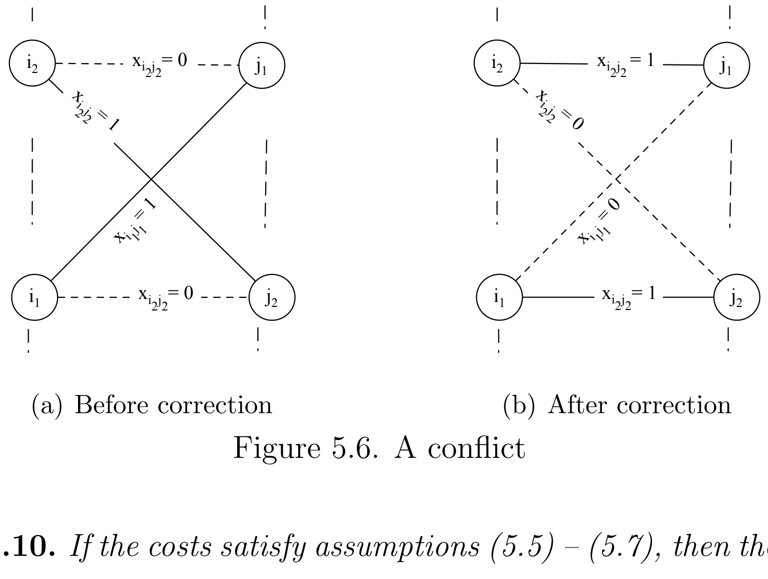
i.	$i_1 < i_2 \leq j_2 < j_1$	**iii.**	$i_1 \leq j_2 < j_1 \leq i_2$	**v.**	$j_2 \leq i_1 < i_2 \leq j_1$
ii.	$j_2 < j_1 \leq i_1 < i_2$	**iv.**	$j_2 \leq i_1 < j_1 \leq i_2$	**vi.**	$i_1 \leq j_2 < i_2 \leq j_1$.

Any one of these cases eventually represents a conflict having the form illustrated in Figure 5.6(a). Solid edges are conflicting, whereas dashed ones represent their conflict-free equivalents. Observe that, constraints (4.20) and constraints (4.21) are still satisfied after *correcting* the conflict, i.e. after replacing conflicting (solid) edges with the new (dashed) edges. Proposition 5.9 shows that such change does not increase total cost. One can also run into these six cases for the given particular conflict definition and cost assumption in the scheduling of moving resources under similar spatial constraints, such as the crane scheduling problem in container terminals [11]. Hence, it is possible to benefit from their arguments in proving that this particular case can be solved in polynomial time.

Proposition 5.9. *The correction cost is nonincreasing under the assumptions (5.5) – (5.7).*

Proof. It is a direct conclusion of Proposition 5.7 because the assigned arcs can be interpreted as the arcs carrying unit flow in a four layered network. Actually, the bipartite noncrossing matching problem can be treated as a special version of MCNFP where the number of layers is only four and all flow upper bounds are equal to 1. □

Then, as a consequence of Proposition 5.9, it is possible to show that correcting the conflicts in an optimal solution of the ordinary bipartite matching relaxation results in an optimal conflict-free $V_1(G)$-perfect matching, i.e. it is possible to solve the problem in polynomial time.



(a) Before correction (b) After correction

Figure 5.6. A conflict

Proposition 5.10. *If the costs satisfy assumptions (5.5) – (5.7), then the $V_1(G)$-perfect matching problem has an optimal conflict-free solution.*

Proof. Consider an optimal solution $\mathbf{x}^*$ of the ordinary bipartite matching relaxation and conflicting edges $\{i_1, j_1\}$ and $\{i_2, j_2\}$, which means $x^*_{i_1,j_1} = 1$ and $x^*_{i_2,j_2} = 1$, and either $i_1 < i_2$ and $j_2 < j_1$ or $i_2 < i_1$ and $j_1 < j_2$. Without loss of generality we can assume that $i_1 < i_2$ and $j_2 < j_1$. It is possible to correct a conflict by a simple operation. Replace the edges $\{i_1, j_1\}$ and $\{i_2, j_2\}$ of the matching with the new edges $\{i_2, j_1\}$ and $\{i_1, j_2\}$ (i.e. set $x^*_{i_1 j_1} = x^*_{i_2 j_2} = 0$ and $x^*_{i_2 j_1} = x^*_{i_1 j_2} = 1$).

This operation is illustrated with Figure 5.6. The conflict represented by solid edges in Figure 5.6(a) is corrected by replacing them with the dashed ones to obtain the conflict-free matching of Figure 5.6(b). Observe that the feasibility of the matching is preserved at vertices i_1, i_2, j_1, j_2. Consequently, only cases (i) and (ii) or cases (iii) – (vi) when with $c_{j_1 j_2} = 0$, $c_{i_1 j_1} = 0$, $c_{i_1 i_2} = 0$ and $c_{i_2 j_2} = 0$ can occur in an optimal solution of the relaxation, since otherwise it is possible to create a new $V_1(G)$-perfect matching with one fewer conflict and smaller total cost after implementing the correction operation of Figure 5.6, which contradicts the optimality of matching $\mathbf{x}^*$. Therefore, the elimination of the conflicts in an optimal solution of the ordinary bipartite matching problem results in an alternative optimal conflict-free matching. Besides, this can be done in polynomial time in the number of edges. □

Notice that Proposition 5.10 has an implicit assumption as well: two correction operations can be implemented, which may not be possible if $\{i_1, j_1\}$ or $\{i_2, j_2\}$ are missing in the graph. However, this is not possible in our case since we assume that the bipartite graph G is complete.

6. PRE-OPTIMIZATION PROCEDURES

In this chapter several pre-optimization procedures which can be applied prior to any exact solution method, are described. Particularly, decreasing the size of the problem in advance is of high importance to enhance the performance of the subsequent exact solution algorithm. For this purpose, efficient and cheap preprocessing strategies are developed for MCNFP and MCFPC, and a probing scheme is provided for MCFPC and APC. They delete the nonpromising arcs which cause infeasibility with respect to the conflict constraints in case they are selected in a feasible solution. Due to the special structure of MFPC which ensures that the problem produces always a feasible solution which is equal to zero flow, the developed preprocessing and probing techniques are not applicable for this problem. Also, heuristics are developed to find an initial feasible solution of MCFPC and APC in order to obtain a "good enough" bound for the optimal objective value of the considered problems.

6.1. Reducing the Number of Crossings on Layered Networks[9]

Now, consider the MCNFP formulation. An optimal solution of the MCFP relaxation defined by (4.2) – (4.5), which is obtained after dropping compatibility constraints (4.6) and (4.7), and replacing $u_{ij}x_{ij}$ with u_{ij} in (4.8), can have crossings. The efficiency of any exact solution algorithm can be improved if some of the potential crossings can be detected and deleted in advance. The following proposition and its corollary provide a tool in this direction.

Proposition 6.1. *An arc $(p, q) \in A_{l(l+1)}(N)$ is crossed by an arc*

i. *$(r, s) \in A_{l(l+1)}(N)$ with $r > p$ and $s < q$ in an optimal solution $\mathbf{f}^*$ of the MCFP if*

$$-\sum_{\{j \in V_l(N): j \leq p\}} \sum_{\{(j,i) \in A_{l(l+1)}(N): i<q\}} f^*_{ji} < \sum_{\{i \in V^-_{l+1}(N): i<q\}} b_i$$

[9]An earlier version of this section appears in [12] as a part of its content.

ii. *$(r,s) \in A_{l(l+1)}(N)$ with $r < p$ and $s > q$ in an optimal solution $\mathbf{f}^*$ of the MCFP if*

$$-\sum_{\{j\in V_l(N):j\geq p\}} \sum_{\{(j,i)\in A_{l(l+1)}(N):i>q\}} f_{ji}^* < \sum_{\{i\in V_{l+1}^-(N):i>q\}} b_i$$

Proof. We only show part (i), since the proof of part (ii) is similar. Consider the flow balance equation of the vertices of layer $l+1$ with demand b_i (i.e. the set $V_{l+1}^-(N)$) and add them side by side for vertices $i < q$ to obtain

$$\sum_{\{i\in V_{l+1}^-(N):i<q\}} b_i = \sum_{\{i\in V_{l+1}^-(N):i<q\}} \sum_{(i,j)\in A_{l+1l+2}(N)} f_{ij}^* - \sum_{i\in V_{l+1}^-(N):i<q} \sum_{(j,i)\in A_{l(l+1)}(N)} f_{ji}^*.$$

The second summation on the right hand side can be split into two for arcs $(j,i) \in A_{l(l+1)}(N)$ respectively for $j > p$ and $j \leq p$ which results in

$$\sum_{\{i\in V_{l+1}^-(N):i<q\}} b_i = \sum_{\{i\in V_{l+1}^-(N):i<q\}} \sum_{(i,j)\in A_{l+1l+2}(N)} f_{ij}^* - \sum_{\{i\in V_{l+1}^-(N):i<q\}} \sum_{\{(j,i)\in A_{l(l+1)}(N):j\leq p\}} f_{ji}^*$$

$$-\sum_{\{i\in V_{l+1}^-(N):i<q\}} \sum_{\{(j,i)\in A_{l(l+1)}(N):j>p\}} f_{ji}^*.$$

Notice that the first two terms on the right hand side represent the difference between the total outflow from the demand vertices of layer $l+1$ which are below vertex q, and the total inflow to the same vertices from the vertices of layer l which are below vertex

p including p as well. Then,

$$\begin{aligned}
\sum_{\{i\in V^-_{l+1}(N):i<q\}} b_i &\geq -\sum_{\{i\in V^-_{l+1}(N):i<q\}} \sum_{\{(j,i)\in A_{l(l+1)}(N):j\leq p\}} f^*_{ji} \\
&\quad - \sum_{\{i\in V^-_{l+1}(N):i<q\}} \sum_{\{(j,i)\in A_{l(l+1)}(N):j>p\}} f^*_{ji} \\
&\geq -\sum_{\{j\in V_l(N):i<q\}} \sum_{\{(j,i)\in A_{l(l+1)}(N):j\leq p\}} f^*_{ji} \\
&\quad - \sum_{\{i\in V^-_{l+1}(N):i<q\}} \sum_{\{(j,i)\in A_{l(l+1)}(N):j>p\}} f^*_{ji}
\end{aligned}$$

follows since $f^*_{ij} \geq 0$ for all $(i,j) \in A_{l+1l+2}(N)$, and

$$\sum_{\{j\in V_l(N):i<q\}} \sum_{\{(j,i)\in A_{l(l+1)}(N):j\leq p\}} f^*_{ji} \geq \sum_{\{i\in V^-_{l+1}(N):i<q\}} \sum_{\{(j,i)\in A_{l(l+1)}(N):j\leq p\}} f^*_{ji}.$$

Therefore, if the condition of the assertion is true, then

$$\begin{aligned}
0 &> \sum_{\{i\in V^-_{l+1}(N):i<q\}} b_i + \sum_{\{j\in V_l(N):i<q\}} \sum_{\{(j,i)\in A_{l(l+1)}(N):j\leq p\}} f^*_{ji} \\
&\geq -\sum_{\{i\in V^-_{l+1}(N):i<q\}} \sum_{\{(j,i)\in A_{l(l+1)}(N):j>p\}} f^*_{ji}
\end{aligned}$$

and

$$0 < \sum_{\{i\in V^-_{l+1}(N):i<q\}} \sum_{\{(j,i)\in A_{l(l+1)}(N):j>p\}} f^*_{ji}$$

follows consequently. Hence, there must exist an arc $(r,s) \in \{(j,i) \in A_{l(l+1)}(N) : j > p, i < q\}$ with $f_{rs} > 0$. □

Corollary 6.1. *An arc $(p,q) \in A_{l(l+1)}(N)$ with positive flow cannot exist in an optimal solution of MCNFP if one of the following conditions holds.*

i.

$$-\sum_{\{j\in V_l(N):j\leq p\}}\ \sum_{\{(j,i)\in A_{l(l+1)}(N):i<q,i\in V^-_{l+1}(N)\}} \min\{u_{ji},-b_i\} < \sum_{\{i\in V^-_{l+1}(N):i<q\}} b_i$$

ii.

$$-\sum_{\{j\in V_l(N):j\geq p\}}\ \sum_{\{(j,i)\in A_{l(l+1)}(N):i>q,i\in V^-_{l+1}(N)\}} \min\{u_{ji},-b_i\} < \sum_{\{i\in V^-_{l+1}(N):i>q\}} b_i$$

Proof. Directly follows from Proposition 6.1 as a consequence of the fact that $0 \leq f^*_{ji} \leq \min\{u_{ji}, -b_i\}$ for $(j,i) \in A_{l(l+1)}(N)$, $i \in V^-_{l+1}(N)$. □

First of all, notice that this rule is related to the satisfaction of the total demand of a subset of vertices in $V_{l+1}(N)$. Besides, although it provides a sufficient condition for an arc to be crossed, all possible crossings cannot be prevented in an optimal solution of the MCFP relaxation by the condition described in Corollary 6.1. Nevertheless, it can reduce the number of crossings by deleting arcs in the network. An arc $(p,q) \in A_{l(l+1)}(N)$ is crossed by another arc $(i,j) \in A_{l(l+1)}(N)$ with $p < i \leq n_l$ and $1 \leq j < q \leq n_{l+1}$ if the total demand associated with vertices $1 \leq j < q$ cannot be satisfied by the total inflow to them from vertices $1 \leq i \leq p$ as stated in case (i) of Corollary 6.1. Case (ii) deals with the situation that (p,q) is crossed by an arc (i,j) with $1 \leq i < p$ and $1 \leq j < q \leq n_{l+1}$.

It is also possible to state supply versions of Proposition 6.1 and Corollary 6.1. We give their statement in the following without proof for the sake of completeness, since their proofs are very similar and can be done by rewording the arguments for the supplies instead of the demands.

Proposition 6.2. *An arc $(p,q) \in A_{l(l+1)}(N)$ is crossed by an arc*

i. *$(r,s) \in A_{l(l+1)}(N)$ with $r > p$ and $s < q$ in an optimal solution $\mathbf{f}^*$ of the MCFP if*

$$\sum_{\{i \in V_{l+1}(N): i \geq q\}} \sum_{\{(j,i) \in A_{l(l+1)}(N): j > p\}} f^*_{ji} < \sum_{\{j \in V_l^+(N): j > p\}} b_j$$

ii. *$(r,s) \in A_{l(l+1)}(N)$ with $r < p$ and $s > q$ in an optimal solution $\mathbf{f}^*$ of the MCFP if*

$$\sum_{\{i \in V_{l+1}(N): i \leq q\}} \sum_{\{(j,i) \in A_{l(l+1)}(N): k < p\}} f^*_{ji} < \sum_{\{j \in V_l^+(N): j < p\}} b_j$$

Corollary 6.2. *An arc $(p,q) \in A_{l(l+1)}(N)$ with positive flow cannot exist in an optimal solution of MCNFP if one of the following conditions holds.*

i.

$$\sum_{\{i \in V_{l+1}(N): i \geq q\}} \sum_{\{(j,i) \in A_{l(l+1)}(N): j > p, j \in V_l^+(N)\}} \min\{u_{ji}, b_j\} < \sum_{\{j \in V_l^+(N): j > p\}} b_j$$

ii.

$$\sum_{\{i \in V_{l+1}(N): i \leq q\}} \sum_{\{(j,i) \in A_{l(l+1)}(N): j < p, j \in V_l^+(N)\}} \min\{u_{ji}, b_j\} < \sum_{\{j \in V_l^+(N): j < p\}} b_j$$

This time, this rule is related to the satisfaction of the total supply of a subset of vertices in $V_l(N)$. An arc $(p,q) \in A_{l(l+1)}(N)$ is crossed by another arc $(i,j) \in A_{l(l+1)}(N)$ with $1 \leq p < i \leq n_l$ and $1 \leq j < q \leq n_{l+1}$ if the total outflow from vertices $1 \leq p < i \leq n_l$ to vertices $1 \leq q \leq j \leq n_{l+1}$ as stated as case (i) in Corollary 6.2. Case (ii) deals with the situation that (p,q) is crossed by an arc (i,j) with $1 \leq i < p \leq n_l$ and $1 \leq q < j \leq n_{l+1}$.

As a result, the efficiency of an exact solution algorithm may increase because the network size can be reduced considerably using these rules. The preprocessing that Corollary 6.1 suggests can be stated formally as Algorithm 6.1 given in Figure 6.1. The process Corollary 6.2 suggests is very similar and the corresponding algorithm is not included.

```
**Algorithm 6.1: Preprocessing for MCNFP**
**Input:** A layered network $N = (V(N), A(N))$, arc capacities $\mathbf{u}$ and unit flow costs
 $\mathbf{c}$;
**Output:** A preprocessed network;
begin
 for $l = 2, 3, \ldots, L-2$ do
    for all arc $(p, q) \in A_{l(l+1)}(N)$ such that $p \leq n_l - 1$, $q \geq 2$ do
       $D = 0$, $C = 0$;
       for $j = 1, 2, \ldots, q-1$ do
          if $b_j < 0$, then $D \leftarrow D + b_j$;
          for all arc $(i, j) \in A_{l(l+1)}(N)$ do
             if $i \leq p$, then $C \leftarrow C + \min\{u_{ij}, -b_j\}$
          end for
       end for
    end for
    if $C < -D$, then Delete $(p, q)$;
    else $D = 0$, then $C = 0$;
    for $j = q+1, q+2, \ldots, n_{l+1}$ do
       if $b_j < 0$, then $D \leftarrow D + b_j$;
       for all arc $(i, j) \in A_{l(l+1)}(N)$ do
          if $i \geq p$, then $C \leftarrow C + \min\{u_{ij}, -b_j\}$;
       end for
    end for
    if $C < -D$, then Delete $(p, q)$;
 end for
end
```

Figure 6.1. Preprocessing for MCNFP.

6.2. Preprocessing on General Networks[10]

The aim is to detect the arcs which are impossible to carry positive flow in any feasible solution of MCFPC and delete them from the network permanently at the beginning. Deleting arcs through preprocessing is previously utilized in Section 6.1 by exploiting the layered structure of the network. Here, a generalized scheme which is applicable for any type of network is presented.

To determine whether an arc $(i, j) \in A(N)$ can be deleted or not, we begin by assuming a positive flow along arc (i, j). Then, the arcs in the conflict list of (i, j) are forced to carry zero flow to obey the *conflict restrictions.* We need to check two conditions for each arc $(p, q) \in \delta_C(i, j)$:

i. If vertex p is a supply vertex, then the total capacities of the outgoing arcs besides the elements of $\delta_C(i, j)$ must be larger than or equal to the supply of p, i.e.

$$b_p \leq \sum_{(p,v)\in A(N)\setminus\delta_C(i,j)} u_{pv}.$$

ii. If vertex q is a demand vertex, then the total capacities of the incoming arcs besides the elements of $\delta_C(i, j)$ must be larger than or equal to the demand of q, i.e.

$$-b_q \leq \sum_{(w,q)\in A(N)\setminus\delta_C(i,j)} u_{wq}.$$

If any one of these two conditions are violated, then we have to send additional amount of flow that conflicts with the current flow on arc (i, j) to satisfy flow balance. So, we conclude that arc (i, j) cannot carry positive flow on a MCFPC solution and this arc is deleted permanently. This procedure is repeated for all arcs in the network and each arc is either deleted or left as it is. Preprocessing steps are formally listed as Algorithm 6.2 given in Figure 6.2.

[10]An earlier version of this section appears in [13, 15] as a part of their contents.

```
Algorithm 6.2: Preprocessing for MCFPC
Input:Network $N = (V(N), A(N))$;
Output: The reduced arc set $A(N)$;
begin
 OUTER LOOP:
 for all arc $(i, j) \in A(N)$ do
   INNER LOOP:
   for all arc $(p, q) \in \delta_C(i, j)$ do
     sum = 0;
     if $b_p > 0$ then
       for all arc $(p, v) \in A(N)$ do
         if $(p, v) \notin \delta_C(i, j)$, then $sum \leftarrow sum + u_{pv}$;
       end for
       if $b_p >$ sum, then delete arc $(i, j)$ and break;
     end if
     if $b_q < 0$ then
       for all arc $(w, q) \in A(N)$ do
         if $(u, q) \notin \delta_C(i, j)$, then $sum \leftarrow sum + u_{wq}$;
       end for
       if $-b_q >$ sum, then delete arc $(i, j)$ and break;
     end if
   end for
   END OF INNER LOOP
 end for
 END OF OUTER LOOP
end
```

Figure 6.2. Preprocessing for MCFPC.

6.3. Probing

We begin with the probing scheme developed for MCFPC. First, the flow value of an arc (i, j) is forced to a nonzero value and all conflicting arcs are cleared up by setting their flow capacities to zero. If the relaxed problem ignoring the *conflict restrictions* is infeasible under these conditions, then (i, j) cannot carry positive flow in a feasible solution and we delete this arc permanently. This procedure is repeated for all arcs in the network. Probing requires to solve $|A(N)|$ MCFP relaxations. It makes possible deleting more arcs at the expense of more computation time when compared to preprocessing. Clearly, integer flow is assumed here, which is guaranteed for integer capacities. A step-by-step description of probing scheme is given as Algorithm 6.3 in Figure 6.3.

We can implement the same probing procedure for APC after necessary adjustments. Briefly, an edge is selected and forced to be in a conflict-free assignment by deleting all its conflicting edges from the graph. Then, if the relaxed problem turns out to be infeasible then the selected edge cannot appear in any feasible assignment and thus we can delete it permanently from the graph. This procedure is repeated once for every edge at the beginning of any algorithm.

6.4. Finding Initial Solutions

Diving heuristics and local search base heuristics are proposed to find an initial solution for MCFPC and APC, respectively. These solutions provide finite upper bounds to the optimal objective value of these problems and obtaining a good initial upper bound given by the objective value of a feasible solution is essential to the performance of the subsequent exact solution method.

6.4.1. Diving Heuristics

We can use a simple diving heuristic to attain a feasible solution of MCFPC as fast as possible. First, the MCFP relaxation allowing conflicts is solved and one of the

```
Algorithm 6.3: Probing for MCFPC
Input:Network $N = (V(N), A(N))$;
Output: The reduced arc set $A(N)$;
begin
 for all arc $(i,j) \in A(N)$ do
    create a copy of flow capacities, $u_{ij}^{original} = u_{ij}$ for all $(i,j) \in A(N)$;
    set $l_{ij} = 1$;
    set $u_{pq} = 0$ for all $(p,q) \in \delta_C(i,j)$;
    solve MCFP relaxation;
    if MCFP is infeasible then
       delete arc $(i,j)$;
    else
       set $l_{ij} = 0$;
    end if
    set $u_{pq} = u_{pq}^{original}$ for all $(p,q) \in \delta_C(i,j)$;
 end for
end
```

Figure 6.3. Probing for MCFPC.

conflicting flows is set to zero and the MCFP relaxation is solved again. All conflicting flows are cleared in this manner. If there is no conflicting pair, a feasible solution is found. If the problem turns out to be infeasible or the relaxation produces a higher lower bound than that of the incumbent solution, the last arc whose flow is forced to be zero, is forced to have positive flow. Then we delete the arcs conflicting with it.

Essentially, we are doing a depth-first search on a BB tree to find a feasible solution fast. Notice that it provides the optimal solution if all nodes of the tree are processed. We need three variables to backtrack in the tree: k denotes the depth of the tree, a_k keeps the arc that is used to branch at depth k and binary flag d_k which is set to 0 if a_k have 0 flow and to 1 if the arcs in its conflict list are forced to have zero flow. Algorithm 6.4 gives a detailed description of these steps in Figure 6.4.

The success of the heuristic relies on how arc a_k is selected. The flow with the highest unit cost or the one that conflicts with the highest number of arcs perform better than arbitrary selection. We hope to find a small lower bound by deleting the arc with the highest unit cost, and a conflict-free solution in the smallest possible number of steps by deleting the most infeasible arc with the highest number of conflicts with respect to the relaxed MCFP solution. The first rule favors the quality of the bound while the second one intends to find any feasible solution with a smaller number of branching operations. However, we cannot claim that the second rule always performs faster than the first rule because counting the number of conflicts for each flow requires additional time. This heuristic can be stopped when the first or r^{th} conflict-free solution is found, where r is a parameter. Alternatively, it can return the best conflict-free feasible flow, if found, within a time limit. In our experiments, we apply both arc selection rules separately and stop the heuristic when the first conflict-free flow is obtained. The smallest of the returned bounds is selected as the initial upper bound.

6.4.2. Local Search Based Heuristic

An initial feasible solution of APC is found through a straightforward perturbation based local search procedure. First of all, an initial perfect matching is constructed

```
Algorithm 6.4: Diving heuristic
Input: Network $N = (V(N), A(N))$;
Output: An upper bound, $\bar{z}$, for MCFPC;
begin
 1. Initialize: Create a copy of flow capacities, $u_{ij}^{original} = u_{ij}$ for all $(i,j) \in A(N)$,
    $k = 0$, $\bar{z} = \infty$;
 2. Solve MCFP relaxation;
    if infeasible, then go to step 4;
    else Let $\mathbf{f}$ and $\underline{z}$ be respectively an optimal solution and its value;
    if $\underline{z} \geq \overline{z}$, then go to step 4;
    else go to step 3;
 3. Branch:
    if $\mathbf{f}$ is conflict-free, then go to step 5;
    else find an arc $(i,j)$ whose flow violates conflict constraints (i.e. $f_{ij}, f_{pq} > 0$
    for some $(p,q) \in \delta_C(i,j)$);
    Set $k \leftarrow k+1$, $a_k = (i,j)$, $u_{ij} = 0$, $d_k = 0$, and go to step 2;
 4. Prune:
    if $k = 0$, then go to step 6;
    else if $d_k = 0$, then $u_{a_k} = u_{a_k}^{original}$, $l_{a_k} = 1$, $u_{pq} = 0$ for all $(p,q) \in \delta_C(a_k)$
    where $d_k = 1$, and go to step 2;
    else $l_{a_k} = 0$, $u_{pq} = u_{pq}^{original}$ for all $(p,q) \in \delta_C(a_k) \setminus \bigcup_{i \in J} \delta_C(a_i)$ where
    $J = \{i : i \in \{1, 2, \ldots, k-1\}$ and $d(i) = 1\}$;
    Set $k \leftarrow k-1$, go to step 2;
 5. Update upper bound: $\bar{z} = \underline{z}$, go to step 4;
 6. Return $\overline{z}$.
end
```

Figure 6.4. Diving heuristic for computing an initial upper bound on MCFPC.

with the Hungarian algorithm. Then a local search is performed by exchanging conflicting edges with their compatible neighbors. When it is not possible to obtain any improvement, the current solution is randomly permuted and again the local search procedure is run. This process is repeated for a number of times and the best feasible solution (i.e. the feasible solution with the smallest upper bound) is accepted as the initial solution and its objective is used as the initial upper bound.

7. BENDERS DECOMPOSITION ALGORITHM

In this chapter[11] , a mature Benders decomposition (BD) algorithm for solving MFPC is described. BD is a well-known approach based on the separation of a complicated formulation into a master problem and a subproblem [85]. The algorithm consists of generating constraints for the master problem as needed by using the output of the subproblem which in turn uses the solution of the master problem as its input. The iterations continue until all the necessary cuts are added to reach the optimum of the original formulation. It has been applied to a wide variety of optimization problems related to planning, scheduling, routing, etc. especially in the last two decades. A recent survey summarizes both theoretical and practical approaches utilized to accelerate BD [86]. Briefly, the authors discuss the impact of the choice of the mathematical model, the decomposition strategy, the solution procedure to solve the master problem and the subproblem, and the quality of the generated cuts. An earlier study, [87], compiles the BD applications in detail for fixed-charge network design problems.

In MFPC formulations, $\boldsymbol{x}$ variables can be considered as the variables which complicate the problem because once they are fixed, the formulation becomes an LP problem and it can be solved by any LP solver. As a matter of fact, it is the well-known MFP, and there exist polynomial-time algorithms to solve it [6]. Due to this special structure, we can decompose MFPC$_S$, the strong MFPC formulation, into a master problem (MP$_S$) which contains only $\boldsymbol{x}$ variables and a subproblem (SP) including $\boldsymbol{f}$ and v variables. The master problem for MFPC$_S$ can be written as

$$\text{MP}_S\text{:}\quad \max \quad 0 + v^* \tag{7.1}$$

$$\text{s.t.} \quad x_{ij} + x_{kl} \leq 1 \qquad (k,l) \in \delta_C(i,j);\ (i,j) \in A(N) \tag{7.2}$$

$$x_{ij} = 0,1 \qquad (i,j) \in A(N). \tag{7.3}$$

[11]An earlier version of this chapter appears in [14] as a part of its content.

The first term of (7.1), which is zero, is the term depending on $\boldsymbol{x}$ in the original objective function (4.14) and the second term v^* is the optimal objective value of the following SP that utilizes $\bar{\boldsymbol{x}}$ values which are the output of MP$_S$:

$$\begin{aligned}
\text{SP}(\bar{\boldsymbol{x}}):\ \max \quad & v && && (7.4)\\
\text{s.t.} \quad & \sum_{(i,j)\in A(N)} f_{ij} - \sum_{(j,i)\in A(N)} f_{ji} = \begin{cases} v & i = s \\ 0 & i \in V(N)\backslash\{s,t\} \\ -v & i = t \end{cases} && && (7.5)\\
& f_{ij} \le u_{ij}\bar{x}_{ij} && (i,j)\in A(N) && (7.6)\\
& f_{ij} \ge 0 && (i,j)\in A(N). && (7.7)
\end{aligned}$$

Notice that SP($\bar{\boldsymbol{x}}$) has always a finite optimum if all arc capacities are finite. Also, it is always feasible because $\boldsymbol{f} = \mathbf{0}$ is the trivial solution for the problem. Hence, the dual problem of SP($\bar{\boldsymbol{x}}$) is always feasible and has a finite optimum. If we assign dual variables $\boldsymbol{\pi}$ to constraint set (7.5), and $\boldsymbol{\mu}$ to constraint set (7.6), the dual formulation becomes

$$\begin{aligned}
\text{DSP}(\bar{\boldsymbol{x}}):\ \min \quad & \sum_{(i,j)\in A(N)} \mu_{ij}u_{ij}\bar{x}_{ij} && && (7.8)\\
\text{s.t.} \quad & \pi_i - \pi_j + \mu_{ij} \ge 0 && (i,j)\in A(N) && (7.9)\\
& \pi_t - \pi_s = 1 && && (7.10)\\
& \pi_i \text{ unr }, \mu_{ij} \ge 0 && i \in V(N), (i,j)\in A(N). && (7.11)
\end{aligned}$$

By weak duality, the objective value of an optimal solution of DSP($\bar{\boldsymbol{x}}$) gives an upper bound on the optimal objective value of SP($\bar{\boldsymbol{x}}$) implying that

$$v^* \ \le \sum_{(i,j)\in A(N)} \mu^*_{ij}u_{ij}x_{ij}. \tag{7.12}$$

At this point, we should remind that DSP($\bar{\boldsymbol{x}}$) is the minimum cut formulation and the minimum cut formulation always yields an integer optimal solution under integrality assumption of the capacities u_{ij} although it is formulated as an LP problem. In fact, the arcs in the minimum cut, i.e. the arcs oriented from the connected component containing s to the one with t have μ^*_{ij} values equal to one and the remaining ones have zero values. In other words, the optimal dual multipliers with value one form the minimum cut $\mathcal{MC}$, i.e. $\mu^*_{ij} = 1$ for $(i,j) \in \mathcal{MC}$ where $\boldsymbol{\mu}^*$ is an optimal solution. As a result, inequality (7.12) can be rewritten as

$$v^* \leq \sum_{(i,j)\in\mathcal{MC}} u_{ij}x_{ij}. \tag{7.13}$$

Given the optimal dual multipliers μ^*_{ij}, $(i,j) \in A(N)$, inequality (7.12) represents a Benders optimality cut, which is added to the MP$_S$'s constraints in the next iteration. Observe that we do not generate any Benders feasibility cut because the dual problem is always feasible and finite, as stated before. According to the classical iterative approach, MP$_S$ is resolved after the addition of every Benders cut. Since we know that MP$_S$ and SP($\bar{\boldsymbol{x}}$) produce upper and lower bounds, respectively, the algorithm is terminated when the optimal objective values produced by the two problems are equal.

Although a general BD framework is described to solve MFPC$_S$, we observed that it can be improved from the algorithmic and implementation aspects within the context of our problem. In the following sections, we present the procedures designed to obtain stronger Benders cuts and to find efficient connectivity cuts. Moreover, we explain how to produce valid inequalities and an initial upper bound for the master problem. Finally, the impact of the choice of the mathematical model and the implementation of BD as a single branch-and-cut tree are discussed.

7.1. Strong Connectivity Cuts

Previously, we have mentioned that DSP($\bar{\boldsymbol{x}}$) is always feasible, and thus produces only Benders optimality cuts. This is not surprising as MFPC$_S$ itself always produces a feasible flow (i.e. zero flow is a trivial feasible solution). However, when the selected arcs (with $\bar{x}_{ij} = 1$) do not provide a directed path from s to t, it means that there is a cut which is also a minimum cut separating the source and the sink, and producing zero flow. In this situation, we need to make sure that at least one of the arcs in this minimum cut $\mathcal{MC}$ is selected in the optimal solution by adding the *connectivity cut*

$$\sum_{(i,j)\in\mathcal{MC}} x_{ij} \geq 1. \tag{7.14}$$

The idea behind the cuts of type (7.14) is the fact that zero flow is very unlikely to be the maximum. So, we can accelerate the algorithm by avoiding zero flows as much as possible with the help of connectivity cuts. We should note that they are previously utilized [88] to ensure feasibility. Besides, they can be considered as the stronger forms of combinatorial Benders cuts

$$\sum_{(i,j):\bar{x}_{ij}=0} x_{ij} + \sum_{(i,j):\bar{x}_{ij}=1} (1 - x_{ij}) \geq 1. \tag{7.15}$$

which are proposed in [89].

