DECOMPOSITION ALGORITHMS FOR THE ELEMENTARY SHORTEST PATH PROBLEM IN NETWORKS CONTAINING NEGATIVE CYCLES

By

DEVARAJA VIGNESH RADHA KRISHNAN

Bachelor of Engineering
Anna University
Chennai, Tamil Nadu, India
2012

Submitted to the Faculty of the
Graduate College of
Oklahoma State University
in partial fulfillment of
the requirements for
the Degree of
MASTER OF SCIENCE
December, 2015

DECOMPOSITION ALGORITHMS FOR THE ELEMENTARY SHORTEST PATH PROBLEM IN NETWORKS CONTAINING NEGATIVE CYCLES

Committee members:

Dr. Balabhaskar Balasundaram

Thesis Advisor

Dr. Arash Pourhabib

Dr. Chaoyue Zhao

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to my advisor Dr. Balabhaskar Balasundaram for his support, motivation and patience. His guidance helped me during the research period and writing of this document. I could not have imagined having a better advisor.

Besides my advisor, I would like to thank the rest of my thesis committee: Dr. Arash Pourhabib and Dr. Chaoyue Zhao for their insightful comments and encouragement. Their questions helped me to attain a better approach towards the research.

My sincere thanks also goes to Dr. Mamane Souley Ibrahim, who graciously shared his research test-bed with me. The research moved forward in the right direction because of his support.

Last but not least, I would like to thank my parents, sister, grandparents and friends for supporting me spiritually throughout the research. I am indebted to them for my life.

Disclaimer: Acknowledgements reflect the views of the author and are not endorsed by committee members or Oklahoma State University.

TABLE OF CONTENTS

LIST OF TABLES

Table Page

LIST OF FIGURES

Figure Page

CHAPTER 1

INTRODUCTION

1.1 Elementary shortest path problem with negative cycles

The *Elementary Shortest Path Problem* (ESPP) in a directed network seeks a directed path from a specified source node to a specified sink node such that the sum of arc weights of all the arcs on the path is a minimum. A network is said to contain a negative cycle if it contains a directed cycle, with sum of its arc weights being negative. ESPP in networks without negative cycles is a well-solved problem. However, finding an elementary shortest path in networks containing negative cycles is an NP-hard problem [1].

Algorithms solving ESPP seek a directed walk of the shortest length. A directed walk can be visualized as a directed path with directed cycles attached to it. If the directed cycles attached to the path are not negative cycles, then it is possible to remove those cycles from the walk without increasing the objective function and a shortest path can be identified. However, in the presence of a negative cycle, it is no longer possible to remove the cycle without increasing the objective function. In order to find a directed path in a network containing negative cycles, it is necessary to restrict walks from revisiting nodes. This additional requirement makes this problem more difficult to solve. This thesis proposes and investigates a new approach to solve the *Elementary Shortest Path Problem with Negative Cycles* (ESPPNC).

1.2 Motivation

The *Longest Path Problem* (LPP) is used to find the longest distance between two nodes in a graph. LPP is used to calculate critical paths in the parallel precedence-constrained job scheduling problem i.e. scheduling jobs on identical processors when certain jobs have to be completed before beginning certain other jobs, such that all the jobs are completed in the minimum amount of time while still respecting the constraints [2]. In this problem, a feasible path is a subgraph in which each task must wait for the one before it to get completed. Any path with two parallel tasks is excluded from the solution because they may violate the precedence constraints. On the contrary, no job on the longest path can start before the previous tasks are finished; so each task is finished as soon as the longest path to its end point is finished. Consequently, the whole job is finished as the longest path is completed.

Unlike the shortest path problem, LPP is an NP-hard problem [3]. Some effective approximation algorithms are available in the literature which can solve LPP [4, 5]. In order to find the longest path in a graph with nonnegative arc weights, all arc weights can be multiplied by -1 and a shortest path algorithm can be used. During this procedure negative cycles may be created. So an ESPPNC algorithm can also be used to find a longest path.

ESPPNC also arises as a sub-problem in column generation schemes for the vehicle routing problem with time-windows [6, 7]. When this problem is solved using a branch-and-price algorithm, arcs with negative reduced cost can be involved and consequently paths with negative reduced cost can also be involved. However, the optimal solution is identified when there are no more negative reduced cost paths. So, finding a most negative reduced cost path is necessary to validate the optimality of a solution, but since there are arcs with negative reduced cost, negative cycles can be present. An ESPPNC model can be used here to identify improving columns.

1.3 Organization

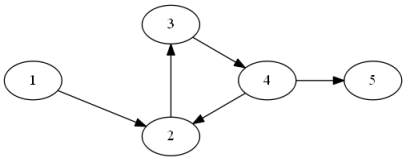
This thesis is organized as follows: The remainder of this chapter discusses the ESPPNC, including formal definitions and a review of the classical algorithms used to solve the ESPP, before providing the reasons why the classical approaches fail to correctly solve the ESPPNC. This chapter also presents the approaches in the literature to solve the ESPPNC and introduces the research problem. Chapter 2 introduces our decomposition ideas and the resulting decomposition and branch-and-cut algorithms for this problem.

Chapter 3 is devoted to the computational study, which presents the objectives of the experiments, provides implementation details, summarizes the results and discusses the findings. Finally, Chapter 5 presents the conclusions of this thesis and directions for future research.

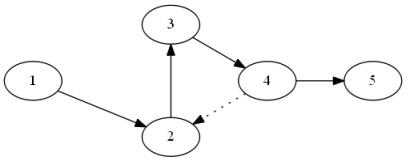
1.4 Definitions and notations

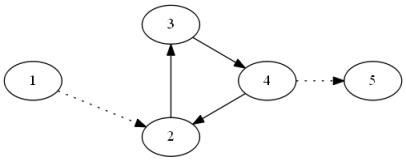
Consider a simple, finite and directed graph $G = (N, A)$, consisting of N, a set of nodes and A, a set of arcs. This graph G contains n nodes and m arcs ($|N| = n$ and $|A| = m$). For each arc $(i, j) \in A$, i is the tail node and j is the head node. The arc is oriented from the tail node to the head node. The cost or weight of an arc $(i, j) \in A$ is c_{ij}. In many practical cases, the networks have nonnegative arc weights, but in some cases networks can have negative arc weights. For each node i, $\Gamma^+(i)$ represents all nodes which are out-neighbors of node i. Similarly $\Gamma^-(i)$ represents all nodes which are in-neighbors of node i. Formally, $\Gamma^+(i) = \{j \in N \mid (i, j) \in A\}$ and $\Gamma^-(i) = \{j \in N \mid (j, i) \in A\}$.

Definition 1 *In a directed graph, a directed walk is a sequence of nodes $v_1, v_2, ..., v_k$, such that $(v_i, v_{i+1}) \in A$, for $i = 1, 2, ..., k - 1$.*



(a)





(b) (c)

Figure 1.1: (a) A directed walk between node 1 and 5; (b)A directed path between node 1 and 5; (c)A directed cycle 2-3-4-2

In a directed walk, the nodes need not be distinct i.e., there may be repetition of nodes. In the above definition, v_1 and v_k are the end nodes of the walk.

Definition 2 *A directed path is a directed walk without any repeated nodes.*

Definition 3 *A directed cycle is a closed directed path* $v_1, v_2, ..., v_k$, *such that* $(v_k, v_1) \in A$.

Every directed walk from nodes s to node i can be decomposed into a directed path from node s to i and *arc-disjoint* directed cycles [8, p.79-81].

1.5 Classical approaches for the shortest path problem

The ESPP without negative cycles is a well-solved problem, which has several strongly polynomial time algorithms [8]. The classical combinatorial algorithms typically solve the single source shortest path problem by finding a shortest path tree, a directed out-tree rooted at the source node that contains shortest paths from the source node s to all the other nodes. None of the classical approaches work correctly in the presence of negative cycles. In this section, the working principle of the classical algorithms as well as the reason they fail to find a shortest path in the presence of negative cycles

is discussed.

The elementary shortest path problem can be solved in polynomial time in directed acyclic graphs, by scanning out-neighbors of nodes in a topological order, but this does not apply to the graphs that contain a directed cycle as no topological ordering exists for those graphs. For such non-acyclic graphs, the well-known *Dijkstra's Algorithm* [9] can be used, but it can fail to correctly identify a shortest path in the presence of an arc with negative weight.

