

ABSTRACT

Title of dissertation: Mathematical Programming Models
 for Influence Maximization
 on Social Networks

Rui Zhang, Doctor of Philosophy, 2016

Dissertation directed by: Professor Subramanian Raghavan
 The Robert H. Smith School of Business
 and Institute for Systems Research

In this dissertation, we apply mathematical programming techniques (i.e., integer programming and polyhedral combinatorics) to develop exact approaches for influence maximization on social networks. We study four combinatorial optimization problems that deal with maximizing influence at minimum cost over a social network. To our knowledge, all previous work to date involving influence maximization problems has focused on heuristics and approximation.

We start with the following viral marketing problem that has attracted a significant amount of interest from the computer science literature. Given a social network, find a target set of customers to seed with a product. Then, a cascade will be caused by these initial adopters and other people start to adopt this product due to the influence they receive from earlier adopters. The idea is to find the minimum cost that results in the entire network adopting the product.

We first study a problem called the Weighted Target Set Selection (WTSS) Problem. In the WTSS problem, the diffusion can take place over as many time periods as

needed and a free product is given out to the individuals in the target set. Restricting the number of time periods that the diffusion takes place over to be one, we obtain a problem called the Positive Influence Dominating Set (PIDS) problem. Next, incorporating partial incentives, we consider a problem called the Least Cost Influence Problem (LCIP). The fourth problem studied is the One Time Period Least Cost Influence Problem (1TPLCIP) which is identical to the LCIP except that we restrict the number of time periods that the diffusion takes place over to be one.

We apply a common research paradigm to each of these four problems. First, we work on special graphs: trees and cycles. Based on the insights we obtain from special graphs, we develop efficient methods for general graphs. On trees, first, we propose a polynomial time algorithm. More importantly, we present a tight and compact extended formulation. We also project the extended formulation onto the space of the natural variables that gives the polytope on trees. Next, building upon the result for trees---we derive the polytope on cycles for the WTSS problem; as well as a polynomial time algorithm on cycles.

This leads to our contribution on general graphs. For the WTSS problem and the LCIP, using the observation that the influence propagation network must be a directed acyclic graph (DAG), the strong formulation for trees can be embedded into a formulation on general graphs. We use this to design and implement a branch-and-cut approach for the WTSS problem and the LCIP. In our computational study, we are able to obtain high quality solutions for random graph instances with up to 10,000 nodes and 20,000 edges (40,000 arcs) within a reasonable amount of time.

Mathematical Programming Models for Influence Maximization
on Social Networks

by

Rui Zhang

Dissertation submitted to the Faculty of the Graduate School of
the University of Maryland, College Park in partial fulfillment
of the requirements for the degree of
Doctor of Philosophy
2016

Advisory Committee:

Professor Subramanian Raghavan, Chair/Advisor

Professor Michael Ball

Professor Bruce Golden

Professor MohammadTaghi HajiAghayi (Dean's Representative)

Professor William Rand

© Copyright by
Rui Zhang
2016

Dedication

To my wife, Dr. Yang Wang (王瑒)

To my mother, Xiujin Chen (陈秀金)

To my father, Yanding Zhang (张延丁)

For all the sacrifices that they have made and all the love that they have given to ensure that I could focus on what I want to achieve, I am forever in their debt.

Acknowledgments

First and foremost, my deepest gratitude is to my advisor, Professor Subramanian Raghavan, for inspiring and motivating me to work on this thesis. He has always been there and provided continuous advice and support to me on research and life in general throughout the course of my PhD study. I would like to thank him for his patience and continuous encouragement. I have benefited from his enthusiasm and immense knowledge. I hope one day I will become as good an advisor to my students as Professor Raghavan has been to me.

I would also like to express my very sincere gratitude to Professor Bruce Golden who has been another source of advice whenever I need it. He has offered me opportunities to work on interesting research topics. Moreover, I would like to thank my other committee members, Professor Michel Ball, Professor William Rand and Professor MohammadTaghi HajiAghayi for serving on my thesis committee, coordinating their schedules and giving me input and feedback. It has been a great pleasure to work with and learn from these extraordinary individuals.

The PhD program in the Smith School of Business has been a wonderful place for me. I would like to thank Justina Blanco for her excellent job in taking care of all the administrative details. She plays a huge role in keeping the program running smoothly. I would also like to acknowledge and thank Weiming Zhu (朱未名) and Wenfeng Wang (王文峰) for their friendship. We are comrades in arms.

Most importantly, none of this would have been possible without the love and patience of my family. My mother, Xiujin Chen (陈秀金), and my father, Yanding Zhang

(张延丁), have been a constant source of love, concern, support and strength all these years – both spiritually and materially. Without them, I would never have enjoyed so many opportunities. I really appreciate what they have done for me.

I am very grateful to my wife and best friend, Yang Wang (王瑒). She has always stood by me through all the ups and downs. I am so fortunate to have her unflagging love, encouragement and support. She has made me become a better person. Words cannot express the gratitude I owe her.

I would also like to thank my daughter, Olivia Yuening Zhang (张玥凝). Being a father has been my greatest source of achievement, pride and inspiration. I believe at least one chapter of this thesis was done during those sleepless nights when she was sleeping in my arms.

This thesis is the starting point of my journey.

Table of Contents

List of Tables	vii
List of Figures	viii
1 Introduction	1
2 Weighted Target Set Selection on Social Networks	8
2.1 Introduction	8
2.2 Algorithm for the WTSS Problem on Trees	11
2.3 A Tight and Compact Extended Formulation on Trees	17
2.3.1 Polytope of the WTSS problem on Trees	28
2.4 The WTSS Problem on Cycles	37
2.5 A Branch-and-Cut Approach for General Networks	41
2.6 Computational Experience	49
2.7 Conclusions	59
3 Generalizations of the Dominating Set Problem on Social Networks	62
3.1 Introduction	62
3.1.1 Problem Definition	64
3.1.2 Related Literature	64
3.1.3 Our Contributions	67
3.2 The TPIDS Problem	69
3.2.1 Algorithm for the TPIDS Problem on Trees	69
3.2.2 Good Formulations for the TPIDS Problem on Trees	76
3.3 The PIDS Problem	81
3.3.1 Algorithm for the PIDS Problem on Trees	81
3.3.2 A Tight and Compact Extended Formulation on Trees	91
3.3.3 Projection of the Dummy Node Formulation	106
3.3.4 Polyhedral Study of the PIDS Problem on General Graphs	115
3.3.5 Computational Experiments	125
3.4 Conclusions	129

4	Tailored Incentives and Least Cost Influence Maximization on Social Networks	131
4.1	Introduction	131
4.1.1	Problem Definition	132
4.1.2	Related Literature	134
4.1.3	Our Contributions	135
4.2	Problem Complexity	137
4.2.1	Unequal Influence Factors	142
4.3	LCIP on Trees	144
4.3.1	Greedy Algorithm	144
4.3.2	Dynamic Programming Algorithm.	148
4.3.3	Totally Unimodular Formulation	155
4.3.4	A Tight and Compact Extended Formulation	159
4.3.5	Polytope of the LCIP on Trees	164
4.4	From Trees to General Graphs: A Branch-and-Cut Approach	170
4.5	Computational Experiments	182
4.6	Conclusions	196
5	The One Time Period Least Cost Influence Problem	198
5.1	Introduction	198
5.1.1	Problem Definition	198
5.1.2	Our Contributions	200
5.2	The 1TPLCIP on Trees	201
5.2.1	Dynamic Programming Algorithm.	201
5.2.2	A Tight and Extended Formulation	208
5.2.3	Polytope of the 1TPLCIP on Trees	215
6	Conclusions and Future Work	223
6.1	Future Work	224
6.1.1	Proportion Requirements	224
6.1.2	Marked Targets	225
6.1.3	Latency Constraints	225
6.1.4	Combinatorial Games	227

List of Tables

2.1	Comparisons of LP relaxations of Ack2 and BIP2.2 for the WTSS problem, a Greedy Heuristic, and Optimal Integer Solutions for 1000-Node Instances.	49
2.2	Comparison of Brand-and-Cut Settings for the WTSS problem on 10000-Node Instances.	52
2.3	Analyzing the Effect of Graph Density on the Branch-and-Cut Procedure for the WTSS problem.	55
2.4	Heuristic for the WTSS problem Fixed Budget on 200-Node Instances. . .	55
2.5	Optimality Gaps for Greedy and Shakarian Heuristic on the WTSS problem for 100% Adoption on 200-Node Instances.	58
3.1	LP Relaxations of BIP_{saxena} and BIP_{dummy} with 200-node instances.	126
3.2	running time of four formulations in seconds with 200-node instances . .	127
3.3	Linear Programming Relaxation Comparison. (Rel.Imp. = $\frac{z_{dummy} - z_{saxena}}{z_{saxena}}$) . .	128
4.1	Summary of complexity results.	137
4.2	LP Relaxation of MIP4.7 and BIP4.6 Formulation for the LCIP with 1000-Node Instances.	184
4.3	Comparison of Brand-and-Cut Settings for the LCIP on 10000-Node Instances.	187
4.4	Analyzing the effect of graph density on the branch-and-cut procedure for the LCIP.	189
4.5	Heuristic for the LCIP Fixed Budget on 200-Node Instances.	189
4.6	Optimality Gaps for Heuristics for 100% Adoption for the LCIP on 200-Node Instances.	192
4.7	Advantage of Partial Incentives of the LCIP with 200-Node Instances . .	194

List of Figures

1.1	Connections between Problems.	2
2.1	(a) A WTSS instance (b) The final star (c) The solution of Algorithm 1 . . .	17
2.2	(a) A WTSS instance (b) A fractional optimal solution	18
2.3	(a) G_t (b) A valid solution to BIP2.1	21
2.4	Illustration of Notation in Theorem 2.2	31
2.5	Illustration of the Cases for the Algorithm on Cycles	40
2.6	An example of cycle	41
2.7	Transforming a 0-1 knapsack problem to the WTSS problem with unequal influence on stars	60
3.1	A TPIDS problem instance.	71
3.2	TPIDS (a) After compress star 5, 6, 7, 8, 9 and 10. (b) The last star.	72
3.3	The solution obtained by our DP algorithm for the TPIDS.	75
3.4	A PIDS problem instance.	82
3.5	(a) After compress star 1, 2 and 3. (b) The last star.	85
3.6	The solution obtained by our DP algorithm.	89
3.7	(a) A PIDS instance (b) A fractional solution returned by LP_{saxena} and $LP_{\text{baïou}}$	92
3.8	(a) An original edge (b) A transformed edge (c) Transformed graph of Figure 3.7.	93
3.9	Node j is a (a) free child (b) expensive child (c) core child of node i for u variables.	99
3.10	A PIDS problem instance for Theorem 3.4. and Dual variable values.	105
3.11	Theorem 3.5 necessity proof: $T = \emptyset$ and $ S^u + S^v > 1$ example.	109
3.12	Theorem 3.5 necessity proof: $ T > 1$ example.	110
3.13	Theorem 3.5 necessity proof: $ T = 1$ and $S_j \neq \emptyset$ example.	110
3.14	Theorem 3.5 necessity proof: $ T = 1$ and $S_2 \neq \emptyset$ example.	111
3.15	Theorem 3.5 necessity proof: $ T = 1$ and $S^+ \neq \emptyset$ example.	111
3.16	Illustration for notations in facet-defining proof of inequality (3.84).	117
4.1	Illustration of the reduction from independent set	138
4.2	An Illustration of the reduction from target set selection	141

4.3	Transforming a 0-1 knapsack problem to the LCIP problem with unequal influence on stars	143
4.4	Greedy Algorithm for LCIP on a Tree	145
4.5	The DP Algorithm for LCIP on a Tree	153
4.6	Categorization of incoming influence when and $g_i \geq 2$	156
4.7	Categorization of incoming influence when $g_i = 1$	157
4.8	Illustration for Theorem 4.7	161
4.9	A pathological example of a cycle in influence propagation	171
4.10	Influence Direction Symmetry	175
4.11	Influence Allocation Symmetry	177
4.12	Illustration of Conditions 3 and 4	179
5.1	(a) A 1TPLCIP instance (b) A fractional optimal solution	207
6.1	Time Period Example.	226
6.2	A WTSS problem instance and its optimal solution	228

Chapter 1: Introduction

Recently, the dynamic processes for the diffusion process of influence has attracted significant interest from algorithmic researchers. Over the past ten years, the following viral marketing problem has attracted a significant amount of interest. Assume we want to promote a new product over a given social network and wish this product will be adopted by most people in this network. We can initialize the diffusion process by "targeting" some influential people. Then, a cascade will be caused by these initial adopters and other people start to adopt this product due to the influence they receive from earlier adopters. But how should we select these influential people who are targeted initially? Domingos and Richardson [2001] studied the problem in a probabilistic setting, and provided heuristic solutions. Kempe et al. [2003] were the first ones to model this problem as an optimization problem by using the threshold model proposed by Granovetter [1973], showed it is NP-hard to find the optimal initial set, and developed approximation algorithms for the problem. Subsequently, several different variants of this problem have been studied. However, to the best of our knowledge, all previous work are from approximation algorithm perspective. Chen et al. [2013]'s recent monograph nicely summarizes most of the relevant work in the area. We want to apply mathematical programming techniques (i.e., integer programming and polyhedral combinatorics) to develop exact approaches

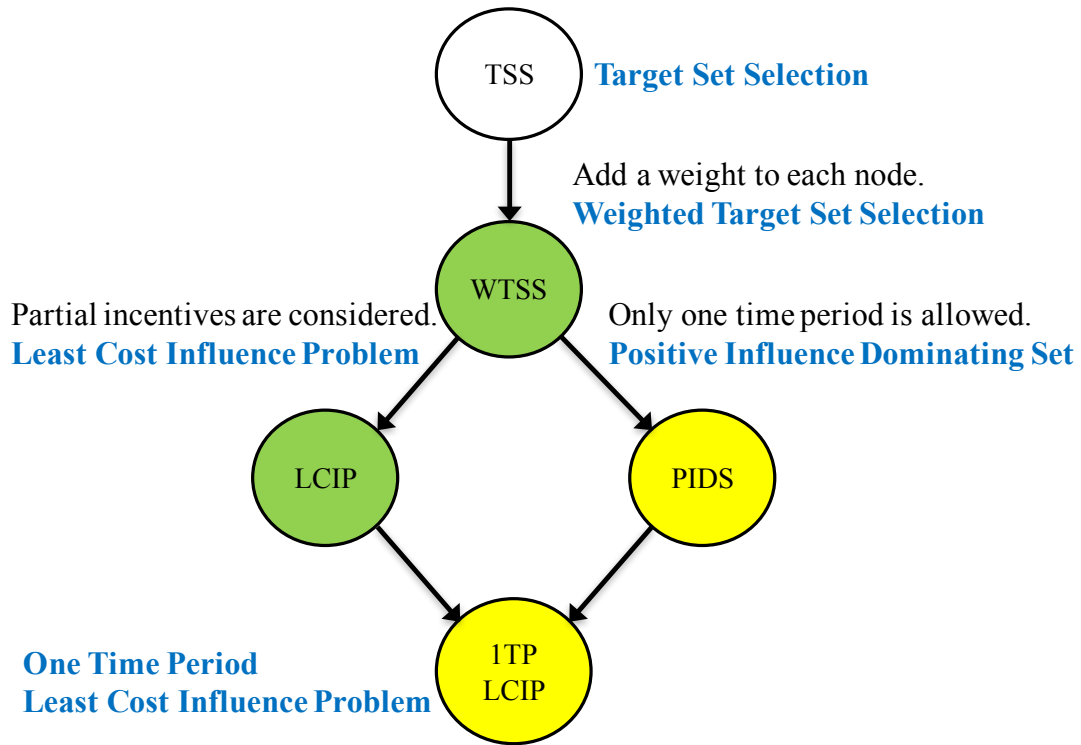


Figure 1.1: Connections between Problems.

for solving these problems optimally. Our research is motivated by the desire to develop mathematical programming approaches for these problems and a better understanding of the underlying polytopes. The purpose is to provide meaningful benchmarks by solving non-trivial instances. Consequently, the results can be used to evaluate the heuristics and approximation algorithms proposed for these problems. In the event the social networks analyzed are of smaller size our approach may be used to solve these smaller problems to optimality.

In this thesis, we study four combinatorial optimization problems for influence maximization on social networks. The connections between them are shown in Figure 1.1. First, we consider the Weighted Target Set Selection (WTSS) problem. Given a social network, finding a target set of customers of the smallest possible size that could lead the

whole network to be influenced through the influence diffusion process is referred to as the Target Set Selection (TSS) problem [Chen, 2009]. In the TSS problem, given a connected undirected graph, each node is associated with a threshold, denoted by g_i , which takes values between 1 and the degree number of the node, denoted by $deg(i)$. All nodes are set as in an "inactive" state initially. A selected subset of nodes, the target set, is switched to the "active" state. Next, the states of nodes are updated step by step with respect to the following rule: an inactive node i becomes active if at least g_i of its neighbors are active in the previous step. The goal is to find the minimum size target set while ensuring that all nodes are active by the end of this process. In the WTSS problem, for each node $i \in V$, there is a weight, denoted by b_i , which models the fact that different nodes require differing levels of effort to become initial active nodes.

The WTSS problem is closely related to two other problems that deal with influence maximization on social networks. In the WTSS problem the diffusion process continues until complete. However, if we restrict the number of steps/time periods that the diffusion takes place over to be one, we obtain a weighted version of a problem called the Positive Influence Dominating Set (PIDS) problem (proposed by Wang et al. [2009]). In the PIDS problem, either a node is selected at a cost of b_i , or it requires g_i of its neighbors to be selected.

The WTSS problem also connects to another problem called the Least Cost Influence Problem (LCIP) proposed by Günneç [2012]. The problems are similar, except that instead of paying the full amount b_i to a node that is selected in the target set we are allowed to provide partial incentives/payments to incentivize a node to adopt a product. In the LCIP, each node has a threshold b_i and a neighbor j of node i exerts an influence of

d_{ij} on node i if j adopts the product before node i . In other words, the sum of the payment p_i given to a node i and the incoming influence from neighbors who have already adopted this product should be greater than or equal to its threshold b_i . (For the WTSS, $d_{ij} = d_i$ for all neighbors j of node i and thus $g_i = \lceil \frac{b_i}{d_i} \rceil$.)

The fourth problem is the One Time Period Least Cost Influence Problem (1TPL-CIP) which combines the PIDS problem and the LCIP together. With assumption that all neighbors of node i exert an same influence of d_{ij} (i.e., $d_i = d_{ij}$) on node i if they adopt the product before node i , from the perspective of the the PIDS problem, we allows partial incentives instead of only providing the whole product. From that of LCIP, we restrict the number of steps/time periods to be one.

In Chapter 2, we study the WTSS problem. Motivated by the desire to build strong formulations for the WTSS problem we first study the WTSS problem on trees. Our contributions in this regard are three-fold. First, we propose a polynomial algorithm for the WTSS problem on trees. The algorithm uses a dynamic programming approach. More importantly, we present a tight and compact extended formulation for the WTSS problem on trees. We also project it into natural nodes space and give the polytope of the WTSS problem on trees. The projection leads to a set of valid inequalities whose separation procedure is discussed as well.

Next, building upon the result for trees---we derive the polytope of the WTSS problem on cycles; as well as a polynomial time algorithm for the WTSS problem on cycles. This formulation is in the space of the natural variables, and is based on lifting a set of valid inequalities that are natural analogues in cycles to inequalities in the tree polytope.

This leads to our contribution on general graphs. Using the observation that the

influence propagation network must be a directed acyclic graph (DAG), the extended formulation for trees can be embedded into a formulation on general graphs, where an additional exponentially sized set of constraints is added to ensure that the arcs selected form a DAG. We show that when the underlying graph is a DAG, the extended formulation on DAGs is a tight formulation (i.e., provides integer solutions on the target set selection variables). We use this to design and implement a branch-and-cut approach for the WTSS problem on general graphs. In our computational study, we are able to obtain high quality solutions for random graph instances with up to 10,000 nodes and 20,000 edges (40,000 arcs) within a reasonable amount of time.

In Chapter 3, we study the PIDS problem. First, we show that the PIDS problem on trees can be solved in linear time. Furthermore, we provide a tight and compact extended formulation for the PIDS problem on trees that provides integral solutions on the natural (node) variables. A natural question then concerns the projection of this extended formulation onto the space of the natural node variables. We provide a complete description of the projection, thus obtaining the polytope of the PIDS problem on trees. Interestingly the formulation for trees is also valid on general graphs, and provides a strong formulation for the PIDS problem. This leads to our third contribution. The projection on the space of the natural variables gives rise to an exponential size class of valid inequalities. We provide a polynomial time separation procedure for this class of valid inequalities. Furthermore, we show the conditions under which this class of valid inequalities is facet defining. Computational experience with this formulation on general graphs is discussed.

In Chapter 4, the LCIP is discussed. We first study the complexity of the LCIP. We show that the LCIP is NP-complete. We then consider several special conditions including

(1) equal influence from neighbors, (2) 100% adoption, and (3) restricting the problem to trees. Specifically, we show that the LCIP is NP-complete even on bipartite graphs, when all neighbors exert equal influence, and we do not require 100% adoption. When we require 100% adoption the problem remains NP-complete (and is in fact APX-hard). For trees, when neighbors exert *unequal* influence and we require 100% adoption, the problem remains NP-complete.

Then, we focus on the case when neighbors exert equal influence and 100% adoption is required. We study the LCIP on trees. Our contributions in this regard are three-fold. First, we propose two polynomial algorithms for the LCIP on trees. We describe a greedy algorithm which has $O(|V|\log|V|)$ running time. Second, we show a dynamic programming (DP) algorithm that has a better $O(|V|)$ running time. The DP algorithm decomposes a tree into several "star" subproblems. For each star subproblem, it finds at most two solution candidates. After all subproblems are examined, a backtracking procedure is used to determine the final solution. More importantly, the DP algorithm also works in the unequal influence case, although the running time is no longer polynomial (it is dependent on that of the mixed 0-1 knapsack problem). Third, we present two strong formulations. One is a totally unimodular (TUM) formulation for the LCIP on trees. This TUM formulation is built on the influence propagation network, i.e., influence traveling over arcs, and makes use of special structures about the amount of influence passing along an arc. The other one is an extended formulation making use of the natural payment variables and the directed influence variables. For the latter one, we project it onto the natural payment variable space and give a complete description of its polytope.

This leads to our contribution on general graphs. Using the observation that the in-

fluence propagation network must be a directed acyclic graph (DAG), both formulations for trees can be embedded into a formulation on general graphs, where an additional exponentially sized set of constraints is added to ensure that the arcs selected form a DAG. We use this to design and implement a branch-and-cut approach for the LCIP on general graphs. In our computational study, we are able to obtain high quality solutions for random graph instances with up to 10,000 nodes and 20,000 edges (40,000 arcs) within a reasonable amount of time.

Chapter 5 presents the 1TPLCIP. First, we show that the 1TPLCIP on trees can be solved in linear time. Furthermore, we provide a tight and compact extended formulation for the 1TPLCIP problem on trees that provides integral solutions on the payment variables. We are able to project this extended formulation onto the payment space and provide a complete description of the 1TPLCIP polytope on trees. The proposed formulation for trees is also valid on general graphs, and provides a strong formulation for the 1TPLCIP.

In Chapter 6, we provide concluding remarks and direction for future work.

Chapter 2: Weighted Target Set Selection on Social Networks

2.1 Introduction

In this chapter we focus on the Target Set Selection (TSS) problem proposed by Chen [2009]. Given a connected undirected graph $G = (V, E)$. For each node $i \in V$, there is a threshold, denoted by g_i , which is between 1 and $\deg(i)$, the degree of node i . All nodes are inactive initially. We select a subset of nodes, the target set, and they become active. After that, in each step, we update the state of nodes by the following rule: an inactive node i becomes active if at least g_i of its neighbors are active in the previous step. The goal is to find the minimum cardinality target set while ensuring that all nodes are active by the end of this activation process.

Chen showed that the TSS problem is hard to approximate within a polylogarithmic factor. He also provided a polynomial time algorithm for the TSS on trees. Chiang et al. [2013] provided a linear-time algorithm for the TSS problem on block-cactus graphs. They also showed that the problem is polynomially solvable on chordal graphs when $g_i \leq 2$ and on Hamming graphs when $g_i = 2$. Ben-Zwi et al. [2011] showed that for a graph with treewidth bounded by ω the TSS problem can be solved in V^ω time. Ackerman et al. [2010] provided some combinatorial bounds for the TSS problem under majority ($g_i \geq 0.5 \times \deg(i)$) and strict majority ($g_i > 0.5 \times \deg(i)$) influences. In passing they

also described an integer programming model for the TSS problem. Spencer and Howarth [2015] consider a problem of providing incentives to consumers to promote "green" (i.e., environmental friendly) behavior. In that context they consider the TSS problem (although they refer to it as the Min-Cost Complete Conversion (MCC) problem). To model the influence propagation process they use a time-indexed integer programming formulation with as many time periods as the number of nodes in the network. This model grows very rapidly and is not a computationally viable model (in their experiments they were only able to solve problems with 30 nodes and 75 edges). The same time-indexed integer program is described in Shakarian et al. [2013]. Shakarian et al. [2013] also propose a heuristic for the TSS problem. A serious issue with the previously discussed integer programming formulations in Ackerman et al. [2010], Spencer and Howarth [2015], and Shakarian et al. [2013] is the fact that they are weak; and even on trees their linear programming (LP) relaxation provides fractional solutions.