Since we are not obliged but free to assign positive f_{ij} value if $\bar{x}_{ij} = 1$, it is to our advantage to increase the number of selected arcs. That is exactly why the second term of the left-hand side of the combinatorial cut (7.15) is not needed for our problem. Also, the variables on the left-hand side of (7.14) form a subset of the variables in the first term of the left hand side of (7.15) implying that constraint (7.14) is stronger than constraint (7.15).

Furthermore, when the s-t connection cannot be constructed for given $\bar{x}_{ij},\ (i,j) \in A(N)$, there may exist multiple cuts that disconnect the source and the sink. Among them, the one containing the smallest number of arcs provides the strongest connectivity cut. In order to find this cut, we make use of the following proposition which is valid for the ordinary minimum cut problem on a connected network.

Proposition 7.1. *Let us define new arc capacities $u'_{ij} = mu_{ij} + 1$, where $m = |A(N)|$. The minimum cut with respect to the capacities u'_{ij} is the minimum cut with respect to the capacities u_{ij} containing the minimum number of arcs.*

Proof. First, we will show that the minimum cut with u'_{ij} is also a minimum cut with capacities u_{ij}. Now, consider the objective function to be minimized with the modified capacities, which is $\sum_{(i,j)\in A(N)} \mu_{ij} u'_{ij}$. When u'_{ij} values are replaced with $mu_{ij} + 1$, it becomes

$$m \sum_{(i,j)\in A(N)} \mu_{ij} u_{ij} + \sum_{(i,j)\in A(N)} \mu_{ij}. \tag{7.16}$$

As the minimum cut has to contain at least one arc, and $u_{ij} \geq 1\ (i,j) \in A(N)$, it follows that $\sum_{(i,j)\in A(N)} \mu_{ij} u_{ij} \geq 1$. Thus, the minimum value that the first term of (7.16) can take is m. Moreover, the maximum number of arcs in a cut cannot exceed the total number of arcs. Then, the second term of (7.16) can be at most m. Therefore, the optimum is determined by the first term which is a multiple of the objective function with respect to the original capacities. So, we conclude that the resulting minimum cut is optimal with respect to both u_{ij} and u'_{ij}. If there exist multiple minimum cuts providing the same value for the first term, the second term of (7.16) breaks the tie, and favors the one containing the minimum number of arcs. $\square$

The difference of DSP($\bar{\boldsymbol{x}}$) from the classical minimum cut formulation stems from the existence of $\bar{x}_{ij}$ parameters. Then, it is possible to reach a similar conclusion for DSP($\bar{\boldsymbol{x}}$).

Proposition 7.2. *The minimum cut, i.e. an optimal solution of DSP($\bar{\boldsymbol{x}}$), with respect to the newly defined arc capacities $u'_{ij}\bar{x}_{ij}$ where $u'_{ij}\bar{x}_{ij} = mu_{ij}\bar{x}_{ij} + 1$ is the minimum cut with respect to the capacities $u_{ij}\bar{x}_{ij}$ containing the minimum number of arcs.*

Proof. Directly follows from the proof of Proposition 7.1. □

When the subproblem produces zero flow, a strong connectivity cut is produced in addition to the Benders cut. Although two optimization problems must be solved in this case, two cuts are added in one step, which can increase the efficiency of the method.

7.2. Strong Benders Cuts

As a result of the maximum flow-minimum cut duality, DSP obtained for fixed $\bar{\boldsymbol{x}}$ is the minimum cut problem, which is a member of highly degenerate network optimization problems. When DSP($\bar{\boldsymbol{x}}$) provides multiple optima, alternative Benders optimality cuts exist, as expected. In this case, employing a strategy that selects the strongest Benders cut among them improves the performance of the algorithm [88]. A formal definition of the strength of a cut in a given optimization problem $\min_{\boldsymbol{y}\in Y,\ z\in R}\{z : z \geq f(\boldsymbol{u}) + \boldsymbol{y}g(\boldsymbol{u})\ \boldsymbol{u} \in U\}$ is provided in [90]: the cut $z \geq f(\boldsymbol{u_1}) + \boldsymbol{y}g(\boldsymbol{u_1})$ is *stronger* than (or *dominates*) the cut $z \geq f(\boldsymbol{u_2}) + \boldsymbol{y}g(\boldsymbol{u_2})$ if $z \geq f(\boldsymbol{u_1}) + \boldsymbol{y}g(\boldsymbol{u_1}) \geq f(\boldsymbol{u_2}) + \boldsymbol{y}g(\boldsymbol{u_2})\ \boldsymbol{y} \in Y$, satisfying the inequality strictly for at least one $\boldsymbol{y} \in Y$. The cut that dominates all the other cuts is called *Pareto-optimal.* Since finding the Pareto-optimal cut involves solving an optimization problem and finding a point in the relative interior of Y [88], there is a trade-off between the strength of the generated cut and the time spent to find it. In other words, it is worthwhile as long as the overall saved time exceeds the devoted time in order to produce strong cuts. For example, [90] and [88] develop procedures to find pareto-optimal cuts efficiently by exploiting the special structures of the dual subproblems. On the other hand, the dual problems are solved using a two-phase approach to strengthen the obtained Benders cuts [91, 92].

For our problem, we adopt the line of reasoning proposed in [92] in order to strengthen Benders cuts. When we consider the objective function (7.8) of DSP($\bar{\boldsymbol{x}}$) , it can be observed that arcs $(i,j) \in A(N)$ with $\bar{x}_{ij} = 0$ have no contribution to the objective value regardless of the values of μ_{ij}. Our aim is to find the alternative optimum that assigns as many zero values as possible to these μ_{ij} variables in order to obtain a strengthened cut of type (7.12). Due to the fact that the right-hand side of inequality (7.12) provides an upper bound on the maximum flow v^*, this strategy allows us to insert better bounds to the MP$_S$ formulation. For this purpose, upon determining the optimal objective value of DSP($\bar{\boldsymbol{x}}$) (or equivalently SP($\bar{\boldsymbol{x}}$)) denoted by $z(\bar{\boldsymbol{x}})$, the following optimization problem called the modified dual subproblem (mDSP($\bar{\boldsymbol{x}}$)), is solved.

$$\begin{aligned} \text{mDSP}(\bar{\boldsymbol{x}})\text{:}\ \ \min \quad & \sum_{(i,j)\in A(N)} \mu_{ij} u_{ij} && (7.17)\\ \text{s.t.} \quad & \sum_{(i,j)\in A(N)} \mu_{ij} u_{ij} \bar{x}_{ij} = z(\bar{\boldsymbol{x}}) && (7.18)\\ & \text{constraints (7.9), (7.10), and (7.11).} \end{aligned}$$

Although we expect to produce a strengthened Benders optimality cut, it requires to solve two optimization problems, namely DSP($\bar{\boldsymbol{x}}$) and mDSP($\bar{\boldsymbol{x}}$). According to the objective function (7.17), the strong Benders cut is equivalent to the minimum cut with respect to given $u_{ij}\bar{x}_{ij}$ values where its total flow capacity with respect to u_{ij} is the smallest among the multiple minimum cuts. Instead of solving two optimization problem, the strong Benders cut can be generated by changing the parameters as described in the next proposition.

Proposition 7.3. *Let us replace $\bar{x}_{ij}$ with $\bar{x}'_{ij} = U\bar{x}_{ij} + 1$ where $U = \sum_{(i,j)\in A(N)} u_{ij}$. Then the optimal solution of DSP($\bar{\boldsymbol{x}}'$) is equivalent to the optimal solution of mDSP($\bar{\boldsymbol{x}}$).*

Proof. Consider the objective function to be minimized with the modified parameters, which is $\sum_{(i,j)\in A(N)} \mu_{ij} u_{ij} \bar{x}'_{ij}$. When $\bar{x}'_{ij}$ values are replaced with $U\bar{x}_{ij} + 1$, it becomes

$$U \sum_{(i,j)\in A(N)} \mu_{ij} u_{ij} \bar{x}_{ij} + \sum_{(i,j)\in A(N)} \mu_{ij} u_{ij}. \tag{7.19}$$

For any cut, the first term of (7.19) is a multiple of U, which is the largest value that the second term of (7.19) can ever take. Hence, the first term determines the minimum cut ensuring that the optimal solution of DSP($\bar{\boldsymbol{x}}'$) is also optimal for DSP($\bar{\boldsymbol{x}}$). In other words, an optimal solution of DSP($\bar{\boldsymbol{x}}'$) satisfies constraint (7.18). If there exist multiple feasible solutions providing the same value for the first term, the second term of (7.19), which is equal to objective function (7.17) of mDSP($\bar{\boldsymbol{x}}$), favors the alternative optimal solution having the minimum total cut capacity with respect to original flow capacities u_{ij}. $\square$

Upon solving DSP($\bar{\boldsymbol{x}}'$), the attained optimal dual values μ^*_{ij}, $(i,j) \in A(N)$ are used to generate Benders optimality cuts of type (7.12). According to the experimental results presented in Section 6, the overall efficiency of the algorithm is improved by means of the strengthened cuts.

7.3. Valid Inequalities and Initial Upper Bound

In branch-and-cut, a finite upper bound increases the probability of pruning the nodes of the branch-and-bound tree and the overall efficiency could be significantly improved. To that end, we use the optimal objective value of MFP relaxation obtained by omitting the conflict constraints and $\boldsymbol{x}$, as the initial upper bound for v^*.

Besides, the initial form of MP$_S$ neither forces $\boldsymbol{x}$ values to be positive nor includes a constraint that connects v^* to the rest of the formulation. Under these conditions, it is certain that an *s*-*t* disconnected subnetwork is produced. In order to increase the number of selected arcs, we can add valid inequalities. For example, there exists a minimum cut corresponding to the solution of the MFP relaxation. We derive one valid inequality of type (7.13) and one of type (7.14) from this cut, and add them to the initial master problem.

Moreover, it is possible to anticipate some of the cuts that will be added in the future and add them to the formulation at the beginning. We apply a breadth first search that begins from the source s, continues by labeling the vertices $V(N)$, and finally stops when the sink t is labeled. Note that the vertex label represents the length (in the number of arcs) of the shortest (directed) path from s to that vertex, and the ones with the same label form a layer. If we end up with l layers, then we have $l-1$ arc sets that separate these layers. Thus, we add two valid inequalities for each arc set, one in the form of Benders optimality cut and the other in the form of connectivity cut, as described before.

Notice that the addition of valid inequalities does not ensure producing s-t connected subnetworks but it carries us a few steps ahead with much less effort. These inequalities will probably be added to MP$_S$ in the forthcoming iterations so this approach is quite useful to shorten the solution time.

7.4. Benders Decomposition for the Other Formulations

In Section 4, MFPC is modeled using weak, strong, and clique formulations. The weak one is obtained by aggregating the constraints of the strong formulation for each arc, and the clique formulation is built using the maximal cliques of the conflict graph. Since the difference of the models originates from the form of conflict constraints, i.e. the constraints involving $\boldsymbol{x}$ variables, the same BD approach can be applied to MFPC$_W$ after changing the master problem with

$$\begin{array}{lll} \text{MP}_W\text{: max} & 0 + v^* & \\ \quad\text{s.t.} & \displaystyle\sum_{(k,l)\in\delta_C ij} x_{kl} + |\delta_C(i,j)| x_{ij} \leq |\delta_C(i,j)| & (i,j) \in A(N) \\ & x_{ij} = 0, 1 & (i,j) \in A(N). \end{array}$$

Similarly, we can implement BD for the clique formulation MFPC$_K$ using the following master problem:

$$
\begin{aligned}
\text{MP}_K\text{:}\ \max \quad & 0 + v^* \\
\text{s.t.} \quad & \sum_{(i,j)\in K} x_{ij} \le 1 && K \in \mathcal{K} \\
& x_{ij} = 0,1 && (i,j) \in A(N).
\end{aligned}
$$

Even though the models represent the same problem, they differ in terms of BD performance. [90] report that the representation with a tighter LP relaxation performs better. Our computational experiments support the idea that the strong formulations generate stronger Benders cuts in naive BD. On the other hand, we observe that MP$_W$ can be solved faster than MP$_S$ and MP$_K$. Therefore, the entire practical BD performance on the weak formulation is much better from the computational point of view. Hence, MFPC$_W$ is employed in benchmarking instead of MFPC$_S$ and MFPC$_K$.

7.5. Implementation

In the ordinary BD, the master problem is solved optimally whenever a Benders cut is added. Undoubtedly, re-optimizing a mixed-integer programming problem in a repeated manner has high computational cost. Instead, we utilize a single branch-and-cut tree for the solution of the master problem, which allows user interruption when an integer solution $\hat{\boldsymbol{x}}$ is found. First, DSP($\hat{\boldsymbol{x}}$) is solved and using its objective function value $z(\hat{\boldsymbol{x}})$, the strong Benders optimality cut is determined. If the optimal value of DSP($\hat{\boldsymbol{x}}$) is zero, u_{ij} values are modified as described in Proposition 7.2, and a strong connectivity cut is obtained. The generated cut(s) is (are) added to the master problem, and the branch-and-cut procedure continues until all integer solutions are explored. Successful implementations of a similar approach are available in the literature within different contexts [93, 94].

The depicted cut generation procedures require the solution of at least two optimization problems for each integer solution. We can speed up the algorithm by solving MFP with an efficient algorithm in order to compute the initial $z(\hat{\boldsymbol{x}})$ value and find the strong connectivity cut. To this end, Goldberg's preflow-push algorithm [95] which is known to be the fastest MFP solver in practice is used. The numerical results confirm that the described implementation considerably improves the quality of the obtained solutions. The final version of BD, which incorporates all the described improvements, is provided more formally as Algorithm 7.1.

```
Algorithm 7.1: MFPC Benders Decomposition
Input: A network $N = (V(N), A(N))$;
Output: Optimal solution for MFPC, $\boldsymbol{f}^*$ and $v^*$;
begin
 1. Initial UB: Solve MFP relaxation on $N$ and set its optimal objective value
    as UB;
 2. Valid Inequalities: Find the minimum cut corresponding to the MFP
    relaxation solution and several other cuts produced by breadth-first search on
    $N$. Generate inequalities of type (7.13) and (7.14) for each cut and add them
    to the master problem;
 3. Solve the master problem. Whenever an integer solution $\hat{\boldsymbol{x}}$ is found,
    i.   Generate a strong Benders optimality cut according to Proposition 7.3;
    ii.  If the maximum flow of SP($\hat{\boldsymbol{x}}$) is 0, then generate a strong connectivity
         cut;
    iii. Add the generated cuts to the master problem and return to the solution
         process;
 4. Return the solution of the master problem.
end
```

Figure 7.1. MFPC Benders Decomposition.

8. BRANCH-AND-BOUND ALGORITHM

The proposed BB in this chapter[12] employs the relaxations of the given problems obtained by omitting the conflict constraints and the variables utilized to model the conflicts, namely the minimum cost flow problem (MCFP), the maximum flow problem (MFP) and the assignment problem (AP) relaxations. The bounds they give are used for pruning purposes and branching is performed only if the optimal solution of the relaxed problem contains conflicting flows in general terms. First, we outline the BB steps for the minimum cost flow problem with conflicts (MCFPC) and then indicate the differences when applied to the maximum flow problem with conflicts (MFPC) and the assignment problem with conflicts (APC).

At each node of the BB tree, MCFP relaxation that is obtained by removing constraints (4.11), replacing (4.12) with $0 \leq f_{ij} \leq u_{ij}$, $(i,j) \in A(N)$ and deleting all $\mathbf{x}$ variables is solved. If the relaxation is infeasible, the current subproblem is fathomed. Otherwise, the objective value of the relaxation gives a lower bound for a conflict-free solution. If this lower bound is not less than the best known upper bound, the current node is pruned. Otherwise, if the resulting solution does not violate constraints (4.11), i.e. there is no conflicting arc pair with positive flow on it, then a feasible solution for MCFPC is obtained and the upper bound can be updated. If none of these conditions are satisfied, the current node is divided into child nodes. The algorithm ends when all the nodes in the tree are pruned and returns the incumbent solution.

The same procedure can also be applied to APC by solving AP relaxations at nodes of the BB tree. Notice that, x_{ij}, $(i,j) \in A(N)$ remains in the relaxation. If a node is not pruned due to the infeasible solution space of the subproblem or the bound of AP relaxation, we check whether a conflict-free assignment is found checking the values of x_{ij} variables.

[12]An earlier version of this chapter appears in [13–15] as a part of their contents.

Described BB steps to solve MCFPC are also valid for MFPC when we use MFP relaxations at every node of the BB tree. Since MFPC is a maximization problem, MFP relaxations of the subproblems provide an upper bound to a conflict-free solution. If this upper bound is not larger than the best known lower bound, i.e. the objective value of the best known conflict-free feasible solution, the current node is pruned. We do not need to check the feasibility of the solution because MFP has always a feasible solution, which is zero flow.

8.1. Branching Rules for Division

The branching rule has a major impact on the performance of the algorithm. Five rules are considered for branching. The first one is based on a conflicting pair of arcs (edges), the second one on a conflicting arc (edge), the third one is a hybrid form of the first two rules, and the last two employs a clique including a conflicting arc (edge) pair.

8.1.1. Conflicting Pair Branching

At node t of the BB tree we choose a pair of conflicting arcs, say arcs (p, q) and (r, s), in an optimal solution of the corresponding minimum cost flow relaxation MCFP$^{(t)}$. We create two subproblems where arcs (p, q) and (r, s) are forced to be zero, respectively. This rule does not partition the parent's feasible solution space; because the case where both (p, q) and (r, s) carry zero flow is allowed in both child nodes. Pair branching rule prevents only one conflicting arc pair, namely arcs (p, q) and (r, s), from appearing in the descendant nodes. However, these arcs can conflict with other arcs at deeper levels and this can result in a very large BB tree. This rule can be implemented as described using MFP relaxations at node t, denoted by MFP$^{(t)}$. When we apply this rule in BB to solve APC, the branching is performed using edges instead of arcs.

8.1.2. Conflicting Arc Branching

Let (p, q) and (r, s) be two conflicting arcs with $f_{pq},\ f_{rs} > 0$ in an optimal solution of the relaxation MCFP$^{(t)}$ or MFP$^{(t)}$. Choose one of them, say arc (p, q), as the branching arc and create two subproblems based on it. In the first subproblem delete arc (p, q), and in the second problem delete all the arcs conflicting with arc (p, q) and force arc (p, q) to carry positive flow. The resulting division provides a partition and one arc at each branching is guaranteed not to conflict with any other arc deeper in the tree. In fact, it is in the same spirit as the variable branching of classical BB with LP relaxation.

This rule is referred as *conflicting edge branching* when applied to an AP relaxation at node t, denoted by AP$^{(t)}$. For branching, we choose two edges $\{p, q\}$ and $\{r, s\}$ with $x_{pq},\ x_{rs} = 1$ in an optimal solution of AP$^{(t)}$.

8.1.3. Conflicting Arc Pair Branching

This rule is a combination of conflicting pair and conflicting arc (edge) branching rules. When we detect a conflicting arc (edge) pair, we consider three cases that can occur in a feasible solution of the child nodes. In other words, one of the conflicting arcs (edges) is forced to be in a feasible solution in one branch, the other is forced to be in a feasible solution in another branch and both of them are forbidden to be in a feasible solution in the third branch. As can be seen this is not a dichotomized rule as the previous ones.

8.1.4. Clique Branching

A more interesting approach can be the consideration of the cliques of the conflict graph $C = (V(C), E(C))$. Recall that a subset of arcs in which every arc conflicts with others form a clique in the conflict graph. At any node of the BB tree, in an optimal solution of the MCFP relaxation, if we have a pair of conflicting arcs with positive flows, we can find a clique of K, preferably with maximum cardinality, including the corresponding vertices of this pair. We partition the vertices of the clique into subsets. For each subset of vertices, a child node is produced by deleting their corresponding arcs from the original network N. This rule is an adaptation of the generalized upper bound (GUB) branching of the classical BB with LP relaxation and expected to produce a more balanced BB tree [96].

Let (p, q) and (r, s) be two conflicting arcs with flows in an optimal solution of the MCFP$^{(t)}$ (or MFP$^{(t)}$) at node t of the BB tree. Determine a clique $K = (V(K), E(K))$ of the conflict graph including $(p, q), (r, s) \in V(K)$. It is better to have K be one of the cliques with the largest cardinality as mentioned. Since finding a maximum cardinality clique is itself an intractable problem, we prefer to use an efficient greedy method for producing maximal cliques. For this purpose we first let (p, q) and (r, s) be the first two elements of $V(K)$, search over the arcs in $\delta_C(p, q)$ and add one to $V(K)$ if it is also an element of $\delta_C(r, s)$. In other words, if we are given an initial vertex pair of the conflict graph C which is known to be connected by an edge and denoted as the subset U, we can employ a greedy heuristic to find a maximal clique K^* including U. A formal description of the greedy heuristic is given as Algorithm 8.1.

Here, we consider two versions of the clique branching rule: *clique*-0 and *clique*-1 branching rules. In the first clique branching rule, named as *clique*-0 branching rule, we partition the vertices of the clique K into two subsets as evenly as possible ensuring that the vertices equivalent to the initially given conflicting *flows* are in separate subsets. For each subset of vertices, a child node is produced by deleting their corresponding arcs from the original network N.

Notice that this rule includes pair branching as a special case, since a conflicting pair is a clique of size two of the conflict graphs. Besides, it becomes arc branching if the partition is made unevenly so that one of the two subsets includes only one of the conflicting arcs and the other contains the rest of the clique, namely the arcs that are in conflict with it.

Algorithm 8.1: Greedy heuristic to determine a maximal cardinality clique

```
Input: A conflict graph $C = (V(C), E(C))$ and a vertex subset $U \subseteq V(C)$ of size two to be included in the clique;
Output: A maximum cardinality clique $K^*$ including vertices of $U$;
begin
 Set $K = U$ and put in $S$ vertices in $V(C)$ that are adjacent with all vertices in $U$;
 while $S \neq 0$ do
    Choose a vertex in $S$, say $w$, with the largest degree in the induced subgraph $C[S]$ of conflict graph $C$;
    $K \longleftarrow K \cup \{w\}$;
    $S \longleftarrow S \setminus \{w\}$;
    Delete from $S$ all vertices that are not adjacent with $w$
 end while
 Set $K^* = K$ and induced subgraph $C[K^*]$ is a maximal clique including $U$
end
```

Figure 8.1. Greedy heuristic to determine a maximal cardinality clique.

In the second clique branching rule, called as *clique*-1 branching rule, we partition the vertices of the clique into more than two subsets. In fact, we add $|V(K)| + 1$ subproblems to the active node list. For each vertex u of K, a child node is created by deleting all the vertices except u and forcing the corresponding arc of u to carry positive flow. Also, one more child node is added where all vertices of K are deleted from the conflict graph. As can be noticed this is not a dichotomized branching rule.

The clique branching rules described here can be safely implemented when we find a pair of conflicting edges in an optimal assignment provided in $\text{AP}^{(t)}$.

8.2. Penalty Calculation

Penalty calculation and the resulting *strong branching* idea which anticipates the outcomes of possible branching strategies and selects the most promising one has become an essential part of the algorithms that solve integer programming problems [35, 96]. Calculated penalties are used to improve the bounds given by the subproblem relaxation and thus increasing the probability of pruning. If we are not able to prune the node, we have the possibility of fixing values of some variables. They also serve as powerful tools when determining the arcs (edges) or pairs that we branch on.

In the context of network flows with constraints, the *penalty* with respect to an arc (i, j) with $f_{ij} > 0$ is a lower bound for the amount of increase in the objective function value when the flow on arc (i, j) is forced to be zero and it is denoted by pn_{ij}. Given an optimal solution of MCFP relaxation, the deletion of arc (i, j) by imposing the constraint $u_{ij} = 0$ makes the current optimal solution infeasible and requires dual simplex iterations to recover the created infeasibility. From a theoretical perspective, pn_{ij} value corresponds to the change in the objective value after one dual simplex iteration.

8.2.1. Penalties from Basic Spanning Tree

We need the basis information of the optimal solution in order to calculate penalties. The basis is represented in the form of either matrices/tableaux or spanning trees. Since we use network simplex algorithm to solve MCFP relaxations, all the information is kept in the underlying basic spanning tree, $T = (V(T), A(T))$, of the current optimal solution. In [50], a similar penalty calculation that exploits the spanning tree structure is applied for the transportation problem with exclusionary side constraints, where some pairs of supply nodes are not allowed to send flow to a certain demand node simultaneously. The described penalty calculation method in [50] is for the uncapacitated flows. In our case, we develop an elaborate penalty calculation routine which is modified for flows with finite arc capacities.

As MCFP solver, we use the network simplex algorithm of LEMON (Library for Efficient Modeling and Optimization in Networks, [97]), which is a C++ template library offering efficient implementations for combinatorial optimization tasks with graphs and networks. Since the network simplex algorithm of LEMON works with basic spanning trees, the basic spanning tree T corresponding to the optimal solution of MCFP relaxation is kept within the source code. In the spanning tree T associated with the optimal solution, some arcs are basic, (i.e. arcs in the tree), and others are nonbasic, (i.e. arcs outside the tree). The basic arcs carry flows that are strictly between the lower and the upper bounds, $l_{ij} < f_{ij} < u_{ij}$ $(i, j) \in A(T)$, only if the basic feasible solution is non-degenerate. If the solution is degenerate, flows of some of the basic arcs might be at one of the bounds. However, the nonbasic arcs always carry flows at their either lower or upper bounds. These two subsets of the nonbasic arcs are denoted by L and U, respectively. Given the optimal spanning tree T, we can calculate pn_{ij} values for $f_{ij} > 0$ by considering two cases:

Case 1. : If $(i, j) \in A(T)$ and if we delete (i, j) by making $u_{ij} = 0$, we obtain an excess supply at node i and a demand deficit at node j by an amount of f_{ij}. When arc (i, j) is deleted from T, two subtrees $T_1 = (V(T_1), A(T_1))$ and $T_2 = (V(T_2), A(T_2))$

are obtained. Suppose that $i \in V(T_1)$ and $j \in V(T_2)$. The resulting excess flow can be either sent from T_1 to T_2 through the nonbasic arcs which are at their lower bounds or pulled back from T_1 to T_2 through the nonbasic arcs which are at their upper bounds. The product of the lowest reduced cost of all nonbasic arcs and f_{ij} gives the penalty for arc (i, j). In other words,

$$\begin{aligned} pn_{ij} = f_{ij} \times \min\{|\bar{c}_{kl}| : &(k,l) \in L,\, k \in V(T_1),\, l \in V(T_2) \text{ or} \\ &(k,l) \in U,\, k \in V(T_2),\, l \in V(T_1)\} \end{aligned} \tag{8.1}$$

where $\bar{c}_{kl} = c_{kl} - \pi_k - \pi_l$ is the reduced cost of $(k, l) \in A(N)$ and π_i, $i \in V(N)$ are the dual variables associated with constraints (4.10).

If $pn_{ij} > 0$, we stop. Zero penalty is implied by a zero reduced cost, which is a sign of degeneracy. Due to the fact that optimal MCFP solutions are often degenerate, the occurence of zero penalties is not rare. So, when $pn_{ij} = 0$, one more dual simplex iteration is carried out and pn_{ij} is recalculated.

Let $(u, v) = arg \min_{(k,l)\in L}\{\bar{c}_{kl}\}$. If $\bar{c}_{uv} = 0$, some or all of the flow f_{ij} is sent from i to j with zero cost along a unique path $P = (V(P), A(P))$ from i to j on $V(T_1) \cup V(T_2) \cup \{(u, v)\}$. The flow along P is restricted by $u_{pq} - f_{pq}$ value if the direction of $(p, q) \in P$ is towards j and constrained by $f_{pq} - l_{pq}$ value otherwise. If the minimum of these values is not less than f_{ij}, then we can send f_{ij} units of flow from i to j at a unit cost of $\bar{c}_{uv} = 0$. As a result, pn_{ij} remains zero. Otherwise, we denote the maximum allowed flow by r and send r units of flow along $P = (V(P), A(P))$ and update the flows of arcs on P. New flow on arc (i, j) is set to $f_{ij} - r$, the spanning tree T is updated by adding arc (u, v) and deleting an arc of P whose flow reaches to one of the bounds. Since the reduced cost of the entering arc is zero, reduced costs of other arcs do not change with this (pivot) operation. For the remaining flow on (i, j), pn_{ij} is recalculated as described above.

Case 2. : If $(i, j) \notin A(T)$, some of the excess supply occured at node i can be sent to node j through the unique path P on the spanning tree T. Suppose that the path P is divided into two as $P_1 = (V(P_1), A(P_1))$ and $P_2 = (V(P_2), A(P_2))$ where P_1 contains arcs directed towards node j and P_2 contains arcs directed towards node i. The flow that can be sent along P is limited by $u_{pq} - f_{pq}$ if the direction of arc (p, q) is the same as the flow's, and restricted by $f_{pq} - l_{pq}$ otherwise. The product of the minimum of these values and the reduced cost of the nonbasic arc (i, j) gives the penalty for arc (i, j). In short,

$$pn_{ij} = |\bar{c}_{ij}| \times \min\{u_{pq} - f_{pq}, (p, q) \in A(P_1) \text{ and } f_{pq} - l_{pq}, (p, q) \in A(P_2)\}.$$

After sending the flow along P, if there is still a positive flow on arc (i, j), the arc (i, j) enters the spanning tree and the blocking arc on path P leaves the spanning tree. Upon updating the spanning tree and reduced costs, we can apply the procedure described in the first case and the sum of the two penalties gives the total penalty of forcing the flow on arc (i, j) to be zero.

These operations are carried out very efficiently if the internal data structures of LEMON are used. In the network simplex algorithm of LEMON, the spanning tree is represented by XTI (Extended Threaded Index) instead of classical ATI (Augmented Threaded Index). Without using the internal data structure for penalty calculations, the burden of the required time to calculate penalties exceeds its advantages. However, XTI representation enables efficient implementation, and penalty calculation procedure becomes a powerful tool. The differences between the ATI and XTI schemes are summarized in [97].

If a subproblem of the BB tree is not pruned, penalties are calculated for the arcs whose flows conflict at least one other arc's flow. Penalty information is used when determining the lower bounds of the produced child nodes, for pruning and variable fixing. Moreover, we make use of them while selecting an arc or a pair of arc for branching purposes.

Note that we do not calculate penalties while solving MFPC as no proper cost structure is defined; unit flow costs are essential to compute reduced costs. Although it is possible to reformulate MFPC as MCFPC problem by assigning negative unit costs to arcs leaving the source and the ones entering the sink, we prefer to utilize a polynomial time algorithm to solve MFPC instead of the network simplex algorithm.

8.2.2. Penalties from the Solution of the Hungarian Algorithm

It is also possible to calculate penalties when we deal with AP relaxation. In this context, penalty pn_{ij} is defined as an estimate of the increase in the lower bound value when edge $\{i, j\}$ is removed from the current optimal assignment, which is obtained by solving the assignment relaxation of the subproblem AP$^{(t)}$. Consider a pair of conflicting edges $\{\{p, q\}, \{r, s\}\}$ of the optimal assignment. The penalty pn_{pq} of removing an edge $\{p, q\}$ is estimated as $pn_{pq} = \beta_p + \beta_q$ where $\beta_p = \min\{\overline{c}_{pi} : i = 1, \ldots, n; i \neq p\}$ and $\beta_q = \min\{\overline{c}_{iq} : i = 1, \ldots, n; i \neq q\}$. Here, $\overline{c}_{ij}$ is the reduced cost of nonbasic edge $\{i, j\}$, i.e. an edge which is not in the optimal assignment of AP relaxation. Notice that the reduced cost values $\overline{c}_{ij}$ are computed by the Hungarian algorithm which is run for solving AP relaxation of the subproblem, AP$^{(t)}$, at search node t.

8.2.3. Pruning by Penalties

Penalties are calculated at node t if the optimal solution of MCFP$^{(t)}$ or AP$^{(t)}$ relaxation contains a conflicting flow or edges and its optimal objective value is less than the best known upper bound, $\underline{z}^{(t)} < \bar{z}$. Calculated penalties allow us to estimate how far a conflict-free solution is from the solution of the relaxed problem.