1.5.1 Label-correcting algorithms

A label-correcting algorithm typified by the *Bellman-Ford Algorithm* [10, 11, 12] is efficient for finding shortest paths in networks with directed cycles and negative arc weights, but not in the networks containing negative cycles. Label-correcting algorithms are iterative approaches that work on the basis of *distance labels*. In label-correcting algorithms, the distance labels on nodes are updated as better (shorter) paths are discovered in each iteration, until the termination condition is reached. The nodes are guaranteed to receive optimal distance labels only after the algorithm termination. The optimality condition on which the label-correcting algorithms are based is as follows:

Theorem 1 [*8, p.*136] *Consider a directed network* $G = (N, A)$ *with no directed negative weight cycles. For each node* $j \in N$*, let* $d(j) < \infty$ *be the distance label that denotes the length between source node* s *and* j *on a directed path. Then,* $d(j)$ *is optimal if and only if* $d(j) \leq d(i) + c_{ij}\ \forall (i, j) \in A$.

This condition states that the shortest path distance label of node j is not greater than shortest path length up to node i along with the length of arc (i, j). *Reduced arc lengths or reduced costs* with respect to distance label vector d is associated with each arc $(i, j) \in A$ as $c_{ij}^d = c_{ij} + d(i) - d(j)$. Some of the properties associated with reduced arc length are as follows [8, p. 44] :

Property 1 *For any directed cycle* W, $\sum_{(i,j)\in W} c_{ij}^d = \sum_{(i,j)\in W} c_{ij}$.

Property 2 *d is optimal if and only if,* $c_{ij}^d \geq 0 \ \forall (i,j) \in A$

The generic label-correcting algorithm shown in Algorithm 1, works based on this optimality condition. This algorithm arbitrarily chooses any arc $(i,j) \in A$ that violates the optimality condition ($c_{ij}^d < 0$) and updates the distance lable $d(j)$ accordingly.

Algorithm 1 Generic label-correcting Algorithm

Require: $G = (N, A)$ without any negative cycles, source node s and c_{ij}

1: $d(s) := 0$, $pred(s) := 0$

2: $d(j) := \infty \ \forall j \in N \setminus \{s\}$

3: **while** some arc $(i,j) \in A$ satisfies $d(j) > d(i) + c_{ij}$ **do**

4: $\quad d(j) := d(i) + c_{ij}$, $pred(j) := i$

5: **end while**

In the presence of negative cycles Algorithm 1 does not terminate. Consider a directed cycle W which contains arcs that satisfies the optimality condition. Property 2 implies that $\sum_{(i,j)\in W} c_{ij}^d \geq 0$. On the contrary, Property 1 implies that $\sum_{(i,j)\in W} c_{ij}^d = \sum_{(i,j)\in W} c_{ij} \geq 0$. So, in the presence of negative cycles no distance labels could satisfy the Properties 1 and 2. Because of this reason, the optimality condition which serves as the basis of label-correcting algorithms is no longer reached in the presence of negative cycles. Figure 1.2 shows a directed graph with a negative cycle 1-2-1 of cost -7. Generic label-correcting algorithm keeps on traversing this cycle, because with each iteration the new distance label of node 1 and 2 will be lesser by 7 units than the previous label.

1.5.2 Linear programming

Flow on an arc $(i,j) \in A$ represented by x_{ij}, is the decision variable of the Linear Programming (LP) formulation for the ESPP. If the arc (i,j) is on a shortest path, then

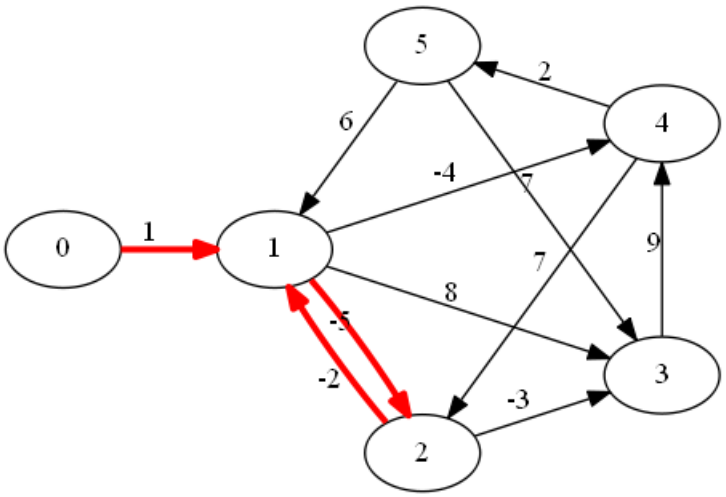


Figure 1.2: A cycle detected from a graph after termination of Bellman-Ford algorithm

x_{ij} is 1 and 0 otherwise, as every extreme point optimal solution to this formulation is binary. The following is the LP formulation for the ESPP.

Formulation 1

$$Min \sum_{(i,j)\in A} c_{ij}x_{ij} \tag{1.1}$$

Subject to:

$$\sum_{j\in\Gamma^+(i)} x_{ij} - \sum_{j\in\Gamma^-(i)} x_{ji} = \begin{cases} +1, i = s \\ -1, i = t \\ 0, otherwise \end{cases} \tag{1.2}$$

$$0 \leq x_{ij} \leq 1 \ \forall (i,j) \in A \tag{1.3}$$

If this LP formulation is used to solve ESPP, the extreme point optimal solutions are integral because of *the totally unimodular structure of the node-arc incidence matrix* [8, p. 449].

The above formulation is sufficient to solve ESPP on a directed graph with negative arc weights, but if negative cycles are present then the optimal solution obtained may not necessarily be a directed path; it is however guaranteed to be a directed walk

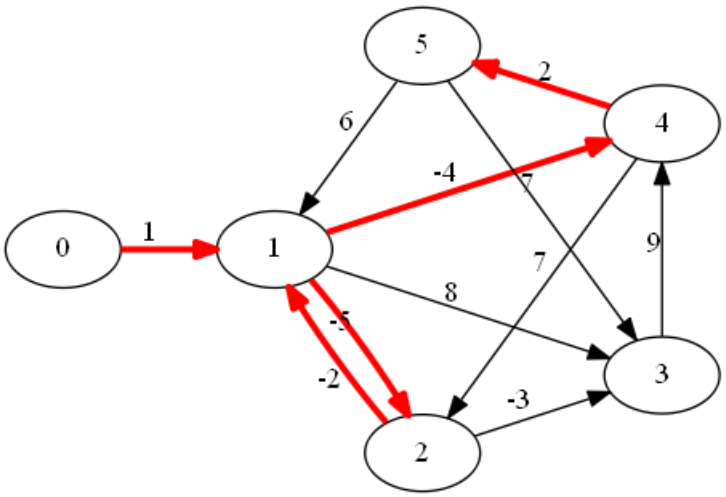


Figure 1.3: A solution to the basic LP formulation

from source node to sink node which satisfies the flow conservation constraints. An example for this case is shown in Figure 1.3. The optimal solution here is a directed walk from the source node to the sink node with a negative cycle attached to it (1-2-1). The optimal solution is not a directed path, but still the flow conservation constraints are satisfied.

These classical approaches can fail to produce correct results while solving the ESPPNC. Presenting and analyzing two new approaches to solve the ESPPNC is the objective of this thesis.

1.6 Literature review

The ESPPNC is a class of problem that has received very sparse attention in this field. In this section, the studies focused on solving the ESPPNC are summarized before the new algorithms are presented.

Drexl [13] proposed a method by which violated *Subtour Elimination Constraints* (SEC) can be identified while solving the ESPP in a branch-and-cut framework. The conventional approach for identifying violated SECs was maximum flow algorithms [14]. The minimum cut which has a strong relationship with maximum flow was used

in [15] to identify violated SECs in the asymmetric traveling salesman problem. In [13], a different approach for the separation problem was proposed. This approach is based on the graph construct called *strong component*.

Definition 4 *[13] A strong component is a strongly connected subgraph $G' = (N', A')$ of graph $G = (N, A)$, such that there are no nodes $i \in N'$ and $j \in N \setminus N'$ containing a directed $i - j$ path and a $j - i$ path.*

A nontrivial strong component is a strong component containing more than one node. The separation procedure presented in [13] is based on the fact that a subtour contains exactly one non trivial strong component. Based on the results, the time taken for separation problem (identifying violated SECs) was shown to be comparatively much better than maximum flow algorithms. The approach proposed in this paper is not limited only to ESPP. It is also applicable to various NP-hard problems, including *shortest Hamiltonian path problem*, *Traveling Salesman Problem* (TSP) and its variations.

Ibrahim et al. [16] analyzed two types of flow based formulations to find an elementary shortest path in directed networks containing negative cycles. This work was mainly focused on presenting two different exact formulation approaches. The first formulation was closely associated with the asymmetric TSP [15]. SEC lie at the heart of this flow-based formulation.