Deviating from previous literature, we consider the *weighted* TSS (WTSS) problem. In the WTSS problem, for each node $i \in V$, there is a weight, denoted by b_i , which models the fact that different nodes require differing levels of effort to become initial adopters (in practice, it is reasonable to assume different individuals require different amounts of effort to be convinced).

We first study the WTSS problem on trees. Our contributions in this regard are three-fold. First, we propose a polynomial algorithm for the WTSS problem on trees. The algorithm uses a dynamic programming approach and decomposes a tree into several "star" subproblems. For each star subproblem, it finds at most two solution candidates. After all subproblems are examined, a backtracking procedure is used to determine the

final solution. As will be evident after our description in Section 2.2, our dynamic programming algorithm significantly differs from Chen [2009], and his algorithm can be viewed as a special case of ours. His algorithm requires no backtracking procedure as it takes advantage of the observation that in the non-weighted case, leaf nodes will never be selected in the target set (this is not true in the weighted case). This allows the target set selection to be determined in one shot without any backtracking. More importantly, we present a tight and compact extended formulation for the WTSS problem on trees. The key idea in this formulation is the addition of a dummy node on each edge in the graph that cannot be selected as part of the target set and has threshold 1. We also project the extended formulation onto the space of the natural node variables that gives the polytope of the WTSS problem on trees. The projection leads to an exponentially sized set of valid inequalities whose polynomial time separation is discussed. We build upon the result for trees in two ways.

We derive the polytope of the WTSS problem on cycles; as well as a polynomial time algorithm for the WTSS problem on cycles. This formulation is in the space of the natural variables, and is based on lifting a set of valid inequalities that are natural analogues in cycles to inequalities in the tree polytope.

This leads to our contribution on general graphs. Using the observation that the influence propagation network must be a directed acyclic graph (DAG), the extended formulation for trees can be embedded into a formulation on general graphs, where an additional exponentially sized set of constraints is added to ensure that the arcs selected form a DAG. We show that when the underlying graph is a DAG, the extended formulation on DAGs is a tight formulation (i.e., provides integer solutions on the target set selection

variables). We use this to design and implement a branch-and-cut approach for the WTSS problem on general graphs. In our computational study, we are able to obtain high quality solutions for random graph instances with up to 10,000 nodes and 20,000 edges (40,000 arcs) within a reasonable amount of time.

The rest of this chapter is organized as follows. Section 2.2 presents a polynomial-time algorithm for the WTSS problem on trees. Section 2.3 describes our tight and compact extended formulation for the WTSS problem on trees. In Section 2.3.1 we derive the polytope of the WTSS problem on trees (i.e., the convex hull of the feasible target set vectors on trees) by projecting the extended formulation onto the space of the natural variables. Section 2.4 derives the polytope of the WTSS problems on cycles (i.e., the convex hull of the feasible target set vectors on cycles) and presents a polynomial-time algorithm for the WTSS problem on cycles. In Section 2.5 we show how to apply the extended formulation derived for trees to general graphs and design a branch-and-cut approach based upon it. Section 2.6 discusses our computational experience applying the branch-and-cut approach. Section 2.7 provides concluding remarks.

2.2 Algorithm for the WTSS Problem on Trees

We present an algorithm to solve the WTSS problem on trees. In this method, we decompose the tree into subproblems. Each subproblem is used to find the most promising solution candidates (at most two) and one of them will be part of the final solution of the tree. A subproblem is defined on a star network which has a single central node and (possibly) multiple leaf nodes. In this tree, each non-leaf node is a central node for a star

Algorithm 1 Algorithm for the WTSS problem on trees

- 1: Arbitrarily pick a node as the root node of the tree
 - 2: Define the order of star problems based on the bottom-up traversal of the tree
 - 3: **for** each star subproblem **do**
 - 4: StarHandling
 - 5: **end for**
 - 6: SolutionBacktrack
-

network. By solving the subproblem, we have one solution candidate for the situation that there is influence coming into the central node along the link which connects the star to the rest of the tree (this kind of influence is referred to as external influence) and one solution candidate for the no external influence situation. Next, the star is compressed into one single leaf node for the next star network. This process is repeated until we are left with a single star. The last star should have the root node of the tree as the central node. After we exhaust all subproblems, a backtracking method is used to identify a final solution which combines the solution candidates of those star subproblems for the tree. The pseudocode of the proposed algorithm is shown in Algorithm 1.

For a star subproblem, we try to find the solution that makes all nodes active while the total cost is the minimum. Observe that a leaf node only requires one active neighbor to make itself active. Therefore, in this star network, if the central node is selected, all nodes become active. However, there is another possibility. Denote the central node by c and refer to this star as star c . Sort all leaf nodes in ascending order according to their cost. Then, if the total cost of the g_c smallest cost leaf nodes is less than the central node's cost, b_c , we can select the first g_c leaf nodes to activate the central node. Then, the active central node will activate the remaining leaf nodes. Moreover, because the central node might receive influence from the rest of this tree, it is possible that we only need

Algorithm 2 StarHandling

Require: a star

- 1: **if** $b_c < \sum_{l \in (s_{g_c-1})} b_l$ **then**
 - 2: $X_I^c \leftarrow c$ and $X_{NI}^c \leftarrow c$.
 - 3: The compressed node's cost is 0.
 - 4: **else if** $b_c > \sum_{l \in s_{g_c}} b_l$ **then**
 - 5: $X_I^c \leftarrow s_{(g_c-1)}$ and $X_{NI}^c \leftarrow s_{g_c}$.
 - 6: The compressed node's cost is b_{g_c} .
 - 7: **else**
 - 8: $X_I^c \leftarrow s_{(g_c-1)}$ and $X_{NI}^c \leftarrow c$.
 - 9: The compressed node's cost is $b_c - \sum_{l \in (s_{g_c-1})} b_l$.
 - 10: **end if**
-

to select the first $(g_c - 1)$ leaf nodes to activate the central node. Hence, we have two solution candidates for a star subproblem. If the central node receives external influence, the candidate solution is the one with smaller cost between the central node and the first $(g_c - 1)$ leaf nodes. Otherwise, the candidate solution is either the central node or the first g_c leaf nodes whichever has a smaller cost.

After we determine the solution candidates for the current star subproblem, it is compressed into a single leaf node for the next star subproblem. This new single node has its threshold set to 1. Its cost is determined as follows. There are three cases.

Case 1: If the central node's cost is smaller than the total cost of the first $(g_c - 1)$ smallest leaf cost nodes, then, it is always cheaper to select the central node for current star. So, this new node has cost 0 because the next star's central node could receive influence from current star's central node in the final solution of the tree and it is free from the next star's point of view.

Case 2: If the central node's cost is bigger than the total cost of the first g_c smallest leaf cost nodes, this new node has cost equal to the cost of the g_c th smallest cost leaf node.

Case 3: If the central node's cost is not in the above two cases, this new node's cost is

equal to the difference between the central node's cost and the total cost of the first $(g_c - 1)$ cheapest leaf nodes.

For the latter two cases, if the current star receives external influence in the final solution of the tree, then, the cost for the current star is the total cost of the first $(g_c - 1)$ cheapest leaf nodes. If it does not receive external influence, the cost of the current star is the minimum of the total cost of the first g_c cheapest leaf nodes and the central node's cost. But in both cases (external influence or no external influence), we have to at least pay a cost equal to the total cost of the first $(g_c - 1)$ cheapest leaf nodes. Then, for the next star, if it wants to receive influence from the current star, it must pay the incremental amount which corresponds to the current star receiving no external influence. We summarize the above procedure in Algorithm 2 StarHandling with the following notation: let X_I^c denote the solution candidate with external influence, X_{NI}^c denote the solution candidate without external influence, s_{g_c} be the g_c cheapest leaf nodes and $s_{(g_c-1)}$ be the $(g_c - 1)$ cheapest leaf nodes.

After we obtain the solution of the last star which has the root node as its central node, we invoke a backtracking procedure to choose the solution from the solution candidates for each star subproblem and piece them together to obtain the final solution for this tree. In the last star subproblem, we choose the smaller of the central node and the g_c cheapest leaf nodes (in the last star there is only one solution candidate because there is no external influence) as the solution. Now, for each leaf node in this star, we know if there is external influence coming into it or not. For instance, if the central node is selected, then, the central node sends out influence to all its leaf nodes. If the g_c cheapest leaf nodes are selected, then, these g_c cheapest leaf nodes do not receive external influence but the

Algorithm 3 SolutionBacktrack

Require: the last star and its solution X

- 1: $X^* \leftarrow X$.
- 2: **if** X is r **then**
- 3: $\forall l \in L(r) \cap NL$ call With-Influence(l, X^*).
- 4: **else**
- 5: $\forall l \in s_{gr} \cap NL$ call No-Influence(l, X^*).
- 6: $\forall l \in \{L(r) \setminus s_{gr}\} \cap NL$ call With-Influence(l, X^*).
- 7: **end if**
- 8: $C^* = \sum_{i \in X^*} b_i$
- 9: **return** C^*, X^* .
- 10: **function** With-Influence(c, X)
- 11: $X \leftarrow (X \setminus c) \cup X_I^c$.
- 12: $\forall l \in L(c) \cap X_I^c \cap NL$ call No-Influence(l, X).
- 13: $\forall l \in L(c) \setminus X_I^c \cap NL$ call With-Influence(l, X).
- 14: **return** X .
- 15: **end function**
- 16: **function** No-Influence(c, X)
- 17: $X \leftarrow (X \setminus c) \cup X_{NI}^c$.
- 18: $\forall l \in L(c) \cap X_{NI}^c \cap NL$ call No-Influence(l, X).
- 19: $\forall l \in L(c) \setminus X_{NI}^c \cap NL$ call With-Influence(l, X).
- 20: **return** X .
- 21: **end function**

remaining leaf nodes do. With this information we can now proceed down the tree, incorporating the partial solution at a node based on whether it receives external influence or not (which we now know). This backtracking procedure is described in Algorithm 3 SolutionBacktrack. We use $V(c)$ for to denote the set of nodes, $L(c)$ to denote the set of leaf nodes in the star c , NL to denote the set of non-leaf nodes in the tree, r to denote the root of the tree (as determined by Algorithm 1), X^* denotes the final solution of the tree and C^* its cost.

In this algorithm, we have two recursive functions: With-Influence and No-Influence. They choose the solution for a star c and recursively choose solutions for stars whose central nodes are leaf nodes of the star c . Although it is possible to prove the correctness of this

algorithm directly, we defer the proof until the next section. There we will provide a tight and compact extended formulation for the WTSS problem, and use linear programming duality to prove its correctness.

Proposition 2.1. *The WTSS problem on trees can be solved in $O(|V|)$ time.*

Proof. Proof of Proposition 2.1. There are at most $|V|$ stars. For each star, we need to find g_i cheapest children and it takes $O(\deg(i))$ time. For the whole tree, this is bounded by $O(|V|)$ time. In the backtracking procedure, we pick the final solution for each node which takes $O(|V|)$ time over the tree. Therefore, the running time for the dynamic algorithm is linear with respect to the number of nodes. \square

In Figure 2.1(a), we have an instance of the WTSS problem. There are 11 nodes and the numbers beside a node are its cost and threshold. The root of this tree is node 4. Stars 1, 2, and 3 correspond to Case 1, 2, and 3 respectively. Star 1 has $X_{NI}^1 = X_I^1 = \{1\}$. Star 2 has $X_{NI}^2 = \{7, 8, 9\}$ and $X_I^2 = \{7, 8\}$. Star 3 has $X_{NI}^3 = \{3\}$ and $X_I^3 = \{10\}$. After we compress these stars, we have the final star in Figure 2.1(b) and the solution $X^4 = \{1, 2\}$. Next, we start the backtracking procedure. From star 1, we have $X^* = \{1, 2\}$. From star 2, it has no external influence; so, we select the solution X_{NI}^2 . Thus, $X^* = \{1, 7, 8, 9\}$. From star 3, it has external influence; so, we select the solution X_I^3 . Thus, $X^* = \{1, 7, 8, 9, 10\}$ and $C^* = 4 + 7 + 8 + 9 + 10 = 38$. Figure 2.1(c) illustrates the solution obtained by Algorithm 1, where the selected nodes are shaded grey.

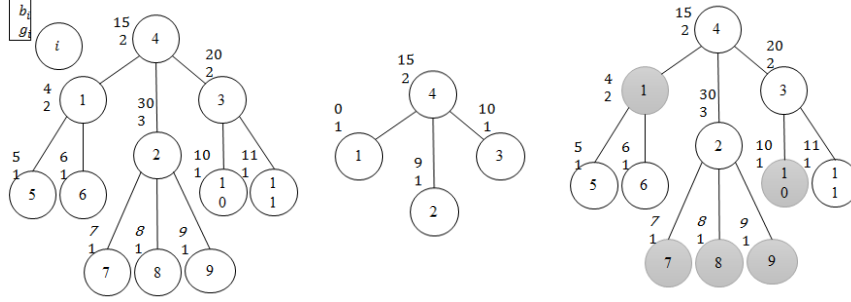


Figure 2.1: (a) A WTSS instance (b) The final star (c) The solution of Algorithm 1

2.3 A Tight and Compact Extended Formulation on Trees

We first discuss a formulation that immediately comes to mind for the WTSS problem on trees. For each node $i \in V$, let x_i be a binary decision variable denoting whether node i is selected in the target set. For each edge $\{i, j\} \in E$, create two binary arc variables h_{ij} and h_{ji} to represent the direction of influence propagation. If node i sends influence to node j , h_{ij} is 1 and 0 otherwise. For any node $i \in V$, let $a(i)$ denote the set of node i 's neighbors.

$$\text{(Ack) Minimize} \quad \sum_{i \in V} b_i x_i \quad (2.1)$$

$$\text{Subject to} \quad h_{ij} + h_{ji} = 1 \quad \forall \{i, j\} \in E$$

$$\sum_{j \in a(i)} h_{ji} + g_i x_i \geq g_i \quad \forall i \in V \quad (2.2)$$

$$x_i \in \{0, 1\} \quad \forall i \in V \quad (2.3)$$

$$h_{ij}, h_{ji} \in \{0, 1\} \quad \forall \{i, j\} \in E. \quad (2.4)$$

This formulation is first discussed in Ackerman et al. [2010]. The objective function (2.1) minimizes the total cost. Constraint (2.2) makes sure on each edge influence is only

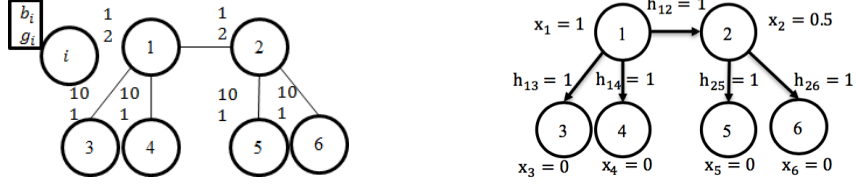


Figure 2.2: (a) A WTSS instance (b) A fractional optimal solution

propagated in one direction. Constraint (2.2) says that a node $i \in V$ must be selected or have g_i or more incoming arcs. We note that this formulation is weak, and Figure 2.2 shows its LP relaxation provides fractional solutions. Figure 2.2(a) provides a WTSS instance, and Figure 2.2(b) describes a fractional optimal solution to the LP relaxation of Ack. It has $x_1 = 1, x_2 = 0.5$. Additionally, $h_{12} = h_{13} = h_{14} = h_{25} = h_{26} = 1$. All other decision variables are zero.

We now present a tight and compact extended formulation for the WTSS problem on trees. Furthermore, we prove the correctness of Algorithm 1. From the input graph G , we create a new graph G_t by adding one dummy node to each edge in G . For each edge $\{i, j\} \in E$, insert a dummy node d . Let D denotes the set of dummy nodes. Since the dummy nodes have effectively split each edge into two in the original graph, we replace each of the original edges $\{i, j\} \in E$ by two edges $\{i, d\}$ and $\{d, j\}$ in the new graph G_t . Let E_t denote the set of edges in G_t ($G_t = (V \cup D, E_t)$). The dummy nodes cannot be selected in the target set (they can be viewed as having large costs), and all have threshold 1 (thus if one of it's neighbors is activated the dummy node will become activated and propagate the influence to the other neighbor). As before, for each node $i \in V$ binary decision variable x_i denotes whether node i is selected in the target set (these are the natural node variables). For each edge $\{i, d\} \in E_t$, where $i \in V$ and $d \in D$ (notice G_t is

bipartite and E_t only contains edges between the nodes in V and D), create two binary arc variables y_{id} and y_{di} to represent the direction of influence propagation. If node i sends influence to node d , y_{id} is 1 and 0 otherwise. As before, for any node $i \in V \cup D$ $a(i)$ denotes the set of node i 's neighbors. We can now write the following compact extended formulation for the WTSS problem on trees.

$$\text{(BIP2.1) Minimize} \quad \sum_{i \in V} b_i x_i \quad (2.5)$$

$$\text{Subject to } (u_d) \quad \sum_{i \in a(d)} y_{id} \geq 1 \quad \forall d \in D \quad (2.6)$$

$$(w_{id}) \quad x_i \leq y_{id} \quad \forall i \in V, d \in a(i) \quad (2.7)$$

$$(z_{id}) \quad y_{id} + y_{di} = 1 \quad \forall \{i, d\} \in E_t \quad (2.8)$$

$$(v_i) \quad \sum_{d \in a(i)} y_{di} + g_i x_i = g_i \quad \forall i \in V \quad (2.9)$$

$$x_i \in \{0, 1\} \quad \forall i \in V \quad (2.10)$$

$$y_{id}, y_{di} \in \{0, 1\} \quad \forall \{i, d\} \in E_t \quad (2.11)$$

We refer to the above formulation as BIP2.1. Constraint (2.6) says that each dummy node has at least one incoming arc (since dummy nodes cannot be selected and have threshold 1). Constraint (2.7) says that if a node is selected, then, it sends out influence to all its neighbors (notice this type of constraint would not be valid in Ack, since there could be two neighboring nodes that are in the target set). Constraint (2.8) makes sure on each edge influence is only propagated in one direction. Constraint (2.9) says that a node $i \in V$ must be selected or have exactly g_i incoming arcs. Clearly if a node is selected, then from constraint (2.7) it has no incoming arcs and constraint (2.9) is satisfied. On the other hand if a node $i \in V$ is not selected it must have more than g_i incoming arcs. Since

we have dummy nodes, we can reverse some of them to have exactly g_i incoming arcs. Constraints (2.10) and (2.11) are binary constraints. We now show the validity of BIP2.1.

Proposition 2.2. *BIP2.1 is a valid formulation for the WTSS problem on trees.*

Proof. Proof of Proposition 2.2. From the discussion above it should be clear that if (\mathbf{x}, \mathbf{y}) is a feasible solution to BIP2.1, then \mathbf{x} must be a feasible target set (one that activates all nodes in the graph). We now show how any feasible target set vector \mathbf{x} can be extended to a feasible solution (\mathbf{x}, \mathbf{y}) to BIP2.1 proving validity. Let P be the set of nodes selected in a feasible target set. For each p in P , we set x_p as 1 and set the remaining x variables as 0. Next, all outgoing arcs of a node p in P are set to 1 and all its incoming arcs are set to 0, i.e. $y_{pd} \leftarrow 1$ and $y_{dp} \leftarrow 0 \forall p \in P, d \in a(p)$ (this ensures constraint (2.7) is satisfied). Next, for any dummy node d that has y values set for only one of its adjacent edges we set the y values as follows. Without loss of generality, let node i and j be adjacent to dummy node d and y_{id} is set to 1 now. Then we set $y_{jd} \leftarrow 0$ and $y_{dj} \leftarrow 1$ to propagate the influence. After that we check for any new nodes $i \in V$ that have been activated by incoming influence (arcs). If so, node i sends out influence to those adjacent dummy nodes that do not send influence to it, i.e. $y_{id} \leftarrow 1$ and $y_{di} \leftarrow 0$ for all $d \in a(i)$ such that y_{id} and y_{di} are not set yet. We repeat this whole procedure until all nodes are active and all x and y value are set. Observe that the procedure so far satisfies constraints (2.6), (2.7), and (2.8). At this point, there might be a node i in V that has more than g_i incoming arcs. But we can always reverse an appropriate number of these incoming arcs to have exactly g_i incoming arcs; and thus satisfy constraint (2.9). Figure 2.3 illustrates the outcome of the procedure described in Proposition 2.2. □

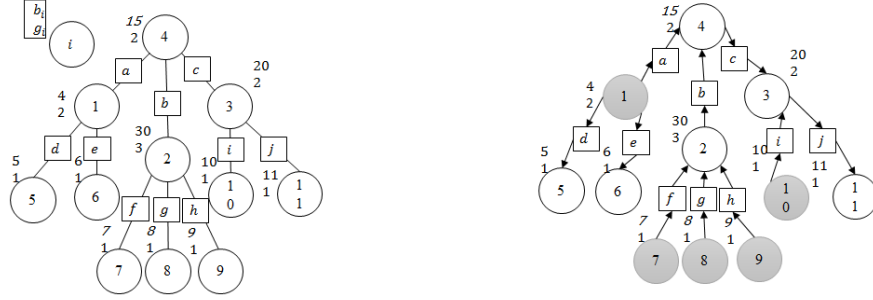


Figure 2.3: (a) G_t (b) A valid solution to BIP2.1

The linear relaxation of BIP2.1 is the following linear programming problem:

$$(LP2.1) \quad \text{Minimize} \quad \sum_{i \in V} b_i x_i \quad (2.12)$$

$$\text{Subject to} \quad (2.6), (2.7), (2.8), (2.9) \quad (2.13)$$

$$x_i \geq 0 \quad \forall i \in V \quad (2.14)$$

$$y_{id} \geq 0 \quad \forall i \in V, d \in a(i) \quad (2.15)$$

$$y_{di} \geq 0 \quad \forall d \in D, i \in a(d) \quad (2.16)$$

We refer to this linear programming problem as LP2.1. The dual to LP2.1 is as follows:

$$(DLP2.1) \quad \text{Maximize} \quad \sum_{d \in D} u_d + \sum_{i \in V} g_i v_i + \sum_{\{i,d\} \in E_t} z_{id} \quad (2.17)$$

$$\text{Subject to } (y_{id}) \quad w_{id} + u_d + z_{id} \leq 0 \quad \forall i \in V, d \in a(i) \quad (2.18)$$

$$(y_{di}) \quad v_i + z_{id} \leq 0 \quad \forall d \in D, i \in a(d) \quad (2.19)$$

$$(x_i) \quad g_i v_i - \sum_{d \in a(i)} w_{id} \leq b_i \quad \forall i \in V \quad (2.20)$$

$$u_d \geq 0 \quad \forall d \in D \quad (2.21)$$

$$w_{id} \geq 0 \quad \forall i \in V, d \in a(i) \quad (2.22)$$

We have u_d , w_{id} , z_{id} and v_i as dual variables for constraint sets (2.6), (2.7), (2.8), and

(2.9) respectively. We refer to the dual linear problem as DLP2.1. Let $\text{conv}(X)$ denote the convex hull of feasible target set vectors \mathbf{x} , and let ETSS denote the feasible region of LP2.1.

Theorem 2.1. *Given a tree, LP2.1 has an optimal solution with x binary and $\text{Proj}_{\mathbf{x}}(\text{ETSS}) = \text{conv}(X)$.*

Proof. Proof of Theorem 2.1. It should be clear that the solution of Algorithm 1 (via Proposition 2.2) provides a feasible solution to BIP2.1 and thus LP2.1. We will now construct a dual feasible solution to DLP2.1 that satisfies the complementary slackness (CS) conditions. The CS conditions of LP2.1 and DLP2.1 are the following:

$$(1 - \sum_{i \in a(d)} y_{id})u_d = 0 \quad \forall d \in D \quad (2.23)$$

$$(x_i - y_{id})w_{id} = 0 \quad \forall i \in V, d \in a(i) \quad (2.24)$$

$$(b_i - g_i v_i + \sum_{d \in a(i)} w_{id})x_i = 0 \quad \forall i \in V \quad (2.25)$$

$$(-v_i - z_{id})y_{di} = 0 \quad \forall d \in D, i \in a(d) \quad (2.26)$$

$$(-w_{id} - u_d - z_{id})y_{id} = 0 \quad \forall i \in V, d \in a(i) \quad (2.27)$$

Let L be the set of leaf nodes in G_t , and let r be the root of the tree as determined by Algorithm 1 (the central node in the last star in Algorithm 1). We will start from the leaf nodes and then follow the sequence of stars that Algorithm 1 considers (the proof works with any sequence, but for expositional reasons it is best to consider the same sequence of stars). For the purposes of this proof we will construct a specific form of the extended solution from Algorithm 1. Recall as we backtrack for each central node of a star encountered we know whether it obtains external influence or not in the final solution. Thus, even prior

to inserting dummy nodes, we simply direct the arc from central node c 's parent to it in the case of external influence, and the arc to node c 's parent in the case of no influence. For any leaf node that is picked we direct the arc from the leaf to its parent and for leaf nodes that are not picked we direct the arc from its parent to it. It is easy to observe (before dummy nodes are inserted and the solution extended) that in a solution obtained by Algorithm 1 a node i that is not selected has by design exactly g_i incoming arcs. We now insert dummy nodes on each edge. If the influence on the edge propagates from i to j , then with the dummy node d inserted in the middle of the edge the influence propagates from i to d to j with the y values set accordingly. Now to satisfy constraints (2.7) and (2.9) for any node i that is selected and has external influence we simply reverse the arc from node i to the dummy node between it and its parent. An important (and meaningful) consequence of this construction is that if a dummy node p in this extended solution has two incoming arcs (from nodes i and j with node i being node j 's child in Algorithm 1), then node i must have been picked in the solution.