Given a conflicting pair (i, j) and (p, q) where $f_{ij}, f_{pq} > 0$ in MCFP$^{(t)}$ or a conflicting edge pair $\{i, j\}$ and $\{p, q\}$ with $x_{ij} = 1$ and $x_{pq} = 1$ in APC$^{(t)}$, penalties pn_{ij} and pn_{pq} are calculated. Then, $\min\{pn_{ij}, pn_{pq}\}$ is the estimated lower bound on the cost of clearing this conflict. Hence, the lowest increase in the objective value to reach a feasible MCFPC solution, $e^{(t)}$, is equal to the maximum of these estimates for each conflicting flow pair:

$$e^{(t)} = \max\{min\{pn_{ij}, pn_{pq}\} : \; f_{ij}, f_{pq} > 0 \text{ and } (p,q) \in \delta_C(i,j)\}.$$

When APC is considered, the expression $e^{(t)}$ becomes

$$e^{(t)} = \max\{min\{pn_{ij}, pn_{pq}\} : \; x_{ij}, x_{pq} = 1 \text{ and } (p,q) \in \delta_C(i,j)\}.$$

Therefore, node t can be pruned if $\underline{z}^{(t)} + e^{(t)} \geq \overline{z}$. Clearly, larger $e^{(t)}$ values lead to smaller BB tree. If we can make sure that clearing up one conflicting pair does not affect the penalties associated with other conflicting flows, taking the summation instead of the maximum provides much better estimates. In general, clearing one conflict changes reduced costs, penalties and even removes other existing conflicts. So, special cases need further consideration.

8.2.4. Pegging

In some cases, the penalties are not sufficiently large for pruning; but they can still provide useful information for reducing problem size by variable fixing, which is also called *pegging*. At search node t, if $\underline{z}^{(t)} + e^{(t)} < \overline{z}$, all arcs are examined one by one. For an arc (i,j), if $\underline{z}^{(t)} + pn_{ij} \geq \overline{z}$, then $f_{ij} > 0$ (or $x_{ij} = 1$ for APC) is forced, and the flows of the arcs in its conflict list are forced to be zero at node t of the BB tree.

Pegging operations prevent at least one conflict. If all existing conflicting flows have already been removed by pegging, neither pruning nor branching is carried out. Still, the current subproblem t is left in the active node list to be resolved in the future with additional constraints. Otherwise, two child nodes are created with lower bounds equal to $\underline{z}^{(t)} + e^{(t)}$.

8.3. Branching Variable Selection

In BB algorithm for MCFPC and APC, the arc with the highest penalty among the unpegged arcs is selected as the branching arc while applying the *arc branching* rule. The pair whose minimum penalty is maximum is selected as the branching pair when using pair, arc pair and clique branching rules. In other words, among all conflicting pairs $\{(p,q),(r,s)\}$, we select the one, with the *maximum* of $\min\{pn_{pq}, pn_{rs}\}$. This approach is similar to the *strong branching* applied in the classical BB with LP relaxations. *Strong branching* gives priority to arcs that are expected to produce higher increase in the lower bounds and its performance turns out to be better than the other rules such as selecting the arc or arc pair with the (total) highest unit cost or (total) highest number of conflicts it involves.

For APC, in addition to *strong branching*, we also try another conflicting pair selection rule, which is the *largest number of conflicts*. Here, we consider the conflicting edge pair $\{i,j\}$ and $\{k,l\}$ with the largest number of total conflicting edges, namely with the largest $|\delta_C(i,j)| + |\delta_C(k,l)|$ value. For the conflicting edge branching rule, among all edges $\{p,q\}$ conflicting with other edges in an assignment relaxation solution at a leaf node of the BB tree, we select the one with the largest number of conflicting edges, namely, with the largest $|\delta_C(p,q)|$ value.

When MFPC is considered, two straightforward possible rules could be to select the arc with the highest or lowest flow capacity among the candidate arcs. Two more rules are to branch on the arc or arc pair with the highest infeasibility with respect to a conflict-free solution. In other words, for an arc (i,j) where $f_{ij} > 0$ in an optimal solution of relaxation MFP$^{(t)}$, the other positive flows conflicting with f_{ij} are counted. As the counted value increases, (i,j) becomes more infeasible with respect to conflict constraints. Notice that this is different from selecting the arc with the highest $|\delta_C(i,j)|$ value, since it is not solution dependent. Actually, it corresponds to the rule of the ordinary BB with LP relaxation that consists of selecting the most or least infeasible variable whose fractional value is the furthest or closest to the nearest integer number.

Although this arc selection rule is computationally more expensive, it significantly reduces the mean depth of the BB tree before pruning.

Apart from these, we can keep the average difference that each variable produces between the bounds of the nodes when it is used to branch. If we obtain better quality bounds from the relaxation, the probability to prune that node increases. Hence, the arc that gives the highest estimated difference based on the historical data is selected as the branching arc variable. The experiments show that the final variable selection rule works the best for BB that solves MFPC.

8.4. Subproblem Selection

The subproblem with the smallest lower bound and the largest upper bound are selected from the active node list to be processed in the next iteration during the implementation of BB for MCFPC and MFPC, respectively. While solving APC, we carry out experiments with four subproblem selection rules to compare the strategies: depth first search (DFS) and breadth first search (BFS) which respectively select the last inserted and the first inserted subproblems to the active node list in addition to selecting the subproblems with the smallest lower bound (SLB) and with the largest lower bound (LLB).

8.5. Algorithms to Solve Relaxed Problems

Determining how to solve the relaxations of the emerging subproblems is a critical decision which has a considerable impact on the performance of the overall algorithm. Since the obtained relaxed problems belong to the family of network flow problems, we make use of polynomial time algorithms which could be quite advantegous especially as the problem size gets larger. To this end, we utilize Hungarian algorithm to solve AP relaxations. However, variants of simplex algorithm which is known to be exponential time but performs reasonably well on practice could be preferred. The network simplex algorithm which allows computing penalties using the basic spanning trees is selected as MCFP solver.

MFP relaxations of the visited subproblems are solved very efficiently by Goldberg's preflow-push algorithm [95]. Note that this algorithm works properly for zero lower bounds on the arc flows. For this reason, the resulting division of *conflicting arc branching* rule does not provide a partition because the case with $f_{pq} = 0$ is allowed in the feasible solutions of both subproblems. If we force f_{pq} to have a positive value in the second branch by defining a lower bound equal to one, we will not be able to benefit from the available efficient maximum flow solvers. Theoretically, a partition ends up with less number of tree nodes as we avoid replicated subproblems. However, the performance of the overall algorithm depends not only on the theoretical background it is founded but also on the way it is implemented. In the latter case where we impose a positive lower bound to the arcs, we solve the subproblems with an LP solver. However, this option combined with the perfect partition cannot outperform the initially described branching strategy.

8.6. Local Preprocessing

Another way of improving the efficiency of the algorithm for MCFPC is the application of preprocessing procedure locally for each subproblem just before solving the MCFP relaxation. Through local preprocessing, it is probable to eliminate more arcs from the subproblem in question with a little computational effort. Every subproblem is created by deleting some arcs from its parent. Let MCFPC$^{(t)}$ be the subproblem considered at node t and $N^{(t)} = (V(N^{(t)}), A(N^{(t)}))$ be the related network. If it is obtained from its parent MCFPC$^{(t-1)}$ and network $N^{(t-1)} = (V(N^{(t-1)}), A(N^{(t-1)}))$, then the arc subset $A^{(t)} = A(N^{(t-1)}) \backslash A(N^{(t)}))$ represents deleted arcs. So, the supplies of tail nodes must be sent and demands of head nodes of the deleted arcs must be satisfied to obtain a feasible solution. Given the deleted arcs as the input, local preprocessing routine repeats pre-processing (i.e. inner loop of Algorithm 5.1) for the arcs that conflict with the outgoing arcs of the tail nodes and incoming arcs of the head nodes of the deleted arcs if they are supply and demand nodes, respectively. Although the local implementation of probing steps is also possible, it is not efficient to call probing at every node of the BB tree because it requires much more time than preprocessing does in return of a marginal profit. The steps of local preprocessing are described as Algorithm 8.2 in Figure 8.2.

The whole BB procedure proposed to solve MCFPC and APC is given as Algorithm 8.3 in Figure 8.3.

Since MFPC is a maximization problem and the described BB to solve it differs in terms of the called subroutines, BB for MFPC is summarized separately as Algorithm 8.4.

```
Algorithm 8.2: Local preprocessing
Input: A subproblem MCFPC$^{(t)}$, the arc set $A^{(t)}$ that exist in the parent of MCFPC$^{(t)}$ but not in MCFPC$^{(t)}$;
Output: The new reduced arc set of subproblem MCFPC$^{(t)}$;
begin
create a copy of flow capacities, $u_{ij}^{original} = u_{ij}$ for all $(i, j) \in A(N)$;
for all arc $(i, j) \in A^{(t)}$ do
  if $b_i > 0$ then
    for all arc $(k, l) \in \delta_C(i, v)\, v \in V(N)$ do
      if arc $(k, l)$ is unlabeled then
        For arc $(k, l)$, repeat inner loop of Algorithm 6.2;
      end if
      label arc $(k, l)$;
    end for
  end if
  if $b_j < 0$ then
    for all arc $(k, l) \in \delta_C(u, j)\, u \in V(N)$ do
      if arc $(k, l)$ is unlabeled then
        For arc $(k, l)$, repeat inner loop of Algorithm 6.2;
      end if
      label arc $(k, l)$;
    end for
  end if
end for
end
```

Figure 8.2. Local preprocessing.

```
Algorithm 8.3: Branch-and-bound Algorithm for MCFPC and APC
Input: A network $N = (V(N), A(N))$, initial upper bound $\overline{z}$;
Output: Optimal conflict-free flow $\mathbf{f}^*$, and its cost $z^*$;
begin
  1. Initialization: Add MCFPC (or APC) into the active subproblem list $\mathcal{L}$;
  2. Termination test: If $\mathcal{L} = \emptyset$, then go to step 7;
  3. Subproblem selection: Select and remove a subproblem from $\mathcal{L}$, apply local
     preprocessing and solve MCFP (or AP) relaxation on it. If it has a solution,
     then let $\mathbf{f}'$ (or $\mathbf{x}'$) and $z'$ be the optimal one and its objective value;
     i.  If the subproblem is infeasible or $z' \geq \overline{z}$, then go to step 2;
     ii. Else if $\mathbf{f}'$ (or $\mathbf{x}'$) is conflict-free, then set $\overline{z} = z'$, $\mathbf{f}^* = \mathbf{f}'$ (or $\mathbf{x}^* = \mathbf{x}'$) and
         go to step 2;
  4. Penalty Calculation: Calculate penalties, $pn_{ij}$. If they are sufficently large
     to prune the current subproblem, then go to step 2;
  5. Pegging:  Fix the values of variables using calculated penalties. If all
     conflicting flows are cleared up with pegging operation, then add the current
     subproblem to $\mathcal{L}$ and go to step 3;
  6. Branching: Create two subproblems and add them to $\mathcal{L}$. Go to step 3;
  7. Return $\mathbf{f}^*$ (or $\mathbf{x}^*$) and $z^*$.
end
```

Figure 8.3. Branch-and-bound Algorithm for MCFPC and APC.

```
Algorithm 8.4: Branch-and-bound Algorithm for MFPC
Input: A network $N = (V(N), A(N))$;
Output: Optimal solution for MFPC, $\boldsymbol{f}^*$ and $v^*$;
begin
  1. Initialization: Add MFPC into the active subproblem list $\mathcal{L}$, set $\underline{z} = 0$;
  2. Termination test: If $\mathcal{L} = \emptyset$, then go to step 4;
  3. Subproblem selection: Select and remove a subproblem from $\mathcal{L}$ and solve
     its MFP relaxation. Let $\boldsymbol{f'}$ and $v'$ be an optimal flow and its value, respectively;
     i.  If $v' \leq \underline{z}$, then go to step 2;
     ii. Else if $\boldsymbol{f'}$ is conflict-free, then set $\underline{z} = v'$, $\boldsymbol{f}^* = \boldsymbol{f'}$ and go to step 2;
  3. Branching: Create two subproblems and add them to $\mathcal{L}$. Go to step 3;
  4. Set $v^* = \underline{z}$. Return $\boldsymbol{f}^*$ and $v^*$
end
```

Figure 8.4. Branch-and-bound Algorithm for MFPC.

9. RUSSIAN DOLL SEARCH ALGORITHM

Russian Doll Search (RDS) is an efficient search procedure, which explores maximal stable sets of the conflict graph C. Since optimal solution of MCFPC, MFPC and APC can be found by solving their MCFP, MFP and AP relaxation on some maximal stable set on C, this method can be applied to three problems. In this chapter[13] , we describe all the steps for the most general problem, MCFPC, use the notation of $z(S)$ instead of $z^k(S)$ $k = 1, 2, 3$ and point out the problem-specific parts at the end.

RDS evaluates $z(S)$ values of the explored maximal stables sets by solving the problem given with (4.27) – (4.29), i.e. MCFP(S). It continues by adding/removing arcs to/from a partial stable set at hand and returns the maximal stable set S^* whose $z(S^*)$ value is minimum. RDS is first introduced to solve constraint optimization or constraint satisfaction problems by [98]. The key feature which makes RDS applicable to our problem is the hereditary structure of stable sets. RDS approach is reported to produce very successful results in [21] and [99]. The former implements a similar algorithm for the maximum s-plex and the maximum s-defective clique problems which are also hereditary graph properties. The authors of the latter apply a RDS approach to find a maximum weighted subgraph with minimum risk which is originally modeled in a stochastic programming framework. Both papers provide significantly better results than the commercial solvers. In our study, the naive RDS method is improved by the introduction of dynamic candidate sets and allowing the property violating vertices to enter the maximal stable set when we make sure that the optimization procedure is not affected by this operation as a consequence of Proposition 4.1.

9.1. Description of the Algorithm

Substantially, RDS is a BB based combinatorial algorithm that tracks between the levels of the BB tree. We keep a partial stable set $\mathcal{I}$ throughout the algorithm. The corresponding arcs to the vertices in $\mathcal{I}$ do not conflict each other, by definition. The

[13]An earlier version of this chapter appears in [13–15] as a part of their contents.

vertices that do not share an edge with $\mathcal{I}$ are placed into a free vertex set F_l associated with level l. Furthermore, a candidate set C_l which is a subset of F_l is defined for level l. C_l consists of vertices that do not violate the stable set property when added to $\mathcal{I}$. Actually, all elements of F_l satisfy this condition but the size of the candidate set has a major impact on the performance of the algorithm. $C_l = F_l$ is assumed in the naive implementation of RDS. The approaches to downsize the candidate set will be discussed later.

The algorithm starts at level $l = 0$ by inserting all isolated vertices to the initial stable set, $\mathcal{I} = \{i : i \in V(C),\, d_C(i) = 0\}$, and the remaining ones to F_0. $\mathcal{I}^*$ denotes the incumbent solution with cost $z(\mathcal{I}^*)$, and $\overline{z}$ is the global upper bound, which is initially equal to the output of the heuristic that finds an initial feasible solution. MCFP relaxation is solved on the arc set $\mathcal{I} \cup F_l$, at level l. It should be emphasized that the corresponding arcs of $\mathcal{I}$ do not necessarily carry positive flow and any two flows in F_l may conflict with each other while the ones in $\mathcal{I}$ are absolutely conflict-free. If the objective value of the relaxed problem, z_l, is less than $\overline{z}$ and $C_l \neq \emptyset$, branching is performed by adding a vertex $i_l \in C_l$ to $\mathcal{I}$. Branching operation increases the size of $\mathcal{I}$ by one, $\mathcal{I} = \mathcal{I} \cup \{i_l\}$, $i_l \in C_l$. The branching vertex i_l is selected as the first element of C_l and removed from C_l and F_l. The free arc set of the next level is constructed by removing all vertices whose inclusion in $\mathcal{I}$ violates the stable set property: $F_{l+1} = \{j \in F_l : (i, j) \notin E(C), j \neq i\}$.

In the case where $z_l < \overline{z}$ and $C_{l+1} = \emptyset$, the incumbent solution and the global upper bound are updated as $\mathcal{I}^* = \mathcal{I}$, $\overline{z} = z_l$ and pruning operation is performed. The algorithm backtracks to the previous level by removing the last added vertex from $\mathcal{I}$, namely i_{l-1}, and proceeds by adding the next vertex in the candidate set of level $l-1$, C_{l-1}. If the MCFP relaxation is infeasible or produces a lower bound which is not less than the best upper bound, i.e. $z_l \geq \overline{z}$, the current branch is pruned exactly in the same way by removing the last added vertex from $\mathcal{I}$ and adding the vertex of the candidate set of the previous level. In these two cases, the stable sets that could be obtained by branching further would provide either no feasible flow or a feasible flow with a higher cost than the global upper bound.

The order of the vertices in the candidate set is an important factor that affects the size of the RDS search tree. The branches which are formed by adding the initial vertices of C_l are less likely to be pruned. As we approach to the end of the candidate set, more vertices are deleted and the probability of encountering high-cost or infeasible MCFP solutions increases. The goal is to place the vertices that are expected to cause pruning as the initial entries of the candidate set C_l. Following this principle, the vertices are ordered in terms of decreasing unit costs of the corresponding arcs just once at the beginning. The performance of this ordering is better than the other rules: ordering by indices, increasing unit costs, increasing or decreasing degrees.

9.2. Candidate Set Generation

Up to now, we have described the naive RDS algorithm developed to solve MCFPC. Now we present how we reduce the size of the candidate set with the aim of improving efficiency. Recall that the candidate set of level l is determined only when the optimal objective value of MCFP relaxation on $\mathcal{I} \cup F_l$, z_l, is less than $\overline{z}$. The candidate set generation procedure is quite simple: a maximal stable set S_l on the subgraph of C induced by the vertex set F_l, denoted by $C[F_l]$, is found and $F_l \setminus S_l$ is assigned as the candidate set C_l. The vertices of $C[F_l]$ are added one by one to S_l, which is initially an empty set. S_l is filled in a greedy manner if the next vertex is not adjacent to the current set. The size of S_l must be maximized to minimize the size of the candidate set C_l. In order to achieve this, the vertices of $C[F_l]$ are sorted in the increasing order of their degrees with respect to the induced subgraph $C[F_l]$. While constructing S_l, the vertices whose corresponding arcs carry positive flow in the solution of MCFP relaxation are given priority. As a result, the ones with zero flow are more probable to fall into C_l. This preference improves the efficiency of the algorithm when applied with a *dynamic* candidate set, which will be described in the next subsection.

Upon determining S_l, another MCFP relaxation on $\mathcal{I} \cup S_l$ is solved. If the relaxed problem has a solution, we know that it is also a conflict-free solution and $\overline{z}$ is updated if necessary. This step can be thought as an equivalent form of the heuristics, which is applied in the classical branch-and-bound with LP relaxation to find an integer solution from a given fractional solution. There is another advantage of solving MCFP relaxation on $\mathcal{I} \cup S_l$. The corresponding vertices of the nonbasic arcs with nonnegative reduced costs (i.e. $\bar{c}_{ij} \geq 0$) can also be added to the maximal stable set S_l, although this can cause the violation of the stable set property as a consequence of Proposition 4.1. Because we know that those arcs never carry positive flow in an optimal solution even if they are included in the set $\mathcal{I} \cup S_l$. In other words, although the vertices of $\mathcal{I} \cup S_l$ do not constitute a stable set, its arcs with positive flow are guaranteed to form a stable set of the conflict graph. In the final step, C_l is set to $F_l \setminus S_l$. If C_l is returned as an empty set, then the optimal solution and its value is updated, $\mathcal{I}^* = \mathcal{I} \cup S_l$ and $\bar{z} = z(\mathcal{I} \cup S_l)$, and the current branch is pruned.

9.3. Dynamic Candidate Set Approach

The original naive RDS algorithm works with a static candidate set, which is generated only once during branching. However, there are some disadvantages of using static candidate sets in our context. For example, suppose that MCFP relaxation is solved on $\mathcal{I} \cup F_l$, the current branch is pruned for some reason at level l, and the last added vertex i_{l-1} is removed from $\mathcal{I}$. Also suppose that the removal of i_{l-1} creates an infeasibility, i.e. MCFP relaxation on the $\mathcal{I} \cup F_{l-1}$ is infeasible (Notice that, at this moment i_{l-1} is included neither in $\mathcal{I}$ nor in F_{l-1}). In static candidate set approach, there is no possibility of detecting this situation at the time of occurence. Instead, we add every element of C_{l-1} one by one to $\mathcal{I}$ and solve the relaxed problem on $\mathcal{I} \cup F_l$ where $F_l \subset F_{l-1}$. These branches at level l, ends up with an infeasible solution space due to the fact that $\mathcal{I} \cup F_{l-1}$ includes $\mathcal{I} \cup F_l$. A similar case takes place when the removal of a vertex increases the objective value above $\bar{z}$. On the other hand, we can save $|C_{l-1}|$ many relaxations by setting $C_{l-1} = \emptyset$ if we detect this situation exactly when it occurs.

Another source of inefficiency is the vertices of C_l representing the arcs carrying zero flow in the solution of the MCFP relaxation on $\mathcal{I} \cup F_l$. Consider the case where all vertices in C_l are associated with zero flow. Then, there is no need to proceed and branch because the vertices of $\mathcal{I} \cup F_l$ with positive equivalent flow already form a maximal stable set and it is not possible to attain better MCFP solutions in $\mathcal{I} \cup F_{l+1}$. In this case, C_l can be accepted as an empty set and the current branch is pruned instead of performing additional unnecessary iterations. If we keep using the static candidate set approach, either all vertices are added to C_l regardless of their equivalent flow value or the ones with zero flow are left outside while constructing C_l for the first time. The former situation causes inefficiency as explained above, while the latter misses some maximal stable sets which may include an optimal solution of MCFPC. As a hybrid approach, vertices (i.e. arcs of $A(N)$) associated with zero flow can be omitted from the candidate set initially and they can be added if they take positive values later in the algorithm. In order to keep track of the *dynamic candidate set*, another set ZF_l is defined to keep track of the vertices in $F_l \setminus S_l$ with zero flow, while the others are kept in C_l.

The use of the dynamic candidate set is proposed to avoid these mentioned inconveniences, namely the inability to detect non-promising candidate solutions and the unnecessary branching operations with the vertices corresponding to arcs with zero flow. In order to understand better the working principle of the dynamic candidate set, assume that the current branch is pruned at level l, then the last added vertex is deleted from $\mathcal{I}$ and the MCFP relaxation on $\mathcal{I} \cup F_{l-1}$ is solved again. If there is no solution or the objective value is not smaller than the upper bound $\overline{z}$, C_{l-1} is set to the empty set. Otherwise, the elements of C_{l-1} and ZF_{l-1} are checked. The vertices of C_{l-1} with zero flow are moved to ZF_{l-1} and the vertices of ZF_{l-1}, which corresponds to positive flows, are placed into C_{l-1}. Notice that we solve two MCFP relaxations per stable set evaluation. Since the MCFP solver we use, network simplex algorithm of LEMON, is a very efficient implementation and the number of evaluated stable sets in the static approach is much more than the twice of the number in the dynamic set approach; the advantages of the dynamic set approach dominate this drawback. Finally, we should keep in mind that local preprocessing steps, which are previously described in Section 8.6, can be applied just before solving any MCFP relaxation to eliminate vertices representing the free arcs from the free set F_l. Overall RDS algorithm is formally listed as Algorithm 9.1.

```
Algorithm 5.6: RDS Algorithm for MCFPC
Input: A network $N = (V(N), A(N))$ with conflict graph $C = V(C), E(C)$, initial
 upper bound $\overline{z}$;
Output: Optimal independent set $\mathcal{I}^*$ of $C$ and its $z(\mathcal{I}^*)$ value;
begin
 1. Initialization: Set $\mathcal{I} = \{i \in V(C) : d_C(i) = 0\}$, $F_0 = V(C) \setminus \mathcal{I}$, $l = 0$
 2. Bounding: Solve MCFP relaxation on $\mathcal{I} \cup F_l$. Let $\mathbf{f_l}$ and $z_l$ be the optimal
    solution and its objective value
    If $z_l \geq \overline{z}$, then go to step 6
 3. Candidate Set Construction:
    i. Find a maximal independent set $S_l$ on the induced subgraph $C[F_l]$
    ii. Solve MCFP relaxation on $\mathcal{I} \cup S_l$, set $\overline{z} = z(\mathcal{I} \cup S_l)$ if $\overline{z} > z(\mathcal{I} \cup S_l)$
    iii. Enlarge $S_l$ with vertices whose equivalent arcs having nonengative reduced
         costs
    iv. Put the elements of $F_l \setminus S_l$ whose corresponding flows are positive into $C_l$
 4. Maximality Check: If $C_l = \emptyset$, then set $\mathcal{I}^* = \mathcal{I} \cup S_l$ and $\overline{z} = z(\mathcal{I} \cup S_l)$, go
    to step 6
 5. Branching:
    i. Set $i_l = i \in C_l$, add $i_l$ to $\mathcal{I}$ and remove it from $C_l$ and $F_l$
    ii. Set $F_{l+1} = \{j \in F_l : (i,j) \notin E(C)\}$
    iii. Set $l = l + 1$, go to step 2
 6. Pruning:
    i. If $l = 0$, then go to step 8
    ii. Remove $i_{l-1}$ from $\mathcal{I}$, set $l = l - 1$
 7. Candidate Set Update: Solve the MCFP relaxation on $\mathcal{I} \cup F_l$, update $z_l$
    and $\mathbf{f_l}$
    i. If $z_l \geq \overline{z}$, then go to step 6
    ii. Else, rearrange $C_l$ such that it contains the vertices of $F_l \setminus S_l$ with positive
        flow values, go to step 4
 8. Termination: Set $z^* = \overline{z}$. Return $\mathcal{I}^*$ and $z^*$
end
```

Figure 9.1. RDS Algorithm for MCFPC.

Application of the described RDS algorithm to solve MFPC requires some modifications as well. First of all, it is a maximization problem and solving MFPC gives an upper bound. Therefore, we branch if $z_l > \underline{z}$ where z_l is the optimal objective value of MFP relaxation at level l and $\underline{z}$ is the global lower bound. Whenever $C_l = \emptyset$ is encountered or $z(\mathcal{I} \cup S_l)$ provides a better solution than the incumbent, $\underline{z}$ is updated. Notice that we cannot enlarge S_l with the vertices violating stable set property because reduced costs are not defined for the arcs of the network. Also, the vertices of $V(C)$ sorted in the increasing order of the flow capacities once at the beginning to make a more efficient search.

Implementation of RDS to solve APC can be quite efficient because we do not need to use an algorithm to calculate $z(S)$. The selected vertices in S already satisfy the conflict constraints, i.e. constraints (4.22), and give a matching which is not necessarily perfect. Hence, $z(S)$ can be calculated by simply summing up the edge costs over the elements of S. Still, it is necessary to use Hungarian algorithm to solve APC on $z(\mathcal{I} \cup S_l)$ in order to be able to determine the reduced costs.

10. EXPERIMENTS AND RESULTS

This chapter[14] provides the results of the experiments carried out to measure the efficiency of the proposed solution methods.

10.1. Computational Study for the Minimum Cost Noncrossing Flow Problem

We have realized a set of computational experiments on a large set of randomly generated test instances in order to assess the strength of the relaxations (i.e. LP relaxations of the formulations and MCFP relaxation) and the value of the preprocessing scheme.

10.1.1. Test Environment

A NETGEN-like [100] instance generator, which exploits the layered structure of the network, has been developed for generating test instances. After setting the number of layers L and the maximum number of vertices in a layer $n_{\max}$, the vertex number of vertices for layers $2, 3, \ldots, L-1$ are generated uniformly within $[1, n_{\max}]$. Layers 1 and L have a single vertex, namely s and t. Arcs are obtained by connecting the vertices of the adjacent layers, and the crossing ones are determined according to the vertex embedding we use. Then, a skeleton, which guarantees a feasible noncrossing flow, is constructed and its arcs are assigned large enough unit costs in order to prevent them from appearing in an optimal solution.

33 instances are generated with 10, 11, 12,. . . , 20 layers; three instances for each value. $n_{\max}$ is set to 15, 16, 17, 18, 19, and 20 arbitrarily, and exactly one instance is generated for each combination. The properties of the test instances are reported in Table 10.1. The first column includes the instance numbers. Columns 2 – 5 list the basic structural properties; these are the number of layers, maximum number of vertices

[14]An earlier version of this chapter appears in [12–15] as a part of their contents.

at each layer, number of vertices and arcs. Column 6 includes the number of crossing arc pairs. The values given in columns 7 and 9 are the maximum possible number of arcs and arc pairs in the networks respectively. They are equal to $|V(N)|(|V(N)|-1)$ and $|A(N)|(|A(N)|-1)/2$ and used to calculate the arc and crossing arc pair densities reported in columns 8 and 10, whose entries are obtained by dividing the entries of column 4 by the ones of column 7 and the entries of column 5 by the ones of column 9.

The computations are carried out on a workstation with Intel Xeon CPU E5-2687W0 3.10 GHz processor and 64.0 GB RAM, and operating within Microsoft Windows 7 Professional environment. The programs are coded in C++ [101]. The CPU times and objective values are obtained using CPLEX 12.6 [102] with default options on.

10.1.2. Formulations and Relaxations

We start our experiments with the MILP formulations and their relaxations. Based on the averages listed in the last row of Table 10.2, we can say that the weak formulation (i.e. (4.2) – (4.6), (4.8), (4.24)) is the most efficient one in terms of optimal solution time, although the strong formulation (i.e. (4.2) – (4.8)) gives 20.26% higher lower bound. This is probably because its LP relaxation can be solved faster, which means a faster process of the nodes of the branch-and-bound tree. The weakest lower bounds belong to the MCFP relaxation. Although their computation requires the solution of MCFP, it can be done efficiently using one of the known algorithms (e.g. [32]). As a result, a very efficient branch-and-bound algorithm can be developed by taking advantage of this.

10.1.3. The Effect of Preprocessing

In order to judge the effect of the preprocessing, we compare the CPU times of both formulations and relaxations with preprocessing. We prefer not to report preprocessing times since it takes less than 0.001 seconds for all test instances. According to the results summarized in Table 10.3, we can say that the effect of preprocessing

Table 10.1. Properties of the generated test instances.

Instance Number	L	n_{max}	$\lvert V(N)\rvert$	$\lvert A(N)\rvert$	Crossing Arc Pair Number	Max. Num. Arcs	Arc Density (%)	Max. Num. Pairs	Crossing Arc Pair Density (%)
1	10	15	71	617	15,725	4,970	12.41	190,036	8.27
2	10	18	100	1,024	39,942	9,900	10.34	523,776	7.63
3	10	20	83	744	23,782	6,806	10.93	276,396	8.60
4	11	15	90	750	17,734	8,010	9.36	280,875	6.31
5	11	16	96	863	24,081	9,120	9.46	371,953	6.47
6	11	17	91	776	27,704	8,190	9.47	300,700	9.21
7	12	16	99	879	28,706	9,702	9.06	385,881	7.44
8	12	17	103	978	28,867	10,506	9.31	477,753	6.04
9	12	18	107	1,009	37,796	11,342	8.90	508,536	7.43
10	13	15	111	856	17,469	12,210	7.01	365,940	4.77
11	13	17	110	899	24,540	11,990	7.50	403,651	6.08
12	13	19	98	767	19,561	9,506	8.07	293,761	6.66
13	14	16	95	669	12,412	8,930	7.49	223,446	5.55
14	14	18	116	959	21,493	13,340	7.19	459,361	4.68
15	14	20	129	1,278	52,662	16,512	7.74	816,003	6.45
16	15	15	105	774	16,566	10,920	7.09	299,151	5.54
17	15	16	118	1,004	24,251	13,806	7.27	503,506	4.82
18	15	17	144	1,295	46,373	20,592	6.29	837,865	5.53
19	16	15	110	836	14,628	11,990	6.97	349,030	4.19
20	16	16	122	794	11,529	14,762	5.38	314,821	3.66
21	16	20	125	774	13,259	15,500	4.99	299,151	4.43
22	17	15	92	557	9,973	8,372	6.65	154,846	6.44
23	17	16	119	904	18,306	14,042	6.44	408,156	4.49
24	17	17	117	847	15,389	13,572	6.24	358,281	4.30
25	18	17	133	943	18,467	17,556	5.37	444,153	4.16
26	18	18	133	844	10,720	17,556	4.81	355,746	3.01
27	18	19	143	1,085	25,747	20,306	5.34	588,070	4.38
28	19	15	145	1,131	27,812	20,880	5.42	639,015	4.35
29	19	16	130	981	19,430	16,770	5.85	480,690	4.04
30	19	20	189	1,620	44,161	35,532	4.56	1,311,390	3.37
31	20	15	125	872	20,522	15,500	5.63	379,756	5.40
32	20	16	148	939	14,379	21,756	4.32	440,391	3.27
33	20	17	135	951	13,160	20,180	4.71	451,725	2.91

Table 10.2. Formulations and relaxations.