Formulation 2

$$Min \sum_{(i,j)\in A} c_{ij}x_{ij} \tag{1.4}$$

Subject to:

$$Constraint\ (1.2)$$

$$\sum_{(i,j)\in S} x_{ij} \leq |S| - 1\ \forall S \subseteq N, 2 \leq |S| \leq |N| \tag{1.5}$$

$$x_{ij} \in \{0,1\}\ \forall (i,j) \in A \tag{1.6}$$

Formulation 2 has an exponential number of constraints. However, Formulation 3 also introduced in [16], has a polynomial number of constraints. It is a MIP formulation which is closely related to the earlier work done by Maculan et al. [17]. In Formulation 3, a binary variable y_i is associated with each node $i \in N$; $y_i = 1$, if the node i is in the elementary $s-t$ shortest path P and $y_i = 0$ otherwise. The flow through an arc $(i,j) \in A$, from source node s to some other node k is defined by variable z_{ij}^k.

Formulation 3

$$Min \sum_{(i,j)\in A} c_{ij} x_{ij} \tag{1.7}$$

Subject to:

$$\sum_{j\in\Gamma^+(s)} z_{sj}^k - \sum_{j\in\Gamma^-(s)} z_{js}^k = y_k\ \forall k \in N \setminus \{s\} \tag{1.8}$$

$$\sum_{j\in\Gamma^+(i)} z_{ij}^k - \sum_{j\in\Gamma^-(i)} z_{ji}^k = 0\ \forall k \in N \setminus \{s\}, i \in N \setminus \{s,k\} \tag{1.9}$$

$$\sum_{j\in\Gamma^+(k)} z_{kj}^k - \sum_{j\in\Gamma^-(k)} z_{jk}^k = -y_k\ \forall k \in N \setminus \{s\} \tag{1.10}$$

$$\sum_{j\in\Gamma^+(i)} x_{ij} = y_i\ \forall i \in N \setminus \{s,t\} \tag{1.11}$$

$$\sum_{j\in\Gamma^-(i)} x_{ji} = y_i\ \forall i \in N \setminus \{s,t\} \tag{1.12}$$

$$\sum_{j\in\Gamma^+(s)} x_{sj} = 1 \tag{1.13}$$

$$\sum_{j\in\Gamma^-(t)} x_{jt} = 1 \tag{1.14}$$

$$0 \le z_{ij}^k \le x_{ij}\ \forall (i,j) \in A, k \in N \setminus \{s\} \tag{1.15}$$

$$x_{ij} \in \{0,1\}\ \forall (i,j) \in A \tag{1.16}$$

$$y_i \in \{0,1\}\ \forall i \in N \tag{1.17}$$

Directed walks from the source node to the sink node are no longer feasible to Formulation 3. Consider Figure 1.3 which is a directed walk from the source node to the sink node, an optimal solution to the LP formulation of ESPP. In this solution, $y_1 = 1$, because node 1 is in the solution. Note that, node 1 is a 4 degree node with two units of inflow and two units of outflow. However, Constraints (1.11-1.12) ensures that, if $y_i = 1$ then i is a 2 degree node. So, a directed walk which is optimal to the LP formulation is infeasible to Formulation 3.

In a graph with n nodes and m arcs, Formulation 3 yields $(n-1)$ $(n+2m+2)$ constraints. The linear relaxation of Formulation 3 was also investigated in [16]. The performance of the LP relaxation of Formulation 3 was comparativley better than Formulation 2.

When solving the ESPPNC to optimality, Formulation 3 can identify an elementary shortest path even in the presence of directed negative cycles. However, finding the optimal solution by directly solving this MIP model is difficult.
Ibrahim et al. [18] builds on Formulation 3 presented in [16]. In [18], the optimal solution of the LP relaxation of Formulation 3 was identified and gradually strengthened by adding violated inequalities in a cutting plane algorithm framework. Also, in [18] two different families of lifted inequalities (simple lifted valid inequalities and co-cycle lifted valid inequalities) were proposed and investigated. Simple lifted valid inequalities proved to be more effective when used in this exact algorithm.

All the past work in the literature involves exact algorithms that strengthen exact formulations by adding cutting planes. However, a decomposition approach has not been explored to solve ESPPNC. Two algorithms are presented in this thesis that solve ESPPNC using a *decomposition and branch-and-cut technique* (DBC). DBC approaches start with a master relaxation of the original formulation and then strengthen the master relaxation by adding lazy constraints in a branch-and-bound (BB) framework.

CHAPTER 2

DECOMPOSITION ALGORITHMS FOR ESPPNC

2.1 Complete formulation

A complete formulation for the ESPPNC is presented in this section before introducing the master relaxation. Our complete formulation is a different presentation of Formulation 3 discussed in Section 1.6.

In order to model the shortest path problem in a network containing negative cycles, we employ additional constraints called *cycle elimination constraints*. Similar to SEC, for every negative cycle containing q arcs, a constraint enforcing the selection of at most $q-1$ arcs from that cycle is included in the formulation. We refer to this as a *cycle elimination constraint.*

Formulation 4

$$Min \sum_{(i,j)\in A} c_{ij}x_{ij} \tag{2.1}$$

Subject to:

$$Constraint\ (1.2)$$

$$\sum_{(i,j)\in E(W)} x_{ij} \leq |W| - 1\ \forall W \in \mathcal{W} \tag{2.2}$$

$$x_{ij} \in \{0,1\}\ \forall (i,j) \in A, \tag{2.3}$$

where, $\mathcal{W}$ is the collection of all the negative cycles in the network. Constraints (2.2) represents the cycle elimination constraints.

In some graph instances, the number of negative cycles can be exponential. Assuming a bidirected, complete graph with negative arc weights and n nodes, the worst case scenario is a total of 2^{n-1} negative cycles. This exact formulation becomes unmanageable if we aim to solve it directly using a commercial MIP solver. So, a master relaxation to the complete formulation is proposed where the cycle elimination constraints are relaxed, which is then used in a decomposition and branch-and-cut algorithm.

2.2 Master relaxation

Definition 5 *A Path-Cycle tuple (PC-tuple) is a subgraph containing exactly one elementary s-t path and at least one directed cycle, with the path and cycle(s) being pairwise node-disjoint.*

An example of a PC-tuple is shown in Figure 2.1. In order to add a cycle elimination constraint in a cutting plane algorithm, a negative cycle in the solution should be identified first. This process of detecting a negative cycle in the solution is *the separation problem.* The negative cycle elimination constraints are relaxed at the beginning of the DBC algorithm and a Master Relaxation Problem (MRP) is obtained.

Formulation 5 (MRP)

$$Min \sum_{(i,j)\in A} c_{ij}x_{ij} \tag{2.4}$$

Subject to:

$$Constraint\ (1.2)$$

$$\sum_{j\in\Gamma^+(i)} x_{ij} \leq 1 \quad \forall i \neq s,t \tag{2.5}$$

$$\sum_{j\in\Gamma^-(s)} x_{js} = 0 \tag{2.6}$$

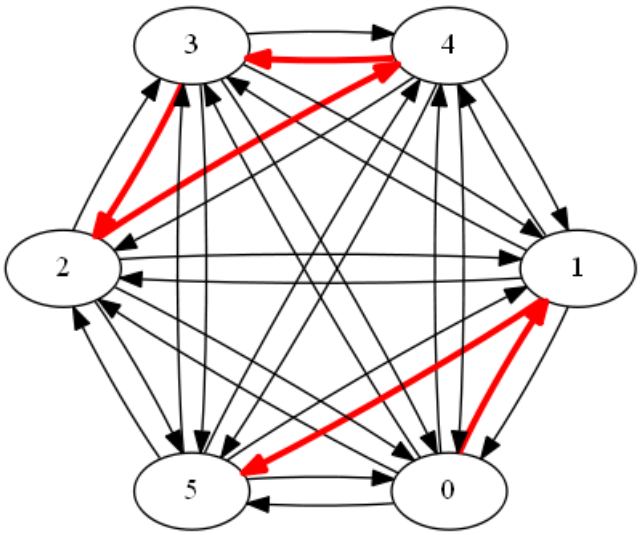


Figure 2.1: A PC-tuple; Total cost: -47 (Path P: -19, Cycle W_1: -28)

$$\sum_{j \in \Gamma^+(t)} x_{tj} = 0 \tag{2.7}$$

$$x_{ij} \in \{0, 1\} \ \forall (i, j) \in A \tag{2.8}$$

If there is no elementary s-t path in the network, then the set of solutions satisfying the above constraints is empty. Apart from relaxing cycle elimination constraints, three new constraints are added to the master relaxation. Constraint (2.5) makes sure that no node has more than single unit of outflow. Constraint (2.6) ensures that there is no inflow into the source node. Constraint (2.7) ensures that there is no outflow from the sink node. Note that this formulation is still a relaxation because every s-t path satisfies Constraints (2.5), (2.6) and (2.7).