For each node l in L and the dummy node d adjacent to node l , we set $v_l = b_l$, $u_d = b_l$, $w_{ld} = 0$ and $z_{ld} = -b_l$. With these choices constraints (2.18), (2.19), and (2.20) associated with l , d and edge $\{l, d\}$, are satisfied and always binding. Consequently, regardless of the primal solution conditions (2.25), (2.26), and (2.27) are satisfied for nodes $l \in L$ and the edge $\{l, d\}$ adjacent to it. Condition (2.24) is satisfied because w_{ld} is 0. For (2.23), if $1 - y_{ld} - y_{id}$ is not equal to 0 where node i is the other node adjacent to node d , it means node l is selected when it can be activated by an incoming arc which never happens in Algorithm 17 (otherwise that would imply in stars solely containing leaf nodes of the original tree Algorithm 17 selects both a leaf node and its parent node at the

same time). Hence, we have $1 - y_{ld} - y_{id} = 0$ and (2.23) is satisfied. Therefore, the complementary slackness conditions are satisfied for this part of the dual solution.

For the center c of the star that Algorithm 1 considers we will show how to select v_c , u_d for the dummy node between node c and its parent, $w_{cd} \forall d \in a(c)$, and $z_{cd} \forall \{c, d\} \in E_t$ while ensuring all the associated complementary slackness conditions are satisfied. Consider the first star that Algorithm 1 considers. The center of this star i must have children dummy nodes that have their u variables set. Relabel these children dummy nodes in ascending order based on their u values. Let S_{g_i-1} be the first $(g_i - 1)$ children dummy nodes and S_{g_i} be the first g_i children dummy nodes. If node i is not the root node r , let p be the parent dummy node of node i and set the value u_p in the following way: We have three cases corresponding identically to the three cases in how a star is handled and compressed in Algorithm 1. If $b_i < \sum_{d \in S_{g_i-1}} u_d$, set $u_p = 0$. If $b_i > \sum_{d \in S_{g_i}} u_d$, set $u_p = u_{g_i}$. Otherwise, set $u_p = b_i - \sum_{d \in S_{g_i-1}} u_d$. We refer to these as Case 1, 2 and 3 respectively. If g_i is equal to $\text{degree}(i)$, then, there is no S_{g_i} and we only have Case 1 and 3. By setting u_p in the aforementioned way for a dummy node p which is the parent of the central node of a star encountered by Algorithm 1, (2.23) is satisfied. To show this, we only need to worry about the value of u_p when the dummy node p has two incoming arcs. Let i and j be the two adjacent nodes to this dummy node p (with i the central node of the current star and j its parent in Algorithm 1). As we argued above, node i must have been selected by Algorithm 1. But then we have $u_p = 0$ because node p 's value would be set based on Case 1, thus satisfying (2.23). Now we will focus on conditions (2.24), (2.25), (2.26) and (2.27).

Now, we consider two situations. Sort the dummy nodes adjacent to node i based

on their u values in ascending order. Let F_{g_i} be the first g_i dummy nodes. In situation 1, node i is selected in the solution. So, $x_i = 1$. Then, we set v_i as the g_i th smallest value in $\{u_d : d \in a(i)\}$. Let $W = g_i v_i - b_i$. Because node i is selected in the solution, b_i is smaller than $\sum_{d \in F_{g_i}} u_d$ and W has a positive value due to the fact that $u_{(g_i)}$ is bigger than $\frac{b_i}{g_i}$. We have $y_{id} = 1$ for all $d \in a(i)$ due to (2.7) and $y_{di} = 0$ for all $d \in a(i)$ due to (2.8). So, (2.18) and (2.20) must be binding based on condition (2.25) and condition (2.27). Meanwhile, we need to satisfy (2.19). We set $z_{id} = -v_i$ and $w_{id} = v_i - u_d$ for all $d \in F_{g_i}$. Then, $z_{id} = -u_d$ and $w_{id} = 0$ for all $d \in \{a(i) \setminus F_{g_i}\}$. Lastly, let $w = W - \sum_{d \in F_{g_i}} w_{id}$. Pick any dummy node \bar{d} in $a(i) \setminus F_{g_i}$ and set $w_{i\bar{d}} = w$ and $z_{i\bar{d}} = -u_{\bar{d}} - w$. Here, we reset the value of $w_{i\bar{d}}$ to distribute the excess amount of W in order to satisfy that $\sum_{d \in a(i)} w_{id} = W$ and ensure (2.20) is binding. For any $d \in a(i)$, (2.18) is binding. Furthermore, (2.19) is binding for any $d \in F_{g_i}$ and is satisfied as an inequality for any $d \in a(i) \setminus F_{g_i}$. So, condition (2.24) is satisfied because $x_i = y_{id} = 1$ and condition (2.26) is satisfied because $y_{di} = 0$. We have (2.18) and (2.20) binding, so, condition (2.25) and condition (2.27) are satisfied. Therefore, the CS conditions are satisfied for this part of the dual solution.

In situation 2, node i is not selected but it has exactly g_i incoming arcs. We have $x_i = 0$ and condition (2.25) is satisfied. Let I be the set of dummy nodes that sends influence to node i , i.e. $y_{di} = 1$ for all d in I . We set $v_i = u_d$ where $d = \arg \max\{u_d : d \in I\}$. Then, for all $d \in I$, we also have $y_{di} = 1$ and $y_{id} = 0$ due to (2.8). So, (2.19) must be binding due to condition (2.26). Therefore, we set $z_{id} = -v_i$ and $w_{id} = 0$ for all $d \in I$. Let $W = g_i v_i - b_i$. If W has a positive value, it means (2.20) is violated. So, following the ascending ordering of u_d for all d in I , we set $w_{id} = \min\{v_i - u_d, W - \sum_{j \in I} w_{ij}\}$ to get (2.20) satisfied. Because node i is not selected, it implies that $b_i \geq \sum_{d \in I} u_d$. So,

$v_i g_i - b_i \leq v_i g_i - \sum_{d \in I} u_d$. Therefore, we can distribute W in this way and ensure that (2.18) holds. Then, for all $d \in I$, (2.18) is satisfied, (2.19) is binding and (2.20) is satisfied. Condition (2.24) is satisfied because $x_i = y_{id} = 0$, Condition (2.26) is satisfied because $z_{id} = -v_i$ and condition (2.27) is satisfied because $y_{id} = 0$. Next, for all $d \in a(i) \setminus I$, $y_{id} = 1$ and $y_{di} = 0$ due to (2.8) and (2.9). Hence, (2.18) must be binding due to condition (2.27). Then, we set $z_{id} = -u_d$ and $w_{id} = 0$ for all $d \in a(i) \setminus I$. So, for all $d \in a(i) \setminus I$, (2.18) is binding, and (2.19) is satisfied because $u_d \geq u_{(g_i)}$. Condition (2.24) is satisfied because $w_{id} = 0$, condition (2.26) is satisfied because $y_{di} = 0$ and condition (2.27) is satisfied because $z_{id} = -u_d$ and $w_{id} = 0$. So, the CS conditions are satisfied for this part of the dual solution.

We repeat this procedure for each star that Algorithm 1 encounters and obtain a dual feasible solution to DLP2.1 that satisfies the complementary slackness conditions associated with our primal solution obtained from Algorithm 1. \square

We illustrate the procedure for finding the dual solution with the solution in Figure 2.3.(b). Firstly, for leaf nodes (5, 6, 7, 8, 9, 10, 11) and dummy nodes adjacent to them (d, e, f, g, h, i, j), we have $v_5 = 5, v_6 = 6, v_7 = 7, v_8 = 8, v_9 = 9, v_{10} = 10, v_{11} = 11$ and $u_d = 5, u_e = 6, u_f = 7, u_g = 8, u_h = 9, u_i = 10, u_j = 11$. For those edges between them, we have $w_{5d} = 0, w_{6e} = 0, w_{7f} = 0, w_{8g} = 0, w_{9h} = 0, w_{10i} = 0, w_{11j} = 0$ and $z_{5d} = -5, z_{6e} = -6, z_{7f} = -7, z_{8g} = -8, z_{9h} = -9, z_{10i} = -10, z_{11j} = -11$.

Then, we have nodes 1, 2 and 3 ready for the next step. We start with node 1. Then, node 1's parent dummy node is node a and $u_a = 0$ because node 1 is a Case 1 node. Node 1 is in situation 1. We have $v_1 = 5$ because $g_1 = 2$ and the second smallest value

among $\{0, 5, 6\}$ is 5. $W = 5 * 2 - 4 = 6$. Hence, $z_{1a} = -5$, $w_{1a} = 5$ and $z_{1d} = -5$, $w_{1a} = 0$ for edges $(1, a)$ and $(1, d)$. Then, $w_{1e} = 6 - 5 = 1$ and $z_{1e} = -(6 + 1) = -7$.

Next, node 2's parent dummy node is node b and $u_b = 9$ because node 2 is a Case 2 node. Node 2 is in situation 2. We have $v_2 = 9$ because it has incoming arcs $(f, 2)$, $(g, 2)$ and $(h, 2)$ and the biggest value among $\{7, 8, 9\}$ is 9. Hence, $z_{2f} = -9$, $z_{2g} = -9$, $z_{2h} = -9$ and $w_{2f} = 0$, $w_{2g} = 0$, $w_{2h} = 0$. $W = 27 - 30 = -3$. Then, $z_{2b} = -9$ and $w_{2b} = 0$.

We now look at node 3. Node 3's parent dummy node is node c and $u_c = 10$ because node 3 is a Case 3 node. Node 3 is in situation 2. So, $v_3 = 10$ because it has incoming arcs $(c, 3)$ and $(i, 2)$ and the biggest value among $\{10, 10\}$ is 10. So, $z_{3c} = -10$, $z_{3i} = -10$ and $w_{3c} = 0$ $w_{3i} = 0$. $W = 20 - 20 = 0$. Then, $z_{2b} = -11$ and $w_{2b} = 0$.

Now, node 4 is ready. It is in situation 2. So, $v_4 = 9$ because it has incoming arcs $(a, 4)$ and $(b, 4)$ and the biggest value among $\{0, 9\}$ is 9. Then, $z_{4a} = -9$, $z_{4b} = -9$ and $w_{3c} = 0$, $w_{3i} = 0$. $W = 18 - 15 = 3$. Start from node a . $w_{4a} = \min\{9 - 0, 3\} = 3$. Also, $z_{4c} = -10$ and $w_{4c} = 0$. The sum of $2v_4$, z_{3c} , z_{3i} and z_{3j} is -10 . The dual objective value is 38 which is exactly equal to the primal objective value.

Interestingly, we can show that any extreme point of LP2.1 with x variables binary also has the y variables binary.

Lemma 2.1. *Every extreme point of LP2.1 with x variables binary has y variables binary.*

Proof. Proof of Lemma 2.1. Assume this is not true and there is an extreme point $(\mathbf{x}^*, \mathbf{y}^*)$ where \mathbf{y}^* is fractional. Observe when $x_i^* = 1$, $y_{id}^* = 1$ and $y_{di}^* = 0$ for all $d \in a(i)$ due to constraints (2.7) and (2.8). Since the x variables are binary, this implies fractional y

values must correspond to arcs adjacent to a node i which is not in the target set. Because constraint (2.9) is an equation and g_i only takes integer value, there are at least two fractional incoming arcs for node i . Let (j, i) and (k, i) be these two incoming arcs. Then, we can find a path which starts from arc (i, j) and ends at a dummy node which has one incoming arc with value 1. All arcs in this path take fractional values. We call this path P_j . Also, let \bar{P}_j be the reverse path of P_j . Similarly, we can also find the path P_k and its \bar{P}_k . These two paths do not contain common nodes except node i because we are given a tree. Then, let ϵ be a very small positive value such that $0 \leq y_e^* \pm \epsilon \leq 1$ for all arcs $e \in P_j \cup P_k \cup \bar{P}_j \cup \bar{P}_k$. Then, we can construct two feasible solutions in the following way: First, both of them have same values as $(\mathbf{x}^*, \mathbf{y}^*)$ except these variables y_e for all $e \in P_j \cup P_k \cup \bar{P}_j \cup \bar{P}_k$. Then, one has $y_e = y_e^* + \epsilon$ for all $e \in P_j$, $y_e = y_e^* - \epsilon$ for all $e \in \bar{P}_j$, $y_e = y_e^* - \epsilon$ for all $e \in P_k$, and $y_e = y_e^* + \epsilon$ for all $e \in \bar{P}_k$. The other one has $y_e = y_e^* - \epsilon$ for all $e \in P_j$, $y_e = y_e^* + \epsilon$ for all $e \in \bar{P}_j$, $y_e = y_e^* + \epsilon$ for all $e \in P_k$, and $y_e = y_e^* - \epsilon$ for all $e \in \bar{P}_k$. Therefore, $(\mathbf{x}^*, \mathbf{y}^*)$ is not an extreme point because it is a convex combination of these two constructed feasible solutions. \square

2.3.1 Polytope of the WTSS problem on Trees

In this section, we derive the polytope of the WTSS problem on trees. The extended formulation is projected onto the space of the node (i.e., x) variables by projecting out all arc (i.e., y) variables. Usually, there are two approaches for projecting out variables. One approach is Fourier-Motzkin elimination (which can easily be applied to project out the x variables). A more elegant method, proposed by Balas and Pulleyblank [1983], is based

upon a theorem of the alternatives. We will follow this approach.

Given the extended formulation, we first replace constraint (2.9) by its greater than or equal to form. It is easy to check that the LP relaxation of this new form is equivalent to that of the old one in the x variable space. Given a feasible solution (\mathbf{x}, \mathbf{y}) of the old form, it is a feasible solution for the new form already because the new form is a relaxation of the old one. Given a solution (\mathbf{x}, \mathbf{y}) for the LP relaxation of the new form. If a constraint of type (2.9) for a node i is not binding, we can make it binding by reducing the total value of $\sum_{d \in a(i)} y_{di}$. To do so, take an appropriate number of y_{di} and reduce their value to ensure this constraint of type of (2.9) binding. This will result an increase in corresponding y_{id} to $1 - y_{di}$. Now, we have a feasible solution for the old form with the constraint of type (2.9) binding. Since there are no costs on the y variables there is no change in cost.

We first substitute out all y_{id} variables by $1 - y_{di}$ because $y_{id} + y_{di} = 1$. Then, we have the following formulation and denote its feasible region as P_d .

$$\text{Minimize} \quad \sum_{i \in V} b_i x_i \quad (2.28)$$

$$\text{Subject to } (u_d) \quad -\sum_{i \in a(d)} y_{di} \geq -1 \quad \forall d \in D \quad (2.29)$$

$$(v_{id}) \quad -y_{di} - x_i \geq -1 \quad \forall i \in V, d \in a(i) \quad (2.30)$$

$$(w_i) \quad \sum_{d \in n(i)} y_{di} + g_i x_i \geq g_i \quad \forall i \in V \quad (2.31)$$

$$y_{di} \geq 0 \quad \forall \{i, d\} \in E_t \quad (2.32)$$

$$0 \leq x_i \leq 1 \quad \forall i \in V \quad (2.33)$$

Based on Theorem 2 in Balas and Pulleyblank [1983], the projection cone W is described by the following linear inequalities:

$$w_i - u_d - v_{id} \leq 0 \quad \forall i \in V \ \& \ d \in a(i) \quad (2.34)$$

$$w_i \geq 0, u_d \geq 0, v_{id} \geq 0 \quad \forall i \in V \ \& \ d \in a(i) \quad (2.35)$$

where u_d , v_{id} and w_i are dual multipliers corresponding to constraints (2.29), (2.30) and (2.31) respectively. If P_d can be represented as $\{\mathbf{Ax} + G\mathbf{y} \geq \mathbf{b}\}$. Then, for any feasible vector $(\mathbf{w}, \mathbf{u}, \mathbf{v})$ to W , it defines a valid inequality: $(\mathbf{w}, \mathbf{u}, \mathbf{v})^T \mathbf{Ax} \geq (\mathbf{w}, \mathbf{u}, \mathbf{v})^T \mathbf{b}$ in the space of the node (x) variables. Furthermore, the projection of P_d is defined by the valid inequalities defined by the extreme rays of W .

Theorem 2.2. *The vector $\mathbf{r} = (\mathbf{w}, \mathbf{u}, \mathbf{v}) \in W$ is extreme if and only if there exists a positive α such that one of the following three cases holds true:*

1. $u_d = \alpha$ for one $d \in D$. All other w, u, v are 0.
2. $v_{id} = \alpha$ for one $\{i, d\} \in E_t$. All other w, u, v are 0.
3. $w_i = \alpha$ for all $i \in S$ where $S \subseteq V$ and S is connected in the original graph G .

Then $u_d = \alpha$ for all $d \in D(S)$ where $D(S)$ is the set of dummy nodes belonging to $G_t(S)$, the induced subgraph of S in the transformed graph G_t . In addition, either $v_{id} = \alpha$ or $u_d = \alpha$ for all $d \in a(S) \setminus D(S)$. All other w, u, v are 0.

Proof. Proof of Theorem 2.2. For ease of exposition, we use Figure 2.4 to illustrate the notation in Case 3. Figure 2.4(a) shows the original graph G and Figure 2.4(b) shows the transformed graph G_t . Here, w_1, w_2, w_3 and w_4 have positive values α . Thus, S contains node 1, 2, 3 and 4. (Notice S is connected in the original graph G .) The induced subgraph $G_t(S)$ is shown in Figure 2.4(c). Based on $G_t(S)$, we obtain $D(S) = \{a, b, c\}$

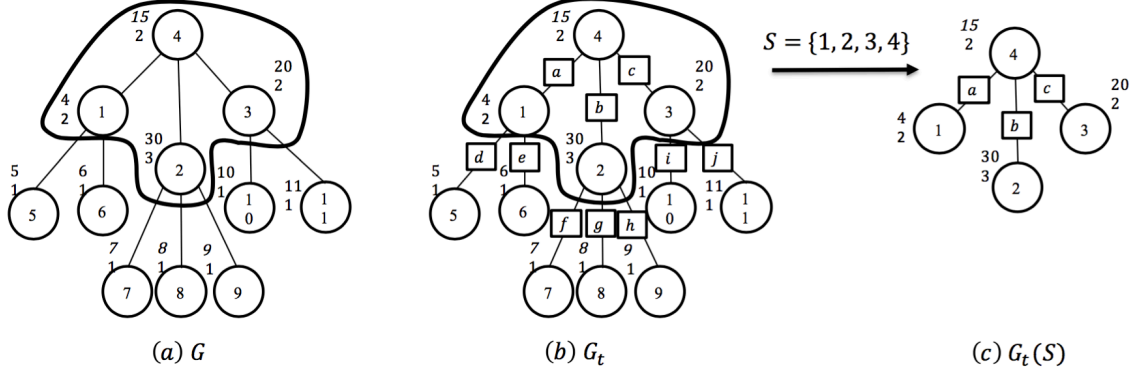


Figure 2.4: Illustration of Notation in Theorem 2.2

and $a(S) \setminus D(S) = \{d, e, f, g, h, i, j\}$. Note, $a(S)$ denotes the set of nodes adjacent to set S .

Recall that a polyhedral cone C is the intersection of a finite number of half-spaces through the origin, and a pointed cone is one in which the origin is an extreme point. A ray of a cone C is the set $R(\mathbf{y})$ of all non-negative multipliers of some $\mathbf{y} \in C$, called the direction (vector) of $R(\mathbf{y})$. A vector $\mathbf{y} \in C$ is extreme, if for any $\mathbf{y}^1, \mathbf{y}^2 \in C$, $\mathbf{y} = \frac{1}{2}(\mathbf{y}^1 + \mathbf{y}^2)$ implies $\mathbf{y}^1, \mathbf{y}^2 \in R(\mathbf{y})$. A ray $R(\mathbf{y})$ is extreme if its direction vector \mathbf{y} is extreme.

Sufficiency. Let $\mathbf{r} \in W$ be of the form Case 1 and assume that $\mathbf{r} = \frac{1}{2}(\mathbf{r}^1 + \mathbf{r}^2)$ for some $\mathbf{r}^1, \mathbf{r}^2 \in W$. Then, except u_d^1 and u_d^2 , all other directions are 0. Then, $\mathbf{r}^1, \mathbf{r}^2$ are in $R(\mathbf{r})$. So, \mathbf{r} is extreme.

Case 2 is similar to Case 1.

Let $\mathbf{r} \in W$ be of the form Case 3 and assume that $\mathbf{r} = \frac{1}{2}(\mathbf{r}^1 + \mathbf{r}^2)$ for some $\mathbf{r}^1, \mathbf{r}^2 \in W$. So, for any component in \mathbf{r} with value 0, their corresponding components in \mathbf{r}^1 and \mathbf{r}^2 are also 0. Given i and d , let $p_{id}^k, k = 1, 2$, represent the positive component u_d^k or $v_i^k, k = 1, 2$. Then, we have $w_i^1 + w_i^2 = 2\alpha$ and $p_{id}^1 + p_{id}^2 = 2\alpha$ for all $d \in a(i)$ and for all $i \in S$. Then, if there is a pair d_1 and d_2 , we have $p_{id_1}^1 > p_{id_2}^1$ if and only if $p_{id_1}^2 < p_{id_2}^2$. But

constraint (2.34) imposes that $w_i^k \leq p_{id}^k$, $k = 1, 2$. Hence, $p_{id_1}^k = p_{id_2}^k = \alpha_k$, $k = 1, 2$, for all $d_1, d_2 \in a(i)$. Otherwise, either constraint (2.34) would be violated or $w_i^1 + w_i^2 < 2\alpha$ because w_i^k would take the smaller value between $p_{id_1}^k$ and $p_{id_2}^k$, $k = 1, 2$. Therefore, $\mathbf{r}^1, \mathbf{r}^2$ are in $R(\mathbf{r})$. Therefore, \mathbf{r} is extreme.

Necessity. Let \mathbf{r} be an extreme vector of W . Let $C_{\mathbf{r}} = \{S \subseteq V : w_i > 0 \ \forall i \in S \ \& \ S \text{ is connected in the original tree graph } G \text{ and maximal}\}$, $S_d = \{d \in D : u_d > 0\}$ and $S_{id} = \{\{i, d\} \in E_t : v_{id} > 0\}$ based on this \mathbf{r} . First, we consider the situation where $C_{\mathbf{r}} = \emptyset$ and assume $|S_d| + |S_{id}| > 1$. Let \mathbf{r}^1 contain all but one of the positive components in \mathbf{r} with double their values. Let \mathbf{r}^2 contain the one positive component omitted by \mathbf{r}^1 in \mathbf{r} with double its value. Then, $\mathbf{r}^1, \mathbf{r}^2 \in W$ and $\mathbf{r} = \frac{1}{2}(\mathbf{r}^1 + \mathbf{r}^2)$. So, if $|S_d| + |S_{id}| > 1$, \mathbf{r} is not extreme, contrary to the assumption. We conclude that if $C_{\mathbf{r}} = \emptyset$, then $|S_d| + |S_{id}| = 1$ and thus \mathbf{r} is in the form of either Case 1 or Case 2.

Now consider the situation when $C_{\mathbf{r}} \neq \emptyset$. When $|C_{\mathbf{r}}| > 1$, it means there is more than one connected component. Consider any set $S \in C_{\mathbf{r}}$. Then, \mathbf{r}^1 has values $w_i^1 = 2w_i$ and $u_d^1 = 2u_d, v_{id}^1 = 2v_{id}$ for all $d \in a(i)$ and all $i \in S$ and 0s in other components. Then, let \mathbf{r}^2 be $2\mathbf{r} - \mathbf{r}^1$. Hence, we have $\mathbf{r} = \frac{1}{2}(\mathbf{r}^1 + \mathbf{r}^2)$ and $\mathbf{r}^1, \mathbf{r}^2$ are different in at least one direction. So, if $|C_{\mathbf{r}}| > 1$, \mathbf{r} is not extreme, contrary to the assumption.