Instances		Strong Formulation			Weak Formulation			MCFP Relaxation	
No.	z^*	CPU Optimum (sec.)	CPU Relaxation (sec.)	Lower Bound	CPU Optimum (sec.)	CPU Relaxation (sec.)	Lower Bound	CPU Relaxation (sec.)	Lower Bound
1	12,525	36.42	0.05	6,356.9	120.12	0.03	4,343.3	0.00	3,732.0
2	27,697	1,644.14	0.36	20,405.1	597.78	0.04	18,617.0	0.01	18,211.0
3	20,930	76.48	0.08	11,767.6	55.63	0.03	9,194.2	0.01	8,749.0
4	36,005	44.85	0.21	13,573.5	38.41	0.05	8,956.4	0.00	7,689.0
5	20,731	179.08	0.08	7,395.7	179.93	0.03	5,147.9	0.00	4,713.0
6	33,793	33.34	0.09	8,867.7	58.77	0.03	6,116.8	0.00	4,997.0
7	19,360	1,497.88	0.07	15,063.6	846.02	0.05	13,591.5	0.00	13,002.0
8	16,783	52.65	0.21	7,413.8	43.55	0.03	6,049.0	0.01	5,847.0
9	30,472	50.51	0.13	9,684.2	46.15	0.09	5,817.0	0.00	4,715.0
10	28,272	56.26	0.13	18,667.9	52.42	0.02	16,537.7	0.01	16,088.0
11	33,980	126.07	0.20	22,755.8	81.51	0.03	20,632.4	0.01	19,825.0
12	26,264	17.09	0.21	12,482.1	89.30	0.03	9,427.8	0.01	8,144.0
13	45,081	110.02	0.17	21,667.8	89.01	0.06	18,011.7	0.00	14,476.0
14	15,073	1,107.39	0.09	6,267.6	741.07	0.04	4,972.0	0.00	4,728.0
15	22,162	193.07	0.41	7,619.5	1,139.20	0.10	5,426.7	0.00	5,183.0
16	27,435	808.50	0.06	16,279.7	772.31	0.03	13,679.0	0.00	12,375.0
17	36,773	2,480.90	0.11	18,934.7	1,824.77	0.03	15,358.4	0.00	14,033.0
18	56,305	17.79	0.20	25,659.4	20.52	0.06	18,726.9	0.01	15,344.0
19	21,666	386.88	0.05	11,699.2	200.76	0.02	9,410.8	0.00	8,564.0
20	41,783	21.45	0.03	35,661.5	24.71	0.02	33,877.9	0.00	33,406.0
21	43,085	20.68	0.17	31,782.8	40.32	0.03	27,980.7	0.00	26,181.0
22	27,308	14.27	0.04	13,886.8	35.59	0.02	11,644.2	0.00	8,815.0
23	25,277	1,013.86	0.17	12,397.9	585.06	0.06	9,806.9	0.00	8,747.0
24	29,701	654.81	0.08	19,957.1	1,817.86	0.02	17,153.8	0.00	16,084.0
25	27,641	12.49	0.06	15,794.8	18.75	0.03	13,470.2	0.01	12,803.0
26	97,928	7.65	0.22	73,421.1	22.08	0.08	61,444.4	0.00	47,289.0
27	40,357	383.21	0.12	22,516.4	271.14	0.04	17,350.2	0.01	14,807.0
28	25,141	68.47	0.08	13,829.7	58.20	0.03	11,626.5	0.01	11,108.0
29	37,767	833.74	0.13	20,836.8	536.03	0.05	15,618.7	0.00	12,549.0
30	50,254	65.11	0.38	36,129.3	145.42	0.11	32,449.0	0.00	31,690.0
31	22,431	6.35	0.05	13,524.9	19.34	0.02	11,606.8	0.00	10,175.0
32	58,713	80.68	0.08	42,226.8	104.81	0.03	36,960.4	0.02	33,595.0
33	40,462	667.09	0.09	21,700.0	732.59	0.03	18,026.5	0.00	16,428.0
Average	33,308	386.94	0.14	19,279.63	345.73	0.04	16,031.29	0.00	14,366.42

Table 10.3. The effect of preprocessing.

Instances		Strong Formulation			Weak Formulation			MCFP Relaxation	
No.	Deleted arcs (%)	CPU Optimum (sec.)	CPU Relaxation (sec.)	Lower Bound	CPU Optimum (sec.)	CPU Relaxation (sec.)	Lower Bound	CPU Relaxation (sec.)	Lower Bound
1	17.18	150.96	0.03	10,084.1	130.07	0.01	9,004.5	0.00	7,573.0
2	22.56	807.76	0.14	23,531.1	710.11	0.04	22,343.6	0.00	21,771.0
3	32.53	58.83	0.08	17,617.6	65.47	0.01	16,370.0	0.00	15,417.0
4	34.00	46.78	0.09	26,024.5	30.83	0.02	22,215.4	0.00	18,233.0
5	33.37	199.91	0.05	13,353.6	154.49	0.02	11,405.4	0.00	10,459.0
6	39.95	29.14	0.05	17,744.5	34.02	0.02	14,853.7	0.00	11,790.0
7	17.75	813.49	0.05	15,517.5	736.54	0.02	13,927.2	0.00	13,211.0
8	27.20	54.24	0.06	11,002.1	26.60	0.02	10,051.0	0.00	9,780.0
9	41.43	46.01	0.04	19,005.1	49.86	0.02	17,008.1	0.00	13,809.0
10	26.99	47.25	0.04	23,194.2	67.05	0.01	22,016.2	0.00	21,062.0
11	27.03	84.58	0.08	26,659.7	82.01	0.02	24,549.0	0.00	22,621.0
12	31.94	36.89	0.07	17,829.4	43.48	0.03	14,982.2	0.01	12,828.0
13	31.09	73.31	0.14	30,034.7	137.09	0.02	26,735.0	0.00	21,627.0
14	22.42	1,549.86	0.07	10,500.2	902.51	0.01	9,372.4	0.00	8,876.0
15	37.72	270.41	0.07	16,377.9	246.04	0.02	14,871.8	0.00	13,818.0
16	25.97	629.98	0.05	20,596.6	801.53	0.02	18,740.5	0.00	16,511.0
17	27.29	855.85	0.06	29,049.0	1,415.23	0.02	26,260.6	0.00	23,116.0
18	40.62	11.62	0.05	40,105.0	19.06	0.02	36,289.8	0.01	32,746.0
19	23.92	120.99	0.03	16,242.7	162.94	0.00	14,291.0	0.00	13,226.0
20	20.15	15.63	0.03	39,794.9	25.82	0.02	38,818.5	0.02	37,431.0
21	39.66	25.94	0.01	40,188.5	34.07	0.01	38,722.2	0.00	35,262.0
22	33.75	16.27	0.02	21,957.7	19.31	0.01	20,464.6	0.00	15,803.0
23	25.66	834.77	0.04	19,232.2	623.70	0.02	17,545.0	0.00	15,479.0
24	26.21	641.70	0.03	24,720.9	750.19	0.02	23,241.9	0.00	22,516.0
25	32.77	10.27	0.06	23,398.6	8.97	0.02	21,655.9	0.01	18,863.0
26	26.78	15.27	0.09	81,890.1	24.32	0.03	74,967.9	0.00	61,555.0
27	24.06	242.85	0.12	29,029.3	346.73	0.04	25,195.9	0.01	21,840.0
28	34.39	58.08	0.05	21,149.7	50.28	0.02	20,268.8	0.00	18,964.0
29	21.71	660.31	0.08	27,530.7	764.88	0.02	24,636.9	0.00	21,660.0
30	30.43	89.73	0.06	45,592.5	71.62	0.02	43,524.6	0.00	42,261.0
31	33.83	7.73	0.03	19,353.6	16.16	0.02	18,220.8	0.00	14,838.0
32	30.56	64.70	0.06	51,554.2	78.23	0.02	49,395.0	0.00	43,606.0
33	32.07	599.35	0.06	27,870.6	502.49	0.02	25,607.8	0.00	23,013.0
Average	29.48	277.89	0.06	25,991.91	276.72	0.02	23,865.25	0.00	21,259.55

is remarkable. First of all, the values in the second column indicate that, on the average, 29.48% of the arcs have been deleted. There is also a considerable decrease in the running times. The average CPU time decreases by 28.18% and 19.96% for the strong and weak formulations respectively. The weak formulation is 0.4% faster with preprocessing compared to the strong formulation. There is a higher improvement in the efficiency of the strong formulation due to preprocessing, which makes its average performance comparable with that of the weak formulation after preprocessing. An interesting observation is related to the lower bounds; preprocessing makes them 34.82%, 48.87%, and 48.08% higher on the average.

10.2. Computational Study for the Minimum Cost Flow Problem with Conflicts

10.2.1. Test Environment

In order to evaluate the performances of the proposed algorithms, the instances which are guaranteed to have at least one feasible solution are produced by NETGEN generator [100]. First, a feasible skeleton network is constructed. This skeleton includes at least one feasible solution for the Minimum Cost Flow Problem (MCFP) incurring a high cost so that this solution is unlikely to be an optimal one. Then, random arcs with all the necessary parameters are added to the network to obtain a MCFP instance. Finally, the conflict lists are generated arbitrarily by ensuring that none of the arcs in the skeleton conflict with each other. This final step makes the skeleton feasible for MCFPC; at least one feasible solution is guaranteed.

The size of the test instances are characterized by three parameters: the number of nodes $n = |V(N)|$, the arc density $p = \frac{m}{n(n-1)}$ with $m = |A(N)|$, and the conflict density $d = \frac{2w}{m(m-1)}$ with $w = |E(C)|$. The parameters are set to the following values: $n \in \{40, 50, 60, 70, 80\}$, $p \in \{0.3, 0.4, 0.5, 0.6\}$ and $d \in \{0.2, 0.3, 0.4, 0.5\}$. As a result $5 \times 4 \times 4 = 80$ instances are generated to be considered in the computational study. Parameter values outside this range render too small or too large instances, which makes either all algorithms find an optimal solution within a few seconds or none of them produce a feasible solution. For reporting the results we have prepared summary tables (i.e. Table 10.4 – Table 10.10). They are based on Table A.1 – Table A.5 given in the Appendix. They include computational results of the experiments for each individual instance.

BB algorithm with three branching rules, i.e. pair, clique and arc branching, and RDS are tested on the generated instances and the results are compared with the MILP solver of CPLEX (version 12.9) on both strong and weak formulations. Even though the superiority of specially designed algorithms over general purpose CPLEX is not surprising, the breakthrough of the recent versions of CPLEX should not be disregarded. Besides, no solution method solving MCFPC exists in the literature to the best of our knowledge. Therefore, CPLEX is selected as the benchmark to measure the efficiency of the algorithms.

Prior to running each method, including CPLEX solver, previously described pre-optimization procedures are applied to obtain a reduced network and an upper bound. They are applied to all algorithms for the sake of fairness. Dual simplex method of CPLEX is selected because our experiments show that its average running time is around 5 seconds shorter than primal simplex and network simplex algorithms of CPLEX which perform almost equally. The MCFP relaxations of BB and RDS algorithms are solved using the network simplex solver of LEMON (version 1.3.1). All experiments are carried out on a computer with Microsoft Windows 7 Professional operating system and Intel Xeon CPU E5-2687W 3.10 GHz processor with 64.0 GB RAM. The running time is limited by one hour.

10.2.2. The Effect of Preprocessing and Probing

Sequentially applied preprocessing and probing steps clear all conflicts in 4 of the instances and delete 6.73% of arcs on the average by spending 0.50 seconds in the average. Probing and preprocessing account for 97.5% and 2.5% of the total time, respectively. Although the average percentage of the deleted arcs is not remarkable, the standard deviation is quite high. It seems to be independent of the number of vertices (n), whereas the group averages over 20 instances for each value of p and d indicate that the percentage of deleted arcs can be increased by reducing arc density p or increasing conflict density d (see Table 10.4 and Table 10.5). In other words, these pre-optimization procedures can be useful tools to reduce the size of the networks particularly with low p and high d values in a negligible amount of time.

Table 10.4. Average percentage of deleted arcs: the effect of arc density.

p	Preprocessing(%)	Probing(%)	Total(%)
0.3	9.11	7.01	16.12
0.4	5.76	4.04	9.81
0.5	0.71	0.11	0.82
0.6	0.08	0.07	0.16
Avg	3.92	2.81	6.73

Table 10.5. Average percentage of deleted arcs: the effect of conflict density.

d	Preprocessing(%)	Probing(%)	Total(%)
0.2	0.45	0.36	0.81
0.3	0.74	1.08	1.83
0.4	4.92	2.29	7.22
0.5	9.56	7.50	17.05
Avg	3.92	2.81	6.73

10.2.3. The Effect of Diving

The diving heuristic is applied with two different rules as mentioned previously in Section 6.4.1 following the probing procedure. It is stopped when a conflict-free solution is found but since we provide a 60-second time limit for each rule, it takes 120 seconds in the worst case to find an initial feasible solution. According to the experimental results, finding an initial solution using these two diving heuristics takes an overall average of 27.36 CPU seconds and at least one rule provides an upper bound for all instances. The average CPU times calculated for the same n, p and d values are listed in Table 10.6. Note that for each n value, the mean is evaluated over 16 instances. The required time to find a conflict-free solution becomes longer as n or p increases, as expected. However, an interesting result is the negative correlation between the CPU time of the diving heuristic and the conflict density. At the first glance, one might think that more time is needed to get rid of the conflicts as d increases. However, larger d means larger conflicts per arc and it allows us to remove more arcs when an arc is forced to carry positive flow.

Table 10.6. Mean CPU time spent by the diving heuristic.

n	Avg. CPU Time (s)	p	Avg. CPU Time (s)	d	Avg. CPU Time (s)
40	6.19	0.3	10.35	0.2	59.40
50	14.29	0.4	16.11	0.3	20.50
60	21.50	0.5	34.85	0.4	13.58
70	38.33	0.6	45.00	0.5	13.12
80	57.40				
Avg	27.36	Avg	27.36	Avg	27.36

10.2.4. Performance Assessment of the New Exact Solution Methods

10.2.4.1. Most Effective Branching Rule. The number of test instances which can be solved to optimality within one hour is counted for every method and listed in the first row of Table 10.7. For test instance k, let LB_i^k be the lower bound given by method i and $maxLB^k$ is the maximum of the lower bounds given by all methods. Notice that if a method finds an optimal solution the lower bound becomes equal to the optimum objective value. The performance measure

$$LBdev_i^k = 100 \times \frac{(maxLB^k - LB_i^k)}{maxLB^k} \tag{10.1}$$

is calculated for every instance k and method i. When we take the average of $LBdev_i^k$ values over k, we obtain a statistic that shows the quality of the lower bounds produced by algorithm i,

$$LBdev_i = \frac{\sum_{k=1}^{80} LBdev_i^k}{80}. \tag{10.2}$$

The average of percent deviations of the lower bounds from the best value, i.e. $LBdev_i$, are listed in the second row of Table 10.7. A similar measure can be calculated for the upper bounds but the quality of the upper bound mostly depends on the quality of the diving heuristic, which is common for all algorithms, so it is not calculated here. According to the results, there is a strong correlation between the number of instances solved optimally and the average deviation from the best lower bound. BB with pair branching rule is able to solve 7 instances optimally within one hour with the worst lower bound performance among all methods. Pair branching produces successful results compared to earlier versions of CPLEX used for the transportation problem with exclusionary side constraints in [50]. Since pair branching produces very large BB trees even for small MCFPC instances, it cannot outperform CPLEX which solves optimally 31 and 59 instances with strong and weak formulations, respectively. Since more instances are solved optimally, the average deviation from the best bound is smaller for the weak formulation as indicated in the table. In theory, the relaxation of the weak one is solved faster producing worse lower bound compared to the strong formulation. Clearly, its advantages outweigh the drawbacks and CPLEX's performance on weak formulation is set as the benchmark for the new algorithms and the results of the strong formulation are not considered.

Both clique and arc branching rules of BB give better results than CPLEX on strong formulation but BB with clique branching, which finds optimum solution of 53 instances, has been outperformed by CPLEX on weak formulation in terms of both performance indicators. Clique branching rule will not be discussed any more due to the fact that BB approaches have a lot in common theoretically and arc branching rule is more successful. The number of instances solved to optimality by RDS and BB-Arc (i.e. BB with arc branching rule) are 67 and 70, respectively. The first row of Table 10.7 includes the number of test instances solved optimally within one hour of running time. The numbers of the second row are the $LBdev_i$ values (i.e. relative percent deviation of the lower bounds) for different methods. At sum, this table reveals that BB provides the highest lower bound for every single instance finding the maximum number of optimum solutions and RDS is the second most successful algorithm in terms of both performance measures.

Table 10.7. Solution performance of different methods: total number of solved instances and $LBdev_i$ values.

CPLEX-Strong	CPLEX-Weak	BB-Arc	BB-Pair	BB-Clique	RDS
31	59	70	7	53	67
53.37	17.12	0.00	83.16	26.76	13.81

<u>10.2.4.2. The Efficiency of the New Methods.</u> In order to assess the effect of the parameters, the overall results given in Table 10.7 are grouped in terms of n, p and d values. One of the performance measures is sufficient for aggregation since the two of them are strongly correlated with each other. Table 10.8 reveals that RDS becomes less successful as the number of nodes increases. Although there seems to be a downward trend in the performance of BB-Arc and CPLEX-Weak as the number of vertices n increases, it is not significant. An increase in n causes slight decreases in the performance of RDS due to the fact that it indirectly increases the number of arcs, i.e. the number of vertices in the conflict graph. Finding maximal stable sets becomes more time-consuming as the number of vertices in the conflict graph increases.

According to Table 10.9, all methods show better performance as conflict density increases and perform worse as arc density increases. A higher arc density directly affects the depth of BB and RDS search trees, i.e. we need to do more branching to find a conflict-free solution or a maximal stable set of the conflict graph. Also, one additional arc results in two additional variables and one more constraint in the MILP formulation, hence CPU times of CPLEX increase. Notice that the slope of the change is steeper for CPLEX, i.e. the performance of CPLEX is more sensitive to changes in p. On the other hand, increasing conflict density does not change the number of variables

Table 10.8. The effect of the number of vertices.

n	BB-Arc	RDS	CPLEX-W
40	15	15	15
50	16	15	14
60	13	13	9
70	14	13	12
80	12	11	9
Total	70	67	59

or constraints in the weak formulation but some of the new constraints dominate the previous ones providing a restricted solution space. The reason why CPLEX is faster with higher d values is the improved quality of lower bounds produced by the more restricted LP relaxation. Also, branching rules associated with BB and RDS allow them to remove more arcs per subproblem as the value of d increases.

Table 10.9. The number of optimally solved instances aggregated over p and d values.

p	BB-Arc	RDS	CPLEX	d	BB-Arc	RDS	CPLEX
0.3	19	19	18	0.2	12	9	7
0.4	20	18	15	0.3	18	18	15
0.5	16	16	15	0.4	20	20	18
0.6	15	14	11	0.5	20	20	19
Total	70	67	59	Total	70	67	59

Another indicator that measures the efficiency of the algorithm is the CPU times. First we detected 59 test problems that are solved optimally by all three solution approaches within one hour CPU time limit. We take the average CPU times of these instances in order to evaluate the algorithms appropriately. According to Table 10.10, where the average CPU times taken over optimally solved 59 instances are reported, BB is 31 and RDS is 10 times faster than CPLEX.

Table 10.10. Average CPU times in seconds.

BB-Arc	RDS	CPLEX-W
23.28	73.03	719.74

10.2.5. The Effect of Local Procedures

It is also possible to evaluate the effect of each procedure mounted inside the algorithms. For instance, local application of the preprocessing steps prior to each subpoblem, both in BB and RDS, has an obvious positive impact on the efficiency of the solution procedures. It removes 7.97% of the arcs per subproblem on the average and the range is $[1.05, 18.71]$. The instances with high percentages, closer to the upper limit of the range, are observed to be solved quickly. Another procedure is the penalty calculation which caused 0.27% of the prunings (i.e. 0.27% of all pruned subproblems is a consequence of the improvement made in the lower bound due to the penalty calculations). The high frequency of degenerate solutions and the early application of local preprocessing can be accounted for this ignorable percentage attributed to penalties. Moreover, the average percentage of all arcs which are fixed by pegging per subproblem is 1.55%. Although this percentage seems very low, it is a consequence of the local preprocessing steps which delete a considerable number of nonpromising arcs. The power of pegging can be better observed when the local preprocessing procedures are turned off. If we do not call the described local procedures, 4.64% of all arcs are fixed per subproblem through pegging, on the average. Due to the fact that local preprocessing steps improve the overall performance, they are utilized in the final form of the algorithm.

10.3. Computational Study for the Maximum Flow Problem with Conflicts

10.3.1. Test Environment

The performance of every algorithm is evaluated over a set of randomly generated instances which are guaranteed to have at least one nonzero feasible solution for MFPC. As the initial step of the arc generation process, an arbitrary directed path from s to t is constructed, and the minimum possible flow capacities are assigned so that the permitted flow on this path is unlikely to be the maximum flow from source to sink. Then, additional arcs with various flow capacities are defined on the network. Finally, the conflict lists are created randomly by ensuring that none of the arcs in the initially created s-t path conflict with each other. This final step provides at least one feasible solution with nonzero objective value for MFPC, and the small capacities of the arcs on the path make it a less attractive choice.

There exist three parameters determining the size of a test instance: the number of vertices $n = |V(N)|$, the arc density $p = \frac{m}{n(n-1)}$ where $m = |A(N)|$, and the conflict density $d = \frac{2w}{m(m-1)}$ with $w = |E(C)|$. The values of these parameters are selected from the following sets: $n \in \{40, 50, 60, 70, 80\}$, $p \in \{0.3, 0.4, 0.5, 0.6\}$, and $d \in \{0.3, 0.4, 0.5, 0.6\}$. For every combination of parameter values, two test instances are generated, which makes a total of $2 \times 5 \times 4 \times 4 \times = 160$ instances. We remark that, parameter values outside the selected ranges render too small or too large instances, for which either all algorithms find an optimal solution within a few seconds or none of them produces a feasible solution. The results are presented in Table 10.11 – Table 10.18 in an aggregated fashion. The details can be found in Table A.6 – Table A.12 given in the Appendix. They include computational results for each individual instance.

New BD, BB and RDS algorithms are tested on the generated instances by comparing the results with the MILP solver of CPLEX (version 12.7) on the strong (MFPC$_S$), weak (MFPC$_W$), and clique (MFPC$_K$) formulations. In addition, the performance of the built-in Benders decomposition algorithm of CPLEX is compared with that of our BD algorithm. No exact solution procedure solving MFPC exists in the literature to the best of our knowledge. Therefore, CPLEX is selected as the benchmark to assess the efficiency of the algorithms.

All of the algorithms are implemented in C++ environment [101] and the solution of the MFP relaxation within the algorithms is obtained by Goldberg's preflow-push algorithm implemented in LEMON (version 1.3.1) [97]. The experiments are carried out with one hour CPU time limit on a workstation with Microsoft Windows 7 Professional operating system, Intel Xeon CPU E5-2687W 3.10 GHz processor and 64.0 GB RAM.

10.3.2. The Effect of the Formulation

From a theoretical point of view, solving the LP relaxation of MFPC$_S$ takes longer while producing a better upper bound compared to MFPC$_W$'s. Hence, we expect that solving the strong MILP formulation yields an optimal solution by creating fewer subproblems, which are processed considerably faster than those obtained from the weak MILP representation. According to the experimental results of our experiments, the optimal solutions of 38 and 86 instances out of 160 are found within the allowed CPU time limit by solving the strong and weak MILPs, respectively. Examining the common 38 instances that are optimally solved with both formulations indicates that MFPC$_W$ is solved about 8 times faster than MFPC$_S$, on the average. When we take into account the significant reduction in the number of constraints and the capability of CPLEX to improve the quality of the bounds, the numerical results are not surprising.

CPLEX is able to solve the clique formulation, MFPC$_K$, for 40 instances out of 160, and it takes 15.74 seconds on he average to find the set of maximal cliques that covers the edges of the conflict graph using the heuristic procedure described in Section 4.5.2. Although the number of conflict constraints in MFPC$_K$ is about one-eighth of those in MFPC$_S$, it is still considerably larger than that of MFPC$_W$. So, the performance of CPLEX on the clique formulation is similar to that of the strong formulation, MFPC$_S$. To sum up, the advantages of MFPC$_W$ outweigh its drawbacks, and the results obtained by CPLEX on the weak formulation are set as the benchmark for the new algorithms.

10.3.3. The Comparison of Different Benders Decomposition Implementations

As explained before, BD can be applied on the strong, weak, and clique formulations using either the classical iterative approach in which the master problem and the subproblem are solved iteratively or by means of a single-search tree where the generated cuts are added by interrupting the ongoing branch-and-cut algorithm that solves the master problem. Since there exist many alternatives, a set of experiments is carried out with the naive version of BD (without strong cuts) to decide which combination works the best. Among them, we have first combined MFPC$_S$ with the iterative BD approach. This method is able to find the optimal solutions of 38 instances (see Table 10.11), which shows that this combination is far worse than our benchmark.

Table 10.11. Number of optimally solved instances in 1 hour.

Method	Instances
Iterative (MFPC$_S$)	38
Single Tree (MFPC$_S$)	97
Single Tree (MFPC$_K$)	85
Single Tree (MFPC$_W$)	121
CPLEX BD (MFPC$_W$)	43

In the next set of experiments, BD is applied to MFPC$_S$ again where the produced Benders cuts are gradually added using a single-search tree. 97 instances are solved optimally this time, and the results suggest that we should adopt the single-search tree implementation. Then, BD is applied on MFPC$_K$ and MFPC$_W$ using a single-search tree, and the optimal solutions of 85 and 121 instances are found, respectively, within one hour.

Finally, we compare the performance of the built-in Benders decomposition of CPLEX implemented on MFPC$_W$, which appears to be the most suitable formulation for BD. Although it gives slightly better results than the iterative approach in terms of the number of instances solved optimally, it is not even close to the performance of BD implementations on a single-search tree.

Table 10.12. Average number of generated cuts and average CPU times.

Method	Benders Cuts	CPU time (s)
Iterative (MFPC$_S$)	34.86	819.89
Single Tree (MFPC$_S$)	54.39	51.25
Single Tree (MFPC$_K$)	56.93	106.86
Single Tree (MFPC$_W$)	56.29	28.56
CPLEX BD (MFPC$_W$)	46.07	704.75

For every method, we take average number of added Benders optimality cuts and solution times over 28 instances whose optimal solutions are found by all four methods, and list them in Table 10.12. In theory, single-search tree adds a cut whenever it finds an integer solution for the master problem, and continues from the current node of the search tree at which the integer solution is found. In contrast, the iterative BD method generates a Benders optimality cut and resolves a more constrained master problem from the very beginning. Results indicate that we end up with consistently fewer Benders cuts before the optimal solution is found with the iterative BD. This outcome supports the claim that the iterative approach generates stronger cuts. However, the inefficiency originating from repeatedly solving the master problem cannot be compensated by the strength of the obtained cuts. On the other hand, the results show that the quality/strength of the produced Benders cuts does not change significantly across different formulations. It is not surprising due to the fact that the subproblem remains unchanged for all formulations. The average CPU times clearly reveal that the single-search tree approach with weak representation outperforms the others, and it appears to be the best way of implementing Benders decomposition for our problem.

10.3.4. Improvements for the Benders Decomposition Algorithm

The utilization of strong cuts, valid inequalities, and an initial upper bound does not have the same impact on the performance of BD applied to different formulations. As the results in Table 10.13 indicate, the number of instances whose optimal solution is found within one hour CPU time limit increases from 121 to 128 in the case of MFPC$_W$, and from 85 to 87 for MFPC$_K$. However, the aforementioned improvements do not help in solving MFPC$_S$. When the experimental results are analyzed in detail, it is observed that the master problem of MFPC$_S$ becomes more difficult to be solved with the addition of valid inequalities, and the solution time of BD is influenced in a negative way. This finding is also valid in the case of MFPC$_K$ for some instances where the number of constraints is relatively higher.

Table 10.13. The number of optimally solved instances for different versions of BD.

Algorithm	MFPC$_S$	MFPC$_K$	MFPC$_W$
Naive BD	97	85	121
Improved BD	84	87	128

Since BD on the weak formulation MFPC$_W$ gives the best results, we continue our analysis with MFPC$_W$. For 121 instances which are solved optimally with both naive and improved BD, the average CPU time is reduced from 544.00 to 408.53 seconds. In other words, the proposed improvements decrease the solution time by 24.90%, on the average.

According to the results of the improved BD experiments, 62% of the created subproblems provide a nonzero feasible flow from s to t for the original problem MFPC$_W$, whereas the percentage of subproblems that do not contain a directed s-t path is 38%.

10.3.5. Efficiency of the Exact Algorithms

In order to measure the performance of the exact algorithms, we check the quality of the bounds as well as the number and CPU times of the test instances that can be solved optimally within one hour CPU time limit. For an instance k, let LB_i^k and UB_i^k denote the lower and upper bounds computed by method i, and $maxLB^k$ and $minUB^k$ denote the maximum of the lower bounds and the minimum of the upper bounds given by all methods. Notice that if an algorithm finds an optimal solution, the lower and upper bounds become equal to the optimal objective value. The performance indicators

$$LBdev_i^k = 100 \times \frac{(maxLB^k - LB_i^k)}{maxLB^k} \tag{10.3}$$

and

$$UBdev_i^k = 100 \times \frac{(UB_i^k - minUB^k)}{minUB^k} \tag{10.4}$$

measure the deviations from the best known bounds, and are calculated for every instance k and method i. When we take the average of $LBdev_i^k$ and $UBdev_i^k$ values over 160 instances, we obtain the following two statistics showing the quality of the lower and upper bounds produced by algorithm i:

$$LBdev_i = \frac{\sum_{k=1}^{160} LBdev_i^k}{160} \tag{10.5}$$

and

$$UBdev_i = \frac{\sum_{k=1}^{160} UBdev_i^k}{160}. \tag{10.6}$$

Obviously, the algorithms with the least $LBdev_i$ and $UBdev_i$ values are the most successful ones in terms of the bound quality. The results are summarized in Table

10.14 for methods BB, RDS, and BD on MFPC$_W$, which is denoted by BD$_W$, and compared with the results of CPLEX on the weak formulation denoted as CPLEX$_W$.

Table 10.14. Solution performance of different methods.

Algorithm	Instances	CPU times	*LBdev*	*UBdev*
$CPLEX_W$	86	656.73	3.12	69.93
BD_W	128	106.62	2.37	5.08
BB	117	85.95	0.96	2.11
RDS	128	44.56	1.70	106.27

The number of test instances optimally solved within one hour is determined for every method, and listed in the second column of Table 10.14. The developed algorithms are well ahead of the CPLEX$_W$ while BD$_W$ and RDS seem to be the most successful ones in finding optimal solutions. For the purpose of comparing the speed of the algorithms, the average CPU times are calculated over 84 instances whose optimal solutions are found by all four methods. They are written in the third column of Table 10.14. RDS is almost 15 times faster than CPLEX$_W$ while BB and BD$_W$ are 7.5 and 6 times more efficient than CPLEX$_W$, respectively.

The average percent deviations of the bounds from the best bounds, i.e. $LBdev_i$ and $UBdev_i$, are provided in the fourth and fifth columns. According to the results, the quality of the produced lower bounds does not differ too much but BB appears to be the algorithm that gives the best lower bounds (or best feasible solutions). CPLEX$_W$ and RDS are not capable of providing good upper bounds. Since RDS is working according to the depth-first search principle, this outcome is expected. The other two methods yield relatively better upper bounds where BB turns out to be the winner among the proposed solution methods.

10.3.6. Effect of Instance Size on the Performance

In order to assess the effect of the instance size on the performance of the methods, the results given in Table 10.14 are grouped in terms of n, p and d. Although it is possible to do this aggregation over four different presented performance measures, they are not independent from each other. Hence, it suffices to consider only one of them, namely the number of optimal solutions found within one hour time limit. Table 10.15 reveals that there is an obvious downward trend in the efficiency of the solution methods as n increases, and the rate of degradation is the highest for CPLEX$_W$. While all of them are able to solve all test instances with 40 vertices, CPLEX$_W$ can solve only 2 out of 32 instances with 80 vertices. The performance of the remaining algorithms deteriorates as well, but considerably slower. They are capable to find optimal solutions of around half of the instances even for $n = 80$.

Table 10.15. The effect of the number of vertices.

n	*CPLEX*$_W$	*BD*$_W$	*BB*	*RDS*
40	32	32	32	32
50	28	31	29	30
60	17	27	22	26
70	7	22	19	23
80	2	16	15	17
Total	86	128	117	128

According to Table 10.16, all methods perform better as the conflict density increases and worse as the arc density increases. A higher arc density directly affects the depth of the BB and RDS search trees, i.e. we need to do more branching to find a feasible solution. When we consider the weak MILP formulation, an additional arc results in two additional variables and one more constraint hence augmenting the required CPU times of CPLEX$_W$ and BD$_W$. Notice that the rate of change is steeper for CPLEX$_W$; the performance of CPLEX$_W$ is more sensitive to changes in p.

Table 10.16. The effect of the arc density

p	$CPLEX_W$	BD_W	BB	RDS
0.3	32	38	34	37
0.4	22	33	31	35
0.5	19	30	28	31
0.6	13	27	24	25
Total	86	128	117	128

On the other hand, a higher conflict density does not change the number of variables or constraints in the weak formulation but some of the new constraints dominate the previous ones providing a more restricted solution space. For higher d values, more restricted LP relaxations speed up CPLEX$_W$ and the master problem of BD$_W$. Also, branching rules associated with BB and RDS allow them to remove more arcs per subproblem as d increases. The computational results in Table 10.17 point out that the algorithms become more successful with increasing d values.

Table 10.17. The effect of the conflict density

d	$CPLEX_W$	BD_W	BB	RDS
0.3	16	20	14	19
0.4	19	31	25	31
0.5	24	37	38	38
0.6	27	40	40	40
Total	86	128	117	128

One interesting result is obtained for $d = 0.3$ where the number of optimally solved instances by CPLEX$_W$, (i.e. 16), is larger than the one of those solved by BB, which is 14. This fact could be perceived as an indicator that CPLEX$_W$ is more successful than BB for d values equal to 0.3. In order to make sure that this is really the case, we check the other performance measures for this subgroup of instances and report the results in Table 10.18. $LBdev$ and $UBdev$ values confirm that the produced bounds are significantly better for the BB algorithm and the average solution time calculated over 14 common instances solved by both CPLEX$_W$ and BB is considerably shorter for BB. The superiority of other efficiency indicators does not support the claim that CPLEX$_W$ outperforms BB when $d = 0.3$.