Proposition 1 *Every feasible solution to the MRP corresponds to a PC-tuple.*

A solution feasible to the master relaxation is shown in Figure 2.1. The node set for the path P from the source node to the sink node is {0, 1, 5} and also a cycle W_1 can be found whose node set is {2, 3, 4}. This solution is a PC-tuple.

A PC-tuple containing cycles with positive weight can be feasible to the MRP. But it cannot be optimal to the MRP because the objective function value can then

be improved by removing the arcs of the positive weight cycle from the solution. However, a PC-tuple containing exclusively negative cycles can be optimal to the MRP. In such PC-tuples containing negative cycles, any arc $(i, j) \in A$ with $x_{ij} = 1$ and not on the directed path P, is on a negative cycle. For example, in Figure 2.1 $x_{24} = 1$, but the path in that PC-tuple is 0-1-5 and the arc (2,4) is not on it. This means that arc (2,4) is a part of a negative cycle. This enables a simpler approach to the separation problem. Just detecting an arc in the PC-tuple which is not on the $s-t$ path of that PC-tuple leads to the detection of a negative cycle. This separation approach is described in Algorithm 2. Note that the separation problem is solved after obtaining an integer feasible solution to the MRP. The steps of the algorithm are as follows:

1. Given a PC-tuple, the path from the source node to the sink node is traversed and all the nodes on the path are marked.

2. All the nodes are scanned in the increasing order of the node label. The first unmarked node i incident to an arc (i, j), such that $x_{ij} = 1$ is selected. By the foregoing discussion, the arc (i, j) is a part of a negative cycle W.

3. By traversing along the negative cycle W that contains the arc (i, j), all the internal nodes of that negative cycle are identified and marked.

Note that the cycle is identified from the set of unmarked nodes by scanning in the increasing order of the node label. Because of this, if there is more than one negative cycle in a PC-tuple, the cycle containing the node with the smallest index number is identified first.

2.2.1 Lazy constraints

Large number of cycle elimination constraints can be involved in Formulation 4 and only some of the constraints are violated during the runtime of the algorithm, while

Algorithm 2 Separation

Require: A graph $G = (N, A)$ containing a PC-tuple corresponding to $x \in \{0,1\}^m$, source node s, sink node t

Unmark all nodes in N, mark root node s
$i := s$; PATH := $\emptyset$;CYCLE := $\emptyset$
while $i \neq t$ **do**
    if node i is incident at an arc (i, j), such that $x_{ij} = 1$ **then**
        $PATH := PATH \cup \{i\}$, mark node i
        $i := j$
    end if
end while
for i=1 to n **do**
    if i is unmarked **then**
        if node i is incident to an arc (i, j), such that $x_{ij} = 1$ **then**
            $CYCLE := CYCLE \cup \{i\}$, mark node i, $k := i$ and $i := j$
            while $j \neq k$ **do**
                if node i is incident to an arc (i, j), such that $x_{ij} = 1$ **then**
                    $CYCLE := CYCLE \cup \{i\}$, mark node i
                    $i := j$
                end if
            end while
            Break for-loop
        end if
    end if
end for
Return CYCLE

the remaining constraints are not violated. These constraints need not be included in the formulation, but the violation of a constraint can be verified only during the running of the algorithm. The solver package we use has built-in methods for dealing with this scenario, known as *lazy constraints*.

Using this feature, a set of constraints are pooled together and a BB algorithm is executed. If one of the cycle elimination constraints is found to be violated in an intermediate step using the separation algorithm, then that particular constraint is enforced. This approach enables dealing with large number of constraints [19].

2.3 Decomposition and branch-and-cut approach

The structure of the proposed decomposition and branch-and-cut approach is discussed in this section. Branch-and-cut technique is one of the most widely used tools in integer programming to solve NP-hard problems. Our algorithm begins by solving the master relaxation of the original formulation which is discussed in Section 2.2. Each node of the BB tree can represent one of the following cases:

- An elementary $s-t$ path, in which case the BB node is pruned by feasibility and the incumbent solution is updated as required.
- A fractional solution, in which case we continue branching.
- An infeasible LP relaxation, in which case we prune that BB node by infeasibility.
- A PC-tuple feasible to the MRP, then a sequence of steps are followed as shown below:

    1. The separation problem is solved and the negative cycle is detected.
    2. The cycle elimination constraint is added to the current formulation as a lazy constraint and the LP relaxation is re-solved.

3. The re-solved LP may terminate yeilding an $s-t$ path (in which case the BB node is pruned by feasibility) or a fractional solution (in which case we continue branching) or a PC-tuple (in which case steps 1-3 are repeated).

This algorithm terminates when there are no nodes left in the BB tree to branch. At that point, the incumbent solution is the optimal elementary $s-t$ path.

CHAPTER 3

COMPUTATIONAL STUDY

This chapter presents the results of our computational experiments followed by discussions of specific noteworthy observations. The objective of the experiments in this chapter is to investigate the scalability and computational performance of the DBC approaches. This includes extracting the runtimes of the algorithms for solving various parts of the problem on different test-beds; comparing the computational results of the DBC algorithms with direct solution of Formulation 3 using a commercial MIP solver.

3.1 Approaches in implementing cutting planes

In the implementation of the DBC approach, the lazy constraints are added using the *callback function*. Two different approaches to implement cutting planes in the callback function of DBC algorithms are discussed in this section.

Single lazy constraint (DBC-SC): Consider a PC-tuple detected in the callback function. Assume this PC-tuple contains more than one negative cycle. A single negative cycle is detected from that PC-tuple and a cycle elimination constraint is added that eliminates the identified PC-tuple. This means that one lazy constraint is added for every PC-tuple detected in the DBC algorithm. Recall that the negative cycle containing the node with minimum node label is identified in this setting.

Multiple lazy constraints (DBC-MC): If a PC-tuple containing more than one negative cycle is detected in the callback function, then all the negative cycles present

in that PC-tuple are used to generate cuts. Cycle elimination constraints are added to eliminate this PC-tuple. This means that one lazy constraint is added for each negative cycle within every PC-tuple detected in the DBC algorithm. This approach is expected to strengthen the relaxation more than the DBC-SC approach that has enumerated the same number of BB nodes as more cutting planes are added. However, we cannot strictly guarantee this as the search tree is also different under these two settings and are not directly comparable.

3.2 Implementation details

The DBC algorithms and the MIP formulation were implemented in C++, and Gurobi™ optimizer 6.0.2 was used to solve all the instances. All experiments were conducted on a 64-bit computer equipped with a 2.20 GHz Intel® core™ i3-2328M CPU and 4.00 GB RAM. The operating system on the computer was Windows® 8.1. Data was collected and stored using Microsoft® Excel®. The results were analyzed and visualized using Microsoft® Excel® and MATLAB®. The runtimes were extracted in seconds. The maximum time limit for any approach to solve any instance was set to 3000 seconds.

3.3 Experiments

Computational experiments are executed on two different test-beds to compare the performances of DBC-SC, DBC-MC and direct solution of Formulation 3 using Gurobi. In both test-beds, test instances are directed networks with a source node, a sink node; and a list of integer arc weights with unrestricted sign. The first test-bed (Test-bed 1) is a set of 17 instances used by Ibrahim et al. [18] to study their exact formulation. The largest network in this test-bed consists of 200 nodes and 370 arcs. The second test-bed (Test-bed 2) consists of 14 dense networks (bidirected, complete graphs). In this test-bed, arc weights are randomly generated between a lower and upper bound.

Some networks consist of negative arc weights for all the arcs, while the remaining networks contain arcs with nonnegative arc weights. The largest network in the second test-bed consists of 120 nodes and 14280 arcs.

Performance profiles [20] are used for evaluating the performance of the three approaches. Let $t_{p,u}$ be the time taken by algorithm u to solve the network instance p. Let t_p^* denote the lowest time taken by any algorithm to solve the network instance p. The ratio $r_{p,u}$ is defined as

$$r_{p,u} = \frac{t_{p,u}}{t_p^*} \tag{3.1}$$

Let $\rho_u(\tau)$ be the probability that the ratio $r_{p,u}$ for an algorithm is within a factor τ of the best ratio value.

$$\rho_u(\tau) = \frac{no.\ of\ instances\ such\ that\ r_{p,u}}{total\ no.\ of\ instances} \leq \tau \tag{3.2}$$

For each approach, $\rho_u(\tau)$ is plotted for which the runtime is within a factor τ of the best runtime. The top curve of the plot is the approach that solves most of the instances in a time that is within a factor τ of the best runtime.