When $|C_{\mathbf{r}}| = 1$ (so the set of nodes with $w_i > 0$ in the original tree G form a single connected component) and $S \in C_{\mathbf{r}}$, define $S_0 = \{\{j, d\} \in E_t : p_{jd} > 0 \ \& \ j \in V \setminus S\}$. (Recall that we use p_{jd} to generically represent either of the positive components u_d or v_{jd} .) If $S_0 \neq \emptyset$, let \mathbf{r}^1 have $w_i^1 = 2w_i$ and $u_d^1 = 2u_d, v_{id}^1 = 2v_{id}$ for all $d \in a(i)$ and all $i \in S$ and 0s in other components. Then, let \mathbf{r}^2 be $2\mathbf{r} - \mathbf{r}^1$. \mathbf{r}^2 is feasible and it has at least one positive component because S_0 is not empty and its corresponding components are

not in \mathbf{r}^1 . Hence, we have $\mathbf{r} = \frac{1}{2}(\mathbf{r}^1 + \mathbf{r}^2)$ and $\mathbf{r}^1, \mathbf{r}^2$ are different in at least one direction. So, if $|C_{\mathbf{r}}| = 1$ and $S_0 \neq \emptyset$, \mathbf{r} is not extreme, contrary to the assumption.

When $|C_{\mathbf{r}}| = 1$, define $S_1 = \{\{i, d\} \in E_t : i \in S, u_d > 0 \oplus v_{id} > 0\}$ and $S_2 = \{\{i, d\} \in E_t : i \in S, u_d > 0 \ \& \ v_{id} > 0\}$. If $S_2 \neq \emptyset$ and let $\alpha^1 = 2 \times \min\{w_i, u_d, v_{id} : i \in S, (i, d) \in S_2\}$, then, we make \mathbf{r}^1 have $w_i^1 = \alpha^1$ for all $i \in S$. For $\{i, d\} \in S_2$, we have $v_{id}^1 = \alpha^1$. Also, for $\{i, d\} \in S_1$, if $u_d > 0$, we have $u_d^1 = \alpha^1$. Otherwise, we have $v_{id}^1 = \alpha^1$. The remaining components are 0s. Then, let \mathbf{r}^2 be $2\mathbf{r} - \mathbf{r}^1$. Hence, we have $\mathbf{r} = \frac{1}{2}(\mathbf{r}^1 + \mathbf{r}^2)$ and $\mathbf{r}^1, \mathbf{r}^2$ are different in at least one direction because the component which is $\arg \min\{w_i, u_d, v_{id} : i \in S, \{i, d\} \in S_2\}$ is positive in \mathbf{r}^1 but zero in \mathbf{r}^2 . Thus we must have $|S_2| = \emptyset$.

Next, if $|C_{\mathbf{r}}| = 1$ and $|S_2| = \emptyset$, consider a node $i \in S$ and let $w_i = \alpha$. Define $S^+ = \{\{i, d\} \in S_1 : p_{id} > \alpha\}$. When $S^+ \neq \emptyset$, consider an edge $\{i, d\} \in S^+$ with $u_d > 0$. We can make \mathbf{r}^1 have $u_d^1 = 2 \times (u_d - \alpha)$ and 0s in the other components. Then, let \mathbf{r}^2 be $2\mathbf{r} - \mathbf{r}^1$ (note that \mathbf{r}^2 is feasible). Hence, we have $\mathbf{r} = \frac{1}{2}(\mathbf{r}^1 + \mathbf{r}^2)$ and $\mathbf{r}^1, \mathbf{r}^2$ are different in at least one direction because \mathbf{r}^1 only has one positive component and \mathbf{r}^2 has at least two positive components. Thus, if $|C_{\mathbf{r}}| = 1$ then $S^+ = \emptyset$.

Next, define $S_v = \{\{i, d\} \in G_t(S) : v_{id} > 0\}$. If $S_v \neq \emptyset$, consider an edge $\{i, d\} \in S_v$ with j adjacent to d . We decompose S into two connected components (in the original tree graph G) by using the edge $\{i, j\}$ (removing $\{i, j\}$ in G separates S into two connected components). Let S_i be the one containing node i . We make \mathbf{r}^1 have $w_i^1 = 2w_i$, $u_d^1 = 2u_d$, and $v_{id}^1 = 2v_{id}$ for all i in S_i and d in $a(i)$. Also, it has 0s in the remaining components. Then, let \mathbf{r}^2 be $2\mathbf{r} - \mathbf{r}^1$ (which is feasible). Hence, we have $\mathbf{r} = \frac{1}{2}(\mathbf{r}^1 + \mathbf{r}^2)$ and $\mathbf{r}^1, \mathbf{r}^2$ are different in at least one direction. Thus, if $|C_{\mathbf{r}}| = 1$ then $S_v = \emptyset$.

Consider an edge $\{i, j\}$ in the original graph G where both $i, j \in S$; and the edges $\{i, d\}$ and $\{d, j\}$ in $G_t(S)$ associated with $\{i, j\}$. Here only $u_d > 0$ because $S_v = \emptyset$. So, $w_i = w_j = u_d$ because $S^+ = \emptyset$. Given S is a connected component, we have $w_i = w_j$ for i, j in S ; and so $u_d = \alpha$ for $d \in D(S)$. This proves that if $C_r \neq \emptyset$, then \mathbf{r} must be in the form of Case 3. \square

Applying Theorem 2 in Balas and Pulleyblank [1983]---Case 1 extreme directions generate the valid inequality $0 \geq -1$ which is not very useful, while Case 2 extreme directions generate $-x_i \geq -1$ or the trivial inequalities, $x_i \leq 1$ for all $i \in V$. Given an extreme ray \mathbf{r} of the form described in Case 3, define $V_i = \{d \in a(i) : v_{id} > 0\}$ for all $i \in S$ based on \mathbf{r} . Consequently, Case 3 extreme directions generate the following valid inequality in the original graph G :

$$\sum_{i \in S} (g_i - |V_i|)x_i \geq \sum_{i \in S} g_i - |a(S)| - |S| + 1 \quad \forall i \in S \subseteq V \quad \& \quad S \text{ is connected}$$

$$\& \quad |V_i| = 0, 1, \dots, V_i^m.$$

Here $a(S)$ denotes the neighbors of set S in the original graph G . Noting that given a set S , there are many extreme rays \mathbf{r} satisfying Case 3, and giving rise to sets V_i . For a given set S the tightest valid inequality is obtained when the coefficient of x_i on the left hand side has the smallest value. This is obtained when $|V_i| = V_i^m = |a(i) \cap a(S)|$ (in other words V_i^m represents the number of neighbors of node i in the original graph G that are not in the set S). After removing dominated inequalities, the projection of P_d onto the x space is:

$$|a(S)| + (|S| - 1) + \sum_{i \in S} (g_i - V_i^m)x_i \geq \sum_{i \in S} g_i \quad \forall S \subseteq V \quad \& \quad S \text{ is connected} \quad (2.36)$$

$$0 \leq x_i \leq 1 \quad \forall i \in V. \quad (2.37)$$

Note we have rearranged inequality (2.36) to make its meaning clearer. Intuitively, given a connected component S , its corresponding inequality (2.36) can be interpreted as follows. The first two terms are the total number of edges associated with nodes in S . They represent the maximum amount of influence that come from the arcs ($|a(S)|$ arcs from outside of S and $|S| - 1$ internal to S). The right hand side is the total amount of influence required for this connected component S . Thus, the third term accounts for the fact that if a node i in S is selected, its threshold (i.e., g_i) is satisfied and those edges from $a(S)$ to node i cannot be used to satisfy the requirement of the other nodes in $S \setminus \{i\}$.

To illustrate inequality (2.36), we use the instance in Figure 2.2. Recall the LP relaxation of Ack has a fractional optimal solution with $x_1 = 1$, $x_2 = 0.5$ and all other $x_i = 0$. However, applying inequality (2.36) with $S = \{2, 5, 6\}$ gives $1 + 2 + (2 - 1)x_2 + (1 - 0)x_5 + (1 - 0)x_6 \geq 2 + 1 + 1$. Simplification gives $x_2 + x_5 + x_6 \geq 1$ which is violated by the fractional optimal solution.

Proposition 2.3. *The valid inequalities (2.36) can be separated in $O(|V|^3)$ time.*

Proof. Proof of Proposition 2.3. When the size of S is h , the separation procedure of inequalities (2.36) can be stated as the following optimization problem: Given a tree $G = (V, E)$ and an integer $h \leq |V|$, each node i in V has a weight which takes value from $\{c_{ij} : j = 0, 1, \dots, \deg(i)\}$ based on the number of neighbors not in the selected component, denoted by j . The goal is to find a connected component S with h nodes whose weight $\sum_{i \in S} c_{ij}$ is the minimum.

In Blum [2007], a dynamic programming algorithm is proposed for a generalized k -minimum spanning (k -MST) problem when the given graph is a tree. Its definition is as follows: Given a tree $G_B = (V_B, E_B)$ and an integer $k < |V_B|$, each node i in V_B has a weight w_i and each edge in E_B has a weight e_{ij} , the goal is to find a subtree T of k edges whose weight $\sum_{i \in T} w_i + \sum_{(i,j) \in T} e_{ij}$ is the minimum.

Given an instance of the separation problem with the cardinality of S specified to be h , we show that it is equivalent to the generalized k -MST problem on trees. From the input graph G , we transform it into graph G_t which was used to derive the extended formulation. Thus, $G_t = (V \cup D, E_t)$. Recall that we have a fractional solution \mathbf{x} and $\deg(i)$ denotes the degree of node i . For all i in V , set its weight $w_i = (1 - x_i)(\deg(i) - g_i)$. Then, for all d in D , set its weight $w_d = 0$. Lastly, for each (i, d) in E_t , we have edge weight $e_{id} = x_i - 1$. We also set $k = 2(h - 1)$ as the target cardinality. For a node $i \in V$ and a given tree T , let E_i^T be the number of its adjacent edges in T . The objective value of T is: $\sum_{i \in T \cap V} (1 - x_i)(\deg(i) - g_i) + \sum_{(i,d) \in T} (x_i - 1) = \sum_{i \in T \cap V} (\deg(i) - g_i - \deg(i)x_i + g_i x_i + E_i^T x_i - E_i^T) = \sum_{i \in T \cap V} \{(g_i + E_i^T - \deg(i))x_i - g_i + (\deg(i) - E_i^T)\}$. Let \bar{T} denote $T \cap V$. In the original graph $\deg(i) - E_i^T$ is identical to V_i^m . If we consider the nodes in \bar{T} , $\sum_{i \in \bar{T}} V_i^m$ is equal to $|a(\bar{T})|$; the cardinality of the set of neighbors of \bar{T} in the original graph. Thus, the objective value is equal to $\sum_{i \in \bar{T}} (g_i - V_i^m)x_i - \sum_{i \in \bar{T}} g_i + |a(\bar{T})|$. This is exactly the constraint (2.36) with $S = \bar{T}$ after we move $\sum_{i \in S} g_i$ to the left hand side and move $|S| - 1$ to the right hand side. If the objective value is smaller than $1 - h$, we have found a violated inequality. Otherwise, there is no violated inequality for sets S with cardinality h .

The time complexity of Blum's algorithm is $O(k^2|V_B|)$. It returns the values of the

best l -cardinality trees in G_B for all l values in the range $0 \leq l \leq k$. Thus, we only need to run it once by setting $k = |V_B| - 1$ ($k = |V_B|$ is trivial). Hence, in the worst case, the time complexity is $O(|V|^3)$ because k is bounded by $|V_B|$ and $|V_B| = 2|V|$. \square

2.4 The WTSS Problem on Cycles

In this section, we study the WTSS problem on cycles. We present its polytope and give a polynomial time algorithm to solve the problem. We work on the original graph and provide a formulation in the space of the natural variables. Based on the idea of inequalities (2.36), we have the following formulation for cycles:

$$\text{(CIP}_{\text{wtss}}\text{)} \quad \text{Minimize} \quad \sum_{i \in V} b_i x_i \quad (2.38)$$

$$\text{Subject to} \quad |V| + \sum_{i \in V} g_i x_i \geq \sum_{i \in V} g_i \quad (2.39)$$

$$2 + (|S| - 1) + \sum_{i \in S} (g_i - V_i^m) x_i \geq \sum_{i \in S} g_i \quad \forall S \subset V \quad (2.40)$$

& S is connected

$$x_i \in \{0, 1\} \quad \forall i \in V \quad (2.41)$$

In a cycle, a node either requires one or two neighbors. We call a node i a 1-node if its g_i value is 1. Similarly, a node is called a 2-node if its g_i value is 2. Let G_1 be the set of 1-nodes and G_2 be the set of 2-nodes. For the constraint (2.39), moving the first term $|V|$ to the right hand side, we can rewrite it as $\sum_{i \in G_1} x_i + 2 \sum_{i \in G_2} x_i \geq |G_2|$. Then, increasing the coefficient of all 1-node variables from one to two obtains a valid inequality: $2 \sum_{i \in G_1} x_i + 2 \sum_{i \in G_2} x_i \geq |G_2|$. Dividing both sides by two gives $\sum_{i \in V} x_i \geq \frac{|G_2|}{2}$. Finally, the left hand side must be an integer, so, replacing the right hand side by its ceiling

value provides a lifted valid inequality: $\sum_{i \in V} x_i \geq \lceil \frac{|G_2|}{2} \rceil$. Furthermore, when $|G_2| = 0$, it is trivial to see that the optimal solution is to pick the node with the smallest cost. Thus, we have the following valid inequality (instead of (2.39)):

$$\sum_{i \in V} x_i \geq \max\{1, \lceil \frac{|G_2|}{2} \rceil\}. \quad (2.42)$$

Next, we start removing redundant constraints in (2.40).

Case 1: When S has at most one 2-node, moving $2 + (|S| - 1)$ to the right hand side makes the right hand side 0 or -1 because $\sum_{i \in S} g_i$ is $|S| + 1$ or $|S|$. Thus, it is dominated by the non-negativity constraints.

Case 2: In a cycle, given a set S , there are two nodes adjacent to nodes not in S . We call these two nodes as end-nodes. Without loss of generality, let nodes j and k be end-nodes and node j be a 1-node. Also, in S , node h is the node adjacent to j . So, based on S , we have $(g_k - 1)x_k + g_h x_h + \sum_{i \in S \setminus \{j, k, h\}} g_i x_i \geq \sum_{i \in S} g_i - |S| - 1$ which is the summation of $x_h \geq 0$ and the inequality based on $S \setminus \{j\}$: $(g_k - 1)x_k + (g_h - 1)x_h + \sum_{i \in S \setminus \{j, k, h\}} g_i x_i \geq \sum_{i \in S \setminus \{j\}} g_i - |S| = \sum_{i \in S} g_i - |S| - 1$ because $g_j = 1$. The latter inequality dominates the former one. In other words, if not both these two end nodes are 2-nodes, the inequality given by this set S is redundant.

Case 3: When S has at least three 2-nodes and the two end-nodes are 2-nodes. Let nodes j and k be end-nodes and node h be a 2-node in between j and k . We can break S into two segments: one contains the segment of nodes from node j to node h (denoted by S_j) and the other one contains the nodes from node h to node k (denoted by S_k). Then, based on S , we have $x_j + x_k + 2x_h + \sum_{i \in S \setminus \{j, k, h\}} g_i x_i \geq \sum_{i \in S} g_i - |S| - 1$. Also, S_j gives $x_j + x_h + \sum_{i \in S_j \setminus \{j, h\}} g_i x_i \geq \sum_{i \in S_j} g_i - |S_j| - 1$ and S_k gives $x_k + x_h + \sum_{i \in S_k \setminus \{k, h\}} g_i x_i \geq$

$\sum_{i \in S_k} g_i - |S_k| - 1$. Thus, the inequality based on S is redundant because it can be obtained by the summation of the two inequalities based on S_j and S_k .

Combining the above three cases, we conclude that constraint (2.40) can be simplified as the follows:

$$\sum_{i \in S^2} x_i \geq 1 \quad \forall S^2 \subset V \quad (2.43)$$

where S^2 is a segment of nodes which has exactly two 2-nodes as its end-nodes.

Theorem 2.3. *Inequalities (2.42), (2.43) and $0 \leq x_i \leq 1 \quad \forall i \in V$ form an integral polytope for the WTSS problem on cycles.*

Proof. Proof of Theorem 2.3. As $0 \leq x_i \leq 1$ are upper and lower bound constraints, from Proposition 2.1 in Nemhauser and Wolsey [1988, p. 540] it suffices to prove inequalities (2.42) and (2.43) together form a totally unimodular matrix (TUM). Notice, however, that the constraint matrix defined by inequalities (2.42) and (2.43) form a row interval matrix (i.e., the non-zero entries in each row are 1's that appear consecutively). Thus, from Corollary 2.10 in Nemhauser and Wolsey [1988, p. 544] it is a TUM. \square

The number of sets S^2 is equal to $|G_2|$, the number of 2-nodes. Thus, the number of constraints is $1 + |G_2| + 2|V| = O(|V|)$. Hence, we have a tight and compact formulation for the WTSS problem on cycles.

We can make use of Algorithm 1 to solve the WTSS problem on cycles. First, we conduct two pre-processing steps. For each $S^2 \subset V$, if it contains more than one 1-node, we can replace these 1-nodes by the one with the smallest cost among them. It is easy to see that in an optimal solution, at most one 1-node will be selected and it should be

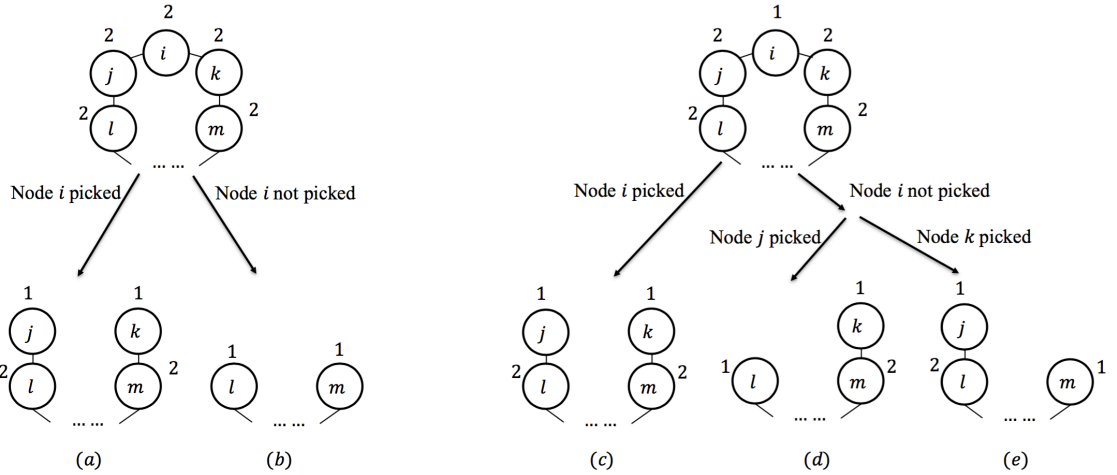


Figure 2.5: Illustration of the Cases for the Algorithm on Cycles

the one with the smallest cost. Now, each S^2 contains at most one 1-node. The second pre-processing step is that if this 1-node's cost is not the smallest one in this S^2 , we can delete this 1-node and connect the 2-nodes to each other. This is because, if this 1-node is selected, then, switching it with the 2-node with the smaller cost results in a solution no worse than the current one. Thus, at this point, each S^2 has at most one 1-node and if it has one, this 1-node has the strictly lowest cost among these three nodes.

After pre-processing, if there are no 1-nodes left in the graph, we can arbitrarily pick a node i . Without loss of generality, let those two nodes adjacent to node i be nodes j and k as shown in the top left of Figure 2.5. Then, we solve two possibilities for node i . One assumes that node i is selected and after accounting for the influence propagation from node i requires that the tree shown in Figure 2.5(a) be solved to ascertain the solution in this case. The other one assumes nodes j and k are selected at the same time and after accounting for the influence propagation from nodes j and k requires that the tree shown in Figure 2.5(b) be solved to ascertain the solution in this case.

If there are 1-nodes in the graph, we can arbitrarily pick a 1-node i . Without loss of

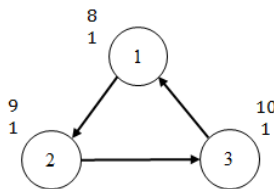


Figure 2.6: An example of cycle

generality, let those two closest 2-nodes adjacent to 1-node i be nodes j and k as shown in the top right of Figure 2.5. Then, we consider two possibilities for node i . First, we assume that node i is selected and after accounting for the influence propagation from node i it requires that the tree shown in Figure 2.5(c) to be solved to ascertain the solution in this case. If node i is not selected, then because node j and k are 2-nodes at least one of them must be selected in the target set. Thus we first consider the case that node j is selected when node i is not selected; and then we consider the case that node k is selected when node i is not selected. These require the solution of the trees shown in Figures 2.5(d) and 2.5(e) respectively.

Overall, we call Algorithm 1 at most three times and the pre-processing can be done in $O(|V|)$ time. Therefore, we have the following proposition:

Proposition 2.4. *The WTSS problem on cycles can be solved in $O(|V|)$ time.*

2.5 A Branch-and-Cut Approach for General Networks

The extended formulation BIP2.1 (discussed in Section 2.3) can be embedded into a larger model and applied to general graphs. The idea is based on the simple observation that the influence propagation process can be modeled as a directed acyclic graph (DAG).

We note that BIP2.1 is not a valid formulation on any graph that contains a cycle. Fig-

Figure 2.6 provides an example illustrating the problem caused by a cycle. In this figure, we have a directed influence cycle of node 1, 2 and 3. Each of them has threshold 1 and has one directed incoming arc. This means all 3 nodes become active by being influenced by the others. However, in this cycle, nothing indicates which node is the first active one to initialize the influence propagation process. In other words, the target set is empty. This is not a feasible solution.

We now discuss how BIP2.1 can be embedded in a bigger model to apply it to general graphs. We first show that BIP2.1 is a valid formulation for a DAG (with appropriate modifications since the original graph is a DAG) and its LP relaxation returns integral solutions on DAGs. First, we observe that the WTSS problem can be solved trivially on DAGs.

Proposition 2.5. *The WTSS problem on a DAG can be solved in $O(|V|)$ time.*

Proof. Proof of Proposition 2.5. Given a DAG, the optimal solution of the WTSS problem is trivial to find. For a node i , set $x_i = 1$ if the number of its incoming arcs is smaller than its threshold. Otherwise, set $x_i = 0$. Notice that, the WTSS must contain these nodes (set to $x_i = 1$) at a minimum. We now argue that this is a feasible solution. Consider the nodes of the DAG in the topological order. By design when we reach a node, its predecessors are all active. Therefore, if it has more than g_i incoming arcs, this node will become active. Otherwise, it has less than g_i incoming arcs, which means we have selected it in the target set. Since we did not select any nodes other than those which are selected by necessity, this solution is optimal. The time complexity is $O(|V|)$ because we need to scan through all nodes once. □

Given a DAG, we put this data into BIP2.1 in the following way. Without loss of generality, let node i and node j be nodes in the original graph and node d be the dummy node inserted to edge $\{i, j\}$. If in the DAG the edge $\{i, j\}$ is represented as an arc (j, i) , i.e. node j sends influence to node i , then, we set $y_{jd} = 1$ and $y_{dj} = 0$ (to ensure that in BIP2.1 node j cannot receive influence from node i) and leave y_{di} and y_{id} as decision variables in the model. Notice this implies constraint (2.6) can be removed from BIP2 as it is always satisfied. Let $p(i)$ denote the set of dummy nodes which are adjacent to node i while considering the incoming arc (j, i) in the DAG. Then, the linear relaxation of BIP2.1 is the following linear program that we refer to as LPDAG.

$$\text{(LPDAG) Minimize} \quad \sum_{i \in V} b_i x_i \quad (2.44)$$

$$\text{Subject to } (w_{id}) \quad x_i \leq y_{id} \quad \forall i \in V, d \in p(i) \quad (2.45)$$

$$(z_{id}) \quad y_{id} + y_{di} = 1 \quad \forall i \in V, d \in p(i) \quad (2.46)$$

$$(v_i) \quad \sum_{d \in p(i)} y_{di} + g_i x_i = g_i \quad \forall i \in V \quad (2.47)$$

$$x_i, y_{id}, y_{di} \geq 0 \quad \forall i \in V, d \in p(i) \quad (2.48)$$

In the dual to LPDAG (which we refer to as DLPDAG) we have w_{id} , z_{id} and v_i as dual variables for constraint sets (2.45), (2.46), and (2.47) respectively. DLPDAG can be written as follows:

$$\text{Maximize} \quad \sum_{i \in V} g_i v_i + \sum_{i \in V} \sum_{d \in p(i)} z_{id} \quad (2.49)$$

$$\text{Subject to } (y_{id}) \quad w_{id} + z_{id} \leq 0 \quad \forall i \in V, d \in p(i) \quad (2.50)$$

$$(y_{di}) \quad v_i + z_{id} \leq 0 \quad \forall i \in V, d \in p(i) \quad (2.51)$$

$$(x_i) \quad g_i v_i - \sum_{d \in p(i)} w_{id} \leq b_i \quad \forall i \in V \quad (2.52)$$

$$w_{id} \geq 0 \quad \forall i \in V, d \in p(i). \quad (2.53)$$

Theorem 2.4. *LPDAG has an optimal solution with x and y variables binary.*

Proof. Proof of Theorem 2.4. Proposition 2.5 provides a optimal target set which can be converted to a feasible solution to LPDAG (in a similar fashion to how a target set for a tree is converted to a feasible solution to BIP2.1 in Proposition 3.3). Let S be the set of selected nodes. We will now construct a dual feasible solution to DLPDAG that satisfies the complementary slackness(CS) conditions. The CS conditions of LPDAG and DLPDAG are as follows:

$$(x_i - y_{id})w_{id} = 0 \quad \forall i \in V, d \in p(i) \quad (2.54)$$

$$(b_i - g_i v_i + \sum_{d \in p(i)} w_{id})x_i = 0 \quad \forall i \in V \quad (2.55)$$

$$(-v_i - z_{id})y_{di} = 0 \quad \forall i \in V, d \in p(i) \quad (2.56)$$

$$(-w_{id} - z_{id})y_{id} = 0 \quad \forall i \in V, d \in p(i) \quad (2.57)$$

For a node $i \in S$, $x_i = 1$. For these nodes $i \in S$ we also have $y_{id} = 1$ and $y_{di} = 0$ for all d in $p(i)$. For $y_{id} = 1$, constraint (2.50) should be binding. So, we need $w_{id} = -z_{id}$. For $y_{di} = 0$, constraint (2.51) should be satisfied. So, $v_i \leq -z_{id}$ must be satisfied. Let $v_i = -z_{id}$, then, we have $w_{id} = v_i$. Because $x_i = 1$ and condition (2.55), constraint (2.52) must be binding. So, we need $g_i v_i - \sum_{d \in p(i)} w_{id} = b_i$. Thus, $g_i v_i - \sum_{d \in p(i)} v_i = b_i$ by substituting w_{id} by v_i . Hence, we set $v_i = w_{id} = \frac{b_i}{g_i - |p(i)|}$ and $z_{id} = -\frac{b_i}{g_i - |p(i)|}$. Furthermore, $w_{id} = \frac{b_i}{g_i - |p(i)|}$ has a positive value because $|p(i)|$ is strictly smaller than g_i . Otherwise, node i would not be selected. Thus by design this choice of dual variables

satisfies dual feasibility and the complementary slackness conditions.