Table 10.18. Comparison of CPLEX$_W$ and BB for $d = 0.3$.

Algorithm	Instances	CPU times	LBdev	UBdev
CPLEX$_W$	16	577.66	4.17	24.93
BB	14	351.26	2.46	5.18

10.4. Computational Study for the Assignment Problem with Conflicts

10.4.1. Test Environment

In the literature, there is no standard test library for the APC. To the best of our knowledge the only works addressing the APC are [57,58,60]. Hence, we have generated new and larger test instances following the instance generation procedure suggested in these works. Our test bed consists of 135 test problems of various sizes. The number of nodes in the left hand-side of the bipartite graph $G = (V_1(G) \cup V_2(G), E(G))$ is shown with $|V(G)|$ and selected between 15 and 500. The number of distinct conflict pairs, i.e. $|E(C)|$, ranges between $5,000$ and $200,000$.

Now we introduce the details of the computational experiments. The algorithms are coded in C++ [101] and tested on a PC with 2.2 GHz Intel Core i7 processor and 8 GB RAM operations within 64-bit Windows 7 environment. CPLEX 12.7 with default options is used as a subroutine to solve the resulting BIP and LP problems.

The computational experiments are summarized in Table 10.19 – Table 10.26. The numbers reported in all tables, except the 'Average' row, represent the overall average values of 5 randomly generated test instances. The row 'Average' in all tables indicate the overall average values of the corresponding columns. In all tables, the first two columns, i.e. columns '$|V(G)|$' and '$|E(C)|$' include the number of vertices and conflict pairs in the test instances, respectively. We report CPU times in seconds under 'CPU' columns. In our experiments, we have imposed a CPU time limit of one hour. Hence, in Table 10.22 the instances which cannot be solved within the CPU limit of one hour are marked with 'na'.

In Table 10.19 – Table 10.26, rather than reporting the exact values of the upper and lower bounds obtained during our experiments, we present percent deviations from the optimal value. Namely, we calculate percent deviations of upper and lower bounds from the optimal objective value as $100 \times \frac{(z_{ub}-z^*)}{z^*}$ and $100 \times \frac{(z^*-z_{lb})}{z^*}$ respectively. Here, z_{ub} (z_{lb}) stands for the best upper (lower) bound and z^* denotes the optimum value of the APC.

10.4.2. The Effect of the Formulations

Table 10.19 summarizes computational results obtained with the BIP formulations and the results obtained with the RDS algorithm. The third column, i.e. the column 'OPT', is for the average optimal values. The next six columns are for the CPLEX when we solve optimally WEAK and STRONG formulations with a CPU time limit of one hour. The 'Best Sol.' columns, namely the fourth and seventh columns, include an optimal or best feasible solution values during the search of the BB tree with CPLEX. The 'Nodes' columns include the number of nodes processed by CPLEX. The CPU times are given with 'CPU'. The last three columns incorporate results obtained with the RDS algorithm. The tenth column, namely 'Best UB', includes the percent deviation of the incumbent solution from the optimum solution value when the RDS algorithm is stopped. The eleventh column, i.e. 'Explored Nodes', reports the number of nodes of the BB tree explored during the run of the RDS algorithm. The last column stands for the CPU time in seconds.

Considering the results in Table 10.19, we can conclude that WEAK formulation yields an optimal or best solutions faster with an overall CPU time average of 316.91 secs. CPU time, though exploring larger number of nodes, when compared with the STRONG formulation, which requires 371.19 secs. overall average CPU time. Besides, in 23 out of 27 cases, WEAK formulation outperforms STRONG formulation. As a result, we can argue that the BIP performance of the WEAK formulation is better than that of the STRONG formulation. Note that, we cannot reach the optimum within the CPU time limit in all cases. As for example, for a few instances with $|V(G)| = 40$ and $|E(C)| = 60,000$ CPLEX could not yield optimal solution within one hour for both formulations. On the other hand, one can observe that in overall average the RDS algorithm does not outperform the best performing BIP formulation, i.e. WEAK formulation, solved to optimality with CPLEX (671.47 secs. vs 316.91 secs). However, in 15 out of 27 cases, which are indicated in boldface, the RDS algorithm runs faster than CPLEX.

10.4.3. The Effect of the Branching Rules

According to our preliminary experiments, we have observed that, among all combination of subproblem selection rules, i.e. depth first search (DFS), breadth first search (BFS), smallest lower bound (SLB), largest lower bound (LLB), branching rules, i.e. conflicting pair branching, conflicting edge branching, conflicting edge-pair branching, clique-0 branching, clique-1 branching, and conflicting pair selection rules, i.e. largest number of conflicts and strong branching, we obtain the best results in terms of both accuracy and efficiency when DFS subproblem selection rule is applied together with the conflicting edge branching and strong branching strategy. Therefore, we do not report the results obtained with other combinations of these rules.

In Table 10.20 we present the effect of three straightforward branching rules, i.e. conflicting edge branching rule, conflicting pair branching rule and conflicting edge-pair branching, on the performance of the BB algorithm. Columns 3 – 6 are for the results with conflicting edge branching, columns 7 – 11 include the results obtained with conflicting pair branching rule and the last four columns report the results with conflicting edge-pair branching rule. Recall that, in all three cases we use DFS to select subproblems from the active node list $\mathcal{L}$ and strong branching for selecting conflicting pairs. Recall that the columns 'Explored Nodes' list the number of subproblems processed by the BB algorithm. The columns 'Active Nodes' include the number of unprocessed problems when the BB algorithm stops. Notice that when the BB algorithm stops with 0 number of active nodes then the optimum is reached. When we consider the conflicting edge branching rule, we observe that in all cases, except for $|V(G)| = 100$ and $|E(C)| = 350,000$, the optimum is reached. Therefore, we can conclude that conflicting edge branching rule outperforms the other two rules in terms of both accuracy and efficiency.

Table 10.21 includes the results obtained when we apply clique branching rules. Columns 3 – 7 incorporate the results obtained with clique-0 branching rule and columns 8 – 12 report the results obtained with clique-1 branching rule. In the column under 'Cliques' we report the number of maximal cliques of the conflict graph $C = (V(C), E(C))$ generated during the BB algorithm for clique branching. To this end, at each active node of the BB tree, i.e. a problem from $\mathcal{L}$, we perform the greedy heuristic described in Figure 8.1 which detects a maximal cardinality clique including the selected conflicting edge pair. When the size of the maximal clique is larger than or equal to three we apply one of the clique branching rules, for the other case we consider conflicting edge branching rule. As it can be observed from Table 10.21, the clique branching rules do not perform better than conflicting edge branching rule.

Recall that, in the computational experiments summarized in Table 10.20 – Table 10.22 we have employed the strong branching for the conflicting pair selection rule. Next, Table 10.23 includes the results obtained with another conflict pair selection rule, i.e. the largest number of conflicts rule. To be precise, columns 4 – 7 are for the results obtained with strong branching rule and columns 8 – 11 incorporate results with the largest number of conflicts rule. As it can be observed from Table 10.23, strong branching arises to be more efficient than the largest number of conflicts (262.05 secs. vs. 799.09 secs. average CPU times). Moreover, in 24 out of 135 test instances we could not reach the optimum with the largest number of conflicts. However, this value is 5 out of 135 for the strong branching.

10.4.4. The Effect of the Subproblem Selection Rules

In Table 10.22 we present the results obtained with four subproblem selection rules together with the conflicting edge branching rule and strong branching strategy. Columns 3 – 6 are for the DFS rule, columns 7 – 10 incorporate the results with the BFS rule, the columns 11 – 14 include the results with SLB and the last four columns report the results with the LLB. As it can be observed from Table 10.22, with the BFS rule the BB algorithm cannot find an optimal solution in 8 out of 27 cases within the CPU time limit of one hour. Similarly, when SLB (LLB) is applied we cannot reach the optimum for 10 (1) cases. Considering the percentage gap from the optimum value, we can say that both DFS and LLB outperform the other two subproblem selection strategies. However, when we take into account the CPU times, DFS arises to be the best subproblem selection strategy. As a final remark, we should point that when we do not consider the case $|V(G)| = 100$ and $|E(C)| = 350,000$, the overall average CPU time requirement with DFS decreases down to 133.67 secs. while this value is 559.21 secs. for the LLB.

10.4.5. The Effect of the Initial Upper Bound

Considering the outputs reported in Table 10.20 – Table 10.22 and Table 10.23, we can say that it is reasonable to proceed with the combination of DFS, conflicting edge branching and strong branching strategies. As we have already discussed in Section 3.1.1 the quality of the initial upper bound value considerably affects the overall performance of the BB algorithm. Table 10.24 includes the experimental results obtained with the initial upper bound $\overline{z}$ which is set to the solution value calculated by the local search algorithm introduced in Section 3.1.1. and with the initial upper bound $\overline{z}$, which is set to an arbitrarily large value. In columns 3 – 4 we give the results obtained with the local search algorithm. Recall that, when local search finds a feasible solution then the current solution is perturbed in order to search other regions of the solution space.

The number of perturbations is set to $|V(G)|$. Columns 5 – 8 (9 – 12) include results when BB algorithm starts with finite initial upper bound our local search heuristic computes (an arbitrarily large value). As it can be observed, the efficiency of the BB algorithm depends on the quality of the initial upper bound. Observe that, the average CPU time requirement decreases from 296.58 secs. to 262.05 secs. when local search is run to obtain an initial upper bound value $\overline{z}$ for the BB algorithm. Note that the computation times required by the local search heuristic are not taken into account in the CPU times of the BB algorithms, which we report in the corresponding 'CPU' columns.

10.4.6. The Effect of Probing

Next, in Table 10.25, we report the results on the effect of probing on the performance of the BB algorithm. The columns 'Probings' stand for the number of nodes where probing has been applied. Columns 3 – 7 incorporate the results obtained when probing has been applied in the first 100 iterations of the BB algorithm. Columns 8 – 12 (13 – 17) are for the results when probing has been applied in the first $1,000$ $(10,000)$ iterations. As it can be observed from Table 10.25, when probing is applied in the first 100 iterations, a slight decline occurs in the efficiency of the BB algorithm compared to the results reported in columns 3 – 6 of Table 10.23, (262.05 vs. secs. to 285.34 secs. average CPU times). However, when we apply for the first $1,000$ iterations or more, it seems that the efficiency of the BB algorithm deteriorates even further, which can be attributed to the excessive CPU time requirement of the probing procedure.

10.4.7. The Effect of Pegging

Finally, Table 10.26 summarizes the results obtained when pegging heuristic is performed within the BB algorithm and the initial upper bound $\overline{z}$ is set to infinity. The columns 'Peggings' stand for the number of edges in $E(G)$ pegged during the run of the BB algorithm. The columns 3 – 5 stand for the best results obtained with two BIP formulations, i.e. WEAK and STRONG formulations. Columns 6 – 10 include results obtained when pegging heuristic is called within the BB algorithm. Note that the results reported in the last five columns are the best among all results obtained with the BB algorithms presented so far. As it can be observed, the performance of the BB algorithm considerably improves when we employ pegging heuristic together with the optimum initial upper bound values. Namely, the overall average CPU time of the BB with pegging is significantly shorter (247.31 secs. on the average) than the CPU times reported with the Best of BIP Models (i.e. 316.91 secs. on the average). Notice also that BB with pegging is the winner in 21 out of 27 cases.

Table 10.19. Performance of the formulations and the RDS algorithm.

			WEAK formulation - BIP			STRONG formulation - BIP			RDS		
$\|V(G)\|$	$\|E(C)\|$	OPT	Best Sol.	Nodes	CPU	Best Sol.	Nodes	CPU	Best UB	Explored Nodes	CPU
15	5000	2246.2	0.00	12006.0	13.2	0.00	1883.8	50.7	0.00	168330.0	3.6
20	10000	2450.8	0.00	120962.4	130.1	0.00	26721.4	167.9	0.00	362017.2	13.1
30	20000	3247	0.00	9395.8	25.8	0.00	3678.8	33.2	0.00	123854.4	11.5
30	30000	3355.8	0.00	479938.2	1650.3	0.00	158111.0	2471.6	0.00	7419390.0	637.4
40	40000	4205	0.00	3731.0	12.2	0.00	1065.0	18.2	0.00	141223.6	24.7
40	60000	4283.4	0.01	702376.2	3508.5	0.11	131268.0	3600.1	0.29	22953790.0	3402.9
50	50000	5183.4	0.00	734.6	7.6	0.00	204.2	7.1	0.00	127559.2	39.4
50	60000	5186.2	0.00	2118.2	14.9	0.00	422.6	15.4	0.00	715812.8	197.5
60	80000	6163.6	0.00	576.8	10.6	0.00	320.0	11.8	0.00	166867.8	77.6
70	100000	7156.8	0.00	85.6	9.2	0.00	0.0	9.7	0.00	74048.2	54.7
70	150000	7166.8	0.00	6065.4	46.7	0.00	1910.6	80.8	0.01	3024983.4	2070.7
80	200000	8154.2	0.00	1757.0	34.4	0.00	753.2	51.6	0.05	2709098.6	2667.9
90	250000	9157.2	0.00	5669.0	68.5	0.00	1876.6	181.4	0.07	2783784.4	3599.6
100	100000	10118.8	0.00	0.0	7.8	0.00	0.0	10.6	0.00	2966.6	6.5
100	250000	10144	0.00	727.6	23.9	0.00	109.0	32.1	0.01	682440.8	1242.8
100	350000	10160	0.00	10669.0	149.9	0.00	3821.8	421.2	0.42	2746598.4	3599.4
150	200000	15093.6	0.00	0.0	15.3	0.00	0.0	20.9	0.00	169.0	1.3
150	350000	15102.2	0.00	0.0	22.7	0.00	0.0	34.2	0.00	17191.4	117.5
150	500000	15105.8	0.00	0.0	33.8	0.00	0.0	52.0	0.00	47670.8	337.1
200	200000	20077.4	0.00	0.0	39.3	0.00	0.0	43.4	0.00	24.0	0.4
200	400000	20082.8	0.00	0.0	46.5	0.00	0.0	59.1	0.00	372.8	6.5
250	500000	25058.4	0.00	0.0	99.5	0.00	0.0	115.8	0.00	196.2	6.6
250	700000	25057.8	0.00	0.0	106.8	0.00	0.0	131.6	0.00	84.4	2.9
300	100000	30042.4	0.00	0.0	184.1	0.00	0.0	177.9	0.00	3.0	0.2
300	300000	30040.6	0.00	0.0	188.6	0.00	0.0	195.0	0.00	7.0	0.6
400	200000	40016.6	0.00	0.0	608.8	0.00	0.0	591.1	0.00	2.4	0.3
500	200000	50006.2	0.00	0.0	1497.3	0.00	0.0	1437.9	0.00	2.0	6.9
Average		14224.56	0.001	50252.33	316.91	0.004	12301.70	371.19	0.03	1639573.64	671.47

Table 10.20. Performance of the straightforward branching rules

		DFS (conflicting edge branching)				DFS (conflicting pair branching)				DFS (conflicting edge- pair branching)			
		Best	Explored	Active	CPU	Best	Explored	Active	CPU	Best	Explored	Active	CPU
$\|V(G)\|$	$\|E(C)\|$	UB	Nodes	Nodes		UB	Nodes	Nodes		UB	Nodes	Nodes	
15	5000	0.00	81865.8	0.0	1.7	6.85	130143615.8	95.2	3600.0	0.00	81752.2	0.00	2.1
20	10000	0.00	172717.4	0.0	5.0	6.09	86389696	77.6	3600.0	0.00	289275.8	0.00	9.8
30	20000	0.00	51022.4	0.0	2.8	1.62	48490354.2	37.8	3600.0	0.00	148632.6	0.00	9.7
30	30000	0.00	1831427.6	0.0	120.9	2.81	41144032.6	69.2	3600.0	0.00	3300242.8	0.00	222.5
40	40000	0.00	30318.2	0.0	2.7	1.95	26562379.6	53	3600.0	0.00	286967.4	0.00	33.6
40	60000	0.00	11186573.4	0.0	1186.0	2.63	23593288.8	90.6	3600.0	0.00	12981901	0.00	1299.6
50	50000	0.00	20153.4	0.0	2.6	0.92	17748935.2	36.4	3600.0	0.00	164739.6	0.00	26.1
50	60000	0.00	24564.2	0.0	3.7	1.60	17026719.4	62.6	3600.0	0.00	680230.8	0.00	105.1
60	80000	0.00	22308.2	0.0	4.9	1.39	12061340.8	61.4	3600.0	0.00	888075.6	0.00	195.6
70	100000	0.00	7498.8	0.0	2.2	0.40	6598855.6	25.0	2624.8	0.00	389519.8	0.00	118.0
70	150000	0.00	301795.8	0.0	93.8	1.76	8233584.2	122	3600.0	0.00	6615412.2	0.00	2025.3
80	200000	0.00	262232.2	0.0	104.1	0.49	6826339.6	45.4	3600.0	0.21	6877559	17.40	2998.7
90	250000	0.00	2545169.4	0.0	1370.2	0.47	4861258.4	64.4	3600.0	0.24	6630563.4	21.00	3428.9
100	100000	0.00	931.8	0.0	0.6	0.00	8570.8	0.0	6.7	0.00	18823.6	0.00	12.4
100	250000	0.00	986857.4	0.0	536.3	0.27	3494522.6	41.8	3017.1	0.07	3510772.4	12.00	2246.9
100	350000	0.11	5676555.2	13.2	3600.0	0.39	3722199.4	71.4	3600.0	0.37	5544825.6	35.80	3600.0
150	200000	0.00	101.8	0.0	0.2	0.00	291.8	0.0	0.7	0.00	3059	0.00	5.0
150	350000	0.00	2555.2	0.0	3.9	0.00	23220	0.0	47.0	0.00	131993	0.00	207.9
150	500000	0.00	21718.6	0.0	29.5	0.00	456896.8	0.0	958.9	0.03	957757.8	5.20	1437.5
200	200000	0.00	17.8	0.0	0.1	0.00	22.8	0.0	0.1	0.00	116.8	0.00	0.4
200	400000	0.00	195.6	0.0	0.9	0.00	1579.2	0.0	6.8	0.00	2492	0.00	8.5
250	500000	0.00	103.6	0.0	0.7	0.00	231.2	0.0	2.1	0.00	683	0.00	4.3
250	700000	0.00	350.2	0.0	2.2	0.00	522.6	0.0	4.1	0.00	3197.4	0.00	17.8
300	100000	0.00	3.2	0.0	0.1	0.00	3.8	0.0	0.1	0.00	6.6	0.00	0.1
300	300000	0.00	5.6	0.0	0.1	0.00	7.4	0.0	0.1	0.00	12	0.00	0.1
400	200000	0.00	2.0	0.0	0.1	0.00	3.0	0.0	0.2	0.00	5.8	0.00	0.1
500	200000	0.00	2.0	0.0	0.1	0.00	2.4	0.0	0.2	0.00	2.8	0.00	0.1
Average		0.00	860260.99	0.49	262.05	1.10	16199573.11	35.33	1980.34	0.03	1833652.58	3.39	667.26

Table 10.21. Performance of the clique branching rules

		DFS (*clique-0* branching)					DFS (*clique-1* branching)				
		Best	Explored	Active	Cliques	CPU	Best	Explored	Active	Cliques	CPU
$\lvert V(G)\rvert$	$\lvert E(C)\rvert$	UB	Nodes	Nodes			UB	Nodes	Nodes		
15	5000	0.00	888241.2	0	100.8	18.8	0.00	92318.8	0	92.6	2.0
20	10000	0.00	2994138	0	238.4	94.2	0.00	280856.2	0	218.8	9.4
30	20000	0.00	219244.4	0	606	11.7	0.00	95883.2	0	582.2	5.7
30	30000	0.07	32531410.8	2.8	653.8	1955.3	0.00	3289276.2	0	627.4	212.0
40	40000	0.00	350193.2	0	835	33.6	0.00	119034.2	0	763.6	13.3
40	60000	1.82	35058144.2	36.4	1165.6	3600.0	0.11	23974095	7.4	1235	2669.5
50	50000	0.00	101476.6	0	314	15.7	0.00	100067.4	0	330	17.2
50	60000	0.00	290995.6	0	616.8	43.9	0.00	322491	0	617.8	54.6
60	80000	0.00	247993.2	0	346.4	55.8	0.00	352717.8	0	366.2	88.0
70	100000	0.00	47053.6	0	125.2	14.5	0.00	66675.8	0	126	24.1
70	150000	0.00	2306008	0	829.6	715.3	0.00	2117654.2	0	821.8	742.2
80	200000	0.00	1483595.4	0	629.2	683.1	0.00	1721760	0	604.8	870.9
90	250000	0.13	5102471	12.4	501.2	2964.8	0.18	4885938.8	13.2	513.4	2954.7
100	100000	0.00	4235.4	0	9.6	3.0	0.00	5235.6	0	9.8	4.1
100	250000	0.01	2526616	1.8	178	1651.8	0.01	2116577.8	2.2	174.8	1488.8
100	350000	0.28	5420215.4	25.8	433	3600.0	0.35	4741403.6	23.4	462.4	3600.0
150	200000	0.00	555	0	0.6	1.3	0.00	556	0	0.6	1.4
150	350000	0.00	8320.4	0	6.4	17.9	0.00	10687.2	0	6.6	24.3
150	500000	0.00	42078.8	0	23	85.8	0.00	47036.2	0	22.4	101.5
200	200000	0.00	38.2	0	0.2	0.8	0.00	40	0	0.2	0.3
200	400000	0.00	664.8	0	0.4	3.4	0.00	656.6	0	0.4	3.4
250	500000	0.00	228.6	0	0.2	2.1	0.00	217.4	0	0.2	2.3
250	700000	0.00	845	0	0	8.3	0.00	845	0	0	9.0
300	100000	0.00	4.8	0	0	0.1	0.00	4.8	0	0	0.1
300	300000	0.00	11.2	0	0	0.2	0.00	11.2	0	0	0.2
400	200000	0.00	4.6	0	0	0.2	0.00	4.6	0	0	0.2
500	200000	0.00	2	0	0	0.2	0.00	2	0	0	0.2
Average		0.09	3319436.50	2.93	281.98	577.11	0.02	1642298.03	1.71	280.63	477.75

Table 10.22. Performance of the subproblem selection rules

		DFS (conflicting edge)				BFS (conflicting edge)				SLB (conflicting edge)				LLB (conflicting edge)			
$\|V(G)\|$	$\|E(C)\|$	Best UB	Explored Nodes	Active Nodes	CPU	Best UB	Explored Nodes	Active Nodes	CPU	Best UB	Explored Nodes	Active Nodes	CPU	Best UB	Explored Nodes	Active Nodes	CPU
15	5000	0.00	81865.8	0.0	1.7	0.00	84418	0.0	3.1	0.00	83181.6	0.0	74.8	0.0	92908.2	0.0	2.0
20	10000	0.00	172717.4	0.0	5.0	0.00	185606.4	0.0	9.9	0.00	177007.8	0.0	1228.9	0.0	211629.4	0.0	6.6
30	20000	0.00	51022.4	0.0	2.8	0.00	92894.6	0.0	9.7	0.00	49278.8	0.0	87.0	0.0	74318.8	0.0	4.4
30	30000	0.00	1831427.6	0.0	120.9	0.00	2425962	0.0	387.8	na	na	na	na	0.0	2508678.6	0.0	165.7
40	40000	0.00	30318.2	0.0	2.7	0.00	88626.6	0.0	8.1	0.00	25843.6	0.0	26.3	0.0	64263.8	0.0	6.4
40	60000	0.00	11186573.4	0.0	1186.0	na	na	na	na	na	na	na	na	0.0	14126296	0.0	3009.5
50	50000	0.00	20153.4	0.0	2.6	0.00	63957.6	0.0	7.1	0.00	22126.8	0.0	46.8	0.0	53439.8	0.0	8.0
50	60000	0.00	24564.2	0.0	3.7	0.00	113402.4	0.0	19.8	0.00	23726.4	0.0	34.7	0.0	176049	0.0	26.4
60	80000	0.00	22308.2	0.0	4.9	na	na	na	na	0.00	17876	0.0	19.7	0.0	101777.8	0.0	21.8
70	100000	0.00	7498.8	0.0	2.2	0.00	37719.2	0.0	10.14	0.00	6860.4	0.0	5.0	0.0	36856	0.0	11.3
70	150000	0.00	301795.8	0.0	93.8	na	na	na	na	na	na	na	na	0.0	1412460.2	0.0	435.5
80	200000	0.00	262232.2	0.0	104.1	na	na	na	na	na	na	na	na	0.0	851302.8	0.0	375.3
90	250000	0.00	2545169.4	0.0	1370.2	na	na	na	na	na	na	na	na	0.0	13930867	0.0	8898.9
100	100000	0.00	931.8	0.0	0.6	0.00	2412	0.0	1.26	0.00	595.2	0.0	0.4	0.0	2670.4	0.0	2.0
100	250000	0.00	986857.4	0.0	536.3	na	na	na	na	na	na	na	na	0.0	2671102.6	0.0	1492.2
100	350000	0.11	5676555.2	13.2	3600.0	na	na	na	na	na	na	na	na	na	na	na	na
150	200000	0.00	101.8	0.0	0.2	0.00	615.2	0.0	1.0	na	na	na	na	0.0	555	0.0	1.1
150	350000	0.00	2555.2	0.0	3.9	0.01	7491	0.0	11.6	na	na	na	na	0.0	7693	0.0	13.5
150	500000	0.00	21718.6	0.0	29.5	na	na	na	na	na	na	na	na	0.0	33956.6	0.0	56.5
200	200000	0.00	17.8	0.0	0.1	0.00	28.8	0.0	0.2	0.00	16.6	0.0	0.1	0.0	38.2	0.0	0.2
200	400000	0.00	195.6	0.0	0.9	0.00	520.2	0.0	1.9	0.00	150.2	0.0	0.6	0.0	665.2	0.0	2.6
250	500000	0.00	103.6	0.0	0.7	0.00	298.4	0.0	2.0	0.00	119	0.0	0.9	0.0	227.8	0.0	1.6
250	700000	0.00	350.2	0.0	2.2	0.00	921.4	0.0	6.3	0.00	394.2	0.0	2.7	0.0	845	0.0	5.6
300	100000	0.00	3.2	0.0	0.1	0.00	4.4	0.0	0.1	0.00	3.8	0.0	0.1	0.0	4.8	0.0	0.1
300	300000	0.00	5.6	0.0	0.1	0.00	11	0.0	0.2	0.00	7.6	0.0	0.2	0.0	11.2	0.0	0.2
400	200000	0.00	2.0	0.0	0.1	0.00	4.2	0.0	0.2	0.00	3	0.0	0.2	0.0	4.6	0.0	0.1
500	200000	0.00	2.0	0.0	0.1	0.00	2	0.0	0.2	0.00	2	0.0	0.2	0.0	2	0.0	0.1
Average		0.00	860260.99	0.49	262.05	0.00	163415.54	0.00	25.29	0.00	23952.53	0.00	89.91	0.00	1398408.58	0.00	559.51

Table 10.23. Performance of the conflicting pair selection rules

		Strong branching				Largest Number of Conflicts			
		Best	Explored	Active	CPU	Best	Explored	Active	CPU
$\|V(G)\|$	$\|E(C)\|$	UB	Nodes	Nodes		UB	Nodes	Nodes	
15	5000	0.00	81865.8	0.0	1.7	0.00	137638.6	0.0	2.2
20	10000	0.00	172717.4	0.0	5.0	0.00	639306.8	0.0	15.2
30	20000	0.00	51022.4	0.0	2.8	0.00	360271.8	0.0	18.0
30	30000	0.00	1831427.6	0.0	120.9	0.00	34849524	1.4	1736.3
40	40000	0.00	30318.2	0.0	2.7	0.00	545585.6	0.0	50.9
40	60000	0.00	11186573.4	0.0	1186.0	0.62	40129056	9.6	3600.0
50	50000	0.00	20153.4	0.0	2.6	0.00	386157.4	0.0	55.4
50	60000	0.00	24564.2	0.0	3.7	0.00	1835024.2	0.0	266.8
60	80000	0.00	22308.2	0.0	4.9	0.00	1144183.4	0.0	253.4
70	100000	0.00	7498.8	0.0	2.2	0.00	491952.8	0.0	163.5
70	150000	0.00	301795.8	0.0	93.8	0.04	8318581.8	4.4	2581.2
80	200000	0.00	262232.2	0.0	104.1	0.28	8431669.8	16.6	3600.0
90	250000	0.00	2545169.4	0.0	1370.2	0.18	6703573.4	14.6	3600.0
100	100000	0.00	931.8	0.0	0.6	0.00	13432.2	0.0	10.6
100	250000	0.00	986857.4	0.0	536.3	0.00	2053012	3.8	1375.1
100	350000	0.11	5676555.2	13.2	3600.0	0.30	5761404.6	22.4	3600.0
150	200000	0.00	101.8	0.0	0.2	0.00	526.4	0.0	1.3
150	350000	0.00	2555.2	0.0	3.9	0.00	38138.2	0.0	85.6
150	500000	0.00	21718.6	0.0	29.5	0.00	204677.4	0.0	469.3
200	200000	0.00	17.8	0.0	0.1	0.00	45.4	0.0	0.2
200	400000	0.00	195.6	0.0	0.9	0.00	1996.8	0.0	10.9
250	500000	0.00	103.6	0.0	0.7	0.00	1436.8	0.0	11.8
250	700000	0.00	350.2	0.0	2.2	0.00	6718.4	0.0	67.1
300	100000	0.00	3.2	0.0	0.1	0.00	4.2	0.0	0.1
300	300000	0.00	5.6	0.0	0.1	0.00	19	0.0	0.3
400	200000	0.00	2.0	0.0	0.1	0.00	7.2	0.0	0.2
500	200000	0.00	2.0	0.0	0.1	0.00	2.4	0.0	0.1
Average		0.00	860260.99	0.49	262.05	0.05	4150146.18	2.70	799.09

Table 10.24. The effect of the initial upper bound on the performance of the BB algorithm

		Local Search		DFS - (conflicting edge) with initial $\overline{z} < \infty$				DFS - (conflicting edge) with initial $\overline{z} = \infty$			
		Best	CPU	Best	Explored	Active	CPU	Best	Explored	Active	CPU
$\lvert V(G)\rvert$	$\lvert E(C)\rvert$	UB		UB	Nodes	Nodes		UB	Nodes	Nodes	
15	5000	1.81	0.7	0.00	81865.8	0.0	1.7	0.00	87930.8	0.0	2.2
20	10000	1.99	0.8	0.00	172717.4	0.0	5.0	0.00	230899.2	0.0	7.6
30	20000	2.13	0.9	0.00	51022.4	0.0	2.8	0.00	67891.6	0.0	4.4
30	30000	2.18	1.1	0.00	1831427.6	0.0	120.9	0.00	2393118.4	0.0	161.4
40	40000	2.17	1.8	0.00	30318.2	0.0	2.7	0.00	44561.4	0.0	4.4
40	60000	2.42	2.2	0.00	11186573.4	0.0	1186.0	0.00	11444366.6	0.0	1342.5
50	50000	2.10	2.5	0.00	20153.4	0.0	2.6	0.00	30142.6	0.0	4.3
50	60000	2.22	2.9	0.00	24564.2	0.0	3.7	0.00	31976.2	0.0	4.9
60	80000	2.25	3.7	0.00	22308.2	0.0	4.9	0.00	32446.6	0.0	6.8
70	100000	2.31	3.9	0.00	7498.8	0.0	2.2	0.00	11258.0	0.0	3.4
70	150000	2.44	4.1	0.00	301795.8	0.0	93.8	0.00	438974.6	0.0	131.8
80	200000	2.43	4.3	0.00	262232.2	0.0	104.1	0.00	446658.6	0.0	186.5
90	250000	2.45	4.3	0.00	2545169.4	0.0	1370.2	0.00	3049306.0	0.0	1603.2
100	100000	2.47	4.2	0.00	931.8	0.0	0.6	0.00	1624.0	0.0	1.2
100	250000	2.49	4.5	0.00	986857.4	0.0	536.3	0.01	1580952.4	1.6	885.3
100	350000	2.61	4.7	0.11	5676555.2	13.2	3600.0	0.11	6168610.6	16.4	3600.0
150	200000	2.87	5.0	0.00	101.8	0.0	0.2	0.00	204.4	0.0	0.4
150	350000	2.93	5.1	0.00	2555.2	0.0	3.9	0.00	4844.2	0.0	7.9
150	500000	3.16	6.5	0.00	21718.6	0.0	29.5	0.00	27993.6	0.0	43.4
200	200000	3.54	8.2	0.00	17.8	0.0	0.1	0.00	30.6	0.0	0.2
200	400000	3.68	8.6	0.00	195.6	0.0	0.9	0.00	472.6	0.0	1.7
250	500000	4.08	10.5	0.00	103.6	0.0	0.7	0.00	143.8	0.0	1.0
250	700000	4.33	11.3	0.00	350.2	0.0	2.2	0.00	405.6	0.0	2.6
300	100000	4.49	12.8	0.00	3.2	0.0	0.1	0.00	3.8	0.0	0.1
300	300000	4.72	16.0	0.00	5.6	0.0	0.1	0.00	7.6	0.0	0.2
400	200000	5.16	20.5	0.00	2.0	0.0	0.1	0.00	3.0	0.0	0.2
500	200000	7.12	23.7	0.00	2.0	0.0	0.1	0.00	2.0	0.0	0.2
Average		3.06	6.48	0.00	860260.99	0.49	262.05	0.00	966475.14	0.67	296.58