3.3.1 Results from Test-bed 1

The results from Test-bed 1 are shown in Table 3.1. From the results, all three approaches (DBC-SC, DBC-MC and direct MIP) are able to optimally solve all 17 instances. In Table 3.1, the fastest runtime for every network is highlighted in bold font. From the total runtime, it can be seen that both DBC approaches solve all the instances much faster than the direct MIP approach. Looking at the total runtimes of the DBC approaches, DBC-MC performs better than DBC-SC in 15 instances. DBC-SC performs better than DBC-MC in 1 instance and both the DBC approaches solve one instance in the same runtime.

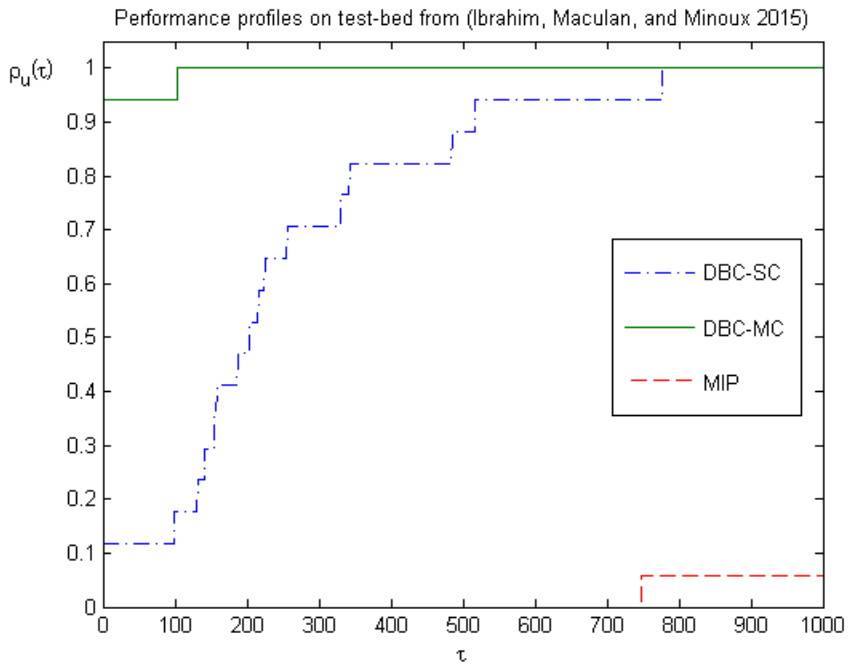


Figure 3.1: Performance profiles on $\tau[0, 1000]$ (Test-bed 1)

The number of callbacks (number of times the callback function is executed) show that the DBC-MC approach solves all the instances with fewer callbacks. This in turn reduces the total runtime of the DBC-MC approach on this test-bed. Note that the number of lazy constraints added in dav763 is much higher than other graphs of the same size. The reason for this behavior is discussed in Section 3.3.3.

The performance profiles of the three approaches on Test-bed 1 is shown in Figure 3.1. From this figure, the probability that DBC-MC is the best approach for a given instance is about 0.94. The probability that DBC-SC is the best approach for a given instance is about 0.12. By this measure DBC-MC is the most favorable algorithm for Test-bed 1.

3.3.2 Results from Test-bed 2

The results from Test-bed 2 are shown in Table 3.2. From the results, the DBC approaches are able to optimally solve all 14 instances. However, the direct MIP

Table 3.1: Comparison of performance on Test-bed 1

Name	$\|N\|$	$\|A\|$	**#Cuts**		**#Callbacks**		**Callback runtime**		**Total runtime**		
			SC	**MC**	**SC**	**MC**	**SC**	**MC**	**SC**	**MC**	**MIP**
g100e5	100	180	24	26	26	19	0.37	0.31	0.89	**0.67**	14.97
g100e6	100	180	18	20	20	13	0.29	0.24	0.74	**0.59**	13.81
g100e8	100	180	40	40	43	23	0.53	0.40	0.96	**0.81**	13.28
g100e9	100	180	42	40	44	27	0.54	0.44	1.01	**0.84**	13.67
g100e10	100	180	35	36	37	33	0.48	0.48	**0.90**	**0.90**	13.37
g100e12	100	180	11	11	14	9	0.19	0.15	0.57	**0.50**	13.3
g100e14	100	180	216	198	219	170	3.08	2.63	3.82	**3.48**	13.9
g100e15	100	180	14	15	16	10	0.25	0.19	0.74	**0.64**	14.78
g100e17	100	180	10	11	12	6	0.16	0.10	0.51	**0.42**	14.46
g200e2	200	342	69	69	70	29	0.86	0.63	1.56	**1.38**	97.61
g200e4	200	342	27	28	29	13	0.39	0.28	1.05	**0.91**	94.28
dav33	200	370	99	46	102	34	1.99	0.90	3.25	**1.83**	100.78
dav194	200	370	23	21	25	8	0.45	0.22	1.14	**0.85**	98.69
dav229	200	370	59	62	61	19	0.89	0.53	1.59	**1.30**	98.95
dav277	200	370	23	23	25	7	0.54	0.24	1.47	**0.97**	102.83
dav280	200	370	48	37	50	12	0.65	0.32	1.38	**0.93**	99.19
dav763	200	370	1975	1975	1979	1915	38.31	42.78	**61.26**	67.55	107.01

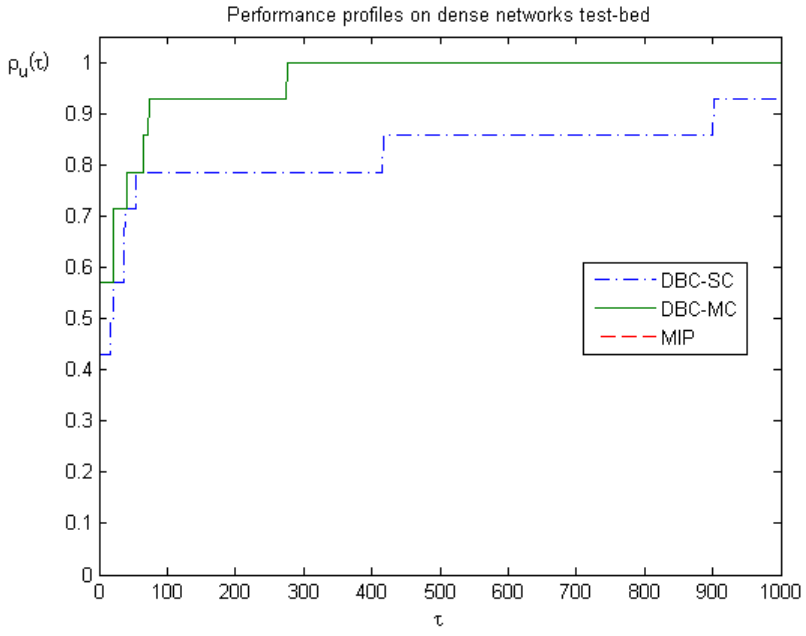


Figure 3.2: Performance profiles on $\tau[0, 1000]$ (Test-bed 2)

approach takes more than 3000 seconds to solve 7 instances. In Table 3.2, the fastest runtime for every network is highlighted in bold font. Like Test-bed 1, both the DBC approaches solve all the instances much faster than the direct MIP approach. Looking at the total runtimes of the three approaches, DBC-MC performs significantly better than DBC-SC in 8 instances. DBC-SC performs marginally better than DBC-MC in remaining 6 instances. Even though the most favorable approach is not as evident as it is in Test-bed 1, on average, DBC-MC appears to be the better approach among the two.

The performance profiles of the three approaches on Test-bed 2 is shown in Figure 3.2. From this figure, it is clear that the DBC approaches perform better than the direct MIP approach. The probability that the direct MIP approach is the best approach to solve any given instance in Test-bed 2 is 0, mainly because it takes more than 3000 seconds to solve 50% of the instances in Test-bed 2.