For a node $i \in V \setminus S$, $x_i = 0$. We set all dual variables associated with node $i \in V \setminus S$ to zero, i.e., $v_i = w_{id} = z_{id} = 0$ for all $d \in p(i)$. Then, it is easy to see this choice of dual variables are feasible. Further conditions (2.54), (2.56), and (2.57) are always satisfied since the dual variables are zero, while condition (2.55) is satisfied since $x_i = 0$ (or all the complementary slackness conditions are satisfied).

Notice with this choice of dual variables, the dual objective value is

$$\sum_{i \in S} (g_i v_i + \sum_{d \in p(i)} z_{id}) = \sum_{i \in S} (g_i v_i - \sum_{d \in p(i)} w_{id}) = \sum_{i \in S} b_i.$$

This proves integrality of the x variables.

We now show all extreme points of LPDAG have y variables binary. Assume this is not true and there is an extreme point $(\mathbf{x}^*, \mathbf{y}^*)$ where \mathbf{y}^* is fractional. Observe when $x_i^* = 1$, $y_{id}^* = 1$ and $y_{di}^* = 0$ for all $d \in p(i)$ due to constraints (2.45) and (2.46). Since the x variables are binary, this implies fractional y values must correspond to arcs adjacent to a node i which is not in the target set. Because constraint (2.47) is an equation and g_i only takes integer value, there are at least two fractional incoming arcs for node i . Let (j, i) and (k, i) be these two incoming arcs and they have value y_{ji}^* and y_{ki}^* , respectively. Let ϵ be a very small positive value such that $0 \leq y_{ji}^* \pm \epsilon \leq 1$ and $0 \leq y_{ki}^* \pm \epsilon \leq 1$. Then, we can construct two feasible solutions in the following way: First, both of them have same values as $(\mathbf{x}^*, \mathbf{y}^*)$ except these four variables y_{ij} , y_{ji} , y_{ki} and y_{ik} . Then, one has $y_{ij} = y_{ij}^* + \epsilon$, $y_{ji} = y_{ji}^* - \epsilon$, $y_{ik} = y_{ik}^* - \epsilon$ and $y_{ki} = y_{ki}^* + \epsilon$. The other one has $y_{ij} = y_{ij}^* - \epsilon$, $y_{ji} = y_{ji}^* + \epsilon$, $y_{ik} = y_{ik}^* + \epsilon$ and $y_{ki} = y_{ki}^* - \epsilon$. Therefore, $(\mathbf{x}^*, \mathbf{y}^*)$ is not an extreme point

because it is a convex combination of these two constructed feasible solutions. \square

Our model for general graphs will introduce additional variables and constraints to model the fact that the influence propagation network implied by a feasible solution to the WTSS problem should be acyclic. In this model, we introduce a new variable set, h_{ij} and h_{ji} for all $\{i, j\}$ in E (these are defined on the original graph). If $h_{ij} = 1$, it means node i gives influence to node j . Otherwise, $h_{ij} = 0$. A new constraint set which ensures that the directed graph formed by \mathbf{h} , denoted by $G(\mathbf{h})$, has to be a DAG is needed as well.

With this in hand, we have the following formulation that we refer to as BIP2.2:

$$(BIP2.2) \text{ Minimize } \sum_{i \in V} b_i x_i \quad (2.58)$$

$$\text{Subject to } (2.6), (2.7), (2.8),$$

$$(2.9), (2.10), (2.11)$$

$$\sum_{(i,j) \in C} h_{ij} \leq |C| - 1 \quad \forall \text{dicycles } C \text{ in } G(\mathbf{h}) \quad (2.59)$$

$$h_{ij} + h_{ji} = 1 \quad \forall \{i, j\} \in E \quad (2.60)$$

$$h_{ij} \leq y_{id}, \quad h_{ji} \leq y_{jd} \quad \forall \{i, j\} \in E \& (i, d), (j, d) \in E_t \quad (2.61)$$

$$h_{ij}, h_{ji} \in \{0, 1\} \quad \forall \{i, j\} \in E. \quad (2.62)$$

The objective function is the same as before. Constraint set (2.59) called k -dicycle inequalities say that $G(\mathbf{h})$ must be a DAG. Constraint set (2.60) ensure that influence must go in one direction between two nodes in the original graph. Because h variables are defined on the original graph G and y variables are on the transformed graph G_t , we need constraint set (2.61) to serve as linking constraints which synchronize the influence propagation process between these two graphs (notice when $h_{ij} = 1, y_{id} = 1$; and when $h_{ji} = 1,$

$y_{jd} = 1$). Note that this is how we obtained LPDAG earlier. Constraint set (2.62) enforce the binary constraint on h variables.

To solve the LP relaxation of BIP2.2 (which we refer to as LP2.2) it is necessary to solve the separation problem for the exponential set of k -dicycle inequalities (2.59). Grötschel et al. [1985] present a separation procedure for k -dicycle inequalities which is based on the shortest path algorithm. We implement this separation procedure in our approach. Further, we make an important observation that in BIP2.2 it suffices to define the h variables as binary (i.e., integrality can be relaxed on the x and y variables). This can be helpful in terms of branching.

Corollary 2.1. *BIP2.2 only needs binary constraint on the h variables.*

Proof. Proof of Corollary 2.1. Once the h variables are binary and satisfy constraints (2.59) and (2.60), we have a DAG and the integrality of x and y follow as a direct consequence of Theorem 2.4. □

We take advantage of Corollary 2.1 in implementing a branching rule. In BIP2.2 we give branching priority to the h variables. We call this the H -branching rule.

We note that it is possible to derive a formulation without the h variables and work directly with the x and y variables (i.e., a formulation that works on the transformed graph directly). This formulation would simply add k -dicycle constraints defined on the transformed graph (i.e., with the y variables) to BIP2.1. However, there are three reasons for focusing (working with) on BIP2.2. First, in terms of its LP relaxation, it is easy to show that any feasible solution to the LP relaxation of BIP2.2 is feasible to the LP relaxation of this alternate model (showing that in terms of LP relaxations it is no worse than the alter-

nate model). Second, BIP2.2 requires fewer binary variables (which makes a significant difference in branch-and-cut). Lastly, with the help of h variables, BIP2.2 has a much smaller supporting graph for the separation procedure which saves significantly on time (in fact solving the LP relaxation of BIP2.2 is 10 times faster than this alternate model).

We now discuss some additional features of the branch-and-cut procedure. In the extended formulation BIP2.2, we use additional variables to model the influence propagation process. Thus, the MIP search process spends a significant amount of time on these variables because of the binary requirement on these variables. However, we actually are only interested in the target set (i.e., the natural node (x) variables). Consequently, for a given \mathbf{x} , we consider the integral portion of the solution (i.e., those nodes with $x_i = 1$) and determine the set of nodes that can be activated using this target set. We call this process as Feasibility Lift. If all nodes are activated by these selected nodes, we have obtained an integral feasible solution and the current node of the branch-and-bound tree can be fathomed. Otherwise, we can continue the branch-and-cut search as usual. One advantage of Feasibility Lift is that we can focus the separation procedure on the subgraph induced by the inactive nodes (because there must be some cycles to help them satisfy their thresholds). We refer to this subgraph as the Inactive Induced Graph. In this way, we have a much smaller supporting graph and can add fewer violated inequalities in the branch-and-cut procedure.

Lastly, we realize that the separation procedure is expensive. Therefore, we modified the branch-and-cut procedure in the following way: In the root node, we do our best to find violated inequalities in order to achieve a better dual bound, (i.e., we focus on the cutting plane method). However, once we enter the branching phrase, the separation

ID	OPT	Ack2	Ack2/OPT	LP2.1	LP2.1/OPT	Greedy	Greedy/OPT
1	1438	561.00	0.39	1282.50	0.89	1651	1.15
2	13171	7148.67	0.54	13022.00	0.99	17086	1.30
3	19528	11524.38	0.59	19476.00	1.00	25467	1.30
4	7598	3828.08	0.50	7429.00	0.98	9712	1.28
5	13236	7354.53	0.56	13084.50	0.99	17604	1.33
6	14591	8622.57	0.59	14454.50	0.99	20070	1.38
7	1709	569.17	0.33	1456.00	0.85	2024	1.18
8	7214	3523.25	0.49	7011.38	0.97	9263	1.28
9	7115	3761.00	0.53	6960.75	0.98	9689	1.36
10	7110	3527.25	0.50	6912.33	0.97	8917	1.25

Table 2.1: Comparisons of LP relaxations of Ack2 and BIP2.2 for the WTSS problem, a Greedy Heuristic, and Optimal Integer Solutions for 1000-Node Instances.

procedure is only invoked when the integrality constraints are satisfied at a node.

2.6 Computational Experience

We now discuss our computational experience with the branch-and-cut approach. We generated networks using the method proposed by Watts and Strogatz [1998] in their pioneering work on social network analytics. As indicated in the Stanford large network dataset collection [Leskovec, 2011], real social networks are sparse. We took this into account as follows. For sparsity, we generated network with average degree number 4. We chose the rewiring probability p as 0.3 (loosely, it is the probability that an edge is reconnected to a uniformly chosen node after initializing a ring with pre-specified average degree), because Watts and Strogatz [1998] showed this corresponds most closely to the social networks they studied. We randomly generated node type g_i from a discrete uniform distribution between $[1, |a(i)|]$ and cost b_i from a discrete uniform distribution between $[1, 100]$. We used CPLEX 12.6 with Python API and ran our tests on a machine with the following specifications: Intel i5 3.40GHz, 24 GB ram, Ubuntu 14.04.

In our first set of experiments we study the strength of the LP relaxation of BIP2.2 (LP2.2). For this, ten 1000-node instances are generated. The objective value of LP2 is compared against the LP relaxation of Ack augmented with k -dicycle inequalities (we refer to this as Ack2).¹ Table 2.1 shows the results. The first column provides the instance number. ``OPT" is the value of the optimal solution that is obtained by solving BIP2.2 (with the best setting for branch-and-cut which will be discussed later in this section). In terms of running time, each instance only takes several seconds (no more than five) with the best setting. ``Ack2" shows the LP relaxation value for Ackerman et al.'s formulation. ``Ack2/Opt" compares this value to the optimal objective value and gives the strength of Ackerman et al.'s formulation. Similarly, ``LP2" and ``LP2/OPT" contain the LP relaxation value for BIP2.2 and compare it to the optimal objective value respectively. Notice LP2 is a much stronger LP relaxation for the WTSS problem. On average, LP2 is able to improve the LP relaxation's objective value by 96%. The minimum improvement and the maximum improvement are 68% and 156%, respectively. Furthermore, the remaining gap of LP2 (i.e., distance to optimal solution) is within 4% over these ten instances on average, while it is 50% for the Ack2 formulation. We can make at least two key observations. One important observation is that the bigger the remaining gap, the bigger the improvement compared to the Ack2 formulation. Instance 1 and 7 have the biggest gap, but they also have the biggest improvement. Another observation is that for most instances, the remaining gap is very small (i.e., LP/OPT is close to 1) which can help the branch-and-cut search significantly in general. ``Greedy" is the objective value of the initial solution found by a greedy heuristic which repeatedly selects the smallest cost node and adds it to the tar-

¹In a slightly different fashion Ack2 is the model proposed in Ackerman et al. [2010] for general graphs.

get set until a solution is found. ``Greedy/OPT" assesses the quality of this greedy initial solution. On average, it is about 28% off the optimal objective value, although the worst one is about 40% off. With regards to running time the LP relaxation of Ack2 needs 0.19 seconds on average while the LP relaxation of BIP2.2 takes 0.31 seconds on average. In summary, although BIP2.2 takes a little more time than Ack2 it is able to provide a much better dual bound.

In order to test different features of our branch-and-cut procedure and to demonstrate the benefit of Feasibility Lift and Inactive Induced Subgraph, we conducted a set of experiments with the branch-and-cut procedure on ten instances with 10,000 nodes and 20,000 edges (after bidirecting these edges we have 40,000 arcs). We construct an initial feasible solution using the greedy algorithm. All runs are capped to a 5-minute time limit.

ID	Optimality Gap (%)			Avg Gap (%)	Max Gap (%)	Summary					
	BIP2.2	All	w.o. H -BR			BIP2.2	All	w.o. H -BR	# of Improved Solutions	# of Best Feasible Solutions	Avg Dual Bound
1	29.11	1.42	1.19			30.38	1.98				1.27
2	31.47	2.29	0.91			31.80	3.94				3.51
3	31.67	0.63	0.46	Avg User Sep. Time (Sec.)		224.62	38.61				40.29
4	30.25	1.71	0.46	Min User Sep. Time (Sec.)		184.98	30.72				29.66
5	29.45	1.20	0.96	Max User Sep. Time (Sec.)		244.31	59.16				52.42
6	30.94	1.76	1.06	Avg Objective Value		76279.30	54048.60				53647.80
7	31.80	3.94	1.13	# of Improved Solutions		4	10				10
8	27.47	1.18	1.58	# of Best Feasible Solutions		0	1				9
9	30.31	2.00	1.39	Avg Dual Bound		53092.38	52988.67				52975.01
10	31.34	3.64	3.51	Best Dual Bound		7	3				0

Table 2.2: Comparison of Brand-and-Cut Settings for the WTSS problem on 10000-Node Instances.

Table 2.2 contains the results for 10000-node instances. We have three settings: ``BIP2.2" is the straight implementation of BIP2.2 formulation with the H -branching rule and k -dicycle separation. ``All" adds Feasibility Lift along with the Inactive Induced Graph separation to the ``BIP2.2" setting. Although we can relax integrality on the x and y variables, they are kept as binary in all settings to get a better dual bound at the root node because we allow CPLEX to add its own cuts. In order to identify the effect of the H -branching rule, ``w.o. H -BR" excludes the H -branching rule from ``All". In the left part of this table, we give detailed information regarding the optimality gap for these 10 instances. In the right part of this table, summary measures (averaged over the 10 instances) can be found. Although none of the instances can be solved in five minutes, the average gap is reduced from 30.38% to 1.98% and the maximum gap is reduced from 31.80% to 3.94% with the help of Feasibility Lift and Inactive Induced Graph. Furthermore, when we remove the H -branching rule and let CPLEX make its own choices, those two measures are reduced to 1.27% and 3.51% respectively. We observe that CPLEX occasionally deviates from the H -branching rule and branches on x variables. Although, we note that instance 8 performs better with the H -branching rule. Next, we show average, minimum and maximum user separation time (in seconds) which is the time used for separating k -dicycle inequalities. Incorporating Inactive Induced Graph significantly reduces the separation time from about 225 seconds to 39 seconds on average. The same story can be found for minimum and maximum user separation time. ``Avg Objective Value" shows the average objective function value of those best feasible solutions found in each setting. ``# of Improved Solutions" is the number of instances that find a better solution compared to the initial solution. Without Feasibility Lift, only four instances can find a

better solution. With it, all ten instances can improve their solution. The solution quality has been improved significantly from about 76,000 to 54,000. ``# of Best Feasible Solutions" is the number of instances that find the best solution among these three settings. The third setting (w.o. H-BR) can obtain nine out of ten best feasible solutions, while the ``All" setting is able to find one of them. Lastly, we show the average dual bound and the number of best dual bound in each setting. We can see that separating on the Inactive Induced Graph only sacrifices the dual bound for a very small amount (about 0.2%), but it saves a large amount of time in the search procedure. Based on the above experiment, we conclude that the setting ``w.o. H-BR" (which incorporates Feasibility Lift and Inactive Induced Subgraph) has the best performance generally.

	Nodes	Avg Degree	1	2	3	4	5	6	7	8	9	10	Avg	Max
Gap (%)	10000	4	0.04	0.06	0.00	0.00	0.00	0.23	0.17	0.11	0.05	0.17	0.08	0.23
	5000	8	2.81	10.46	19.25	23.42	5.71	17.20	3.52	9.24	6.23	25.63	12.35	25.63
	2500	16	14.14	4.56	9.24	8.37	14.72	42.04	9.91	32.99	7.20	14.65	15.78	42.04
Imp. (%)	10000	4	30.23	31.15	31.34	30.01	31.53	30.46	31.16	30.10	30.05	31.45	30.75	31.53
	5000	8	37.72	32.77	27.19	24.07	37.86	28.45	37.11	34.31	39.78	22.13	32.14	39.78
	2500	16	39.80	40.60	43.37	38.59	40.00	14.90	43.42	23.08	46.88	38.86	36.95	46.88

Table 2.3: Analyzing the Effect of Graph Density on the Branch-and-Cut Procedure for the WTSS problem.

Edges	# of Instances	Percentage of Total Cost		Random Adoption Percentage		Greedy Adoption Percentage	
		Avg (%)	Max (%)	Avg (%)	Max (%)	Avg (%)	Max (%)
400	10	10.71	11.67	59.15	68	88.25	94
600	10	7.83	9.95	58.25	85	91.65	94.5
800	10	5.23	7.07	64.20	82	91.45	97
1000	10	5.12	6.85	65.30	79	90.75	98
1200	10	4.99	7.57	67.15	84.5	90.10	95.5

Table 2.4: Heuristic for the WTSS problem Fixed Budget on 200-Node Instances.

Next, we investigate the role graph density plays in the difficulty solving a problem. For this, in addition to the 10,000 node instances we generated for the previous experiment; we generate ten instances with 5,000 nodes and average degree number 8 and ten instances with 2,500 nodes and average degree number 16. They all have 20,000 edges with differing levels of graph density. We give 5 minutes, 1-hour, and 2-hour time limits to the 10,000-node, 5,000-node, and 2,500-node instances respectively (as will be evident graph density plays a huge role in terms of the running time). We use the "w.o. *H*-BR" setting, but add 3-cycles inequalities of 3-cliques a priori (these are easy to add at the outset). The results are shown in Table 2.3. "Gap" is the optimality gap (at termination) and "Imp." is the percentage improvement of the best found solution compared to the initial greedy solution. We also show their average and maximum values in the last two columns. The denser the graph, the bigger the gap. The average gap increases from 0.08% to 12.35% when the number of nodes decreases from 10,000 to 5,000. Then, it increases to 15.78% when the number of nodes becomes 2,500. However, our approach is able to improve the solution quality compared to the initial greedy solution. On average, we can improve the initial solution by about 30%. Given that the time limit is increased considerably, the WTSS problem becomes much harder when the graph density increases. We should note that Instances 6 and 8 with 2500 nodes are particularly bad for the branch-and-cut procedure. With a different initial heuristic procedure (based on a heuristic proposed by Shakarian et al. [2013] that we will discuss in the next set of experiments) the branch-and-cut procedure obtains gaps of 18.39% and 24.58% respectively for these two instances. Unfortunately, this alternate initial heuristic procedure yields significantly worse results on all the other instances.

Finally, we focus on a set of experiments on 200 nodes where we evaluate the cost of the optimal solution against differing measures. We varied graph density and created instances where the number of edges takes values $\{400, 600, 800, 1000, 1200\}$. For each setting, 10 instances are generated and there are 50 instances in total. These instances are solved to optimality. Table 2.4 displays the results. First, we wanted to see what percentage of the total cost results in 100% adoption in the WTSS problem. We compare the optimal objective value to the sum of all node costs, $\sum_{i \in V} b_i$. The results are shown in the "Percentage of Total Cost" column of Table 2.4. We can see that on average, about 11% of the total cost is needed for graphs with 400 edges, and it decreases to about 5% for graphs with 1200 edges. Next, we used two different measures to assess the quality of the solutions obtained by the branch-and-cut approach. The first measure tries to evaluate the benefit of optimal targeting by using the optimal objective value as a budget (upper bound) for a heuristic target set and evaluate the fraction of the graph that is influenced by this heuristic target set. We consider two budget constrained heuristic target sets. The first evaluates the benefit of optimal targeting against a random solution, and the second against a greedy solution. In both cases the budget is exceeded but the set of nodes must satisfy the following property: if any node is removed from this set, then the budget is respected. For the budget constrained target set that is generated randomly, we find 100 random solutions and select the one with the highest adoption (fraction of graph influenced). In the greedy heuristic we greedily pick the node with the smallest cost until the total weight of selected nodes violates the bound. From the column "Random" in Table 2.4, we can see that the random selection strategy does not perform well. The average adoption rate increases from 59% to 67% when the number of edges increases from 400 to 1200. But

		Greedy to Opt Gap		Shakarian to Opt Gap	
Edges	# of Instances	Avg (%)	Max (%)	Avg (%)	Max (%)
400	10	46.38	52.74	13.94	27.94
600	10	49.64	60.33	33.98	105.03
800	10	55.34	100.83	39.58	73.31
1000	10	56.09	80.35	33.83	57.24
1200	10	67.35	92.02	34.28	125.85

Table 2.5: Optimality Gaps for Greedy and Shakarian Heuristic on the WTSS problem for 100% Adoption on 200-Node Instances.

even the best one among the 5,000 samples only has 85% adoption. On the other hand the greedy strategy has much better performance. Column ``Greedy" shows that on average the adoption rate is around 90% with the best at 98%.

The second measure evaluates the cost of optimal targeting by comparing the optimal solution against heuristic solutions that have 100% adoption. We consider two heuristics. The first is our initial heuristic which is a greedy algorithm. The second is based on a heuristic proposed by Shakarian et al. [2013] for the target set selection problem. In this heuristic, a measure $dist$ equal to $deg(i) - g_i$ is computed for each node i in V . At each iteration, the heuristic picks the node with the smallest nonnegative $dist$ and removes it. Then, the graph and $dist$ are updated accordingly. Once all nodes have negative $dist$, the remaining nodes in the network are used as the target set. Since the heuristic does not consider weights, to adopt it for the WTSS problem, we use the node weights to break ties (by choosing the node with the higher cost) when picking the node with the smallest non-negative $dist$. Table 2.5 summarizes the results providing the gap between the heuristic solution and the optimal solution value (the gap is calculated as the difference between the heuristic solution and the optimal solution divided by the value of the optimal solution). The column ``Greedy to Opt Gap" shows that the solutions obtained by the greedy

strategy are about 50% greater than the cost of the optimal ones. Furthermore, this gap increases with graph density. The column "Shakarian to Opt Gap" shows that on average the Shakarian heuristic performs better, although its maximum gap values can be significantly greater than the greedy heuristic. The average gap for the Shakarian heuristic is 14% when the number of edges is 400 and is greater than 30% for the other test sets.