Table 10.25. The effect of the probing on the performance of the BB algorithm

		DFS - (conflicting edge) with probing (first 100 iterations)					DFS - (conflicting edge) with probing (first 1000 iterations)					DFS - (conflicting edge) with probing (first 10000 iterations)				
$\|V(G)\|$	$\|E(C)\|$	Best UB	Explored Nodes	Active Nodes	Probings	CPU	Best UB	Explored nodes	Active Nodes	Probings	CPU	Best UB	Explored nodes	Active Nodes	Probings	CPU
15	5000	0.00	86717.6	0.0	188	1.89	0.00	81989.8	0.0	1860.8	1.85	0.00	38890	0.0	18402	1.42
20	10000	0.00	224101.4	0.0	182.8	6.92	0.00	222982.2	0.0	1763.2	6.96	0.00	216920.6	0.0	17069	7.40
30	20000	0.00	62308.4	0.0	168	3.73	0.00	62308.4	0.0	1614.8	3.97	0.00	62308.4	0.0	15996	7.92
30	30000	0.00	2311662	0.0	162.8	144.93	0.00	2311643.4	0.0	1622	135.26	0.00	2311519.6	0.0	16096	152.91
40	40000	0.00	43641.4	0.0	173.6	4.38	0.00	43641.4	0.0	1649.6	4.70	0.00	43641.4	0.0	15978	17.16
40	60000	0.00	11356356	0.0	164	1145.60	0.00	11356356	0.0	1612.8	1125.17	0.00	11356354	0.0	15945	1194.67
50	50000	0.00	30131	0.0	164.4	4.42	0.00	30131	0.0	1594.4	5.72	0.00	30131	0.0	15508	25.66
50	60000	0.00	31770.2	0.0	171.6	4.80	0.00	31770.2	0.0	1614.4	5.83	0.00	31770.2	0.0	15063	25.83
60	80000	0.00	32333.4	0.0	166	6.79	0.00	32333.4	0.0	1570	8.90	0.00	32333.4	0.0	15256	41.69
70	100000	0.00	11231.2	0.0	157.6	3.57	0.00	11231.2	0.0	1540.4	7.33	0.00	11231.2	0.0	10382	39.37
70	150000	0.00	438340.6	0.0	172.8	123.85	0.00	438340.6	0.0	1577.2	125.65	0.00	438340.6	0.0	15306	173.69
80	200000	0.00	440119.2	0.0	161.6	174.66	0.00	440119.2	0.0	1512.8	178.62	0.00	440119.2	0.0	14756	258.11
90	250000	0.00	2980119	0.0	160.8	1495.29	0.00	2980119	0.0	1463.6	1483.17	0.00	2980119	0.0	14474	1667.35
100	100000	0.00	1624	0.0	147.2	2.25	0.00	1624	0.0	1348.4	13.12	0.00	1624	0.0	2201.2	22.66
100	250000	0.00	1607065.4	2.4	161.2	865.77	0.00	1887626	0.0	1450.4	1027.32	0.01	1475896.6	2.6	14166	978.79
100	350000	0.11	6112040.4	16.8	157.2	3600.00	0.11	5864235	16.2	1524.2	3600.00	0.11	5788123.2	16	14294	3600.00
150	200000	0.00	204.4	0.0	104	3.50	0.00	204.4	0.0	234.8	7.67	0.00	204.4	0.0	234.8	7.23
150	350000	0.00	4844.2	0.0	128.4	11.83	0.00	4844.2	0.0	1216	45.69	0.00	4844.2	0.0	5881.2	185.24
150	500000	0.00	27993.6	0.0	152	47.92	0.00	27993.6	0.0	1291.6	80.28	0.00	27993.6	0.0	11637	379.37
200	200000	0.00	30.6	0.0	31.6	2.41	0.00	30.6	0.0	31.6	2.39	0.00	30.6	0.0	31.6	2.27
200	400000	0.00	472.6	0.0	116	10.05	0.00	472.6	0.0	505.6	38.50	0.00	472.6	0.0	516	37.19
250	500000	0.00	143.8	0.0	101.2	16.41	0.00	143.8	0.0	151.6	24.12	0.00	143.8	0.0	151.6	22.91
250	700000	0.00	405.6	0.0	100.8	18.08	0.00	405.6	0.0	426.8	68.63	0.00	405.6	0.0	426.8	65.29
300	100000	0.00	3.8	0.0	2.8	0.73	0.00	3.8	0.0	2.8	0.77	0.00	3.8	0.0	2.8	0.70
300	300000	0.00	7.6	0.0	6.8	1.73	0.00	7.6	0.0	6.8	1.76	0.00	7.6	0.0	6.8	1.66
400	200000	0.00	3	0.0	2	1.21	0.00	3	0.0	2	1.26	0.00	3	0.0	2	1.16
500	200000	0.00	2	0.0	1.2	1.44	0.00	2	0.0	1.2	1.51	0.00	2	0.0	1.2	1.39
Average		0.00	955691.59	0.71	126.16	285.34	0.00	956687.50	0.60	1081.10	296.52	0.00	936793.84	0.69	9251.23	330.33

Table 10.26. The effect of the pegging on the performance of the BB algorithm

		Best of BIP formulations			DFS- (conflicting edge) with pegging				
$\lvert V(G)\rvert$	$\lvert E(C)\rvert$	Best Solution	Nodes	CPU	Best UB	Explored Nodes	Active Nodes	Peggings	CPU
15	5000	0.00	12006.0	13.2	0.00	67531.4	0.0	53316.8	1.4
20	10000	0.00	120962.4	130.1	0.00	207696.4	0.0	208625.4	6.3
30	20000	0.00	9395.8	25.8	0.00	54125.6	0.0	55941.8	3.0
30	30000	0.00	479938.2	1650.3	0.00	2067364	0.0	2620380.8	129.0
40	40000	0.00	3731.0	12.2	0.00	35776.6	0.0	36538	3.5
40	60000	0.01	702376.2	3508.5	0.00	9826476	0.0	13191069.4	1047.4
50	50000	0.00	734.6	7.6	0.00	22190.2	0.0	18855.4	3.3
50	60000	0.00	2118.2	14.9	0.00	25511.2	0.0	24050.4	3.9
60	80000	0.00	576.8	10.6	0.00	26080	0.0	21758.4	5.6
70	100000	0.00	85.6	9.2	0.00	8344.4	0.0	5048	2.6
70	150000	0.00	6065.4	46.7	0.00	335769	0.0	301437	104.4
80	200000	0.00	1757.0	34.4	0.00	329316.8	0.0	275858.4	141.0
90	250000	0.00	5669.0	68.5	0.00	2054668	0.0	1905472.2	1126.9
100	100000	0.00	0.0	7.8	0.00	1326.8	0.0	426	0.9
100	250000	0.00	727.6	23.9	0.00	1408123	0.0	1080180	875.5
100	350000	0.00	10669.0	149.9	0.00	44831879	0.0	3730789.42	3171.5
150	200000	0.00	0.0	15.3	0.00	201.2	0.0	17.2	0.4
150	350000	0.00	0.0	22.7	0.00	4129.8	0.0	831.6	7.1
150	500000	0.00	0.0	33.8	0.00	23269.6	0.0	4011.6	38.1
200	200000	0.00	0.0	39.3	0.00	30.4	0.0	0.8	0.1
200	400000	0.00	0.0	46.5	0.00	455.4	0.0	11	1.7
250	500000	0.00	0.0	99.5	0.00	134.2	0.0	2.2	0.9
250	700000	0.00	0.0	106.8	0.00	393	0.0	8.8	2.5
300	100000	0.00	0.0	184.1	0.00	3.8	0.0	0.0	0.1
300	300000	0.00	0.0	188.6	0.00	7.6	0.0	0.0	0.1
400	200000	0.00	0.0	608.8	0.00	3	0.0	0.0	0.0
500	200000	0.00	0.0	1497.3	0.00	2	0.0	0.0	0.0
Average		0.001	50252.33	316.91	0.00	2271511	0.0	871652.986	247.31

11. CONCLUSIONS

This thesis provides a comprehensive study on network flow problems with conflict constraints including the minimum cost noncrossing flow problem (MCNFP), the minimum cost flow problem with conflicts (MCFPC), the maximum flow problem with conflicts (MFPC), and the assignment problem with conflicts (APC). It contributes by giving complexity results, finding properties that make MCNFP and APC polynomially solvable, developing different problem representations and the exact solution algorithms benefiting from the particular underlying structures.

11.1. The Minimum Cost Noncrossing Flow Problem

Initially, we first introduce MCNFP and we show that the problem is NP-hard. Then, we describe polynomially solvable special cases applicable to well-known practical problems such as the quay crane scheduling problem encountered in container terminals: the problem becomes polynomially solvable when the traveling distances are used as the set-up costs, which is a common practice. We also propose mixed-integer linear programming formulations. Finally, we develop a preprocessing scheme and experimentally study its effect on the formulations. Chapter 10 resume the experimental results where we solve the mathematical formulations of MCNFP over 33 instances. When we apply the preprocessing scheme to reduce the problem size prior to solving the problem, 29.48% of the arcs are deleted on the average and the solution time is decreased by at least 20%.

11.2. The Minimum Cost Flow Problem with Conflicts

Initially, we provide a brief complexity analysis of MCFPC and indicates the polynomially solvable cases when the underlying conflict graph has known special structures where the number of maximal stable sets is polynomial and they can be generated in polynomial time. Three alternative ways to formulate MCFPC are given: a strong MILP formulation, a weak MILP formulation and a combinatorial optimization representation. Moreover, a branch-and-bound (BB) and Russian Doll Search (RDS) algorithms are proposed. For BB, a penalty calculation procedure taking advantage of the spanning trees representing the basic feasible solutions is described to apply strong branching, improve relaxation bounds and for pegging the variables. A new and efficient approach of RDS that uses dynamic candidate lists instead of the static ones is introduced. To the best of our knowledge, these are the first reported exact solution methods in the literature to solve MCFPC. They are proven to perform significantly better than the commercial MILP solver CPLEX in terms of CPU times and the quality of bounds. When we compare the optimal solution times, BB and RDS are 31 and 10 times faster than CPLEX (version 12.9), respectively.

11.3. The Maximum Flow Problem with Conflicts

Three alternative MILP formulations using three alternative representations of the conflict constraints and a combinatorial optimization problem representation are given. LP relaxations of each MILP have different strengths. Moreover, new Benders decomposition (BD), BB and RDS algorithms are proposed. Although MFPC is studied before in the literature and proven to be NP-hard, these are the first reported exact solution methods for MFPC. They perform significantly better than CPLEX (version 12.7) does on the test instances with moderate sizes. BB and BD provide the best average optimality gaps for the instances that cannot be solved in one hour-time limit and RDS is reported to be the fastest algorithm to find the optimal solution.

11.4. The Assignment Problem with Conflicts

Finally, we have implemented BB and RDS algorithms for the assignment problem with conflicts (APC), which is known to be strongly NP-hard, and introduced a polynomially solvable case. For our BB algorithm we have suggested five branching rules: conflicting pair branching, conflicting edge branching, conflicting edge-pair branching, clique-0 branching and clique-1 branching. We have also tested a pegging technique and several probing strategies. Among all these combinations, the best results are observed for the conflicting edge branching when combined with depth first search, an initial upper bound heuristic and pegging. According to the extensive computational experiments we can say that, with the best performing combination of several strategies, the proposed BB algorithm is 22 % more efficient than the best performing Binary Integer Programming formulation solved with CPLEX 12.7, on the average. On the other hand, we should point out that although the CPU time requirement of the RDS algorithm does not seem to be satisfactory for middle sized instances, RDS outperforms the BIP formulations solved via CPLEX when the number of vertices is at least 200.

11.5. Potential Future Research

When we consider the studied problems one by one, we should state that MCNFP is very likely to be strongly NP-hard and this might be shown using a strongly NP-hard path problem for the reduction such as the transportation problem with conflicts. A Benders decomposition algorithm can be developed in order to solve the minimum cost flow problem with conflicts by making use of the special structure of the problem. The essentials of the described algorithms can be safely implemented for other problems which are available in the literature and deal with conflict constraints. Particularly, RDS is applicable to any problem that can be reduced to an optimization problem on a graph with hereditary property, stable set problem, for example. Besides, the instances with large number of vertices, high arc density and with low conflict density are not expected to be solved efficiently by any exact solution method mentioned here. Therefore, there is a room for efficient heuristics, which should be considered as a new and promising research direction as well.

REFERENCES

1. "Video story on Sino-Lankan Colombo International Container Terminal", `https://newsin.asia`, accessed at May 2019.

2. Golden, B. L., "A minimum-cost multicommodity network flow problem concerning imports and exports", *Networks*, Vol. 5, No. 4, pp. 331–356, Oct. 1975.

3. Even, S., A. Itai and A. Shamir, "On the complexity of time table and multicommodity flow problems", *16th Annual Symposium on Foundations of Computer Science (sfcs 1975)*, IEEE, Oct. 1975.

4. Desrosiers, J., Y. Dumas, M. M. Solomon and F. Soumis, "Chapter 2 Time constrained routing and scheduling", *Handbooks in Operations Research and Management Science*, pp. 35–139, Elsevier, 1995.

5. Ahuja, R. K. and J. B. Orlin, "A capacity scaling algorithm for the constrained maximum flow problem", *Networks*, Vol. 25, No. 2, pp. 89–98, Mar. 1995.

6. Ahuja, R. K., T. L. Magnanti and J. B. Orlin, *Networks Flows*, New-York: Prentice-Hall, 1993.

7. Wayne, K. D., *Generalized Maximum Flow Algorithms*, Ph.D. Thesis, Cornell University, 1999.

8. Kleinberg, J. M., *Approximation Algorithms for Disjoint Path Problems*, Ph.D. Thesis, Massachusetts Institute of Technology, May 1996.

9. Baier, G., E. Köhler and M. Skutella, "The k-Splittable Flow Problem", *Algorithmica*, Vol. 42, No. 3-4, pp. 231–248, May 2005.

10. Cao, B., "Transportation problem with nonlinear side constraints a branch and bound approach", *Math. Methods Oper. Res. (ZOR)*, Vol. 36, pp. 185–197, 1992.

11. Türkoğulları, Y., Z. C. Taşkın, N. Aras and K. Altınel, "Optimal berth allocation, time-variant quay crane assignment and scheduling with crane setups in container terminals", *European Journal of Operational Research*, Vol. 254, pp. 985–1001, 2016.

12. Altınel, İ. K., N. Aras, Z. Şuvak and Z. C. Taşkın, "Minimum cost noncrossing flow problem on layered networks", *Discrete Applied Mathematics*, Vol. 261, pp. 2–21, May 2019.

13. Şuvak, Z., I. K. Altınel and N. Aras, *Minimum cost flow problem with conflicts*, Tech. Rep. No: FBE-IE-03/2018-03, Department of Industrial Engineering, Boğaziçi University, İstanbul, Turkey, 2018.

14. Şuvak, Z., I. K. Altınel and N. Aras, *Exact Solution Algorithms for the Maximum Flow Problem with Additional Conflict Constraints*, Tech. Rep. No: FBE-IE-02/2019-02, Department of Industrial Engineering, Boğaziçi University, İstanbul, Turkey, 2019.

15. Öncan, T., Z. Şuvak and I. K. Altınel, *Assignment problems with conflicts*, Tech. Rep. No: FBE-IE-03/2016-16, Department of Industrial Engineering, Boğaziçi University, İstanbul, Turkey, 2016.

16. Pferschy, U. and J. Schauer, "The maximum flow problem with disjunctive constraints", *Journal of Combinatorial Optimization*, Vol. 26, pp. 109–119, 2013.

17. Darmann, A., U. Pferschy, J. Schauer and G. J. Woeginger, "Paths, trees and matchings under disjunctive constraints", *Discrete Applied Mathematics*, Vol. 159, pp. 1726–1735, 2011.

18. Diestel, R., *Graph Theory*, Springer-Verlag GmbH, 2017.

19. Ford, L. R., Jr. and D. R. Fulkerson, *Flows in Networks*, New Jersey: Princeton University Press, 1962.

20. Garey, M. R. and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-completeness*, San Francisco: Freeman, 1979.

21. Trukhanov, S., C. Balasubramaniam, B. Balasundaram and S. Butenko, "Algorithms for detecting optimal hereditary structures in graphs, with application to clique relaxations", *Comput Optim Appl*, pp. 56–113, 2013, `https://doi.org/10.1007/s10589-013-9548-5`.

22. Gouveia, L., M. Leitner and M. Ruthmair, "Layered graph approaches for combinatorial optimization problems", *Computers & Operations Research*, Vol. 102, pp. 22–38, Feb. 2019.

23. Lee, D.-H., H. Q. Wang and L. Miao, "Quay crane scheduling with non-interference constraints in port container terminals", *Transportation Research Part E: Logistics and Transportation Review*, Vol. 44, No. 1, pp. 124–135, Jan. 2008.

24. Schrijver, A., "On the history of the transportation and maximum flow problems", *Mathematical Programming*, Vol. 91, No. 3, pp. 437–445, Feb. 2002.

25. Ford, L. R., Jr. and D. R. Fulkerson, "Maximal Flow Through a Network", *Canadian Journal of Mathematics*, Vol. 8, pp. 24–32, 1956.

26. Schrijver, A., *Combinatorial Optimization. Polyhedra and Efficiency*, Volume 24 of Algorithms and Combinatorics, Springer-verlag, Berlin, 2003.

27. Kuhn, H. W., "The Hungarian method for the assignment problem", *Naval Research Logistics Quarterly*, Vol. 2, pp. 83–97, 1955.

28. Ortega, F. and L. A. Wolsey, "A branch-and-cut algorithm for the single-commodity, uncapacitated, fixed-charge network flow problem", *Networks*, Vol. 41, No. 3, pp. 143–158, May 2003.

29. Hewitt, M., G. L. Nemhauser and M. W. P. Savelsbergh, "Combining Exact and Heuristic Approaches for the Capacitated Fixed-Charge Network Flow Problem", *INFORMS Journal on Computing*, Vol. 22, No. 2, pp. 314–325, May 2010.

30. Ahuja, R. K., K. Mehlhorn, J. Orlin and R. E. Tarjan, "Faster algorithms for the shortest path problem", *Journal of the ACM*, Vol. 37, No. 2, pp. 213–223, Apr. 1990.

31. Ford, L. R. and D. R. Fulkerson, "Solving the Transportation Problem", *Management Science*, Vol. 3, No. 1, pp. 24–32, Oct. 1956.

32. Goldberg, A. V. and R. E. Tarjan, "Solving minimum cost flow problem by successive approximation", *Mathematics of Operations Research*, Vol. 15, pp. 430–466, 1990.

33. Pferschy, U. and J. Schauer, "The knapsack problem with conflict graphs", *Journal of Graph Algorithms and Applications*, Vol. 13, pp. 233–249, 2009.

34. Grötschel, M., L. Lovasz and A. Schrijver, *Geometric Algorithms and Combinatorial Optimization*, Springer-Verlag Berlin Heidelberg, 1993.

35. Conforti, M., G. Cornuejols and G. Zambelli, *Integer Programming*, Graduate Texts in Mathematics, Springer International Publishing, 2014.

36. Yamada, T., S. Kataoka and K. Watanabe, "Heuristic and Exact Algorithms for the Disjunctively Constrained Knapsack Problem", *Information Processing Society of Japan Journal*, Vol. 43, No. 9, pp. 2864–2870, Sep. 2002.

37. Hifi, M. and M. Michrafy, "A reactive local search-based algorithm for the disjunctively constrained knapsack problem", *Journal of the Operational Research Society*, Vol. 57, No. 6, pp. 718–726, Jun. 2006.

38. Hifi, M. and M. Michrafy, "Reduction strategies and exact algorithms for the

disjunctively constrained knapsack problem", *Computers & Operations Research*, Vol. 34, No. 9, pp. 2657–2673, Sep. 2007.

39. Akeb, H., M. Hifi and M. E. O. A. Mounir, "Local branching-based algorithms for the disjunctively constrained knapsack problem", *Computers & Industrial Engineering*, Vol. 60, No. 4, pp. 811–820, May 2011.

40. Bettinelli, A., V. Cacchiani and E. Malaguti, "A Branch-and-Bound Algorithm for the Knapsack Problem with Conflict Graph", *INFORMS Journal on Computing*, Vol. 29, No. 3, pp. 457–473, Aug. 2017.

41. Jansen, K. and S. Öhring, "Approximation Algorithms for Time Constrained Scheduling", *Information and Computation*, Vol. 132, No. 2, pp. 85–108, Feb. 1997.

42. Jansen, K., "An Approximation Scheme for Bin Packing with Conflicts", *Journal of Combinatorial Optimization*, Vol. 3, No. 4, pp. 363–377, 1999.

43. Gendreau, M., G. Laporte and F. Semet, "Heuristics and lower bounds for the bin packing problem with conflicts", *Computers & Operations Research*, Vol. 31, No. 3, pp. 347–358, Mar. 2004.

44. Khanafer, A., F. Clautiaux and E.-G. Talbi, "New lower bounds for bin packing problems with conflicts", *European Journal of Operational Research*, Vol. 206, No. 2, pp. 281–288, Oct. 2010.

45. Muritiba, A. E. F., M. I., E. Malaguti and P. Toth, "Algorithms for the Bin Packing Problem with Conflicts", *INFORMS Journal on Computing*, Vol. 22, No. 3, pp. 401–415, Aug. 2010.

46. Elhedhli, S., L. Li, M. Gzara and J. Naoum-Sawaya, "A Branch-and-Price Algorithm for the Bin Packing Problem with Conflicts", *INFORMS Journal on Computing*, Vol. 23, No. 3, pp. 404–415, Aug. 2011.

47. Sadykov, R. and F. Vanderbeck, “Bin Packing with Conflicts: A Generic Branch-and-Price Algorithm”, *INFORMS Journal on Computing*, Vol. 25, No. 2, pp. 244–255, May 2013.

48. Capua, R., Y. Frota, L. S. Ochi and T. Vidal, “A study on exponential-size neighborhoods for the bin packing problem with conflicts”, *Journal of Heuristics*, Vol. 24, No. 4, pp. 667–695, Apr. 2018.

49. Sun, M., “A tabu search heuristic procedure for solving the transportation problem with exclusionary side constraints”, *Journal of Heuristics*, Vol. 3, pp. 305–326, 1998.

50. Sun, M., “The transportation problem with exclusionary side constraints and two branch-and-bound algorithms”, *European Journal of Operational Research*, Vol. 140, pp. 629–647, 2002.

51. Syarif, A. and M. Gen, *Journal of Intelligent Manufacturing*, Vol. 14, No. 3/4, pp. 389–399, 2003.

52. Goossens, D. and F. C. R. Spieksma, “The transportation problem with exclusionary side constraints”, *4OR*, Vol. 7, No. 1, pp. 51–60, Jan. 2008.

53. Darmann, A., U. Pferschy and Schauer, *Determining a minimum spanning tree with disjunctive constraints*, pp. 414–423, Algorithmic Decision Theory, ADT 2009, Lecture Notes in Computer Science, vol. 5783, Springer, 2009.

54. Zhang, R., S. N. Kabadi and A. P. Punnen, “The minimum spanning tree problem with conflict constraints and its variations”, *Discrete Optimization*, Vol. 8, pp. 101–205, 2011.

55. Samer, P. and S. Urrutia, “A branch and cut algorithm for minimum spanning trees under conflict constraints”, *Optimization Letters*, Vol. 9, pp. 41–55, 2015.

56. Carrabs, F., R. Cerulli, R. Pentangelo and A. Raiconi, "Minimum spanning tree with conflicting edge pairs: a branch-and-cut approach", *Annals of Operations Research*, May 2018.

57. Öncan, T., R. Zhang and A. P. Punnen, "The minimum cost perfect matching problem with conflict pair constraints", *Computers & Operations Research*, Vol. 40, pp. 920–930, 2013.

58. Öncan, T. and I. K. Altınel, "A branch-and-bound algorithm for the minimum cost bipartite perfect matching problem with conflict pair constraints", *Proc. of the Int. Conf. on Sys. Eng. and Eng. Mngt. (ICSEEM'16)*, 2016.

59. Cao, B. and G. Uebe, "Solving transportation problems with nonlinear side constraints with tabu search", *Comp. Oper. Res.*, Vol. 22, pp. 593–603, 1995.

60. Öncan, T. and İ. K. Altınel, "A Branch-and-Bound Algorithm for the Minimum Cost Bipartite Perfect Matching Problem with Conflict Pair Constraints", *Electronic Notes in Discrete Mathematics*, Vol. 64, pp. 5–14, Feb. 2018.

61. Beaumont, O., N. Bonichon, P. Duchon and H. Larchevêque, "Distributed Approximation Algorithm for Resource Clustering", *Structural Information and Communication Complexity*, pp. 61–73, Springer Berlin Heidelberg, 2008.

62. Christofides, N., A. Mingozzi and P. Toth, *Combinatorial Optimization*, chap. Loading problems, pp. 339–69, Chicester: Wiley, 1979.

63. Goossens, D., A. Maas, F. Spieksma and J. van de Klundert, "Exact algorithms for procurement problems under a total quantity discount structure", *European Journal of Operational Research*, Vol. 178, No. 2, pp. 603–626, Apr. 2007.

64. Buragohain, C., S. Suri, C. D. Toth and Y. Zhou, "Improved throughput bounds for interference-aware routing in wireless networks", *G. Lin (Ed.), Computing and Combinatorics, 13th Annual Conference, COCOON07, LNCS 4598*, pp. 210–221,

2007.

65. Boysen, N., D. Briskorn and F. Meisel, “A generalized classification scheme for crane scheduling with interference”, *European Journal of Operational Research*, Vol. 258, No. 1, pp. 343–357, Apr. 2017.

66. Manerba, D. and R. Mansini, “A branch-and-cut algorithm for the multi-vehicle traveling purchaser problem with pairwise incompatibility constraints”, *Networks*, Vol. 65, No. 2, pp. 139–154, Dec. 2014.

67. Bierwirth, C. and F. Meisel, “A survey of berth allocation and quay crane scheduling problems in container terminals”, *European Journal of Operational Research*, Vol. 202, pp. 615–627, 2010.

68. Bierwirth, C. and F. Meisel, “A follow-up survey of berth allocation and quay crane scheduling problems in container terminals”, *European Journal of Operational Research*, Vol. 244, pp. 675–689, 2015.

69. Carlo, H. J., I. F. A. Vis and K. J. Roodbergen, “Seaside operations in container terminals: literature overview, trends, and research directions”, *Flexible Services and Manufacturing Journal*, Vol. 27, No. 2-3, pp. 224–262, Jun. 2015.

70. Steenken, D., S. Voßand R. Stahlbock, “Container terminal operation and operations research - a classification and literature review”, *OR Spectrum*, Vol. 26, No. 1, pp. 3–49, Jan. 2004.

71. She, J., Y. Tong, L. Chen and C. C. Cao, “Conflict-Aware Event-Participant Arrangement and Its Variant for Online Setting”, *IEEE Transactions on Knowledge and Data Engineering*, Vol. 28, No. 9, pp. 2281–2295, Sep. 2016.

72. Jain, K., J. Padhye, V. N. Padmanabhan and L. Qiu, “Impact of Interference on Multi-Hop Wireless Network Performance”, *Wireless Networks*, Vol. 11, No. 4, pp. 471–487, Jul. 2005.

73. Zhang, R., *Quadratic bottleneck problems: algorithms, complexity and related topics*, Ph.D. Thesis, Simon Fraser University, Department of Mathematics, Canada, 2011.

74. Papadimitriou, C. H., "The NP-completeness of the bandwidth minimization problem", *European Journal of Operational Research*, Vol. 16, pp. 263–279, 1976.

75. Fulkerson, D. R. and O. A. Gross, "Incidence matrices and interval graphs", *Pasific Journal of Mathematics*, Vol. 131, pp. 835–855, 1965.

76. Spinrad, J. P., *Efficient Graph Representations*, Number 19 in Fields Institute Monographs. American Mathematical Society, 2003.

77. Mitchell, J. and V. Polishchuk, "Graphs with few cliques", *Proc. 7th Quadrennial Int. Conf. on the Theory and Appl. of Graphs, 1992, Graph Theory, Combinatorics and Applications*, pp. 945–956, 1995.

78. Moon, J. W. and L. Moser, "On cliques in graphs", *Israel Journal of Mathematics*, Vol. 3, pp. 23–28, 1965.

79. Couturier, J. F., P. Heggernes, P. V. Hof and D. Kratsch, "Minimal dominating sets in graph classes: Combinatorial bounds and enumeration", *Theoretical Computer Science*, Vol. 487, pp. 82–94, 2013.

80. Fomin, F. V., F. Grandoni, A. V. Pyatkin and A. A. Stepanov, "Combinatorial bounds via measure and conquer: bounding minimal dominating sets and applications", *ACM Trans. Algorithms*, Vol. 5, pp. 9:1–9:17, 2008.

81. Tsukiyama, S., M. Ide, H. Ariyoshi and I. Shirakawa, "A new algorithm for generating all the maximal independent sets", *SIAM J. Computing*, Vol. 6, pp. 505–517, 1977.

82. Makino, K. and T. Uno, "New algorithms for enumerating all maximal cliques",

SWAT 2004, Lecture Notes in Computer Science, Vol. 3111, pp. 260–272, 2004.

83. Ausiello, G., P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela and M. Protasi, *Complexity and Approximation: Combinatorial Optimization Problems and Their Approximability Properties*, Springer Verlag, Berlin, 1999.

84. Cook, S. A., "The complexity of theorem-proving procedures", *Proceedings of the third annual ACM symposium on Theory of computing - STOC 71*, ACM Press, 1971.

85. Benders, J. F., "Partitioning procedures for solving mixed variables programming problems", *Numerische Mathematik*, Vol. 4, pp. 238–252, 1962.

86. Rahmaniani, R., T. G. Crainic, M. Gendrau and W. Rei, "The Benders decomposition algorithm: A literature review", *European Journal of Operational Research*, Vol. 259, No. 3, pp. 801–817, Jun. 2017.

87. Costa, A. M., "A survey on benders decomposition applied to fixed-charge network design problems", *Computers & Operations Research*, Vol. 32, No. 6, pp. 1429–1450, Jun. 2005.

88. Magnanti, T. L., P. Mireault and R. T. Wong, "Tailoring Benders decomposition for uncapacitated network design", *Mathematical Programming Studies*, pp. 112–154, Springer Berlin Heidelberg, 1986.

89. Codato, G. and M. Fischetti, "Combinatorial Benders Cuts for Mixed-Integer Linear Programming", *Operations Research*, Vol. 54, No. 4, pp. 756–766, Aug. 2006.

90. Magnanti, T. L. and R. T. Wong, "Accelerating Benders Decomposition: Algorithmic Enhancement and Model Selection Criteria", *Operations Research*, Vol. 29, No. 3, pp. 464–484, Jun. 1981.

91. Roy, T. J. V., “Cross decomposition for mixed integer programming”, *Mathematical Programming*, Vol. 25, No. 1, pp. 46–63, Jan. 1983.

92. Üster, H. and H. Agrahari, “A Benders decomposition approach for a distribution network design problem with consolidation and capacity considerations”, *Operations Research Letters*, Vol. 39, No. 2, pp. 138–143, 2011.

93. Bodur, M., T. Ekim and Z. C. Taşkin, “Decomposition algorithms for solving the minimum weight maximal matching problem”, *Networks*, Vol. 62, No. 4, pp. 273–287, Oct. 2013.

94. Taşkın, Z. C. and M. Cevik, “Combinatorial Benders cuts for decomposing IMRT fluence maps using rectangular apertures”, *Computers & Operations Research*, Vol. 40, No. 9, pp. 2178–2186, Sep. 2013.

95. Goldberg, A. V. and R. E. Tarjan, “A new approach to the maximum-flow problem”, *Journal of the ACM*, Vol. 35, No. 4, pp. 921–940, Oct 1988.

96. Nemhauser, G. and L. Wolsey, *Integer and Combinatorial Optimization*, Wiley, New York, 1988.

97. Kovács, P., *Minimum-cost flow algorithms: An experimental evaluation*, Tech. rep., Egerváry Research Group on Combinatorial Optimization, Budapest, Hungary, 2015.

98. Verfaillie, G., M. Lemaitre and T. Schiex, “Russian Doll Search for solving constraint optimization problems”, *Proceedings of the National Conference on Artificial Intelligence*, pp. 181–187, Citeseer, Princeton, 1996.

99. Rysz, M., M. Mirghorbani, P. Krokhmal and E. L. Pasiliao, “On risk-averse maximum weighted subgraph problems”, *J Comb Optim*, pp. 28– –167, 2014, `https://doi.org/10.1007/s10878-014-9718-0`.

100. Klingman, D., A. Napier and J. Stutz, “NETGEN: A program for generating large scale capacitated assignment, transportation, and minimum cost flow networks”, *Management Science*, Vol. 20, pp. 814–820, 1974.