In order to better compare DBC-SC and DBC-MC, Figure 3.3 shows the performance profiles on $\tau[0, 100]$. From this figure, the probability that DBC-MC is the

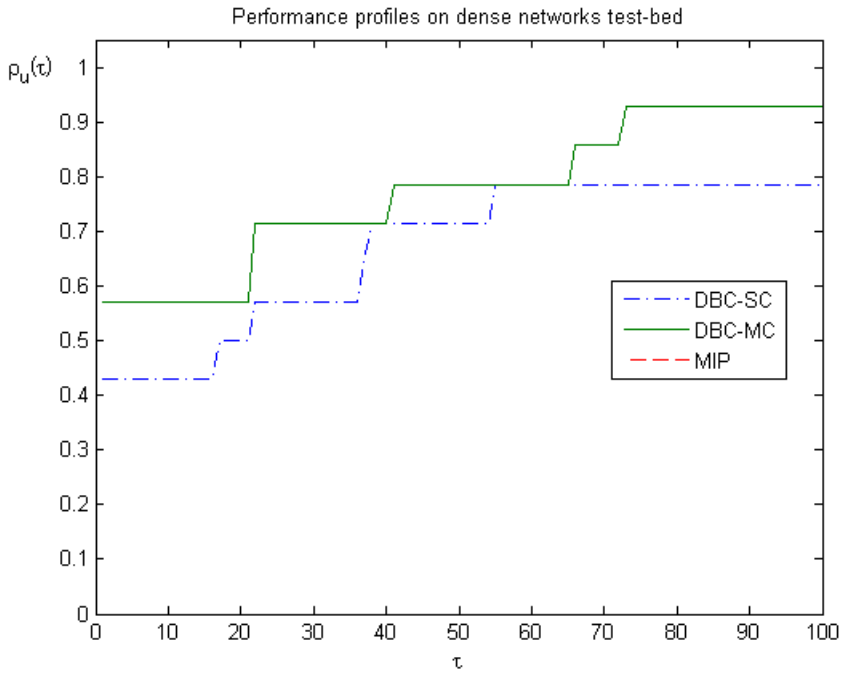


Figure 3.3: Performance profiles on $\tau[0, 100]$ (Test-bed 2)

best approach for a given instance is about 0.58. The probability that DBC-SC is the best approach for a given instance is about 0.42. By this measure DBC-MC is the most favorable algorithm for Test-bed 2.

3.3.3 Number of lazy constraints in dense networks

Intuitively, we expect a lot of negative cycles to be detected in dense instances during the execution of the DBC algorithms, but on the contrary the number of lazy constraints added are much smaller. This phenomenon is explained by the following observations. Consider the dense network fn50 from Test-bed 2, with 50 nodes and 2450 arcs. The lower bound on arc cost is -32 and the upper bound is -1. After solving it in DBC-MC approach, a shortest path is obtained (best OF$= -1536$), after adding 3 lazy constraints. This means that three PC-tuples are detected before obtaining the answer. After closer inspection on these PC-tuples following observations are made:

Table 3.2: Comparison of performance on Test-bed 2

Name	$\|N\|$	$\|A\|$	**#Cuts**		**#Callbacks**		**Callback runtime**		**Total runtime**		
			SC	**MC**	**SC**	**MC**	**SC**	**MC**	**SC**	**MC**	**MIP**
sp20	20	380	3	3	6	6	0.04	0.06	0.49	**0.48**	4.10
fn20	20	380	23	32	27	28	0.27	0.31	**1.02**	1.30	8.09
sp50	50	2450	38	14	43	8	0.72	0.16	5.13	**2.70**	258.16
fn50	50	2450	3	3	5	5	0.06	0.09	**2.28**	2.37	167.77
sp60	60	3540	16	12	18	7	0.33	0.15	4.67	**3.30**	483.44
fn60	60	3540	69	12	74	4	1.40	0.13	8.06	**3.41**	375.53
sp80	80	6320	0	0	2	2	0.02	0.03	**6.16**	6.60	>3000
fn80	80	6320	19	22	21	14	0.54	0.41	7.74	**7.35**	2401.42
sp90	90	8010	3	6	6	4	0.11	0.13	8.01	**7.89**	>3000
fn90	90	8010	16	37	18	24	0.64	1.08	**10.12**	10.33	>3000
sp100	100	9900	31	54	34	30	1.27	1.55	**11.57**	12.32	>3000
fn100	100	9900	19	19	21	12	0.63	0.57	10.68	**10.31**	>3000
sp120	120	14280	26	42	29	23	1.46	1.71	**18.08**	18.45	>3000
fn120	120	14280	17	23	20	16	0.86	0.82	17.14	**16.54**	>3000

1. The path in all of these PC-tuples contains the majority of nodes (always more than 40).

2. Since a large number of nodes are already in the path of PC-tuple, only limited number of nodes are available to form negative cycles. So, if the path in PC-tuple contains a large number of nodes, then the number of negative cyle detected decreases; furthermore, the opposite is also true.

3. The number of nodes in the path of all the PC-tuples stays large (around 40) and this enables detection of very few negative cycles throughout the algorithm, before determining the optimal solution. Hence, very few lazy constraints are added.

In order to validate these observations, the cost of the arc (s, t) in graph fn50 was modified from -29 to -37 (note that -32 was the lower bound on the arc weights). This graph was solved by the DBC-MC algorithm after adding 2432 lazy constraints, a significant increase from 3 lazy constraints before modification. Because of the lowered arc weight, arc (s, t) stayed as the path in many PC-tuples. Since the path in all those PC-tuples had only two nodes, rest of the nodes joined to form negative cycles. The path remained the same for many PC-tuples and that resulted in the identification of 2432 negative cycles, before terminating with the same objective function value as before (-1536). The network dav763 from Test-bed 1 also has a large number of lazy constraints, due to the behavior of the algorithm explained above. The path in many PC-tuples remained the same before finding a solution with a better objective function value.

3.4 Pathological instance

Consider a bidirected, complete graph $G = (N, A)$ in which every arc $(i, j) \in A$ carries negative arc weights between -50 and -1. After finding the optimal $s - t$

path with the DBC-MC algorithm, let the optimal objective function value be -1500. Note that there are a lot of feasible paths from source node s to sink node t (dense graph), furthermore the path containing only arc (s,t) is among the feasible solutions. However, the path containing only arc (s,t) is not the optimal solution, because if it is the optimal solution, then the lowest value objective function can have is -50 (lower bound on arc weight).

Consider a scenario where the arc weight of arc $(s,t) \in A$ alone is changed to -1501 and solved again using the DBC approach. Now, the path containing only arc (s,t) is the optimal solution, but consider the iterations in the DBC algorithms before arriving to that solution. In every PC-tuple of the BB tree, the path will be arc (s,t) . Since, all the arc-weights in G are negative, all other arcs will join to create node-disjoint negative cycles. In this case, the DBC-MC approach will detect a large number of negative cycles and will add cycle elimination constraints for them before finding the optimal solution. This is because, no matter how many cycle elimination constraints are added, it is impossible to find a path shorter than -1501; this means arc (s,t) will continue staying as the path in all the PC-tuples while many negative cycles are identified and exhausted. This is a pathological instance for the DBC algorithms. The direct MIP approach by [16] implemented in Gurobi, solves this instance faster than the DBC approaches.

CHAPTER 4

CONCLUSION AND FUTURE RESEARCH

This thesis explores new approaches to solve ESPPNC. The literature contains exact algorithms to solve ESPPNC. However, a decomposition approach to solve ESPPNC has not been explored. This work is an attempt to solve ESPPNC using a decomposition and branch-and-cut technique. The major contributions of this thesis are exploring the DBC approach and evaluating its performance via computational studies.

Chapter 2 explains the master relaxation of the original formulation. The master relaxation enables an effective separation approach based on identification of negative cycles in PC-tuples. This master relaxation is strengthened by adding cycle elimination constraints in a "lazy" manner in branch-and-bound framework.

Based on how the lazy constraints are implemented, two versions of DBC approach are proposed and explored as a part of the computational studies. These approaches are compared with directly solving the MIP formulation for ESPPNC proposed by Ibrahim et al. [16] using Gurobi on two different test-beds. The results suggest DBC-MC approach to be comparatively superior to solve ESPPNC, except for pathological instances in the test-bed where the direct MIP approach exhibits superior performance.

In light of the experimental results from this thesis, numerical experiments should be redesigned to vary edge density in generating the test-bed. Exploring the performance of the DBC algorithms on an expanded test-bed, including more pathological instances is a potential area for further research. Another area for future research lies

in DBC approaches to solve the single-source shortest path problem with negative cycles. However, DBC approaches to solve this problem should be very different from the DBC algorithms for ESPPNC, because the single-source counterpart cannot be a direct extension of our DBC-SC/-MC approaches; the master relaxation properties do not continue to apply in this setting, which was critical for the DBC algorithms introduced in this thesis.

REFERENCES

[1] M. S. Ibrahim. *Etude de formulations et inégalités valides pour le problème du plus court chemin dans un graphe avec des circuits absorbants*. PhD thesis, Pierre and Marie Curie University, 4 Place Jussieu, 75005 Paris, France, 2007.

[2] R. Sedgewick and K. Wayne. Computer science 226: Data structures and algorithms. *Princeton University, http://www. cs. princeton. edu/courses/archive/spr02/cs226/assignments/assign. html*, 2002.