2.7 Conclusions

In this chapter we studied the weighted version of the TSS problem. We showed that the WTSS problem can be solved polynomially (via dynamic programming) when the underlying graph is a tree. Our dynamic programming algorithm generalizes Chen [2009]'s algorithm and runs in $O(|V|)$ time. More importantly, we present a tight and compact extended formulation whose LP relaxation provides integral solutions for the WTSS problem on trees. We also project this formulation onto the space of the natural node variable space and thus find the polytope of the WTSS problem on trees. Building upon the result for trees we first derive the polytope of the WTSS problem on cycles. Next, by observing the influence propagation network forms a DAG we show how the extended formulation for trees can be embedded into a formulation on general graphs, where an additional exponentially sized set of k -dicycle constraints are added. We design and implement a branch-and-cut approach for the WTSS problem on general graphs. Our computational results show that our formulation improves significantly upon a previous formulation for the problem due to Ackerman et al. [2010]. We are able to find high quality solutions for large random graph instances within a reasonable amount of time.

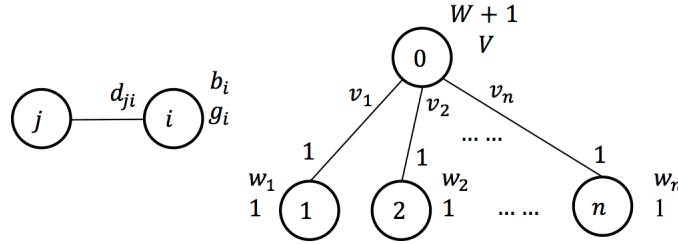


Figure 2.7: Transforming a 0-1 knapsack problem to the WTSS problem with unequal influence on stars

Compared to the heuristic solutions our branch-and-cut procedure provides significantly better upper bounds.

One generalization to the WTSS problem has unequal influence from neighbors. A natural question is whether the tree formulation and algorithm applies to this problem. Unfortunately, as we show in Theorem 2.5, this problem is even NP-complete on stars (i.e., a tree where all nodes are leaves other than a single central node)---contradicting a claim by Ben-Zwi et al. [2011]. Deriving strong formulations for other closely related influence maximization problems is an avenue of our current and future research.

Theorem 2.5. *The WTSS problem with unequal influence is NP-complete on stars.*

Proof. Proof of Theorem 2.5. We prove that when a node i receives unequal influence from its neighbors, the WTSS problem is NP-complete on stars. In the propagation progress in the unequal influence case, the incoming influence of node i is calculated as $\sum_{j \in s(i)} d_{ji}$ where d_{ji} is the amount of influence from node j and $s(i)$ is the set of node i 's active neighbors. Furthermore, each node's threshold value g_i can take a value between 1 and $\sum_{j \in a(i)} d_{ji}$.

The decision version of the 0-1 knapsack problem is defined as follows: Given a set of n items numbered from 1 up to n , each item i with a weight w_i and a value v_i , along

with a maximum weight capacity W , can we select a subset of these items such that a value of at least V will be achieved without exceeding the weight W ?

We construct a star network from this 0-1 knapsack problem. For each item i , we put a node i in the graph as a leaf node. After that, we add one extra node and label it as node 0 which is the central node. All leaf nodes connect to the central node but do not connect to each other. Thus, there are $n + 1$ nodes numbered from 0 up to n in the graph. Then, for each leaf node i , we have $g_i = 1$, $d_{0i} = 1$ and $b_i = w_i$. For the central node 0, we have $g_0 = V$, $d_{i0} = v_i$ for all $i \in a(0)$ and $b_0 = W + 1$. The constructed star is shown in Figure 2.7. The decision question is: Can we select a subset of nodes without total cost exceeding W to activate the whole network? It is easy to see that we should never select the central node 0. So, it is equivalent to ask: Can we select a subset of leaf nodes such that the incoming influence of the central node is at least V and the total cost of those selected leaf nodes does not exceed W ? Therefore, if the answer is "Yes", those selected leaf nodes also solve the 0-1 knapsack problem. □

Chapter 3: Generalizations of the Dominating Set Problem on Social Networks

3.1 Introduction

The dominating set problem is a classic NP-complete problem and is defined as follows: given a graph $G = (V, E)$, we want to find a minimal subset D of V such that every node not in D is adjacent to at least one member of D . Recently, Wang et al. [2009] proposed a new variation of the dominating set problem in the online social network context. Later, Dinh et al. [2014] refined this definition and described two variants. Namely, the Positive Influence Dominating Set (PIDS) problem which requires every node not in D have at least ρ fraction of its neighbors in D , and the Total Positive Influence Dominating Set (TPIDS) problem which requires every node (including nodes in D) have at least ρ fraction of its neighbors in D .

In social networks, the PIDS and TPIDS problems can be applied in viral marketing, politics and can be used to deal with social problems, such as drinking, smoking and drug related problem. Identifying those most influential people can be of great advantages and lower the total cost. Wang et al. [2009] described an application of the PIDS (TPIDS) problem in alleviating college drinking problem. Many college campuses have

binge drinking problems where peer pressure plays a large role in pressuring students to binge drink. An intervention program is used to convert binge drinking students to abstainers. However, it is not realistic to assume that one has enough resources for all binge drinking students. The minimum cardinality PIDS (TPIDS) is a desired outcome in this circumstance. Another example relates to majority on a social network. Suppose we have two competing products and ρ is bigger than 0.5. If a PIDS of all the people has adopted one specific product, it is more likely that this product will be adopted by the whole social network in the end because every person not in this PIDS has more than half of his/her acquaintances who have adopted this product already. Finally, the PIDS problem can be interpreted as a WTSS problem where only one time period is allowed for the influence diffusion process. In Goel et al. [2015], they investigated the diffusion of nearly a billion news stories, videos, pictures, and petitions on the microblogging service Twitter. One important observation is that the vast majority of cascades (over 99%) are terminate within a single time period, supporting a deeper understanding of the PIDS problem.

Although the dominating set has been studied intensively, only limited literature is from the integer programming perspective. For the dominating set problem, Bouchakour et al. [2008] and Saxena [2004] independently presented its polytope for trees and cycles. Baiou and Barahona [2014] showed an extended formulation via facility location and proved that the projection of this formulation on the node space describes the dominating set polytope for cacti graphs. For the PIDS and TPIDS problems, researchers mainly focused on approximability and proved that they are both APX-hard in Wang et al. [2009, 2011], Dinh et al. [2014]. There is no previous literature on mathematical programming formulation for the PIDS and TPIDS problems.

3.1.1 Problem Definition

Deviating from previous literature, we study the PIDS and TPIDS problems with two additional considerations. First, a weight is introduced to each node. This weight models the fact that in real life it takes different effort levels to convince different people. Second, a node can require any positive number of neighbors to be in D (as opposed to a fixed number that is the same for all nodes in the graph). This reflects the situation where different people require different amounts of peer influence to adopt a behavior in practice. Formally, we define the PIDS problem as follows: given an undirected graph $G = (V, E)$, each node i in V has a weight, denoted by b_i , and a threshold requirement, denoted by g_i , taking value between 1 and its degree number, we seek a subset D of V such that a node i not in D is adjacent to at least g_i members of D and the total weight of the nodes in D , denoted by $\sum_{i \in D} b_i$, is minimized. The TPIDS problem is similar except we seek a subset D of V such that *every node* i in V is adjacent to at least g_i members of D .

3.1.2 Related Literature

The TPIDS problem with $\rho = 0.5$ was first introduced by Wang et al. [2009]. They also proposed an algorithm which iteratively adds a dominating set until a TPIDS is found. Wang et al. [2011] proved that the TPIDS problem is APX-hard and proposed a greedy algorithm with $O(|V|^3)$ time complexity and an approximation ratio of $H(\delta)$ where H is the harmonic function and δ is the maximum degree number in the given graph. Raei et al. [2012] proposed a new greedy algorithm which improves the time complexity to $O(|V|^2)$.

Wang et al. [2013] also presented a self-stabilizing algorithm with time complexity of $O(|V|^2)$. Zhang et al. [2012] studied the TPIDS problem in power-law graphs and proved that the greedy algorithm has a constant approximate ratio.

The PIDS problem with $\rho = 0.5$ was first considered by Zou et al. [2009] under a different name "Fast Information Propagation Problem". They proved it to be NP-hard. Zhu et al. [2010] further showed it to be APX-hard and described a greedy approximation algorithm with performance ratio $O(\ln \delta)$.

For both the PIDS and TPIDS problems, Cicalese et al. [2011] proved that they are APX-hard and cannot be approximated to within a factor of $c \ln |V|$ where c is a suitable constant and can be found by construction. They also showed that a greedy strategy has approximation ratio $O(\ln \delta)$ for both problems. Furthermore, a $O(|V|^2)$ dynamic programming algorithm is proposed when the given graphs are trees. Dinh et al. [2014] improved the inapproximability results for both problems. They showed both problems are hard to approximate within $\ln \delta - O(\ln \ln \delta)$. In power-law graphs, they showed that the greedy method targeting the highest degree vertices has a constant factor approximation ratio. They also presented an algorithm for trees which has time complexity with improved $O(|V|)$. Overall, all previous work on the PIDS and TPIDS problems has focused on their approximability.

The dominating set problem has a long history and a tremendous amount of literature (see Haynes et al. [1998a,b]). However, there is very limited literature from the integer programming perspective. Saxena [2004] presented the dominating set polytope for cycles. Bouchakour et al. [2008] described the polytope for a larger class of graphs that can be decomposed by one-cutsets into cycles. Both their formulations only use variables

associated with the nodes in the graph. Baïou and Barahona [2014] showed an extended formulation via facility location (it uses both node and arc variables) and proved that the projection of this formulation onto the node space describes the polytope for cactus graphs. In a cactus graph, each edge is contained in at most one cycle. For instance, every tree is a cactus graph. Cactus graphs also contain cycles and the class of graphs in Bouchakour et al. [2008].

For the dominating set problem on trees, the following two formulations provide integral solutions when their linear programming (LP) relaxations are solved. In other words, these two formulations are the strongest LP formulations for the dominating set problem on trees. First, the node variables formulation from Saxena [2004], Bouchakour et al. [2008] is as follows:

$$\text{Minimize} \quad \sum_{i \in V} x_i \quad (3.1)$$

$$\text{Subject To} \quad \sum_{j \in n(i)} x_j + x_i \geq 1 \quad \forall i \in V \quad (3.2)$$

$$x_i \in \{0, 1\} \quad \forall i \in V \quad (3.3)$$

where $n(i)$ denotes the neighbors of node i for a node i in V . We have decision variable x_i for a node i in V . If node i is selected in D , $x_i = 1$. Otherwise, $x_i = 0$. The objective function (3.1) is to minimize the size of D . The first constraint (3.2) says that either node i or one of its neighbor is in D . Constraint (3.3) is the binary constraint. This formulation is referred to as Saxena's formulation.

The extended formulation via facility location in Baïou and Barahona [2014] is given below:

$$\text{Minimize} \quad \sum_{i \in V} x_i \quad (3.4)$$

$$\text{Subject To} \quad \sum_{j \in n(i)} y_{ji} + x_i \geq 1 \quad \forall i \in V \quad (3.5)$$

$$x_i - y_{ij} \geq 0 \quad \forall i \in V, j \in n(i) \quad (3.6)$$

$$x_i \in \{0, 1\} \quad \forall i \in V \quad (3.7)$$

$$y_{ij} \in \{0, 1\} \quad \forall i \in V, j \in n(i) \quad (3.8)$$

Here, we have additional y_{ij} and y_{ji} variables for each edge $\{i, j\}$ in E . If $y_{ij} = 1$, it means node j is assigned to node i . Otherwise, it is 0. The objective function (3.4) is to minimize the size of D . The first constraint (3.5) says that either node i or one of its neighbor is in D . Constraint (3.6) means that if node i is selected, we can assign node j to it when node j is one of node i 's neighbors. Constraints (3.7) and (3.8) are the binary constraints. We refer to formulation as Baïou's formulation. Note that both the Saxena and Baïou formulations are valid on general graphs.

3.1.3 Our Contributions

First, we show that both the PIDS and the TPIDS problems on trees can be solved in $O(|V|)$ time. We use a dynamic programming approach where a tree is decomposed into subproblems. For each subproblem, we identify the most promising solution candidates for different situations. After all subproblems are examined, a backtracking procedure is used to determine the final solution.

More importantly, for the TPIDS problem, we show that the natural adaptations of these formulations in Bouchakour et al. [2008], Saxena [2004], Baïou and Barahona

[2014] are tight when the given graphs are trees. However, this natural adaptation does not work for the PIDS problem (i.e., the LP relaxations give fractional solutions on trees). For the PIDS problem on trees, a tight and compact extended formulation is presented. The key idea is similar to that in the WTSS problem. We enforce the observation that every node in D should only send out influence. If we enforce this observation directly, it causes a trouble in that two adjacent nodes cannot be in D at the same time; which is not a valid restriction. We work around this problem by applying the edge splitting idea that is introduced in Chapter 2 with the WTSS problem.

This leads to our third contribution. We focus on the PIDS problem and study its polytope. We project the extended formulation onto the natural node variables space and prove that it is stronger than the natural adaptations of those formulations in Bouchakour et al. [2008], Saxena [2004], Baïou and Barahona [2014]. From there, we derive a new set of valid inequalities for the PIDS problem and provide polynomial time separation procedure for it. Based on our computational experiment, our formulation provides much better linear relaxation bound. Our work is a building block for developing exact approaches for the PIDS and the TPIDS problems.

The rest of this chapter is organized as follows. Section 3.2 discusses the TPIDS problem. We present a linear time algorithm and show that the natural adaptations of those formulations in Bouchakour et al. [2008], Saxena [2004], Baïou and Barahona [2014] are tight for the TPIDS problem on trees. Section 3.3 focuses on the PIDS problem. We present a linear time algorithm and the tight and compact extended formulation for the PIDS problem on trees. Also, we project the extended formulation for the PIDS problem onto the natural node variable space which provides not only the complete description

of the polytope for the PIDS problem on trees, but also a novel formulation on general graphs. Finally, computational results are presented on general graphs. Section 3.4 gives some concluding remarks.

3.2 The TPIDS Problem

In this section, we present a linear time algorithm for the TPIDS problem on trees and show that the natural adaptations of the formulations in Bouchakour et al. [2008], Saxena [2004], Baïou and Barahona [2014] are tight for the TPIDS problem on trees.

3.2.1 Algorithm for the TPIDS Problem on Trees

We present an algorithm to solve the TPIDS problem on trees. In this method, we decompose the tree into subproblems. A subproblem is defined on a star network which has a central node and possibly many leaf nodes. Each subproblem is used to find the most promising candidates (at most two) and one of them will be part of the final solution of the tree. In this tree, each non-leaf node is a central node for a star network. By solving the subproblem, we have one candidate for the situation that the central node's parent node is selected and one candidate when its parent node is not selected. Next, the current star is compressed into one single leaf node with updated cost and threshold for its parent star network. This process is repeated until we are left with a single star. The last star should have the root node of the tree as the central node. After we exhaust all subproblems, a backtracking method is used to identify a final solution which combines the candidates of those star subproblems for the tree. The pseudocode of the proposed algorithm is shown

Algorithm 4 Algorithm for the TPIDS and the PIDS problems on trees

- 1: Arbitrarily pick a node as the root node of the tree
 - 2: Define the order of star problems based on the bottom-up traversal of the tree
 - 3: **for** each star subproblem **do**
 - 4: StarHandling
 - 5: **end for**
 - 6: SolutionBacktrack
-

in Algorithm 8.

For a star subproblem, we try to find the solution that satisfies the threshold requirement of the central node while the total cost is the minimum. Denote the central node by c and refer to this star as star c . Let the set of star c 's leaf nodes be $L(c)$. Then, we can select some of star c 's leaf nodes to satisfy the threshold requirement of the central node based on if its parent is selected or not. We define three kinds of sets among $L(c)$: the core set, the free set and the expensive set, denoted by $C(c)$, $F(c)$ and $E(c)$ respectively. The core set has nodes with threshold one including leaf nodes. The other two sets contain leaf nodes with threshold zero. However, nodes have zero cost in the free set while they have positive cost in the expensive set. Hence, nodes in $F(c)$ are always selected. After that, we only select nodes from $E(c)$ and $C(c)$ when it is necessary. Therefore, if the size of $F(c)$ is smaller than the threshold value of node c , we find the ascending order of the remaining nodes in $C(c) \cup E(c)$ based on their cost. Then, if the parent node is selected, the possible solution is the set of the first $g_c - |F(c)| - 1$ nodes in $C(c) \cup E(c)$, denoted by $S_{g_c-1}^c$. And, if its parent node is not selected, the possible solution is consist of the the first $g_c - |F(c)|$ nodes in $C(c) \cup E(c)$, denoted by $S_{g_c}^c$. If $|F(c)|$ is not smaller than the requirement number, we have $S_{g_c}^c = S_{g_c-1}^c = F(c)$. Thus, we have two situations for a star subproblem. One has parent node selected, one does not. For the last star, there is

only one situation because there is no parent node any more.

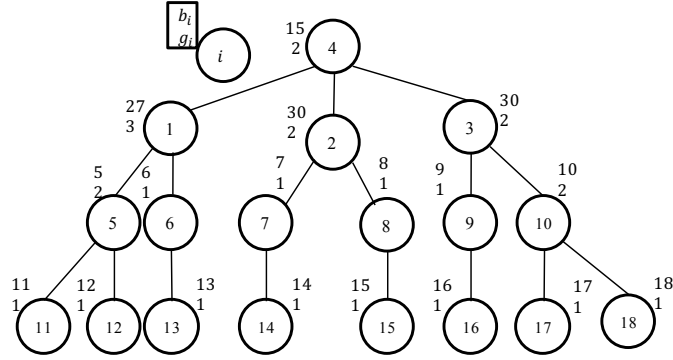


Figure 3.1: A TPIDS problem instance.

Take star 5 in the instance in Figure 3.1 as an example. The $C(5) = \{11, 12\}$, $F(5) = \emptyset$ and $E(5) = \emptyset$. We have $g_5 = 2$ and $|C(1) \cup F(1)| = 2$. Thus, $S_2^5 = S_1^5 = \{11, 12\}$. It is similar for star 6, 7, 8, 9 and 10. $S_1^6 = S_0^6 = \{13\}$. $S_1^7 = S_0^7 = \{14\}$. $S_1^8 = S_0^8 = \{15\}$. $S_1^9 = S_0^9 = \{16\}$. $S_2^{10} = S_1^{10} = \{17, 18\}$.

Once we determine the candidates for the current star subproblem, it is compressed into a single leaf node for its parent star subproblem. Now we need to determine its updated threshold and cost. If this central node's threshold is equal to its degree number in the original tree, this compressed node needs one neighbor. So, the updated threshold, g'_c , is 1 and this node is a core child for its parent star. Otherwise, this compressed node has the updated threshold as 0. Because if this central node's threshold is smaller than its degree number in the original tree, it is possible to select only its children nodes in the tree to satisfy its threshold requirement. For a node c , let node j be one of its children. Then, among the children of node j , the one with the g_j th smallest compressed cost is called its critical grandchild and its cost is denoted as $b'_{j(g_j)}$. The compressed node's cost is decided in the following way: We consider two cases. If the core set is not empty ($C(c) \neq \emptyset$),

the central node's compressed cost is 0. Otherwise, the compressed cost is calculated as the difference between node c 's cost and the total cost of all its critical grandchildren, $b'_c = \max\{b_c - m_c, 0\}$, where $m_c = \sum_{j \in L(c)} b'_{j(g_j)}$. If node j is a leaf in the tree, it has no children. Then, we set $b'_{j(g_j)}$ as ∞ . For the former case, it is required to select the central node for those nodes in the core children set. So, this new compressed node has cost 0 because the next star's central node could receive influence from current star's central node in the final solution of the tree and it is free from the next star's point of view. For the latter case, if the central node is selected, then, those critical grandchildren are not needed to be selected in the TPIDS. Then, for the next star, if it wants to receive influence from the current star, it must pay the incremental amount caused by selecting the central node. For each star, we have two solution candidates. One for the situation that the central node's parent node is selected and one for the situation it is not. We summarize the above procedure in Algorithm 5 with the following notation: Let X_P^c be the solution candidate when the parent node is selected, and X_{NP}^c be the candidate when the parent node is not selected. Let b'_c be the compressed node's cost and g'_c be its updated threshold. Initially, we have $b'_c = b_c$ and $g'_c = g_c$. Finally, let $E_1(c)$ and $E_2(c)$ denote the cheapest $g_c - |F(c)| - 1$ and $g_c - |F(c)|$ nodes in $C(c) \cup E(c)$ respectively.

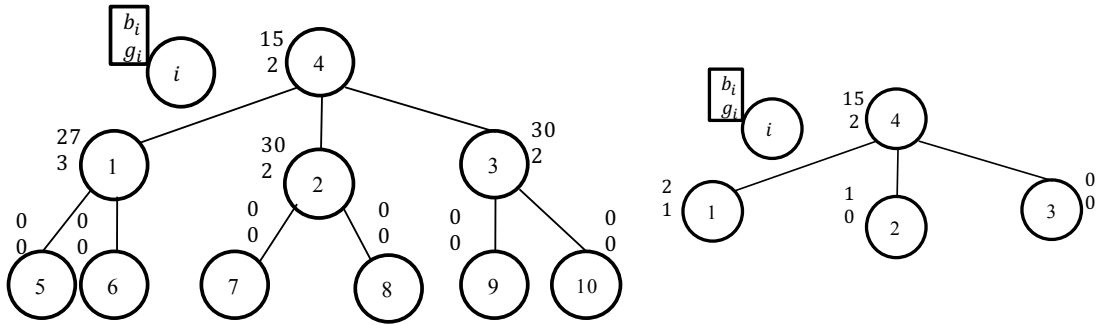


Figure 3.2: TPIDS (a) After compress star 5, 6, 7, 8, 9 and 10. (b) The last star.

Algorithm 5 StarHandling

Require: a star c

- 1: **if** $|F(c)| \geq g_c$ **then**
 - 2: $S_{g_c-1}^c = S_{g_c}^c \leftarrow F(c)$.
 - 3: **else**
 - 4: $S_{g_c-1}^c \leftarrow F(c) \cup E_1(c)$ and $S_{g_c}^c \leftarrow F(c) \cup E_2(c)$.
 - 5: **end if**
 - 6: Let $X_P^c \leftarrow S_{g_c-1}^c$ and $X_{NP}^c \leftarrow S_{g_c}^c$
 - 7: **If** $g_c = \text{deg}(c)$, $g'_c = 1$. **Otherwise**, $g'_c = 0$
 - 8: **If** $C(c)$ is not empty, $b'_c = 0$. **Otherwise**, $b'_c = \max\{b_c - m_c, 0\}$ where $m_c = \sum_{j \in L(c)} b'_{j(g_j)}$.
-

Again, we use the instance in Figure 3.1 as our example. For star 5, the updated threshold is 0 because $g_5 = 2$ and $\text{deg}(5) = 3$. Also, the updated cost is 0 because at least one of its children nodes is a leaf node. Similarly, we compress star 6, 7, 8, 9 and 10. Then, for star 1, we have $C(1) = E(1) = \emptyset$ and $F(1) = \{5, 6\}$. Then, $S_2^1 = \{5, 6\}$ and S_3^1 is not existed. Here, node 1's updated threshold as 1 because $g_1 = \text{deg}(1) = 3$. Its updated cost is 2 because $27 - 12 - 13$ where 12 is the cost of the second smallest child of node 5 and 13 is the cost of the smallest child of node 6. Then, the compressed node 1 is node 4's core child. Similarly, for star 2, we have $C(2) = E(2) = \emptyset$ and $F(2) = \{7, 8\}$. Then, $S_2^2 = \{7, 8\}$ and $S_1^2 = \{7\}$. Its updated cost is 1 and updated threshold is 0. It is node 4's expensive child. For star 3, we have $C(3) = E(3) = \emptyset$ and $F(3) = \{9, 10\}$. Then, $S_2^3 = \{9, 10\}$ and $S_1^3 = \{9\}$. Its updated cost is 0 and updated threshold is 0. It is node 4's free child. After compression, we have a smaller tree in Figure 3.2.(a).

After we obtain the solution of the last star which has the root node as its central node, we invoke a backtracking procedure to choose the solution from the candidates for each star subproblem and piece them together to obtain the final solution for this tree. In the last star subproblem, we first choose nodes in $S_{g_c}^c$ as X_{NP}^r . Then, if $C(c)$ is not empty

or b_c is smaller than m_c , we select the central node. Otherwise, we do not select it. Now, for each leaf node in this star, we know if it is selected or not. We also know if its parent node is selected or not. For instance, if the central node is selected, then, the central node sends influence to all its leaf nodes. Then, all stars with central node in $L(c)$ will pick the candidate where the parent node is selected. Similarly, given the set $S_{g_c}^c$ is selected, then, for a node l in $S_{g_c}^c$, we can proceed to nodes in $L(l)$ and pick the solution candidate where the parent node is selected. And, for a node l in $L(c) \setminus S_{g_c}^c$, if node r is not selected, we can proceed to nodes in $L(l)$ and pick the candidate where the parent node is not selected. With this information we can now proceed down the tree, incorporating the partial solution at a node based on the solution of its parent star. This backtracking procedure is described in Algorithm 6 SolutionBacktrack. Let L denote the set of leaf nodes in the tree, NL denote the set of non-leaf nodes in the tree, r denote the root of the tree (as determined by Algorithm 6), X^* denote the final solution of the tree and C^* its cost.