101. “Visual C++ Documentation”, `https://docs.microsoft.com/en-us/cpp`, accessed at Jun. 2019.

102. *CPLEX User’s Manual: IBM ILOG CPLEX Optimization Studio*, `https://www.ibm.com/support/knowledgecenter`, accessed at May 2019.

APPENDIX A: APPENDICES

A.1. Detailed Computational Results for the Minimum Cost Flow Problem with Conflicts

The following tables, i.e. Table A.1 – Table A.5 include the raw computational results. They are used to prepare the summary tables, i.e. Table 10.4 – Table 10.10, in Section 10.2.

Table A.1. Properties of the test instances and the upper bounds given by the diving heuristics.

Problem Instances				Diving Heuristics				Problem Instances				Diving Heuristics			
Instance No	Number of nodes (n)	Arc density (p)	Conflict density (d)	Rule 1 (UB)	Rule 1 CPU Time (s)	Rule 2 (UB)	Rule 2 CPU Time (s)	Instance No	Number of nodes (n)	Arc density (p)	Conflict density (d)	Rule 1 (UB)	Rule 1 CPU Time (s)	Rule 2 (UB)	Rule 2 CPU Time (s)
1	40	0.3	0.2	19,850	0.04	19,850	2.01	41	60	0.5	0.2	63,000	57.60		60.00
2	40	0.3	0.3	17,200	0.16	17,200	0.10	42	60	0.5	0.3	150,000	2.05	150,000	2.90
3	40	0.3	0.4	35,900	0.80	35,900	0.21	43	60	0.5	0.4	24,000	5.95	24,000	20.70
4	40	0.3	0.5	28,000	0.15	28,000	0.32	44	60	0.5	0.5	32,400	9.03	32,400	7.19
5	40	0.4	0.2	27,950	1.75	27,950	0.43	45	60	0.6	0.2	157,750	19.97	157,750	7.22
6	40	0.4	0.3	47,800	0.12	47,800	1.47	46	60	0.6	0.3	34,250	20.16		60.00
7	40	0.4	0.4	27,850	0.08	27,850	0.10	47	60	0.6	0.4	107,400	9.84	107,400	7.59
8	40	0.4	0.5	27,150	0.21	27,150	0.25	48	60	0.6	0.5	34,100	9.42	34,100	10.47
9	40	0.5	0.2	38,300	4.59	38,300	1.40	49	70	0.3	0.2	125,700	0.58	125,700	3.53
10	40	0.5	0.3	114,150	0.09	114,150	1.42	50	70	0.3	0.3	139,100	1.29	139,100	1.16
11	40	0.5	0.4	63,700	0.17	63,700	0.19	51	70	0.3	0.4	–	–	–	–
12	40	0.5	0.5	70,000	0.03	70,000	0.31	52	70	0.3	0.5	56,000	1.71	56,000	1.84
13	40	0.6	0.2		60.00	35,312	19.50	53	70	0.4	0.2	112,000	57.41		60.00
14	40	0.6	0.3	54,000	1.29	54,000	1.25	54	70	0.4	0.3	181,950	2.59	181,950	4.12
15	40	0.6	0.4	64,000	1.03	64,000	0.36	55	70	0.4	0.4	119,900	1.08	119,900	1.88
16	40	0.6	0.5	19,500	1.18	19,500	1.14	56	70	0.4	0.5	168,000	1.92	168,000	4.02
17	50	0.3	0.2	34,700	13.79	34,700	14.50	57	70	0.5	0.2	123,250	56.12		60.00
18	50	0.3	0.3	48,450	1.25	48,450	0.12	58	70	0.5	0.3	218,950	6.88	218,950	10.98
19	50	0.3	0.4	19,400	1.09	19,400	0.13	59	70	0.5	0.4	126,000	13.01	126,000	13.26
20	50	0.3	0.5	–	–	–	–	60	70	0.5	0.5	135,000	2.99	135,000	3.74
21	50	0.4	0.2	59,900	25.66	59,900	3.50	61	70	0.6	0.2		60.00	157,450	14.15
22	50	0.4	0.3	111,900	0.23	111,900	0.49	62	70	0.6	0.3	116,500	37.21	116,500	23.75
23	50	0.4	0.4	24,650	1.04	24,650	0.51	63	70	0.6	0.4	126,000	20.79	126,000	22.16
24	50	0.4	0.5	–	–	–	–	64	70	0.6	0.5	106,350	23.23	106,350	31.14
25	50	0.5	0.2	89,250	4.73	89,250	22.15	65	80	0.3	0.2	50,200	54.77		60.00
26	50	0.5	0.3	62,450	2.05	62,450	1.25	66	80	0.3	0.3	42,950	33.06	42,950	3.70
27	50	0.5	0.4	79,250	1.09	79,250	2.01	67	80	0.3	0.4	152,200	0.94	152,200	2.01
28	50	0.5	0.5	17,050	6.86	17,050	4.59	68	80	0.3	0.5	–	–	–	–
29	50	0.6	0.2	86,408	8.25		60.00	69	80	0.4	0.2	138,150	11.28		60.00
30	50	0.6	0.3	44,900	5.75	44,900	3.02	70	80	0.4	0.3	118,350	9.72	118,350	9.99
31	50	0.6	0.4	86,050	1.34	86,050	2.48	71	80	0.4	0.4	71,950	9.04	71,950	14.71
32	50	0.6	0.5	15,950	5.66	15,950	5.50	72	80	0.4	0.5	131,600	0.96	131,600	1.29
33	60	0.3	0.2	53,950	7.01	53,950	1.28	73	80	0.5	0.2		60.00	374,550	54.34
34	60	0.3	0.3	52,300	0.99	52,300	1.22	74	80	0.5	0.3		60.00	148,000	23.66
35	60	0.3	0.4	20,950	2.05	20,950	0.50	75	80	0.5	0.4	143,800	21.07	143,800	16.49
36	60	0.3	0.5	52,900	0.08	52,900	1.34	76	80	0.5	0.5	192,000	18.93	192,000	21.51
37	60	0.4	0.2	120,650	3.95	120,650	3.66	77	80	0.6	0.2	167,800	59.01		60.00
38	60	0.4	0.3	90,000	3.12	90,000	1.75	78	80	0.6	0.3	159,800	22.99		60.00
39	60	0.4	0.4	107,950	0.10	107,950	0.85	79	80	0.6	0.4	201,450	36.02	201,450	47.41
40	60	0.4	0.5	47,650	0.10	47,650	1.80	80	80	0.6	0.5	103,650	33.98	103,650	32.59

Table A.2. CPU times of algorithms.

Instance No	CPLEX-Strong CPU Time (s)	CPLEX-Weak CPU Time (s)	BB-Arc CPU Time (s)	BB-Pair CPU Time (s)	BB-Clique CPU Time (s)	RDS CPU Time (s)	Instance No	CPLEX-Strong CPU Time (s)	CPLEX-Weak CPU Time (s)	BB-Arc CPU Time (s)	BB-Pair CPU Time (s)	BB-Clique CPU Time (s)	RDS CPU Time (s)
1	123.49	41.82	4.09	>3600	>3600	20.82	41	>3600	>3600	>3600	>3600	>3600	>3600
2	29.28	17.02	0.58	>3600	96.72	1.27	42	>3600	236.72	3.45	>3600	263.98	10.22
3	22.42	11.60	0.23	>3600	1.29	0.6	43	>3600	>3600	421.43	>3600	>3600	722.22
4	1.36	1.19	0.09	>3600	0.04	0.22	44	>3600	3,596.49	81.48	>3600	39.5	147.23
5	629.20	228.91	41.07	>3600	>3600	98.22	45	>3600	>3600	>3600	>3600	>3600	>3600
6	106.37	40.69	2.7	>3600	382.56	6.18	46	>3600	>3600	>3600	>3600	>3600	>3600
7	2.63	2.23	0.07	>3600	0.19	0.23	47	>3600	>3600	142.8	>3600	2987.86	305.25
8	0.80	0.71	0.01	1.34	0.01	0.02	48	>3600	>3600	197.32	>3600	81.1	339.22
9	>3600	2,027.80	274.08	>3600	>3600	443.81	49	1,595.60	233.36	11.06	>3600	>3600	67.08
10	154.15	45.36	0.44	>3600	16.83	0.89	50	>3600	242.17	0.69	>3600	3.2	2.37
11	10.24	46.63	0.12	>3600	0.09	0.44	51	0.05	0.03	0	0	0	0
12	4.07	3.55	0.03	0.68	0.02	0.1	52	>3600	544.90	4.26	>3600	3.48	10.07
13	>3600	>3600	>3600	>3600	>3600	>3600	53	>3600	>3600	2657.78	>3600	>3600	>3600
14	1,076.57	188.30	7.55	>3600	746.88	46.47	54	>3600	619.77	15.65	>3600	1747.43	38.15
15	347.36	68.11	0.58	>3600	2.46	2.47	55	>3600	191.06	1.22	>3600	1.19	3.15
16	1,201.29	78.00	1.06	>3600	1.77	3.4	56	>3600	457.36	1.34	>3600	1.72	5.16
17	2,487.92	896.72	170.06	>3600	>3600	276.72	57	>3600	>3600	>3600	>3600	>3600	>3600
18	0.62	0.56	0.11	>3600	0.28	0.34	58	>3600	1,095.03	17.91	>3600	2614.2	82.55
19	219.14	130.28	3.31	>3600	19.45	7.28	59	>3600	3,166.74	16.74	>3600	33	45.12
20	0.33	0.21	0	0	0	0	60	>3600	460.99	1.75	>3600	0.85	5.17
21	>3600	>3600	982.41	>3600	>3600	2674.68	61	>3600	>3600	>3600	>3600	>3600	>3600
22	256.76	56.61	0.12	>3600	1.64	1.2	62	>3600	>3600	631.32	>3600	>3600	1665.19
23	2,529.34	419.16	6.12	>3600	23.45	13.95	63	>3600	2,387.09	26.46	>3600	42.44	111.6
24	0.23	0.16	0	0	0	0	64	>3600	2,889.19	27.92	>3600	52.79	70.6
25	>3600	1,323.10	176.76	>3600	>3600	1295.47	65	>3600	>3600	>3600	>3600	>3600	>3600
26	1,817.32	148.69	1.81	>3600	78.1	8.86	66	>3600	>3600	456.52	>3600	>3600	891.21
27	1,364.09	152.66	1.43	>3600	4.65	3.52	67	>3600	128.70	0.48	>3600	0.46	0.85
28	>3600	734.84	33.4	>3600	10.65	63.91	68	5.63	2.07	0	0	0	0
29	>3600	>3600	575.5	>3600	>3600	>3600	69	>3600	>3600	1854.13	>3600	>3600	>3600
30	>3600	1,184.54	49.28	>3600	>3600	168.56	70	>3600	>3600	61.82	>3600	>3600	170.91
31	>3600	79.43	1.04	>3600	1.63	2.44	71	>3600	2,948.32	70	>3600	922.23	193.3
32	>3600	1,148.15	39.7	>3600	24.85	88.82	72	106.10	46.66	0.53	58.5	0.18	1.25
33	>3600	1,155.88	125.86	>3600	>3600	482.79	73	>3600	>3600	>3600	>3600	>3600	>3600
34	1,161.00	81.81	2.21	>3600	147.46	7.25	74	>3600	>3600	>3600	>3600	>3600	>3600
35	2,478.22	189.87	3.63	>3600	12.66	13.61	75	>3600	2,136.78	31.18	>3600	86.4	83.59
36	1,101.95	41.70	0.73	>3600	0.37	1.37	76	>3600	2,957.50	16.74	>3600	8.69	83.41
37	>3600	>3600	1724.22	>3600	>3600	3578.31	77	>3600	>3600	>3600	>3600	>3600	>3600
38	>3600	258.65	3.54	>3600	104.61	10.48	78	>3600	1,210.73	32.48	>3600	534.58	110.63
39	11.73	14.19	0.25	>3600	0.07	0.64	79	>3600	3,493.14	40.25	>3600	194.29	104.67
40	>3600	53.51	0.44	>3600	0.78	2.1	80	>3600	2,547.11	19.41	>3600	21.7	58.4

Table A.3. Number of solved subproblems to find an optimal solution within 1 hour time limit.

Instance No	BB-Arc	BB-Pair	BB-Clique	RDS	Instance No	BB-Arc	BB-Pair	BB-Clique	RDS
1	13,440	732,584	6,077,470	27,466	41	556,096	192,891	232,525	557,992
2	2,163	723,440	451,995	1,597	42	449	127,253	32,505	375
3	667	812,169	4,235	460	43	97,493	94,676	466,058	47,389
4	89	985,377	43	49	44	10,295	139,352	5,541	4,605
5	120,188	685,923	895,606	148,853	45	355,596	107,851	133,252	263,530
6	5,559	767,315	903,061	3,877	46	227,805	115,814	114,536	195,274
7	83	872,893	147	70	47	16,649	128,903	347,495	8,969
8	25	6,483	9	20	48	24,520	58,706	10,415	10,104
9	382,509	617,879	1,346,085	261,654	49	6,731	726,415	864,586	17,493
10	326	1,008,331	12,413	239	50	115	328,910	507	148
11	87	2,169,725	75	124	51	0	0	0	0
12	19	650	7	19	52	943	153,687	677	464
13	3,509,645	491,181	610,014	2,723,137	53	817,768	234,047	479,195	1,195,978
14	4,335	439,998	410,047	9,030	54	2,298	134,180	282,399	2,107
15	272	368,327	1,055	376	55	67	91,660	69	63
16	339	238,087	509	244	56	113	86,965	117	86
17	372,077	631,474	714,233	259,621	57	311,294	93,245	119,138	202,152
18	117	3,399,582	243	115	58	1,888	62,976	232,937	3,309
19	3,261	627,271	18,687	1,652	59	713	45,177	1,553	530
20	0	0	0	0	60	70	46,534	25	55
21	1,246,221	574,151	1,190,047	1,733,193	61	182,881	45,262	60,469	149,266
22	91	4,409,331	1,073	400	62	31,372	37,360	65,492	24,799
23	2,389	328,763	8,769	1,394	63	772	24,980	1,041	1,245
24	0	0	0	0	64	1,033	20,547	1,605	752
25	119,717	410,650	577,328	395,381	65	909,586	223,079	314,780	990,855
26	500	325,473	19,887	1,131	66	142,838	136,894	330,887	73,984
27	307	232,094	777	170	67	48	462,243	37	40
28	5,671	119,010	1,925	2,996	68	0	0	0	0
29	265,443	242,116	358,438	734,005	69	200,831	80,839	98,574	132,708
30	12,600	132,923	420,462	15,100	70	4,691	59,427	140,701	4,373
31	123	182,099	235	102	71	3,533	43,906	53,323	2,657
32	6,581	95,037	4,371	3,812	72	23	15,406	9	20
33	147,036	432,841	1,244,040	209,609	73	272,999	56,552	70,283	146,975
34	1,381	539,415	82,919	1,460	74	265,594	52,201	93,970	285,469
35	1,826	305,121	5,929	1,422	75	601	21,446	1,769	526
36	113	285,557	77	58	76	501	44,565	231	606
37	862,514	333,606	440,984	632,843	77	46,009	23,448	28,649	47,238
38	531	167,480	15,115	440	78	833	22,109	10,639	2,410
39	77	809,449	15	51	79	665	14,341	3,379	590
40	65	174,617	111	70	80	257	14,637	355	242

Table A.4. Upper and lower bounds in 1 hour time limit for instances 1-40.

Instance No	CPLEX-Strong (UB)	CPLEX-Strong (LB)	CPLEX-Weak (UB)	CPLEX-Weak (LB)	BB-Arc (UB)	BB-Arc (LB)	BB-Pair (UB)	BB-Pair (LB)	BB-Clique (UB)	BB-Clique (LB)	RDS (UB)	RDS (LB)
1	19,850	19,850	19,850	19,850	19,850	19,850	19,850	1,492	19,850	2,974	19,850	19,850
2	17,200	17,200	17,200	17,200	17,200	17,200	17,200	794	17,200	17,200	17,200	17,200
3	35,900	35,900	35,900	35,900	35,900	35,900	35,900	608	35,900	35,900	35,900	35,900
4	28,000	28,000	28,000	28,000	28,000	28,000	28,000	1,331	28,000	28,000	28,000	28,000
5	27,950	27,950	27,950	27,950	27,950	27,950	27,950	1,793	27,950	2,476	27,950	27,950
6	47,800	47,800	47,800	47,800	47,800	47,800	47,800	1,053	47,800	47,800	47,800	47,800
7	27,850	27,850	27,850	27,850	27,850	27,850	27,850	2,641	27,850	27,850	27,850	27,850
8	27,150	27,150	27,150	27,150	27,150	27,150	27,150	27,150	27,150	27,150	27,150	27,150
9	38,300	3,896	38,300	38,300	38,300	38,300	38,300	1,478	38,300	2,270	38,300	38,300
10	114,150	114,150	114,150	114,150	114,150	114,150	114,150	4,224	114,150	114,150	114,150	114,150
11	63,700	63,700	63,700	63,700	63,700	63,700	63,700	11,294	63,700	63,700	63,700	63,700
12	70,000	70,000	70,000	70,000	70,000	70,000	70,000	70,000	70,000	70,000	70,000	70,000
13	35,051	2,608	35,051	2,769	35,051	5,888	35,051	1,539	35,051	1,809	22,423	1,384
14	54,000	54,000	54,000	54,000	54,000	54,000	54,000	2,132	54,000	54,000	54,000	54,000
15	64,000	64,000	64,000	64,000	64,000	64,000	64,000	2,749	64,000	64,000	64,000	64,000
16	19,500	19,500	19,500	19,500	19,500	19,500	19,500	990	19,500	19,500	19,500	19,500
17	34,700	34,700	34,700	34,700	34,700	34,700	34,700	1,061	34,700	1,569	34,700	34,700
18	48,450	48,450	48,450	48,450	48,450	48,450	48,450	14,860	48,450	48,450	48,450	48,450
19	19,400	19,400	19,400	19,400	19,400	19,400	19,400	389	19,400	19,400	19,400	19,400
20	65,600	65,600	65,600	65,600	65,600	65,600	65,600	65,600	65,600	65,600	65,600	65,600
21	59,900	2,289	59,900	2,815	7,011	7,011	59,900	1,631	59,900	2,383	7,011	7,011
22	111,900	111,900	111,900	111,900	111,900	111,900	111,900	42,681	111,900	111,900	111,900	111,900
23	24,650	24,650	24,650	24,650	24,650	24,650	24,650	939	24,650	24,650	24,650	24,650
24	68,050	68,050	68,050	68,050	68,050	68,050	68,050	68,050	68,050	68,050	68,050	68,050
25	89,250	89,250	89,250	89,250	89,250	89,250	89,250	4,785	89,250	6,803	89,250	89,250
26	62,450	62,450	62,450	62,450	62,450	62,450	62,450	7,653	62,450	62,450	62,450	62,450
27	79,250	6,705	79,250	79,250	79,250	79,250	79,250	2,973	79,250	79,250	79,250	79,250
28	17,050	222	17,050	17,050	17,050	17,050	17,050	221	17,050	17,050	17,050	17,050
29	85,480	9,722	85,480	22,331	85,480	85,480	85,480	4,453	85,480	7,511	85,480	3,663
30	44,900	2,531	44,900	44,900	44,900	44,900	44,900	2,146	44,900	5,400	44,900	44,900
31	86,050	11,471	86,050	86,050	86,050	86,050	86,050	8,141	86,050	86,050	86,050	86,050
32	15,950	222	15,950	15,950	15,950	15,950	15,950	234	15,950	15,950	15,950	15,950
33	53,950	3,298	53,950	53,950	53,950	53,950	53,950	1,553	53,950	2,716	53,950	53,950
34	52,300	52,300	52,300	52,300	52,300	52,300	52,300	2,874	52,300	52,300	52,300	52,300
35	20,950	20,950	20,950	20,950	20,950	20,950	20,950	615	20,950	20,950	20,950	20,950
36	52,900	52,900	52,900	52,900	52,900	52,900	52,900	1,619	52,900	52,900	52,900	52,900
37	120,650	4,721	120,650	5,378	120,650	120,650	120,650	3,355	120,650	3,885	120,650	120,650
38	90,000	6,720	90,000	90,000	90,000	90,000	90,000	4,144	90,000	90,000	90,000	90,000
39	107,950	107,950	107,950	107,950	107,950	107,950	107,950	13,401	107,950	107,950	107,950	107,950
40	47,650	1,411	47,650	47,650	47,650	47,650	47,650	1,840	47,650	47,650	47,650	47,650

Table A.5. Upper and lower bounds in 1 hour time limit for instances 41 - 80.

Instance No	CPLEX-Strong (UB)	CPLEX-Strong (LB)	CPLEX-Weak (UB)	CPLEX-Weak (LB)	BB-Arc (UB)	BB-Arc (LB)	BB-Pair (UB)	BB-Pair (LB)	BB-Clique (UB)	BB-Clique (LB)	RDS (UB)	RDS (LB)
41	63,000	2,399	63,000	2,762	63,000	4,156	63,000	1,675	63,000	2,033	63,000	929
42	150,000	7,457	150,000	150,000	150,000	150,000	150,000	6,585	150,000	150,000	150,000	150,000
43	24,000	373	24,000	1,147	24,000	24,000	24,000	300	24,000	1,492	24,000	24,000
44	32,400	229	32,400	32,400	32,400	32,400	32,400	195	32,400	32,400	32,400	32,400
45	157,750	3,723	157,750	5,417	157,750	9,386	157,750	3,262	157,750	3,924	157,750	2,118
46	34,250	510	34,250	1,053	34,250	1,866	34,250	349	34,250	893	34,250	213
47	107,400	991	107,400	2,096	107,400	107,400	107,400	638	107,400	107,400	107,400	107,400
48	34,100	342	34,100	1,008	34,100	34,100	34,100	214	34,100	34,100	34,100	34,100
49	125,700	125,700	125,700	125,700	125,700	125,700	125,700	13,171	125,700	21,707	125,700	125,700
50	139,100	24,100	139,100	139,100	139,100	139,100	139,100	13,221	139,100	139,100	139,100	139,100
51	191,150	191,150	191,150	191,150	191,150	191,150	191,150	191,150	191,150	191,150	191,150	191,150
52	56,000	943	56,000	56,000	56,000	56,000	56,000	647	56,000	56,000	56,000	56,000
53	1,531	737	1,531	905	1,531	1,531	112,000	383	112,000	498	2,062	238
54	181,950	6,045	181,950	181,950	181,950	181,950	181,950	4,543	181,950	181,950	181,950	181,950
55	119,900	10,007	119,900	119,900	119,900	119,900	119,900	7,296	119,900	119,900	119,900	119,900
56	168,000	2,078	168,000	168,000	168,000	168,000	168,000	2,094	168,000	168,000	168,000	168,000
57	123,250	3,613	123,250	6,002	123,250	9,473	123,250	3,159	123,250	3,874	123,250	1,748
58	218,950	2,694	218,950	218,950	218,950	218,950	218,950	2,632	218,950	218,950	218,950	218,950
59	126,000	3,317	126,000	126,000	126,000	126,000	126,000	2,896	126,000	126,000	126,000	126,000
60	135,000	3,474	135,000	135,000	135,000	135,000	135,000	3,639	135,000	135,000	135,000	135,000
61	157,450	4,731	157,450	7,052	157,450	16,963	157,450	3,718	157,450	4,644	157,450	1,697
62	116,500	2,901	116,500	5,311	116,500	116,500	116,500	2,857	116,500	4,289	116,500	116,500
63	126,000	3,181	126,000	126,000	126,000	126,000	126,000	2,486	126,000	126,000	126,000	126,000
64	106,350	641	106,350	106,350	106,350	106,350	106,350	749	106,350	106,350	106,350	106,350
65	50,200	577	50,200	769	50,200	1,606	50,200	416	50,200	636	50,200	267
66	42,950	331	42,950	1,295	42,950	42,950	42,950	452	42,950	1,320	42,950	42,950
67	152,200	9,186	152,200	152,200	152,200	152,200	152,200	19,033	152,200	152,200	152,200	152,200
68	271,700	271,700	271,700	271,700	271,700	271,700	271,700	271,700	271,700	271,700	271,700	271,700
69	138,150	4,643	138,150	7,060	138,150	138,150	138,150	3,481	138,150	4,627	138,150	3,853
70	118,350	3,182	118,350	6,034	118,350	118,350	118,350	2,998	118,350	6,501	118,350	118,350
71	71,950	1,352	71,950	71,950	71,950	71,950	71,950	1,524	71,950	71,950	71,950	71,950
72	131,600	131,600	131,600	131,600	131,600	131,600	131,600	131,600	131,600	131,600	131,600	131,600
73	374,550	6,374	374,550	12,799	374,550	23,288	374,550	4,721	374,550	5,693	374,550	2,616
74	148,000	240	148,000	646	148,000	1,596	148,000	220	148,000	358	148,000	155
75	143,800	4,938	143,800	143,800	143,800	143,800	143,800	2,132	143,800	143,800	143,800	143,800
76	192,000	827	192,000	192,000	192,000	192,000	192,000	965	192,000	192,000	192,000	192,000
77	167,800	3,948	167,800	5,698	167,800	5,888	167,800	2,831	167,800	3,438	167,800	1,593
78	159,800	13,639	159,800	159,800	159,800	159,800	159,800	8,186	159,800	159,800	159,800	159,800
79	201,450	4,233	201,450	201,450	201,450	201,450	201,450	3,673	201,450	201,450	201,450	201,450
80	103,650	1,444	103,650	103,650	103,650	103,650	103,650	1,646	103,650	103,650	103,650	103,650

A.2. Detailed Computational Results for the Maximum Flow Problem with Conflicts

Table A.6 – Table A.12 include the computational results on the instance basis. They are used to prepare the concise tables, (Table 10.11 – Table 10.18), in Section 10.4.

Table A.6. Properties of the test instances.

Instance No	Number of vertices (n)	Arc density (p)	Conflict density (d)	Instance No	Number of vertices (n)	Arc density (p)	Conflict density (d)	Instance No	Number of vertices (n)	Arc density (p)	Conflict density (d)	Instance No	Number of vertices (n)	Arc density (p)	Conflict density (d)
1	40	0.3	0.3	41	50	0.4	0.3	81	60	0.5	0.3	121	70	0.6	0.3
2	40	0.3	0.3	42	50	0.4	0.3	82	60	0.5	0.3	122	70	0.6	0.3
3	40	0.3	0.4	43	50	0.4	0.4	83	60	0.5	0.4	123	70	0.6	0.4
4	40	0.3	0.4	44	50	0.4	0.4	84	60	0.5	0.4	124	70	0.6	0.4
5	40	0.3	0.5	45	50	0.4	0.5	85	60	0.5	0.5	125	70	0.6	0.5
6	40	0.3	0.5	46	50	0.4	0.5	86	60	0.5	0.5	126	70	0.6	0.5
7	40	0.3	0.6	47	50	0.4	0.6	87	60	0.5	0.6	127	70	0.6	0.6
8	40	0.3	0.6	48	50	0.4	0.6	88	60	0.5	0.6	128	70	0.6	0.6
9	40	0.4	0.3	49	50	0.5	0.3	89	60	0.6	0.3	129	80	0.3	0.3
10	40	0.4	0.3	50	50	0.5	0.3	90	60	0.6	0.3	130	80	0.3	0.3
11	40	0.4	0.4	51	50	0.5	0.4	91	60	0.6	0.4	131	80	0.3	0.4
12	40	0.4	0.4	52	50	0.5	0.4	92	60	0.6	0.4	132	80	0.3	0.4
13	40	0.4	0.5	53	50	0.5	0.5	93	60	0.6	0.5	133	80	0.3	0.5
14	40	0.4	0.5	54	50	0.5	0.5	94	60	0.6	0.5	134	80	0.3	0.5
15	40	0.4	0.6	55	50	0.5	0.6	95	60	0.6	0.6	135	80	0.3	0.6
16	40	0.4	0.6	56	50	0.5	0.6	96	60	0.6	0.6	136	80	0.3	0.6
17	40	0.5	0.3	57	50	0.6	0.3	97	70	0.3	0.3	137	80	0.4	0.3
18	40	0.5	0.3	58	50	0.6	0.3	98	70	0.3	0.3	138	80	0.4	0.3
19	40	0.5	0.4	59	50	0.6	0.4	99	70	0.3	0.4	139	80	0.4	0.4
20	40	0.5	0.4	60	50	0.6	0.4	100	70	0.3	0.4	140	80	0.4	0.4
21	40	0.5	0.5	61	50	0.6	0.5	101	70	0.3	0.5	141	80	0.4	0.5
22	40	0.5	0.5	62	50	0.6	0.5	102	70	0.3	0.5	142	80	0.4	0.5
23	40	0.5	0.6	63	50	0.6	0.6	103	70	0.3	0.6	143	80	0.4	0.6
24	40	0.5	0.6	64	50	0.6	0.6	104	70	0.3	0.6	144	80	0.4	0.6
25	40	0.6	0.3	65	60	0.3	0.3	105	70	0.4	0.3	145	80	0.5	0.3
26	40	0.6	0.3	66	60	0.3	0.3	106	70	0.4	0.3	146	80	0.5	0.3
27	40	0.6	0.4	67	60	0.3	0.4	107	70	0.4	0.4	147	80	0.5	0.4
28	40	0.6	0.4	68	60	0.3	0.4	108	70	0.4	0.4	148	80	0.5	0.4
29	40	0.6	0.5	69	60	0.3	0.5	109	70	0.4	0.5	149	80	0.5	0.5
30	40	0.6	0.5	70	60	0.3	0.5	110	70	0.4	0.5	150	80	0.5	0.5
31	40	0.6	0.6	71	60	0.3	0.6	111	70	0.4	0.6	151	80	0.5	0.6
32	40	0.6	0.6	72	60	0.3	0.6	112	70	0.4	0.6	152	80	0.5	0.6
33	50	0.3	0.3	73	60	0.4	0.3	113	70	0.5	0.3	153	80	0.6	0.3
34	50	0.3	0.3	74	60	0.4	0.3	114	70	0.5	0.3	154	80	0.6	0.3
35	50	0.3	0.4	75	60	0.4	0.4	115	70	0.5	0.4	155	80	0.6	0.4
36	50	0.3	0.4	76	60	0.4	0.4	116	70	0.5	0.4	156	80	0.6	0.4
37	50	0.3	0.5	77	60	0.4	0.5	117	70	0.5	0.5	157	80	0.6	0.5
38	50	0.3	0.5	78	60	0.4	0.5	118	70	0.5	0.5	158	80	0.6	0.5
39	50	0.3	0.6	79	60	0.4	0.6	119	70	0.5	0.6	159	80	0.6	0.6
40	50	0.3	0.6	80	60	0.4	0.6	120	70	0.5	0.6	160	80	0.6	0.6

Table A.7. CPLEX CPU times (in seconds) for weak (CPLEX$_W$), strong (CPLEX$_S$) and clique (CPLEX$_K$) formulations.