[3] M. R. Garey and D. S. Johnson. Computers and intractability: a guide to the theory of np-completeness. 1979. *San Francisco, LA: Freeman*, 1979.

[4] A. Björklund and T. Husfeldt. Finding a path of superlogarithmic length. *SIAM Journal on Computing*, 32(6):1395–1402, 2003.

[5] R. Uehara and Y. Uno. Efficient algorithms for the longest path problem. In *Algorithms and computation*, pages 871–883. Springer, 2005.

[6] B. Petersen and D. Pisinger. *Shortest paths and vehicle routing*. PhD thesis, Technical University of Denmark, Anker Engelunds Vej 1 Bygning 101A, 2800 Kgs. Lyngby, Denmark, 2011.

[7] L.-M. Rousseau, M. Gendreau, G. Pesant, and F. Focacci. Solving vrptws with constraint programming based column generation. *Annals of Operations Research*, 130(1-4):199–216, 2004.

[8] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Networks flows*. Prentice Hall, Englewood Cliffs, New Jersey, 1993.

[9] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[10] R. Bellman. On a routing problem. *Quarterly of Applied Mathematics*, 16:87–90, 1958.

[11] L. R. Ford and D. R. Fulkerson. *Flows in networks*. Princeton University Press, 1962.

[12] E. F. Moore. *The shortest path through a maze*. Bell Telephone System., 1959.

[13] M. Drexl. A note on the separation of subtour elimination constraints in elementary shortest path problems. *European Journal of Operational Research*, 229(3): 595–598, 2013.

[14] L. A. Wolsey. *Integer programming*. Wiley New York, 1998.

[15] M. Fischetti and P. Toth. A polyhedral approach to the asymmetric traveling salesman problem. *Management Science*, 43(11):1520–1536, 1997.

[16] M. S. Ibrahim, N. Maculan, and M. Minoux. A strong flow-based formulation for the shortest path problem in digraphs with negative cycles. *International Transactions in Operational Research*, 16(3):361–369, 2009.

[17] N. Maculan, G. Plateau, and A. Lisser. Integer linear models with a polynomial number of variables and constraints for some classical combinatorial optimization problems. *Pesquisa Operacional*, 23(1):161–168, 2003.

[18] M. S. Ibrahim, N. Maculan, and M. Minoux. Valid inequalities and lifting procedures for the shortest path problem in digraphs with negative cycles. *Optimization Letters*, 9(2):345–357, 2015.

[19] Gurobi Optimization, Inc, 2014. Gurobi optimizer reference manual, version 6.0, copyright © 2014. `http://www.gurobi.com/documentation/6.0/refman/`.

[20] E. D. Dolan and J. J. Moré. Benchmarking optimization software with performance profiles. *Mathematical programming*, 91(2):201–213, 2002.

[21] J. M. Harris, J. L. Hirst, and M. J. Mossinghoff. *Combinatorics and graph theory*, volume 2. Springer, 2008.

[22] R. W. Floyd. Algorithm 97: Shortest path. *Commun. ACM*, 5(6):345–, June 1962. ISSN 0001-0782. doi: 10.1145/367766.368168. URL `http://doi.acm.org/10.1145/367766.368168`.

[23] J. F. Benders. Partitioning procedures for solving mixed-variables programming problems. *Numerische mathematik*, 4(1):238–252, 1962.

[24] M. Drexl and S. Irnich. Solving elementary shortest-path problems as mixed-integer programs. *OR spectrum*, 36(2):281–296, 2014.

[25] M. Haouari, N. Maculan, and M. Mrad. Enhanced compact models for the connected subgraph problem and for the shortest path problem in digraphs with negative cycles. *Computers & Operations Research*, 40(10):2485–2492, 2013.

APPENDIX A

NETWORK DIAGRAMS

This appendix contains diagrams of some network instances from the test-beds used for testing the performance of the DBC algorithms.

A.1 Test-bed 1

Two networks from Test-bed 1 is shown in this section. With 100 nodes and 180 arcs, g100e5, g100e6, g100e8, g100e9, g100e10, g100e12, g100e14, g100e15 and g100e17 are the smallest networks in Test-bed 1. Figure A.1 shows the network g100e5 with the source node 62 and the sink node 8.

The largest networks in Test-bed 1 are: dav33, dav194, dav229, dav277, dav280 and dav763. These networks contain 200 nodes and 370 arcs. Figure A.2 shows the network dav763 with the source node 48 and the sink node 12.

A.2 Test-bed 2

Networks sp20 and fn20 are the smallest networks from Test-bed 2. These networks contain 20 nodes and 380 arcs. Figure A.3 shows the network fn20 with the source node 1 and the sink node 20.