In this algorithm, we have the recursive function: Pick-Comp. It chooses the solution for a star c and recursively choose solutions for stars whose central nodes are leaf nodes of the star c . Algorithm 7 provides a more detail descriptions of Pick-Comp. Although it is possible to prove the correctness of this algorithm directly, we defer the proof until the next section. There we will provide a tight and compact extended formulation for the TPIDS problem, and use linear programming duality to prove its correctness.

Algorithm 6 SolutionBacktrack

Require: the last star r and its solution X_{NP}^r

- 1: **if** $b_r \leq \sum_{l \in L(r)} b'_{l(g_l)}$ **then**
- 2: $X^* \leftarrow X^* \cup \{r\}$.
- 3: $\forall l \in L(r) \cap NL$ call Pick-Comp(l, X^*, P).
- 4: **else**
- 5: $\forall l \in L(r) \cap NL$ call Pick-Comp(l, X^*, NP).
- 6: **end if**
- 7: $X^* \leftarrow X^* \cup X_{NP}^r$.
- 8: **for** $l \in X_{NP}^r$ **do**
- 9: $\forall h \in L(l) \cap NL$ call Pick-Comp(h, X^*, P).
- 10: **end for**
- 11: **for** $l \in L^r \setminus X_{NP}^r$ **do**
- 12: $\forall h \in L(l) \cap NL$ call Pick-Comp(h, X^*, NP).
- 13: **end for**
- 14: $C^* = \sum_{i \in X^*} b_i$
- 15: **return** C^*, X^* .

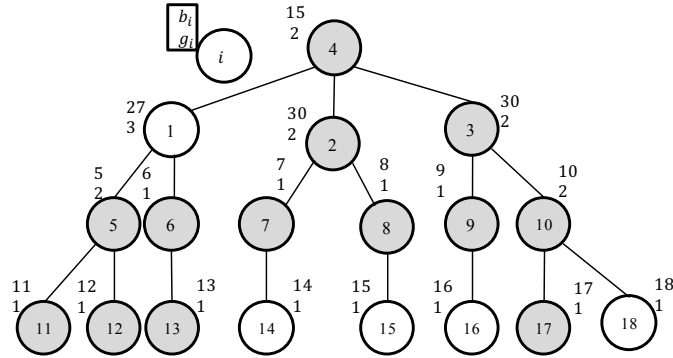


Figure 3.3: The solution obtained by our DP algorithm for the TPIDS.

For star 4 in Figure 3.2.(b), we have $C(4) = \{1\}$, $F(1) = \{3\}$ and $E(7) = \{2\}$. Then, $S_2^4 = \{1, 3\}$ and its core children set is not empty. So, the last star picks S_2^4 to satisfy node 4's threshold. Also, node 4 is selected. From there, star 1, 2 and 3 choose X_P . Then, node 5, 6, 8 and 9 are selected. Next, star 5 and 6 pick X_{NP} . So, node 11, 12 and 13 are selected. Also, star 7, 8, 9, and 10 pick X_P . So, node 17 is selected. Lastly, node 5, 6, 7, 8, 9, and 10 are selected to satisfy leaf nodes. Figure 3.3 shows this optimal solution and the TPIDS is shady.

Algorithm 7 Recursive Functions

```
1: function Pick-Comp( $c, X, Flag$ )
2:   if  $Flag = P$  then
3:      $X' = X_P^c$ .
4:   else
5:      $X' = X_{NP}^c$ .
6:   end if
7:    $X \leftarrow X \cup X'$ .
8:   for  $l \in X'$  do
9:      $\forall h \in L(l) \cap NL$  call Pick-Comp( $h, X, P$ ).
10:  end for
11:  for  $l \in L(c) \setminus X'$  do
12:     $\forall h \in L(l) \cap NL$  call Pick-Comp( $h, X, NP$ ).
13:  end for
14:  return  $X$ .
15: end function
```

Proposition 3.1. *The TPIDS problem on trees can be solved in $O(|V|)$ time.*

Proof. Proof of Proposition 3.1. There are at most $|V|$ stars in a tree. For each star c , let $deg(c)$ denote its degree number. We need to find the g_c cheapest children and it takes time $O(deg(c))$ time (Finding the g_c th order statistics can be done in $O(deg(c))$ by the Quickselect method in Chapter 9 of Stein et al. [2009]. Then, it takes $O(deg(c))$ to go through the list for collecting g_c cheapest children.). For the whole tree, this is bounded by $O(|V|)$. In the backtracking procedure, we only pick the final solution and it takes time $O(|V|)$. Therefore, the running time for the dynamic algorithm is linear with respect to the number of nodes. □

3.2.2 Good Formulations for the TPIDS Problem on Trees

We present two good formulations for the TPIDS on trees. Both formulations are directly adopted from previous formulations for the dominating set problem in the litera-

ture. The first one only uses decision variables in the node space. The second one is an extended formulation which has additional decision arc variables. Both of them are tight and compact for the TPIDS problem on trees.

The first formulation is based on the node variables formulation from Saxena [2004] and Bouchakour et al. [2008]. For each node i , let x_i be 1 if this node is in the TPIDS. Otherwise, it is 0. Then, we have the following formulation on trees (BIP3.1):

$$(BIP3.1) \quad \text{Minimize} \quad \sum_{i \in V} b_i x_i \quad (3.9)$$

$$\text{Subject To} \quad \sum_{j \in n(i)} x_j \geq g_i \quad \forall i \in V \quad (3.10)$$

$$x_i \in \{0, 1\} \quad \forall i \in V \quad (3.11)$$

The objective function (3.9) is to minimize the total cost of the TPIDS. Constraint (3.10) says that for a node i in V , it must have at least g_i neighbors in the TPIDS. Constraint (3.11) is the binary constraint.

The LP relaxation of BIP3.1 denoted by LP3.1 and is as follows:

$$(LP3.1) \quad \text{Minimize} \quad \sum_{i \in V} b_i x_i \quad (3.12)$$

$$\text{Subject To} \quad \sum_{j \in n(i)} x_j \geq g_i \quad \forall i \in V \quad (3.13)$$

$$0 \leq x_i \leq 1 \quad \forall i \in V \quad (3.14)$$

Theorem 3.1. *Given a tree, LP1 has an optimal solution with \mathbf{x} binary.*

Proof. Proof of Theorem 3.1. The constraint matrix defined by constraint set (3.13) is the adjacency matrix of a tree. So, it is a totally unimodular matrix (TUM). The constraint set (3.14) forms a identity matrix. Hence, combining these two constraint sets keeps the

constraint matrix TUM. □

The second formulation is based on the extended formulation via facility location described in Baïou and Barahona [2014]. We adopt it as follows (BIP3.2):

$$(BIP3.2) \quad \text{Minimize} \quad \sum_{i \in V} b_i x_i \quad (3.15)$$

$$\text{Subject To} \quad \sum_{j \in n(i)} y_{ji} \geq g_i \quad \forall i \in V \quad (3.16)$$

$$x_i - y_{ij} \geq 0 \quad \forall i \in V, j \in n(i) \quad (3.17)$$

$$x_i, y_{ij} \in \{0, 1\} \quad \forall i \in V, j \in n(i) \quad (3.18)$$

Recall that we have additional y_{ij} and y_{ji} variables for each edge $\{i, j\}$ in E . If $y_{ij} = 1$, it means node j is assigned to node i in the facility location context. To understand it in the TPIDS setting, we interpret it as node i sending influence to node j . Otherwise, it is 0. The objective function (3.15) is to minimize the total cost. Constraint (3.16) says that each node i in V must have at least g_i incoming arcs. Constraint (3.17) means that if node i is selected in the TPIDS, node i can send influence to node j which is one of node i 's neighbors. Constraint (3.18) is the binary constraint.

The LP relaxation of BIP3.2 is as follows (LP3.2):

$$(LP3.2) \quad \text{Minimize} \quad \sum_{i \in V} b_i x_i \quad (3.19)$$

$$\text{Subject To} \quad (u_i) \quad \sum_{j \in n(i)} y_{ji} \geq g_i \quad \forall i \in V \quad (3.20)$$

$$(v_{ij}) \quad x_i - y_{ij} \geq 0 \quad \forall i \in V, j \in n(i) \quad (3.21)$$

$$(w_i) \quad -x_i \geq -1 \quad \forall i \in V \quad (3.22)$$

$$x_i, y_{ij} \geq 0 \quad \forall i \in V, j \in n(i) \quad (3.23)$$

To show that LP3.2 is a compact formulation, we derive the complementary slackness (CS) conditions through the dual problem of LP3.2. Then, given that we already have a primal solution that is obtained using Algorithm 8, we construct a dual feasible solution and show that this pair of primal and dual solutions satisfies the CS conditions. Let u , v and w be dual variables corresponding to constraint sets (3.20), (3.21) and (3.22) respectively. The dual of LP3.2 is given below (DLP3.2):

$$(DLP3.2) \quad \text{Maximize} \quad \sum_{i \in V} g_i u_i - w_i \quad (3.24)$$

$$\text{Subject To } (x_i) \quad \sum_{j \in n(i)} v_{ij} - w_i \leq b_i \quad \forall i \in V \quad (3.25)$$

$$(y_{ij}) \quad u_j - v_{ij} \leq 0 \quad \forall i \in V, j \in n(i) \quad (3.26)$$

$$u_i, w_i, v_{ij} \geq 0 \quad \forall i \in V, j \in n(i) \quad (3.27)$$

Thus, we obtain the following CS conditions:

$$(\sum_{j \in n(i)} y_{ji} - g_i) u_i = 0 \quad \forall i \in V \quad (3.28)$$

$$(x_i - y_{ij}) v_{ij} = 0 \quad \forall i \in V, j \in n(i) \quad (3.29)$$

$$(1 - x_i) w_i = 0 \quad \forall i \in V \quad (3.30)$$

$$(\sum_{j \in n(i)} v_{ij} - w_i - b_i) x_i = 0 \quad \forall i \in V \quad (3.31)$$

$$(u_j - v_{ij}) y_{ij} = 0 \quad \forall i \in V, j \in n(i) \quad (3.32)$$

Let $\text{conv}(T)$ denote the convex hull of feasible TPIDS vectors \mathbf{x} , and let $ETPIDS$ denote the feasible region of LP3.2.

Theorem 3.2. *Given a tree, LP3.2 has an optimal solution with \mathbf{x} binary and $\text{Proj}_{\mathbf{x}}(ETPIDS) = \text{conv}(T)$.*

Proof. Proof of Theorem 3.2. By Algorithm 8 in Section 3.2.1, we can obtain the value for \mathbf{x} . To determine \mathbf{y} , we have the following procedure: Set $y_{ij} = 1$ for all j in $n(i)$, if $x_i = 1$. The remaining y variables have value 0. It should be clear that we have a feasible solution of BIP3.2 and thus LP3.2. In this proof, we want to show that we can construct a dual feasible solution to DLP3.2 that satisfies the CS conditions above. By now, we have a primal solution (\mathbf{x}, \mathbf{y}) , the compressed cost b' of all nodes, and the order of star subproblems in this execution. Primal constraint (3.21) is always binding and CS condition (3.29) is satisfied. Also, we always set $u_j = v_{ij} \forall i \in V, j \in n(i)$. So, dual constraint (3.26) is always binding and CS condition (3.32) is satisfied. We will focus on the remaining CS conditions (i.e., (3.28), (3.30) and (3.31)) in the following proof.

For a node i in V , let R_i be the set of nodes who send influence to node i (e.g., $x_j = 1$ and $y_{ji} = 1$ for all j in R_i). When node i 's parent node p is in R_i , we give node p a temporary b'_p which will be used in the later calculation. Let $deg(i)$ be the degree number of node i . If $g_i < deg(i)$, let $b'_p = b'_{(g_i)}$ which is the g_i smallest b'_j in node i 's child nodes. If $g_i = deg(i)$, we use $b'_p = b_p$. Now, let $b_m = \max\{b'_j : j \in R_i\}$, the largest b' in R_i . Then, we set $v_{ji} = b_m \forall j \in n(i)$. Repeat this for every node i in V , we set all v variables.

For a node j in $n(i)$, when $x_j = 1$, the v_{ji} value is not smaller than its corresponding cost b'_j in the solution. So, $\sum_{k \in n(j)} v_{jk}$ is not smaller than b_j . Then, we can set $w_j = \sum_{k \in n(j)} v_{jk} - b_j$ to have this constraint binding because w_j is allowed to have any nonnegative value due to the fact that $x_j = 1$ and primal constraint (3.22) is binding. Then, CS conditions (3.30) and (3.31) are satisfied for this case. When $x_j = 0$, the v_{ji} value is not bigger than its b'_j value. Otherwise, this node should be selected in our algorithm. Then, $\sum_{k \in n(j)} v_{jk} \leq b_j$ and we could set $w_j = 0$ to satisfy CS conditions (3.30)

because $x_j = 0$ and primal constraint(3.22) is not binding. Therefore, CS conditions (3.30) and (3.31) are satisfied.

For CS condition (3.28), when the primal constraint (3.20) is binding, we are good with the above setting given that we always set $u_j = v_{ij} \forall i \in V, j \in n(i)$. When the primal constraint (3.20) is not binding for a node i , it means node i has more than g_i neighbors selected. Then, when we follow the above procedure, b_m will be 0. So, $u_i = 0$ and CS condition (3.28) is satisfied in this setting. \square

3.3 The PIDS Problem

In this section, we present a linear time algorithm and the tight and compact extended formulation for the PIDS problem on trees. Also, we project the extended formulation onto the natural node space which provides not only the complete description of the polytope for the PIDS problem on trees, but also a novel formulation on general graphs. We also study the facet-defining conditions for the valid inequalities derived from the projection. The proposed formulation is stronger than the natural adaptations of the formulations in Bouchakour et al. [2008], Saxena [2004], Baïou and Barahona [2014] to the PIDS problem. Then, computational results are presented on general graphs.

3.3.1 Algorithm for the PIDS Problem on Trees

In this section, We present a dynamic programming (DP) algorithm to solve the PIDS problem on trees. In this method, we decompose the tree into subproblems. A subproblem is defined on a star network which has a central node and possibly many leaf

Algorithm 8 Algorithm for the PIDS problems on trees

- 1: Arbitrarily pick a node as the root node of the tree
 - 2: Define the order of star problems based on the bottom-up traversal of the tree
 - 3: **for** each star subproblem **do**
 - 4: StarHandling
 - 5: **end for**
 - 6: SolutionBacktrack
-

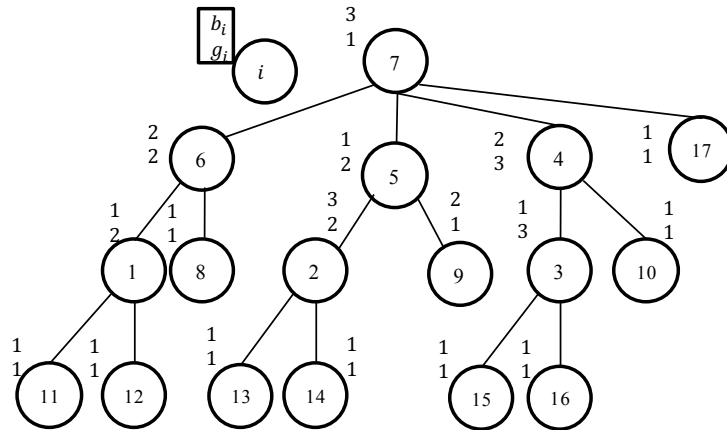


Figure 3.4: A PIDS problem instance.

nodes. Each subproblem is used to find the most promising candidates (at most two) and one of them will be part of the final solution of the tree. In this tree, each non-leaf node is a central node for a star network. By solving the subproblem, we have one candidate for the situation that the central node's parent node is selected and one candidate when its parent node is not selected. Next, the current star is compressed into one single leaf node with updated cost and threshold for its parent star network. This process is repeated until we are left with a single star. The last star should have the root node of the tree as the central node. After we exhaust all subproblems, a backtracking method is used to identify a final solution which combines the candidates of those star subproblems for the tree. The pseudocode of the proposed algorithm is shown in Algorithm 8.

For a star subproblem, we try to find the solution that satisfies the threshold require-

ment while the total cost is the minimum. Denote the central node by c and refer to this star as star c . Let the set of star c 's leaf nodes be $L(c)$. Two solution candidates are obtained for star c . Let X_P^c be the solution candidate when the parent node of star c is selected, and X_{NP}^c be the candidate when the parent node of star c is not selected. We use the instance in Figure 3.4 to illustrate the algorithm. For star 1, the central node is node 1 and its leaf nodes are node 11 and 12 (i.e., $c = 1$ and $L(1) = \{11, 12\}$). Observe that each leaf node of star 1 requires one neighbor. Therefore, in this star network, if the central node is selected, all its leaf nodes have their threshold requirements satisfied. This is the situation i) where node 1 is selected in the PIDS. However, there is another possibility. We can select leaf nodes to satisfy the central node's threshold requirement. Also, we need to consider if the central node's parent is included in the PIDS. Thus, the other possible solution is to select node 11 and 12 in the PIDS no matter if node 1's parent node is include in the PIDS. Given that the cost of node 1 is smaller than the total cost of node 11 and 12, we have $X_P^1 = X_{NP}^1 = \{1\}$. It is similar to star 2 and 3. In star 2, one solution candidate is to select node 2 and the other solution candidate is to select node 13 and 14. But now, we have $X_P^2 = X_{NP}^2 = \{13, 14\}$. In star 3, only one solution candidate is to select node 3 because the threshold value is 3 and it has only two leaf nodes. But, when its parent node is selected, it is possible to have node 15 and 16 instead of node 3 in the PIDS. Here, we have $X_P^3 = X_{NP}^3 = \{3\}$.

Next, once a star's solution candidates are determined, it is compressed into a single leaf node for its parent star subproblem. We need to decide the threshold value and the cost of the compressed leaf node. For the threshold value of the compressed leaf node, if the value is 1, it means the PIDS must contain either this compressed node or its parent node.

Node 1 has 3 neighbors and its threshold value is 2. Thus, when node 1 is not selected, it is possible that its threshold requirement is satisfied by selecting its children node only. Then, we set the threshold of the compressed node of star 1 as 0. For the same reason, we set the threshold of the compressed node of star 2 as 0. However, node 3 has 3 neighbors and its threshold value is 3. Thus, when node 3 is not selected, we must select node 4 in the PIDS. But the cost of node 3 is smaller than the total cost of node 15 and 16. Then, node 3 should be include in the PIDS for sure. Hence, we should have the threshold value of the compress node based on star 3 as 0.

For the cost of the compressed leaf node, we need to compared the cost of the central node to the cost of the solution consisting of leaf nodes. For star 1, the cost of node 1 is 1 which is smaller than 2, the cost of selecting node 11 and 12. Thus, the cost of the compressed node of star 1 is 0 because node 1 should be included in the optimal solution for sure. Otherwise, we would have to pay a bigger cost to cover the threshold requirement of node 1, 11 and 12 by selecting node 11 and 12 in the PIDS. For the same reason, the cost of the compressed node of star 3 is 0. But, in star 2, the cost of node 2 is bigger than that of the total cost of node 13 and 14. Thus, from the perspective of node 5, if it needs influence from node 2, it must pay the incremental cost of 1 which is calculated as the difference between the cost of node 2 and the total cost of node 13 and 14.

After compress star 1, 2 and 3, we obtain a smaller tree in Figure 3.5(a). Star 4 is similar to star 3. Its degree number is equal to its threshold value. But one of its leaf nodes, node 3 has cost 0. It never hurts to include a free node in the PIDS. Hence, we always select a zero cost node in the PIDS. The only solution candidate is to select node 4. But when its parent node is in the PIDS, it is possible to exclude node 4 but include

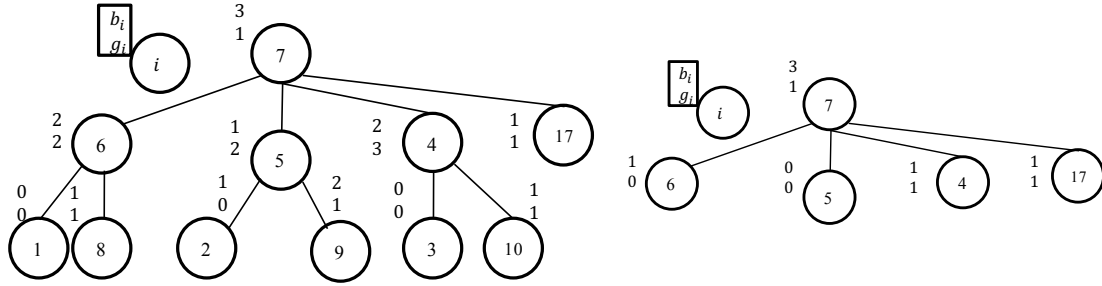


Figure 3.5: (a) After compress star 1, 2 and 3.

(b) The last star.

node 3 and 10 in the PIDS. So, we have $X_P^4 = \{3, 10\}$ and $X_{NP}^4 = \{4\}$. The compressed node of star 4 has threshold as 1 because its degree number is equal to its threshold value. Also, the cost is 1 because the cost of node 4 is 1 unit larger than the total cost of node 3 and 10. In star 5, its threshold value is 2 and it has two leaf nodes. Note that node 2 has cost 1 but its threshold value is 0. That means when node 5 is not selected, we do not need to select node 2. One solution candidate is to select node 5 in the PIDS. When node 5 is not selected, if node 5's parent node is in the PIDS, the solution candidate is to select node 9. Otherwise, we need to selected node 2 and 9. Also, we have $X_P^5 = X_{NP}^5 = \{5\}$. After compress star 5, the threshold is 1 and the cost is 0. For star 6, its threshold value is 2 and it has two leaf nodes. One of its leaf nodes, node 1 has 0 cost and 0 threshold value. Thus, other than selecting node 6, no matter if the parent node is selected, the solution candidate is to select node 1 and 8 given that we assume the zero cost node is always included in the PIDS. Here, we have $X_P^6 = \{1\}$ and $X_{NP}^6 = \{1, 8\}$. After compress star 6, the compressed node has cost as 1 and threshold as 0.

Now, we only have one star left. Star 7 has 4 leaf nodes. There are node 4, 5, 6 and 17. One solution is to select node 7. Although node 7 has threshold value as 1, the other solution is to select node 4 and 17 because each of them has threshold value as 1. Node 7

has cost as 3 and the total cost of node 4 and 17 is 2. The optimal solution of star 7 is to have node 4 and 17 in the PIDS.

Before discuss the backtracking procedure for identifying the final optimal solution, we want to summarize the bottom to top procedure for solving star subproblems. So far, we have seen three kinds of leaf nodes for a star. First, a leaf node has threshold as 1. We call them core children, denoted by $C(c)$. Second, a leaf node has both threshold and cost as 0. We call them free children, denoted by $F(c)$. Lastly, a leaf node has threshold as 0 but it has a positive cost. We call them expensive children, denoted by $E(c)$. For example, in star 7, $C(7) = \{4, 17\}$, $F(7) = \{5\}$ and $E(7) = \{6\}$. To solve a star, other than situation i): selecting the central node in the PIDS. We have two more situations. We assume nodes in $F(c)$ are always selected. If the central node c is not selected, we must select all nodes in $C(c)$. After that, we only select nodes from $E(c)$ when it is necessary. Therefore, if the size of $C(c) \cup F(c)$ is smaller than the threshold number of node c , we find the ascending order of the remaining nodes in $E(c)$ based on their cost. Then, we have situation ii): If node c 's parent node is selected, the possible solution is the set of the first $g_c - |C(c) \cup F(c)| - 1$ nodes in $E(c)$ and $C(c) \cup F(c)$, denoted by $S_{g_c-1}^c$. Lastly, for situation iii): if neither the central node nor its parent node is selected, the possible solution consists of the the first $g_c - |C(c) \cup F(c)|$ nodes in $E(c)$ and $C(c) \cup F(c)$, denoted by $S_{g_c}^c$. See star 5 and 6 in Figure 3.5)(a). If the size of $C(c) \cup F(c)$ is greater than or equal to the threshold number of node c , we have $S_{g_c}^c = S_{g_c-1}^c = C(c) \cup F(c)$. See star 1 and 2 in Figure 3.4. Also, if $deg(c)$ is equal to g_c , we have $S_{g_c}^c = \emptyset$ and $S_{g_c-1}^c = L(c)$. See star 3 in Figure 3.4. Thus, we have three situations for a star subproblem. The central node is selected in the first situation. However it is not selected in the other two situations. For

the last star, there are only two situations because there is no parent node any more.