Instance No	CPLEX$_W$	CPLEX$_S$	CPLEX$_K$	Instance No	CPLEX$_W$	CPLEX$_S$	CPLEX$_K$	Instance No	CPLEX$_W$	CPLEX$_S$	CPLEX$_K$	Instance No	CPLEX$_W$	CPLEX$_S$	CPLEX$_K$
1	20.45	41.17	29.23	41	1,380.51	>3,600	3,002.44	81	>3,600	>3,600	>3,600	121	>3,600	>3,600	>3,600
2	10.02	63.34	54.65	42	213.58	1,916.55	1,048.60	82	>3,600	>3,600	>3,600	122	>3,600	>3,600	>3,600
3	21.19	192.04	109.31	43	796.90	>3,600	>3,600	83	>3,600	>3,600	>3,600	123	>3,600	>3,600	>3,600
4	18.21	108.58	85.89	44	262.28	3,198.91	1,873.82	84	>3,600	>3,600	>3,600	124	>3,600	>3,600	>3,600
5	19.05	98.95	63.54	45	708.83	>3,600	>3,600	85	>3,600	>3,600	>3,600	125	>3,600	>3,600	>3,600
6	12.56	111.32	130.67	46	283.91	3,276.71	3,508.41	86	2,326.10	>3,600	>3,600	126	>3,600	>3,600	>3,600
7	25.16	193.01	141.84	47	54.99	>3,600	>3,600	87	1,708.75	>3,600	>3,600	127	>3,600	>3,600	>3,600
8	34.60	151.26	120.34	48	333.93	>3,600	>3,600	88	1,778.31	>3,600	>3,600	128	>3,600	>3,600	>3,600
9	78.39	248.26	168.96	49	1,773.07	>3,600	>3,600	89	>3,600	>3,600	>3,600	129	>3,600	>3,600	>3,600
10	256.68	599.24	487.56	50	1,295.33	>3,600	>3,600	90	>3,600	>3,600	>3,600	130	>3,600	>3,600	>3,600
11	104.40	586.20	433.31	51	1,019.91	>3,600	>3,600	91	>3,600	>3,600	>3,600	131	>3,600	>3,600	>3,600
12	103.05	537.72	463.09	52	1,094.33	>3,600	>3,600	92	>3,600	>3,600	>3,600	132	>3,600	>3,600	>3,600
13	128.58	1,056.37	564.28	53	960.73	>3,600	>3,600	93	>3,600	>3,600	>3,600	133	>3,600	>3,600	>3,600
14	102.09	500.29	531.66	54	1,151.55	>3,600	>3,600	94	>3,600	>3,600	>3,600	134	3,008.25	>3,600	>3,600
15	41.92	808.04	484.27	55	838.49	>3,600	>3,600	95	3,274.12	>3,600	>3,600	135	2,119.22	>3,600	>3,600
16	106.81	569.48	646.03	56	209.01	>3,600	>3,600	96	>3,600	>3,600	>3,600	136	>3,600	>3,600	>3,600
17	520.09	2,415.34	2,845.37	57	>3,600	>3,600	>3,600	97	>3,600	>3,600	>3,600	137	>3,600	>3,600	>3,600
18	133.86	702.02	491.21	58	>3,600	>3,600	>3,600	98	>3,600	>3,600	>3,600	138	>3,600	>3,600	>3,600
19	215.98	2,027.10	975.95	59	>3,600	>3,600	>3,600	99	944.64	>3,600	>3,600	139	>3,600	>3,600	>3,600
20	191.96	1,298.48	936.78	60	>3,600	>3,600	>3,600	100	1,066.98	>3,600	>3,600	140	>3,600	>3,600	>3,600
21	258.13	1,922.94	1,575.15	61	1,228.67	>3,600	>3,600	101	1,302.17	>3,600	>3,600	141	>3,600	>3,600	>3,600
22	267.59	1,686.52	1,245.22	62	1,567.66	>3,600	>3,600	102	1,242.46	>3,600	>3,600	142	>3,600	>3,600	>3,600
23	103.52	2,205.30	1,478.43	63	1,136.37	>3,600	>3,600	103	946.14	>3,600	>3,600	143	>3,600	>3,600	>3,600
24	46.47	1,709.75	1,281.35	64	1,013.15	>3,600	>3,600	104	448.81	>3,600	>3,600	144	>3,600	>3,600	>3,600
25	1,242.92	>3,600	>3,600	65	2,120.14	>3,600	>3,600	105	>3,600	>3,600	>3,600	145	>3,600	>3,600	>3,600
26	434.00	2,988.04	1,445.87	66	1,763.87	>3,600	>3,600	106	>3,600	>3,600	>3,600	146	>3,600	>3,600	>3,600
27	917.94	>3,600	>3,600	67	441.35	>3,600	>3,600	107	>3,600	>3,600	>3,600	147	>3,600	>3,600	>3,600
28	632.78	>3,600	2,235.80	68	182.60	3,395.31	1,301.42	108	>3,600	>3,600	>3,600	148	>3,600	>3,600	>3,600
29	398.53	>3,600	>3,600	69	548.47	>3,600	>3,600	109	>3,600	>3,600	>3,600	149	>3,600	>3,600	>3,600
30	369.22	>3,600	>3,600	70	261.74	>3,600	>3,600	110	>3,600	>3,600	>3,600	150	>3,600	>3,600	>3,600
31	83.96	>3,600	>3,600	71	267.06	>3,600	>3,600	111	>3,600	>3,600	>3,600	151	>3,600	>3,600	>3,600
32	42.39	2,420.22	1,724.54	72	112.57	>3,600	>3,600	112	2,382.11	>3,600	>3,600	152	>3,600	>3,600	>3,600
33	114.88	677.09	295.20	73	>3,600	>3,600	>3,600	113	>3,600	>3,600	>3,600	153	>3,600	>3,600	>3,600
34	144.97	758.26	405.87	74	>3,600	>3,600	>3,600	114	>3,600	>3,600	>3,600	154	>3,600	>3,600	>3,600
35	108.86	846.41	753.36	75	830.00	>3,600	>3,600	115	>3,600	>3,600	>3,600	155	>3,600	>3,600	>3,600
36	202.82	1,058.94	584.56	76	>3,600	>3,600	>3,600	116	>3,600	>3,600	>3,600	156	>3,600	>3,600	>3,600
37	71.90	881.96	987.26	77	1,696.27	>3,600	>3,600	117	>3,600	>3,600	>3,600	157	>3,600	>3,600	>3,600
38	114.38	1,706.31	1,228.47	78	1,273.12	>3,600	>3,600	118	>3,600	>3,600	>3,600	158	>3,600	>3,600	>3,600
39	89.86	648.98	666.89	79	1,068.26	>3,600	>3,600	119	>3,600	>3,600	>3,600	159	>3,600	>3,600	>3,600
40	60.54	907.97	416.83	80	293.61	>3,600	>3,600	120	>3,600	>3,600	>3,600	160	>3,600	>3,600	>3,600

Table A.8. CPU times (in seconds) for different implementations of Benders decomposition for instances 1-80.

	Naive BD					Improved BD				Naive BD					Improved BD		
Instance No	Iterative (MFPC$_S$)	Single Tree (MFPC$_S$)	Single Tree (MFPC$_K$)	Single Tree (MFPC$_W$)	CPLEX BD (MFPC$_W$)	Single Tree (MFPC$_S$)	Single Tree (MFPC$_K$)	Single Tree (MFPC$_W$)	Instance No	Iterative (MFPC$_S$)	Single Tree (MFPC$_S$)	Single Tree (MFPC$_K$)	Single Tree (MFPC$_W$)	CPLEX BD (MFPC$_W$)	Single Tree (MFPC$_S$)	Single Tree (MFPC$_K$)	Single Tree (MFPC$_W$)
1	65.93	13.00	25.46	5.74	29.63	13.66	18.38	16.34	41	>3,600	>3,600	>3,600	1,512.98	>3,600	2,020.91	2,181.32	781.36
2	282.70	17.57	12.98	4.54	33.71	9.02	5.82	5.11	42	>3,600	748.35	1,059.63	121.10	3,216.24	923.01	1,106.85	158.00
3	774.11	28.55	22.72	10.98	120.01	28.37	16.11	10.99	43	>3,600	1,069.62	1,485.31	276.65	>3,600	881.89	1,332.85	253.63
4	281.38	28.88	18.63	10.27	42.35	15.06	7.96	6.30	44	3,599.66	380.63	786.05	83.68	>3,600	458.50	608.64	73.56
5	552.66	7.80	26.99	8.83	219.70	18.92	12.68	8.58	45	>3,600	345.55	1,030.55	145.99	>3,600	924.02	453.38	99.12
6	16.46	3.70	3.40	7.16	121.70	7.54	5.32	7.62	46	>3,600	198.23	544.10	83.37	>3,600	475.89	760.50	60.30
7	21.11	3.92	16.46	25.27	310.77	14.53	11.23	18.96	47	27.53	74.38	59.94	21.47	2,262.74	244.34	78.34	20.16
8	158.28	5.41	12.03	23.68	89.61	11.29	16.44	26.57	48	336.17	82.29	277.15	40.47	>3,600	156.53	43.09	33.18
9	>3,600	62.98	97.70	23.70	445.25	50.30	27.49	18.99	49	>3,600	3,217.54	2,197.64	566.03	>3,600	2,970.97	1,912.53	483.23
10	>3,600	584.45	655.48	259.52	1,243.65	670.15	318.51	226.66	50	>3,600	1,575.90	1,002.11	636.65	>3,600	2,678.99	545.61	174.91
11	1,708.06	37.13	92.45	34.74	600.87	64.55	130.46	25.93	51	>3,600	611.96	1,075.96	288.60	>3,600	899.62	388.77	224.44
12	1,674.49	45.40	63.26	24.04	323.22	106.16	50.67	22.37	52	>3,600	592.65	978.42	264.61	>3,600	2,708.02	1,359.89	184.08
13	>3,600	36.40	84.46	47.24	445.66	85.96	50.64	38.66	53	>3,600	803.21	1,364.91	141.18	>3,600	1,306.58	819.05	88.98
14	2,192.77	43.84	100.95	33.13	305.00	66.74	63.06	28.24	54	1,760.54	347.88	339.71	72.63	2,716.76	639.85	302.02	84.07
15	12.76	8.83	13.51	24.04	282.14	16.41	10.34	32.95	55	378.02	140.93	226.36	76.24	>3,600	580.22	505.13	52.88
16	543.96	32.57	26.10	33.63	518.70	79.20	48.95	37.24	56	>3,600	163.19	657.84	19.50	>3,600	190.41	286.90	23.88
17	>3,600	558.67	1,020.38	276.67	2,196.66	1,121.36	712.59	182.15	57	>3,600	>3,600	>3,600	>3,600	>3,600	>3,600	>3,600	>3,600
18	>3,600	145.16	304.68	96.03	1,302.10	157.70	94.52	32.31	58	>3,600	>3,600	>3,600	1,304.88	>3,600	>3,600	>3,600	2,721.50
19	1,011.85	56.94	220.48	38.42	388.89	82.13	46.08	36.93	59	>3,600	1,286.72	>3,600	619.27	>3,600	1,996.83	983.22	403.39
20	>3,600	120.49	507.19	49.81	2,021.14	205.36	122.74	51.33	60	>3,600	2,198.47	>3,600	1,545.95	>3,600	>3,600	>3,600	931.24
21	848.00	94.26	357.90	50.23	1,007.60	170.45	130.39	68.56	61	3,120.16	97.13	199.95	86.47	>3,600	158.71	366.13	168.93
22	2,066.63	31.89	172.90	46.94	959.90	219.52	87.92	47.77	62	>3,600	665.82	1,586.22	163.36	>3,600	>3,600	1,224.60	251.02
23	225.11	29.92	73.52	27.58	256.14	81.42	76.72	32.21	63	474.99	107.41	1,140.55	154.78	>3,600	1,744.49	546.27	92.29
24	12.57	8.41	50.11	34.46	392.62	76.24	41.14	33.51	64	>3,600	189.84	1,712.47	75.66	>3,600	213.00	513.05	79.79
25	>3,600	2,595.36	3,150.02	576.31	>3,600	1,469.91	582.47	656.86	65	>3,600	>3,600	>3,600	3,449.85	>3,600	2,972.01	3,463.70	1,165.05
26	>3,600	414.49	635.95	229.27	2,166.22	496.86	186.48	99.83	66	>3,600	1,634.90	2,237.72	908.00	2,964.85	1,180.55	1,148.27	572.83
27	>3,600	893.62	1,588.16	390.47	>3,600	1,947.26	2,133.90	165.28	67	>3,600	676.98	478.95	142.87	>3,600	368.19	515.97	76.83
28	>3,600	566.16	1,348.62	147.65	>3,600	540.09	634.30	165.50	68	>3,600	44.82	105.61	32.64	726.96	269.21	185.38	30.67
29	>3,600	326.59	514.18	55.94	2,426.00	320.19	75.55	71.56	69	1,764.59	104.99	189.67	42.71	2,133.07	458.20	41.23	22.84
30	3,584.92	161.60	644.44	81.32	2,000.72	680.89	404.59	81.89	70	>3,600	83.59	89.20	38.80	3,023.63	112.94	78.06	43.07
31	155.52	213.42	253.81	26.35	>3,600	762.08	394.38	24.63	71	>3,600	170.91	256.70	29.77	>3,600	731.28	285.31	27.32
32	8.71	6.68	9.21	9.36	702.86	22.14	9.30	9.24	72	>3,600	70.67	187.40	28.17	>3,600	302.03	389.63	29.39
33	1,648.48	154.94	214.83	22.50	509.50	121.82	45.86	22.64	73	>3,600	>3,600	>3,600	>3,600	>3,600	>3,600	>3,600	>3,600
34	>3,600	188.92	504.47	117.11	1,257.35	195.17	87.27	43.71	74	>3,600	>3,600	>3,600	3,455.42	>3,600	>3,600	>3,600	982.05
35	>3,600	177.73	198.59	70.48	1,198.05	120.84	190.10	40.48	75	>3,600	761.33	1,955.89	133.21	>3,600	2,182.56	1,743.74	155.02
36	>3,600	119.64	336.07	65.96	2,553.91	160.85	210.26	59.05	76	>3,600	>3,600	>3,600	2,274.51	>3,600	>3,600	2,599.00	825.96
37	910.86	26.08	35.88	28.81	1,516.56	120.87	16.86	41.75	77	>3,600	557.64	1,520.85	181.99	>3,600	1,200.75	413.48	119.22
38	752.36	30.80	100.23	46.39	815.87	111.03	67.05	39.62	78	>3,600	1,046.20	1,657.52	127.22	>3,600	1,110.21	1,358.23	80.28
39	49.66	27.91	86.13	19.87	586.26	54.13	60.03	25.69	79	>3,600	264.48	2,392.56	115.89	>3,600	1,202.51	582.47	72.37
40	0.42	1.79	2.08	10.80	486.04	8.72	2.82	13.18	80	29.17	146.31	425.77	58.41	>3,600	436.35	126.94	49.58

Table A.9. CPU times (in seconds) for different implementations of Benders decomposition for instances 81-160.

	Naive BD					Improved BD				Naive BD					Improved BD		
Instance No	Iterative (MFPC$_S$)	Single Tree (MFPC$_S$)	Single Tree (MFPC$_K$)	Single Tree (MFPC$_W$)	CPLEX BD (MFPC$_W$)	Single Tree (MFPC$_S$)	Single Tree (MFPC$_K$)	Single Tree (MFPC$_W$)	Instance No	Iterative (MFPC$_S$)	Single Tree (MFPC$_S$)	Single Tree (MFPC$_K$)	Single Tree (MFPC$_W$)	CPLEX BD (MFPC$_W$)	Single Tree (MFPC$_S$)	Single Tree (MFPC$_K$)	Single Tree (MFPC$_W$)
81	>3,600	>3,600	>3,600	>3,600	>3,600	>3,600	>3,600	>3,600	121	>3,600	>3,600	>3,600	>3,600	>3,600	>3,600	>3,600	>3,600
82	>3,600	>3,600	>3,600	>3,600	>3,600	>3,600	>3,600	>3,600	122	>3,600	>3,600	>3,600	>3,600	>3,600	>3,600	>3,600	>3,600
83	>3,600	2,743.26	>3,600	2,147.85	>3,600	>3,600	>3,600	1,811.32	123	>3,600	>3,600	>3,600	>3,600	>3,600	>3,600	>3,600	>3,600
84	>3,600	3,069.65	>3,600	1,269.20	>3,600	>3,600	>3,600	754.81	124	>3,600	>3,600	>3,600	>3,600	>3,600	>3,600	>3,600	>3,600
85	>3,600	3,574.48	>3,600	586.03	>3,600	>3,600	2,836.90	676.99	125	>3,600	>3,600	>3,600	>3,600	>3,600	>3,600	>3,600	2,383.48
86	>3,600	1,016.97	2,377.80	246.04	>3,600	960.41	902.37	270.69	126	>3,600	>3,600	>3,600	>3,600	>3,600	>3,600	>3,600	3,292.34
87	2,690.79	948.35	1,587.33	110.96	>3,600	437.11	286.28	96.03	127	>3,600	>3,600	>3,600	824.26	>3,600	>3,600	>3,600	584.75
88	>3,600	337.54	1,893.83	170.62	>3,600	>3,600	>3,600	178.04	128	>3,600	>3,600	>3,600	762.90	>3,600	>3,600	>3,600	663.02
89	>3,600	>3,600	>3,600	>3,600	>3,600	>3,600	>3,600	>3,600	129	>3,600	>3,600	>3,600	>3,600	>3,600	>3,600	>3,600	>3,600
90	>3,600	>3,600	>3,600	>3,600	>3,600	>3,600	>3,600	>3,600	130	>3,600	>3,600	>3,600	>3,600	>3,600	>3,600	>3,600	2,851.28
91	>3,600	>3,600	>3,600	3,427.78	>3,600	>3,600	>3,600	2,646.61	131	>3,600	>3,600	>3,600	>3,600	>3,600	>3,600	>3,600	1,985.23
92	>3,600	>3,600	>3,600	>3,600	>3,600	>3,600	>3,600	2,877.18	132	>3,600	>3,600	>3,600	1,696.83	>3,600	>3,600	>3,600	747.71
93	>3,600	>3,600	>3,600	1,027.98	>3,600	>3,600	>3,600	985.53	133	>3,600	979.01	>3,600	332.31	>3,600	>3,600	>3,600	336.80
94	>3,600	1,392.37	>3,600	696.00	>3,600	>3,600	>3,600	556.91	134	1,399.42	765.90	2,832.15	209.65	>3,600	539.18	795.09	196.45
95	>3,600	1,399.07	>3,600	232.38	>3,600	>3,600	>3,600	242.75	135	>3,600	1,747.61	>3,600	118.50	>3,600	706.34	>3,600	124.64
96	>3,600	1,638.27	3,158.68	330.58	>3,600	>3,600	>3,600	276.14	136	>3,600	818.85	2,093.80	170.65	>3,600	>3,600	>3,600	157.92
97	>3,600	>3,600	>3,600	1,645.20	>3,600	>3,600	>3,600	1,257.11	137	>3,600	>3,600	>3,600	>3,600	>3,600	>3,600	>3,600	>3,600
98	>3,600	>3,600	>3,600	>3,600	>3,600	>3,600	>3,600	>3,600	138	>3,600	>3,600	>3,600	>3,600	>3,600	>3,600	>3,600	>3,600
99	>3,600	1,269.16	1,392.90	154.85	>3,600	1,373.69	1,979.74	176.89	139	>3,600	>3,600	>3,600	>3,600	>3,600	>3,600	>3,600	>3,600
100	>3,600	556.58	3,047.32	178.67	>3,600	2,286.94	408.32	111.56	140	>3,600	>3,600	>3,600	>3,600	>3,600	>3,600	>3,600	>3,600
101	2,838.71	385.74	918.14	117.91	>3,600	3,038.77	1,022.16	107.00	141	>3,600	>3,600	>3,600	1,349.68	>3,600	>3,600	2,499.70	992.71
102	>3,600	582.52	896.35	199.98	>3,600	1,093.66	1,136.51	113.66	142	>3,600	>3,600	>3,600	2,546.24	>3,600	>3,600	>3,600	1,477.15
103	>3,600	225.86	702.19	72.68	>3,600	273.03	1,419.51	59.05	143	>3,600	2,591.82	>3,600	275.53	>3,600	>3,600	>3,600	366.57
104	>3,600	258.01	728.83	45.38	>3,600	1,779.14	1,943.46	61.92	144	>3,600	>3,600	>3,600	526.31	>3,600	>3,600	>3,600	531.03
105	>3,600	>3,600	>3,600	>3,600	>3,600	>3,600	>3,600	>3,600	145	>3,600	>3,600	>3,600	>3,600	>3,600	>3,600	>3,600	>3,600
106	>3,600	>3,600	>3,600	>3,600	>3,600	>3,600	>3,600	>3,600	146	>3,600	>3,600	>3,600	>3,600	>3,600	>3,600	>3,600	>3,600
107	>3,600	>3,600	>3,600	>3,600	>3,600	>3,600	>3,600	1,125.31	147	>3,600	>3,600	>3,600	>3,600	>3,600	>3,600	>3,600	>3,600
108	>3,600	>3,600	>3,600	3,349.54	>3,600	>3,600	>3,600	1,988.66	148	>3,600	>3,600	>3,600	>3,600	>3,600	>3,600	>3,600	>3,600
109	>3,600	>3,600	>3,600	565.70	>3,600	>3,600	>3,600	491.87	149	>3,600	>3,600	>3,600	>3,600	>3,600	>3,600	>3,600	>3,600
110	>3,600	>3,600	>3,600	580.09	>3,600	>3,600	>3,600	585.20	150	>3,600	>3,600	>3,600	3,513.45	>3,600	>3,600	>3,600	3,400.77
111	>3,600	1,005.29	1,722.06	255.33	>3,600	>3,600	>3,600	213.42	151	>3,600	>3,600	>3,600	1,411.01	>3,600	>3,600	>3,600	1,425.92
112	>3,600	319.44	2,020.47	163.80	>3,600	632.30	>3,600	147.65	152	>3,600	>3,600	>3,600	1,322.43	>3,600	>3,600	>3,600	1,489.58
113	>3,600	>3,600	>3,600	>3,600	>3,600	>3,600	>3,600	>3,600	153	>3,600	>3,600	>3,600	>3,600	>3,600	>3,600	>3,600	>3,600
114	>3,600	>3,600	>3,600	>3,600	>3,600	>3,600	>3,600	>3,600	154	>3,600	>3,600	>3,600	>3,600	>3,600	>3,600	>3,600	>3,600
115	>3,600	>3,600	>3,600	>3,600	>3,600	>3,600	>3,600	2,822.26	155	>3,600	>3,600	>3,600	>3,600	>3,600	>3,600	>3,600	>3,600
116	>3,600	>3,600	>3,600	>3,600	>3,600	>3,600	>3,600	>3,600	156	>3,600	>3,600	>3,600	>3,600	>3,600	>3,600	>3,600	>3,600
117	>3,600	>3,600	>3,600	3,573.87	>3,600	>3,600	>3,600	2,509.22	157	>3,600	>3,600	>3,600	>3,600	>3,600	>3,600	>3,600	>3,600
118	>3,600	782.86	>3,600	1,022.20	>3,600	>3,600	>3,600	698.26	158	>3,600	>3,600	>3,600	>3,600	>3,600	>3,600	>3,600	>3,600
119	>3,600	961.76	>3,600	498.70	>3,600	>3,600	>3,600	420.72	159	>3,600	>3,600	>3,600	2,668.07	>3,600	>3,600	>3,600	2,500.11
120	>3,600	>3,600	>3,600	784.37	>3,600	>3,600	2,746.56	812.54	160	>3,600	>3,600	>3,600	2,225.69	>3,600	>3,600	>3,600	1,912.25

Table A.10. CPU times (in seconds) of exact solution algorithms to solve MFPC.

Instance No	BD$_W$	BB	RDS	Instance No	BD$_W$	BB	RDS	Instance No	BD$_W$	BB	RDS	Instance No	BD$_W$	BB	RDS
1	16.34	1.81	1.22	41	781.36	87.44	171.16	81	>3,600	>3,600	>3600	121	>3,600	>3,600	>3,600
2	5.11	1.15	2.68	42	158.00	45.66	42.51	82	>3,600	>3,600	>3,600	122	>3,600	>3,600	>3,600
3	10.99	1.12	0.89	43	253.63	20.11	23.21	83	1,811.32	613.47	632.41	123	>3,600	>3,600	>3,600
4	6.30	0.53	0.36	44	73.56	15.94	12.18	84	754.81	>3,600	950.95	124	>3,600	>3,600	>3,600
5	8.58	0.23	0.37	45	99.12	23.73	8.02	85	676.99	103.38	149.8	125	2,383.48	1,019.57	735.11
6	7.62	0.27	0.23	46	60.30	13.24	7.27	86	270.69	225.92	125.25	126	3,292.34	1,631.64	1,591.30
7	18.96	0.02	0.12	47	20.16	0.2	1.23	87	96.03	1.93	13.73	127	584.75	34.6	104.36
8	26.57	0.08	0.16	48	33.18	2.31	4.45	88	178.04	24.45	20.09	128	663.02	40.95	197.06
9	18.99	3.17	4.7	49	483.23	1,729.33	432.9	89	>3,600	>3,600	>3,600	129	>3,600	>3,600	>3,600
10	226.66	52.81	32.39	50	174.91	>3,600	1,002.80	90	>3,600	>3,600	>3,600	130	2,851.28	>3,600	>3,600
11	25.93	2.67	4.9	51	224.44	741.84	226.06	91	2,646.61	>3,600	1,743.33	131	1,985.23	>3,600	536.33
12	22.37	5.24	3.4	52	184.08	50.36	177.89	92	2,877.18	>3,600	>3,600	132	747.71	957.28	381.84
13	38.66	4.32	1.25	53	88.98	27.89	59.08	93	985.53	674.47	386.49	133	336.80	22.96	29.5
14	28.24	0.76	1.81	54	84.07	16.82	15.41	94	556.91	122.02	391.2	134	196.45	10.2	26.99
15	32.95	0.13	0.42	55	52.88	2.08	6.8	95	242.75	12.48	27	135	124.64	18.5	25.08
16	37.24	1.03	0.8	56	23.88	1.47	3.57	96	276.14	22.59	43.07	136	157.92	7.61	26.88
17	182.15	815.02	97.53	57	>3,600	>3,600	>3,600	97	1,257.11	>3,600	>3,600	137	>3,600	>3,600	>3,600
18	32.31	15.48	23.67	58	2,721.50	>3,600	>3,600	98	>3,600	>3,600	979.93	138	>3,600	>3,600	>3,600
19	36.93	2.39	7.43	59	403.39	121.03	92.66	99	176.89	233.52	48.38	139	>3,600	>3,600	3,239.05
20	51.33	6.9	14.66	60	931.24	240.7	200.68	100	111.56	115.91	70.75	140	>3,600	>3,600	>3,600
21	68.56	3.68	3.35	61	168.93	8.86	42	101	107.00	53.51	40.7	141	992.71	2,054.79	712.22
22	47.77	3.03	2.34	62	251.02	99.51	55.93	102	113.66	8.72	7.97	142	1,477.15	653.31	543.16
23	32.21	0.36	1.36	63	92.29	4.99	19.27	103	59.05	1.92	7.02	143	366.57	39.03	42.04
24	33.51	0.27	1.05	64	79.79	2.71	7.82	104	61.92	1.23	4.06	144	531.03	114.21	133.85
25	656.86	1,971.84	1,046.06	65	1,165.05	>3,600	1,913.83	105	>3,600	>3,600	>3,600	145	>3,600	>3,600	>3,600
26	99.83	44.48	155.61	66	572.83	123.19	210.87	106	>3,600	>3,600	784.78	146	>3,600	>3,600	>3,600
27	165.28	76.66	40.11	67	76.83	43.82	28.31	107	1,125.31	>3,600	842	147	>3,600	>3,600	>3,600
28	165.50	148.52	70.28	68	30.67	28.02	7.94	108	1,988.66	3,515.93	858.8	148	>3,600	>3,600	>3,600
29	71.56	5.09	4.81	69	22.84	9.13	8.1	109	491.87	81.62	101.13	149	>3,600	1,397.73	1,465.65
30	81.89	4.42	7.16	70	43.07	1.36	4.45	110	585.20	163.91	101.46	150	3,400.77	1,722.46	640.91
31	24.63	1.14	2	71	27.32	3.12	2.59	111	213.42	6.1	29.14	151	1,425.92	140.53	263.03
32	9.24	0.02	0.62	72	29.39	0.45	1.65	112	147.65	21.75	21.28	152	1,489.58	60.14	171.55
33	22.64	9.42	5.62	73	>3,600	>3,600	>3,600	113	>3,600	>3,600	>3,600	153	>3,600	>3,600	>3,600
34	43.71	16.9	12.68	74	982.05	>3,600	1,662.31	114	>3,600	>3,600	>3,600	154	>3,600	>3,600	>3,600
35	40.48	7.24	5.8	75	155.02	46.6	48.72	115	2,822.26	>3,600	2,438.30	155	>3,600	>3,600	>3,600
36	59.05	19.55	8.69	76	825.96	1,674.85	526.28	116	>3,600	>3,600	>3,600	156	>3,600	>3,600	>3,600
37	41.75	2.42	0.84	77	119.22	42.45	46.83	117	2,509.22	541.88	732.33	157	>3,600	>3,600	>3,600
38	39.62	4.88	1.25	78	80.28	57.8	49.92	118	698.26	209.45	271.63	158	>3,600	>3,600	>3,600
39	25.69	0.7	0.66	79	72.37	2.39	4.46	119	420.72	20.11	72.56	159	2,500.11	895.8	839.03
40	13.18	0.05	0.19	80	49.58	1.19	4.68	120	812.54	136.13	84.29	160	1,912.25	78.06	513.15

Table A.11. Upper and lower bounds for instances 1-80.

Instance	BD_W		BB		RDS		Instance	BD_W		BB		RDS	
No	UB	LB	UB	LB	UB	LB	No	UB	LB	UB	LB	UB	LB
1	37	37	37	37	37	37	41	37	37	37	37	37	37
2	49	49	49	49	49	49	42	53	53	53	53	53	53
3	24	24	24	24	24	24	43	35	35	35	35	35	35
4	33	33	33	33	33	33	44	38	38	38	38	38	38
5	12	12	12	12	12	12	45	21	21	21	21	21	21
6	34	34	34	34	34	34	46	18	18	18	18	18	18
7	13	13	13	13	13	13	47	24	24	24	24	24	24
8	15	15	15	15	15	15	48	37	37	37	37	37	37
9	35	35	35	35	35	35	49	40	40	40	40	40	40
10	51	51	51	51	51	51	50	57	57	66	55	57	57
11	26	26	26	26	26	26	51	40	40	40	40	40	40
12	50	50	50	50	50	50	52	53	53	53	53	53	53
13	13	13	13	13	13	13	53	25	25	25	25	25	25
14	19	19	19	19	19	19	54	38	38	38	38	38	38
15	25	25	25	25	25	25	55	25	25	25	25	25	25
16	19	19	19	19	19	19	56	18	18	18	18	18	18
17	43	43	43	43	43	43	57	67.42	45	70	49	383	47
18	55	55	55	55	55	55	58	73	73	104	73	493	71
19	33	33	33	33	33	33	59	38	38	38	38	38	38
20	35	35	35	35	35	35	60	51	51	51	51	51	51
21	27	27	27	27	27	27	61	39	39	39	39	39	39
22	34	34	34	34	34	34	62	37	37	37	37	37	37
23	15	15	15	15	15	15	63	26	26	26	26	26	26
24	34	34	34	34	34	34	64	33	33	33	33	33	33
25	45	45	45	45	45	45	65	36	36	44	36	36	36
26	80	80	80	80	80	80	66	53	53	53	53	53	53
27	30	30	30	30	30	30	67	22	22	22	22	22	22
28	48	48	48	48	48	48	68	34	34	34	34	34	34
29	26	26	26	26	26	26	69	27	27	27	27	27	27
30	36	36	36	36	36	36	70	33	33	33	33	33	33
31	24	24	24	24	24	24	71	12	12	12	12	12	12
32	37	37	37	37	37	37	72	19	19	19	19	19	19
33	37	37	37	37	37	37	73	40	39	50	39	213	39
34	48	48	48	48	48	48	74	64	64	67	64	64	64
35	24	24	24	24	24	24	75	26	26	26	26	26	26
36	33	33	33	33	33	33	76	38	38	38	38	38	38
37	21	21	21	21	21	21	77	29	29	29	29	29	29
38	33	33	33	33	33	33	78	37	37	37	37	37	37
39	13	13	13	13	13	13	79	14	14	14	14	14	14
40	37	37	37	37	37	37	80	37	37	37	37	37	37

Table A.12. Upper and lower bounds for instances 81-160.

Instance	BD_W		BB		RDS		Instance	BD_W		BB		RDS	
No	UB	LB	UB	LB	UB	LB	No	UB	LB	UB	LB	UB	LB
81	72.61	40	61	53	375	40	121	121.87	41	92	43	459	50
82	98.69	52	84	68	434	53	122	176.92	55	145	72	809	61
83	40	40	40	40	40	40	123	75.85	34	60	35	453	35
84	39	39	46	39	39	39	124	80.79	52	66	53	424	54
85	26	26	26	26	26	26	125	32	32	32	32	32	32
86	37	37	37	37	37	37	126	53	53	53	53	53	53
87	25	25	25	25	25	25	127	25	25	25	25	25	25
88	19	19	19	19	19	19	128	36	36	36	36	36	36
89	88.01	53	84	53	458	49	129	62.52	31	51	38	289	35
90	136.65	51	103	53	547	54	130	69	69	90	56	388	52
91	41	41	43	41	41	41	131	23	23	25	23	23	23
92	53	53	69	53	365	53	132	37	37	37	37	37	37
93	27	27	27	27	27	27	133	15	15	15	15	15	15
94	53	53	53	53	53	53	134	50	50	50	50	50	50
95	24	24	24	24	24	24	135	15	15	15	15	15	15
96	33	33	33	33	33	33	136	34	34	34	34	34	34
97	39	39	67	39	190	37	137	77.55	54	85	54	382	46
98	55	35	52	38	47	47	138	102.82	49	90	52	434	50
99	23	23	23	23	23	23	139	47.99	29	38	29	36	36
100	36	36	36	36	36	36	140	64.36	50	57	50	388	50
101	25	25	25	25	25	25	141	28	28	28	28	28	28
102	19	19	19	19	19	19	142	50	50	50	50	50	50
103	14	14	14	14	14	14	143	26	26	26	26	26	26
104	20	20	20	20	20	20	144	34	34	34	34	34	34
105	92.49	45	76	48	369	48	145	107.75	40	78	48	478	44
106	83.36	52	67	65	65	65	146	145.68	48	112	51	508	47
107	31	31	35	31	31	31	147	51.94	38	40	38	281	39
108	36	36	36	36	36	36	148	106.51	38	71	55	556	50
109	26	26	26	26	26	26	149	29	25	25	25	25	25
110	33	33	33	33	33	33	150	35	35	35	35	35	35
111	25	25	25	25	25	25	151	23	23	23	23	23	23
112	18	18	18	18	18	18	152	33	33	33	33	33	33
113	81.18	34	69	38	416	39	153	139.49	41	103	33	536	35
114	104.02	48	92	49	433	50	154	202.83	68	133	81	830	56
115	35	35	41	35	35	35	155	90.62	33	69	37	547	37
116	98.88	55	81	55	637	35	156	125.18	37	94	38	689	46
117	25	25	25	25	25	25	157	84.32	38	54	38	598	38
118	50	50	50	50	50	50	158	92.95	37	58	48	655	48
119	24	24	24	24	24	24	159	26	26	26	26	26	26
120	19	19	19	19	19	19	160	36	36	36	36	36	36