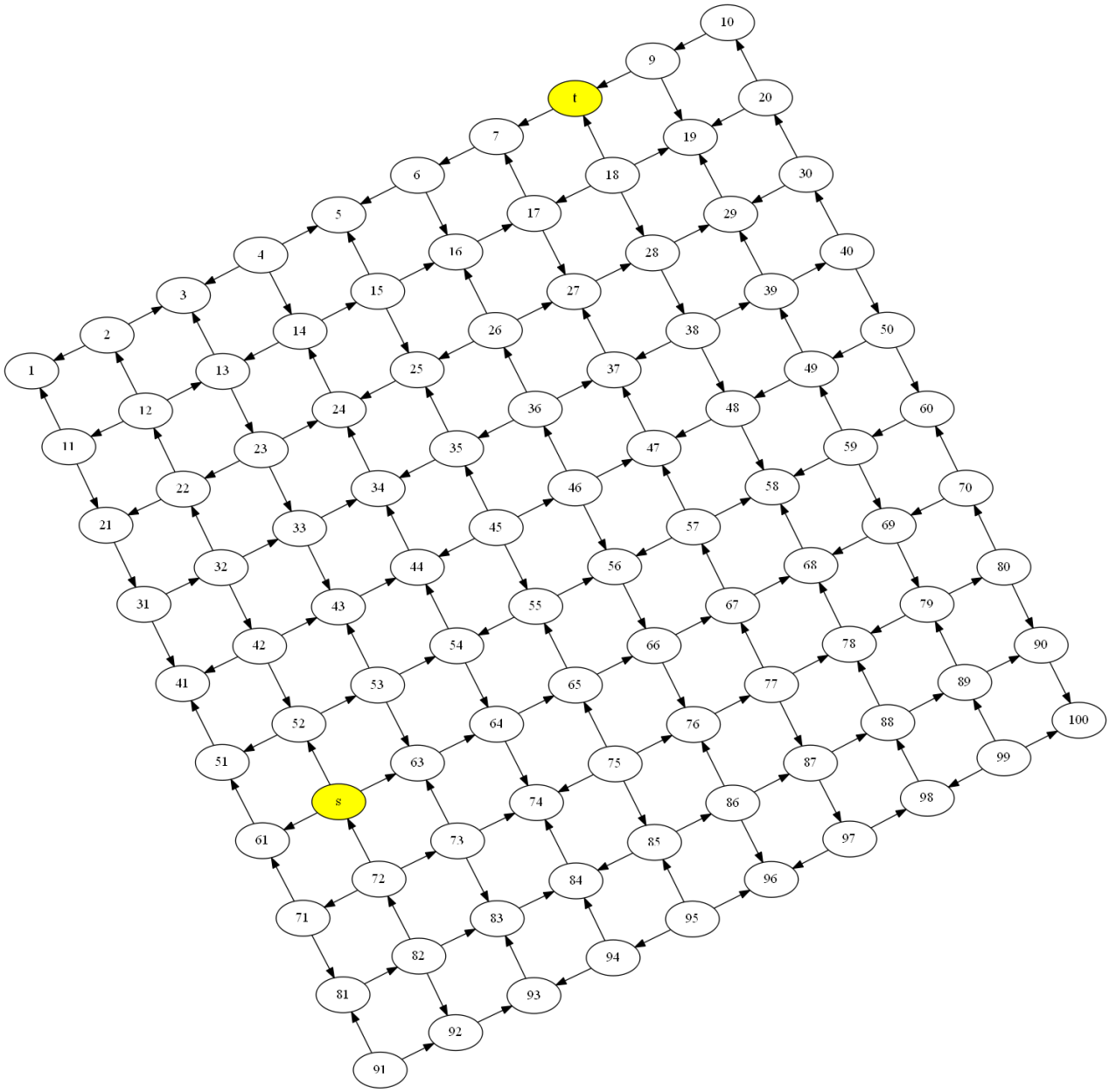


Figure A.1: Network g100e5 from Test-bed 1

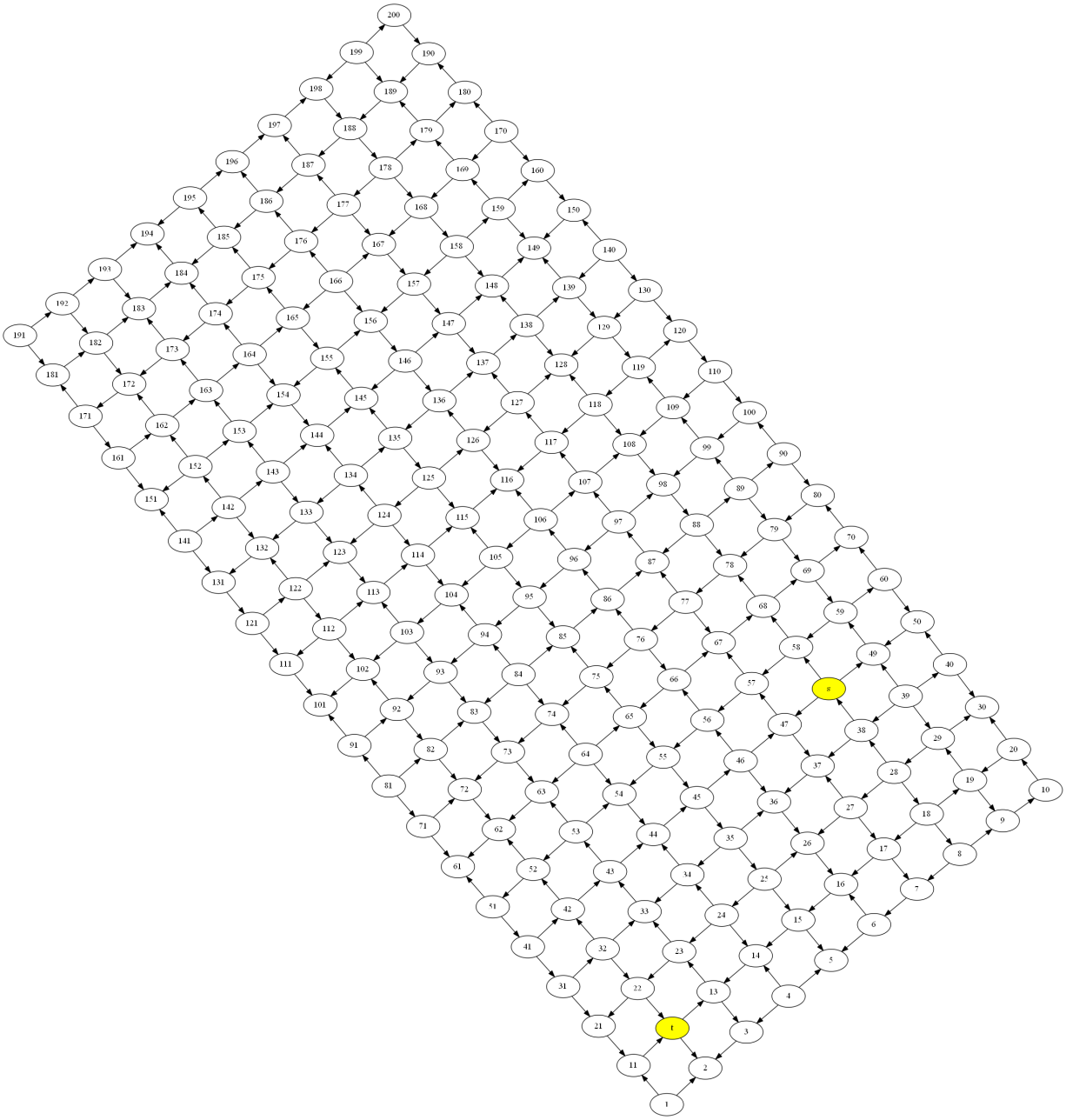


Figure A.2: Network dav763 from Test-bed 1

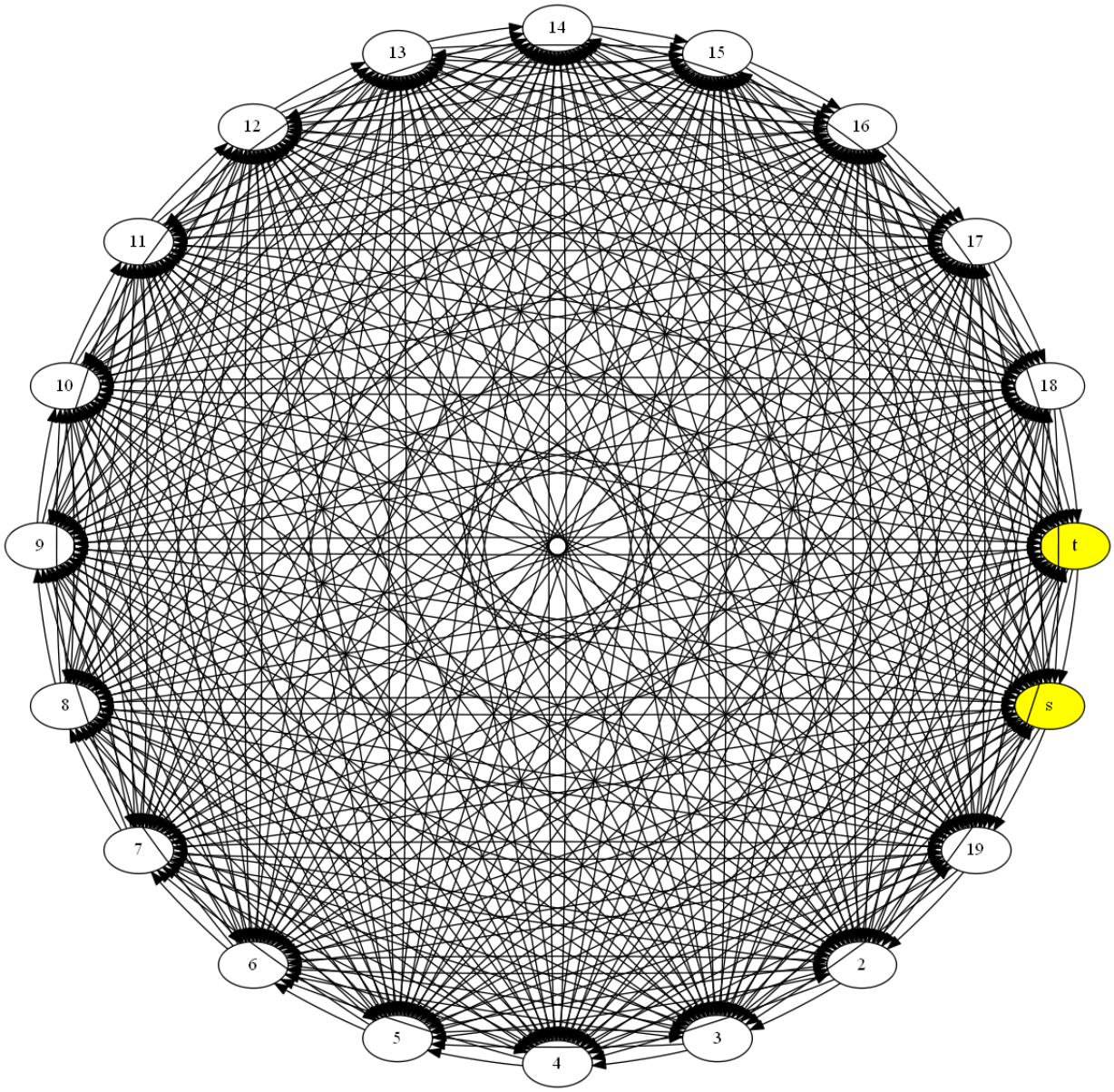


Figure A.3: Network fn20 from Test-bed 2

VITA

Devaraja Vignesh Radha Krishnan

Candidate for the Degree of

Master of Science

Thesis: DECOMPOSITION ALGORITHMS FOR THE ELEMENTARY SHORTEST PATH PROBLEM IN NETWORKS CONTAINING NEGATIVE CYCLES

Major Field: Industrial Engineering and Management

Biographical:

Personal Data: Born in Dindigul, Tamil Nadu, India on August 28, 1990.

Education:

Received the B.E. degree from Anna University, Chennai, Tamil Nadu, India, 2012, in Aeronautical Engineering

Completed the requirements for the M.S. degree with a major in Industrial Engineering and Management at Oklahoma State University in December, 2015.

Experience:

Graduate Teaching Assistant - School of Industrial Engineering and Management, Oklahoma State University (January 2014 - December 2015).

Teaching Assistant -SBM Engineering College, Dindigul, Tamil Nadu, India (August 2012 - May 2013)