Once we determine the candidates for current star subproblem, it is compressed into a single leaf node for its parent star subproblem. For the threshold value, if this central node's threshold is equal to its degree number in the original tree and its cost is bigger than the total cost of nodes in $S_{g_c-1}^c$, the updated threshold, g'_c , is 1. See star 4 in Figure 3.5(a). Otherwise, this compressed node has the updated threshold of 0 because if this central node's threshold is smaller than its degree number in the original tree, it is possible to select only its children nodes in the tree. If the central node's cost is not bigger than the total cost of nodes in $S_{g_c-1}^c$, it is preferred to select the central node.

Next, the updated cost of the compressed node is decided as follows: There are two cases. If the central node's cost is no bigger than the total cost of those nodes in $S_{g_c-1}^c$, this new node has cost 0. Otherwise, this new node has cost equal to $b_c - \sum_{l \in S_{g_c-1}^c} b'_l$. For the former case, it is always cheaper to select the central node for current star. So, this new node has cost 0 because the parent star's central node could receive influence from current star's central node in the final solution of the tree and it is free from the parent star's point of view. For the latter case, if the central node is selected, then, the cost for the current star is the cost of the central node. If the central node is not selected, depending on whether its parent node is selected or not, the cost of the current star is the total cost of those nodes in $S_{g_c}^c$ or that of the nodes in $S_{g_c-1}^c$. But in all three cases, we have to at least pay a cost equal to the total cost of those nodes in $S_{g_c-1}^c$. Then, for the next star, if it wants to receive influence from the current star, it must pay the incremental amount. For each star, we have two solution candidates. One for the situation that the central node's parent node is selected and one for not. Now, we can decide the two solution candidates

Algorithm 9 StarHandling

Require: a star c and let $b'_i = b_i$ if node i is a leaf of the original tree.

- 1: **if** $|C(c) \cup F(c)| \geq g_c$ **then**
 - 2: $S_{g_c-1}^c = S_{g_c}^c \leftarrow C(c) \cup F(c)$.
 - 3: **else if** $\text{deg}(i) = g_c$ **then**
 - 4: $S_{g_c-1}^c \leftarrow C(c) \cup F(c)$ and $S_{g_c}^c \leftarrow \emptyset$.
 - 5: **else**
 - 6: $S_{g_c-1}^c \leftarrow C(c) \cup F(c) \cup E_1(c)$.
 - 7: $S_{g_c}^c \leftarrow C(c) \cup F(c) \cup E_2(c)$.
 - 8: **end if**
 - 9: $\Pi_{S_{g_c-1}^c} = \sum_{l \in S_{g_c-1}^c} b'_l$ and $\Pi_{S_{g_c}^c} = \sum_{l \in S_{g_c}^c} b'_l$.
 - 10: **if** $b_c \leq \Pi_{S_{g_c-1}^c}$ **then**
 - 11: $X_P^c = X_{NP}^c \leftarrow c$.
 - 12: The compressed node's cost, $b'_c = 0$ and the updated threshold $g'_c = 0$.
 - 13: **else**
 - 14: $X_P^c \leftarrow S_{g_c-1}^c$. and $X_{NP}^c \leftarrow \arg \min\{b'_c, \Pi_{S_{g_c}^c}\}$.
 - 15: The compressed node's cost, $b'_c = b'_c - \Pi_{S_{g_c-1}^c}$.
 - 16: **If** $g_c = d(c)$, the updated threshold $g'_c = 1$. **Otherwise**, $g'_c = 0$.
 - 17: **end if**
-

based on the cost of the central node, $S_{g_c}^c$ and $S_{g_c-1}^c$. We summarize the above procedure in Algorithm 9 with the following notation: Let b'_c be the compressed node's cost and g'_c be its updated threshold. Initially, we have $b'_c = b_c$ and $g'_c = g_c$. Finally, let $E_1(c)$ and $E_2(c)$ denote the cheapest $g_c - |F(c)| - |C(c)| - 1$ and $g_c - |F(c)| - |C(c)|$ nodes in $E(c)$ respectively.

After we obtain the solution of the last star which has the root node as its central node, we invoke a backtracking procedure to choose the solution from the candidates for each star subproblem and piece them together to obtain the final solution for this tree. In the last star subproblem, between the central node c and those nodes in $S_{g_c}^c$, we choose the one with smaller cost as the solution. Now, for each leaf node in this star, we know if it is selected or not and if its parent node is selected or not. For instance, if the central node is selected, then, the central node sends influence to all its leaf nodes. Then, all stars

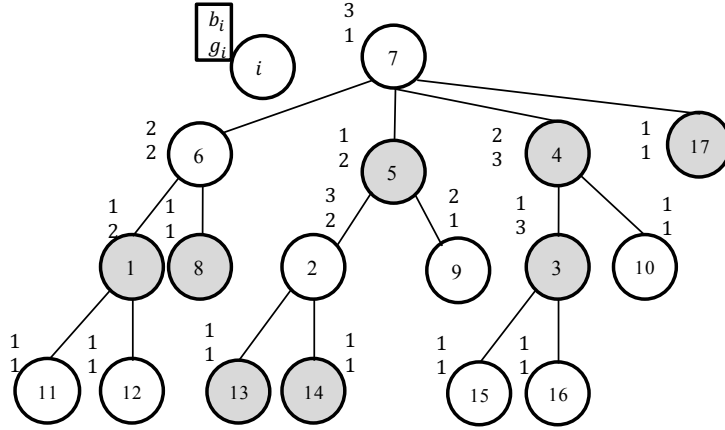


Figure 3.6: The solution obtained by our DP algorithm.

with central node in $L(c)$ will pick the candidate where the parent node is selected. If the set $S_{g_c}^c$ is selected, then, for a node l in $S_{g_c}^c$, we can proceed to nodes in $L(l)$ and pick the candidate where the parent node is selected. And, for nodes in $L(c) \setminus S_{g_c}^c$, they will pick the candidate where the parent node is not selected. With this information we can now proceed down the tree, incorporating the partial solution at a node based on the solution of its parent star. This backtracking procedure is described in Algorithm 10 and the recursive function it uses is also there.

For star 1 in Figure 3.5(b), we have $C(7) = \{4, 17\}$, $F(7) = \{5\}$ and $E(7) = \{6\}$. Then, $S_1^7 = \{4, 5, 17\}$ and its cost is 2 which is smaller than node 1's cost. So, the last star picks S_1^7 as its solution. From there, star 6 choose X_{NP}^6 . Then, node 1 and node 8 are selected. Also, star 3 picks X_P^3 . Then, node 3 is selected. Lastly, star 2 picks X_P^2 , then, node 13 and 14 are selected. Figure 3.6 shows this optimal solution and the PIDS is shaded.

Although it is possible to prove the correctness of this algorithm directly, we defer the proof until the next section. There we will provide a tight and compact extended

Algorithm 10 SolutionBacktrack

Require: the last star and its solution X

- 1: $X^* \leftarrow X$.
- 2: **if** X is r **then**
- 3: $\forall l \in L(r) \cap NL$ call **Pick-Comp**(l, X^*, P).
- 4: **else**
- 5: $\forall l \in L(r) \setminus S_{g_c}$ call **Pick-Comp**(l, X^*, NP).
- 6: **for** $l \in S_{g_c}$ **do**
- 7: $\forall h \in L(l) \cap NL$ call **Pick-Comp**(h, X^*, P).
- 8: **end for**
- 9: **end if**
- 10: $C^* = \sum_{i \in X^*} b_i$
- 11: **return** C^*, X^* .
- 12: **function** **Pick-Comp**($c, X, Flag$)
- 13: **If** $Flag = P, X' = X_P^c$. **Otherwise,** $X' = X_{NP}^c$.
- 14: $X \leftarrow X \cup X'$.
- 15: **if** $X' = c$ **then**
- 16: $\forall l \in L(c) \cap NL$ call **Pick-Comp**(l, X, P).
- 17: **else**
- 18: $\forall l \in L(c) \setminus X'$ call **Pick-Comp**(l, X, NP).
- 19: **for** $l \in X'$ **do**
- 20: $\forall h \in L(l) \cap NL$ call **Pick-Comp**(h, X, P).
- 21: **end for**
- 22: **end if**
- 23: **return** X .
- 24: **end function**

formulation for the PIDS problem, and use linear programming duality to prove its correctness.

Proposition 3.2. *The PIDS problem on trees can be solved in $O(|V|)$ time.*

Proof. Proof of Proposition 3.2. There are at most $|V|$ stars in a tree. For each star c , let $deg(c)$ denote its degree number. We need to find the g_c cheapest children and it takes time $O(deg(c))$ time (Finding the g_c th order statistics can be done in $O(deg(c))$ by the Quickselect method in Chapter 9 of Stein et al. [2009]. Then, it takes $O(deg(c))$ to go through the list for collecting g_c cheapest children.). For the whole tree, this is bounded

by $O(|V|)$. In the backtracking procedure, we only pick the final solution and it takes time $O(|V|)$. Therefore, the running time for the dynamic algorithm is linear with respect to the number of nodes. \square

3.3.2 A Tight and Compact Extended Formulation on Trees

In this section, we present a tight and compact extended formulation for the PIDS problem on trees. The direct adaptations of Saxena's and Baïou's formulations are not tight on trees as will be evident later. A novel edge splitting idea is needed to derive the proposed formulation for the PIDS problem on trees. Furthermore, we also prove that the DP algorithm in Section 3.3.1 gives an optimal solution for the PIDS problem on trees.

First, we present the adaptations of Saxena's and Baïou's formulations for the PIDS problem. They are called natural form formulation ($\text{BIP}_{\text{saxena}}$) and the facility location formulation ($\text{BIP}_{\text{baïou}}$) respectively. Here, we have the same decision variables as defined in Section 3.1.2. Recall that if node i is selected in the PIDS, $x_i = 1$. Otherwise, it is 0. Also, we have y_{ij} and y_{ji} variables for each edge $\{i, j\}$ in E . If node i sends influence to node j , $y_{ij} = 1$. Otherwise, it is 0.

The $\text{BIP}_{\text{saxena}}$ formulation is as follows:

$$(\text{BIP}_{\text{saxena}}) \quad \text{Minimize} \quad \sum_{i \in V} b_i x_i \quad (3.33)$$

$$\text{Subject To} \quad \sum_{j \in n(i)} x_j + g_i x_i \geq g_i \quad \forall i \in V \quad (3.34)$$

$$x_i \in \{0, 1\} \quad \forall i \in V \quad (3.35)$$

The $\text{BIP}_{\text{baïou}}$ formulation is given below:

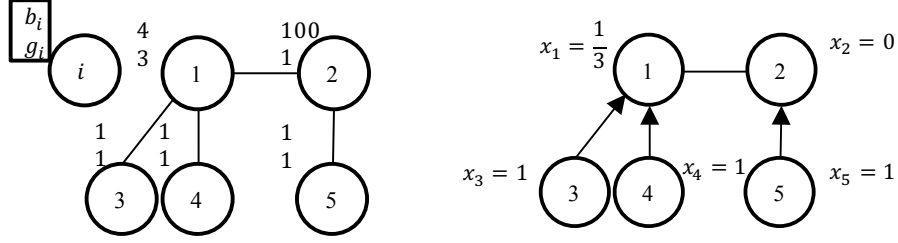


Figure 3.7: (a) A PIDS instance (b) A fractional solution returned by LP_{saxena} and $LP_{\text{baïou}}$

$$(\text{BIP}_{\text{baïou}}) \quad \text{Minimize} \quad \sum_{i \in V} b_i x_i \quad (3.36)$$

$$\text{Subject To} \quad \sum_{j \in n(i)} y_{ji} + g_i x_i = g_i \quad \forall i \in V \quad (3.37)$$

$$x_i - y_{ij} \geq 0 \quad \forall i \in V, j \in n(i) \quad (3.38)$$

$$x_i, y_{ij} \in \{0, 1\} \quad \forall i \in V, j \in n(i) \quad (3.39)$$

In the adaptation procedure, the main idea is changing the coefficient of x_i and the right hand side of constraint (3.2) and (3.5) in both Saxena's and Baiou's formulations, respectively. They are changed from 1 to g_i to reach constraint (3.34) and (3.37) reflecting the fact that a node i in V can require g_i neighbors instead of only 1 neighbor.

In Figure 3.7(a), we have a social network with five nodes. For each node i , its weight and threshold value (b_i and g_i) are listed beside it. For node 1, b_1 is 4 and g_1 is 3. By relaxing the binary constraints in $\text{BIP}_{\text{saxena}}$ and $\text{BIP}_{\text{baïou}}$, we have their linear programming (LP) relaxation which are referred to as LP_{saxena} and $LP_{\text{baïou}}$, respectively. If LP_{saxena} and $LP_{\text{baïou}}$ are used to solve the instance in Figure 3.7(a), both of them return a fractional optimal solution, $x_1 = \frac{1}{3}$, $x_2 = 0$, $x_3 = 1$, $x_4 = 1$ and $x_5 = 1$, with objective value $4\frac{1}{3}$. However, given that this problem can be solved in polynomial time on trees, a perfect integer programming formulation is desirable so that an integral optimal solution may be obtained by solving its LP relaxation. Next, we present a perfect formulation for

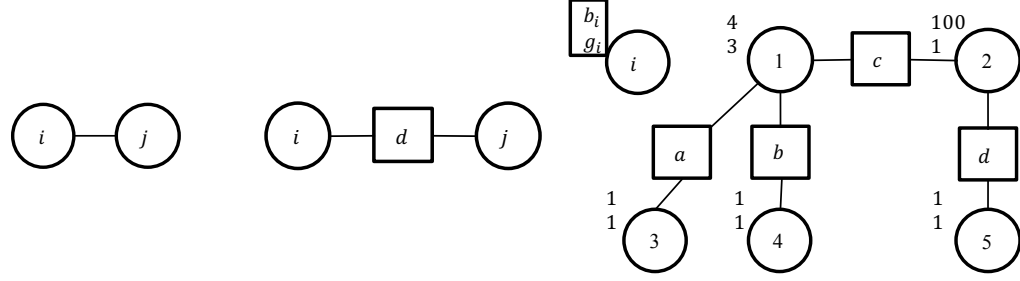


Figure 3.8: (a) An original edge (b) A transformed edge (c) Transformed graph of Figure 3.7.

the PIDS problem on trees.

To obtain the extended formulation, we first create a transformed graph based on the given one. From the input graph G , we create a new graph G_t by adding one dummy node to each edge in G . For each edge $\{i, j\} \in E$, insert a dummy node d . Let D denotes the set of dummy nodes. Since the dummy nodes have effectively split each edge into two, we replace each of the original edges $\{i, j\} \in E$ by two edges $\{i, d\}$ and $\{d, j\}$ in the new graph G_t . The procedure is showed in Figure 3.8(a) and Figure 3.8(b). Let E_t denote the set of edges in G_t ($G_t = (V \cup D, E_t)$). Figure 3.8(c) shows the transformed graph of the one in Figure 3.7 based on this procedure. Dummy nodes are represented by rectangles. Although the influence diffusion process is allowed for two time periods in the transformed graph, the solution is equivalent to a solution with one time period diffusion in the original graph given that the dummy nodes will not be selected in the PIDS.

For each node $i \in V$ we define a binary decision variable x_i that is 1 if node i is selected in the PIDS and 0 otherwise (these are the node variables). For each edge $\{i, d\} \in E_t$, where $i \in V$ and $d \in D$ (notice G_t is bipartite and E_t only contains edges between the nodes in V and D), define two binary arc variables y_{id} and y_{di} . They represent the direction of influence. If node d sends influence node i , y_{di} is 1 and 0 otherwise.

For a node $i \in V \cup D$, let $a(i)$ denote the set of node i 's neighbors in the transformed graph G_t . Furthermore, after inserting the dummy node d into an edge $\{i, j\}$, we call the resulting $\{i, d, j\}$ as an extended edge. We can now write the following tight and compact extended formulation for the PIDS problem and refer to it as the dummy node formulation ($\text{BIP}_{\text{dummy}}$):

$$(\text{BIP}_{\text{dummy}}) \quad \text{Minimize} \quad \sum_{i \in V} b_i x_i \quad (3.40)$$

$$\text{Subject To} \quad x_i \geq y_{dj} \quad \forall i \in V \ \& \ \{i, d, j\} \in E_t \quad (3.41)$$

$$x_i \leq y_{id} \quad \forall i \in V, d \in a(i) \quad (3.42)$$

$$-y_{id} - y_{di} \geq -1 \quad \forall \{i, d\} \in E_t \quad (3.43)$$

$$\sum_{d \in a(i)} y_{di} + g_i x_i \geq g_i \quad \forall i \in V \quad (3.44)$$

$$x_i \in \{0, 1\} \quad \forall i \in V \quad (3.45)$$

$$y_{id}, y_{di} \in \{0, 1\} \quad \forall \{i, d\} \in E_t \quad (3.46)$$

The objective function (3.40) is to minimize the total cost of the PIDS. The first constraint (3.41) says that if node i is selected, then node d can send influence to node j for an extended edge $\{i, d, j\}$ in E_t . Constraint (3.42) means that if node i is selected, it sends influence to all its neighbors. Constraint (3.43) says either i sends influence to d or d sends influence to i or neither send influence to the other. Constraint (3.44) ensures for a node i in V , either it is selected or it has at least g_i in coming arcs. Constraint (3.45) and (3.46) are the binary constraints.

Proposition 3.3. *$\text{BIP}_{\text{dummy}}$ is a valid formulation for the PIDS problem.*

Proof. Proof of Proposition 3.3. First, given any feasible solution of the PIDS problem,

we can make it a solution for BIP_{dummy} . Initially, we set all variables as 0. Based on the feasible solution we are given, if a node i in V is selected, we set $x_i = 1$. Then, we set $y_{id} = 1$ for all d in $a(i)$. Also, we set $y_{dj} = 1$ for all j in $n(i)$ if node j is not selected (i.e., $x_j = 0$). Thus, constraints (3.41) and (3.42) are satisfied. Furthermore, constraint (3.44) is satisfied because we are given a feasible solution. Lastly, only zero-one values are assigned to all variables and at most one of y_{id} and y_{di} is assigned as value 1 for an edge $\{i, d\}$ in E_t . Therefore, constraints (3.43), (3.45) and (3.46) are respected. We obtain a feasible solution for BIP_{dummy} .

Second, it is easy to see that for a given feasible solution of BIP_{dummy} , its x variable part satisfies the definition of the PIDS problem because of constraints (3.41) and (3.44).

□

Next, in Theorem 3.4, we show that BIP_{dummy} is a stronger formulation than BIP_{saxena} and $BIP_{\text{baïou}}$ by comparing the LP relaxation bounds of these three formulations. By relaxing binary constraints, the LP relaxation of BIP_{dummy} is referred to as LP_{dummy} and given below:

$$(LP_{\text{dummy}}) \quad \text{Minimize} \quad \sum_{i \in V} b_i x_i \quad (3.47)$$

$$\text{Subject To } (u_{ij}) \quad x_i - y_{dj} \geq 0 \quad \forall i \in V \ \& \ (i, d, j) \in E_t \quad (3.48)$$

$$(v_{id}) \quad y_{id} - x_i \geq 0 \quad \forall i \in V, d \in a(i) \quad (3.49)$$

$$(w_{id}) \quad -y_{id} - y_{di} \geq -1 \quad \forall \{i, d\} \in E_t \quad (3.50)$$

$$(z_i) \quad \sum_{d \in a(i)} y_{di} + g_i x_i \geq g_i \quad \forall i \in V \quad (3.51)$$

$$x_i \geq 0 \quad \forall i \in V \quad (3.52)$$

$$y_{id}, y_{di} \geq 0 \quad \forall \{i, d\} \in E_t \quad (3.53)$$

Let z_{dummy} , z_{saxena} and $z_{\text{baïou}}$ denote the optimal objective values of LP_{dummy} , $\text{LP}_{\text{saxena}}$ and $\text{LP}_{\text{baïou}}$, respectively.

Theorem 3.3. *In terms of the strength of LP relaxation, $\text{BIP}_{\text{saxena}}$ and $\text{BIP}_{\text{baïou}}$ are equivalent. $\text{BIP}_{\text{dummy}}$ is at least as strong as those two. In other words, $z_{\text{dummy}} \geq z_{\text{saxena}} = z_{\text{baïou}}$.*

Proof. Proof of Theorem 3.3. First, we show that $\text{LP}_{\text{saxena}}$ and $\text{LP}_{\text{baïou}}$ are equivalent. Given any solution of $\text{LP}_{\text{saxena}}$, denoted by x^* , we set $y_{ij}^* = x_i^*$ for each i in V and j in $n(i)$. Then, because x^* satisfies constraint (3.34) ($\sum_{j \in n(i)} x_j^* + g_i x_i^* \geq g_i$), we have $\sum_{j \in n(i)} y_{ji}^* + g_i x_i^* \geq g_i$. Then, we can reduce the value of y^* accordingly to make the constraint binding. So, we have a feasible solution for $\text{LP}_{\text{baïou}}$. Furthermore, given any feasible solution of $\text{LP}_{\text{baïou}}$, denoted by (x^*, y^*) , we just need to take the x^* part and it is a feasible solution for $\text{LP}_{\text{saxena}}$.

Second, we show that the LP_{dummy} is at least as strong as the $\text{LP}_{\text{saxena}}$. Given any feasible solution of LP_{dummy} , denoted by (x^*, y^*) , we just need to take the x^* part and it is a feasible solution for $\text{LP}_{\text{saxena}}$. However, not all feasible solutions of $\text{LP}_{\text{saxena}}$ can be converted to a feasible solution for LP_{dummy} . The counter example is the one shown in Figure 3.7. As we mentioned earlier, $x_1 = \frac{1}{3}$, $x_2 = 0$, $x_3 = 1$, $x_4 = 1$ and $x_5 = 1$ is a feasible and optimal solution for $\text{LP}_{\text{saxena}}$ with objective value $4\frac{1}{3}$. But it is not feasible for LP_{dummy} . After transformation, LP_{dummy} is applied to the instance in Figure 3.8(c). If we want to satisfy constraint (3.51) for node 1, we must have $y_{a1} = y_{b1} = 1$ because $x_2 = 0$ and node 1 needs two incoming arcs given that $x_1 = \frac{1}{3}$. However, y_{1a} and y_{1b} should be at least $\frac{1}{3}$ because $x_1 = \frac{1}{3}$ and constraint (3.49). Then, $y_{a1} + y_{1a} = 1\frac{1}{3} > 1$ and $y_{4b} + y_{b4} = 1\frac{1}{3} > 1$ which violate constraint (3.50). Solving LP_{dummy} for the instance

in Figure 3.8(c), we get the integral solution $x_1 = 1, x_2 = 0, x_3 = 0, x_4 = 0, x_5 = 1,$
 $y_{1a} = 1, y_{ab} = 1, y_{1c} = 1, y_{c2} = 1, y_{a3} = 1, y_{a4} = 1, y_{5d} = 1$ and the remaining y
variables are zeros with objective value 5. \square

More importantly, $\text{BIP}_{\text{dummy}}$ is the strongest formulation for the PIDS problem on trees. It means that we can obtain an optimal integral solution by solving LP_{dummy} instead of $\text{BIP}_{\text{dummy}}$. To prove it, the main idea is that given the primal feasible solution of LP_{dummy} from our DP algorithm, we construct a dual feasible solution for the dual problem of LP_{dummy} . Then, we show the complementary slackness (CS) conditions derived from this pair of primal and dual formulations are satisfied by this pair of primal and dual solutions.

The dual of LP_{dummy} is as follows:

$$(\text{DLP}_{\text{dummy}}) \text{ Max} \quad \sum_{i \in V} g_i z_i - \sum_{\{i,d\} \in E_t} w_{id} \quad (3.54)$$

$$\text{S.T. } (x_i) \quad \sum_{\{i,d,j\} \in E_t} u_{ij} - \sum_{\{i,d\} \in E_t} v_{id} + g_i z_i \leq b_i \quad \forall i \in V \quad (3.55)$$

$$(y_{id}) \quad v_{id} - w_{id} \leq 0 \quad \forall i \in V, d \in a(i) \quad (3.56)$$

$$(y_{di}) \quad -u_{ji} - w_{id} + z_i \leq 0 \quad \forall d \in D, i \in a(d) \quad (3.57)$$

$$u_{ij} \geq 0 \quad \forall i \in V \ \& \ \{i, d, j\} \in E_t \quad (3.58)$$

$$v_{id}, w_{id} \geq 0 \quad \forall \{i, d\} \in E_t \quad (3.59)$$

$$z_i \geq 0 \quad \forall i \in V \quad (3.60)$$

We have u_{ij}, v_{id}, w_{id} and z_i as dual variables for constraint sets (3.48), (3.49), (3.50), and (3.51), respectively. We refer to the dual linear program as $\text{DLP}_{\text{dummy}}$. Let $\text{conv}(X)$ denote the convex hull of feasible PIDS vectors \mathbf{x} , and EPIDS denote the feasible region of LP_{dummy} .

Theorem 3.4. *Given a tree, LP_{dummy} has an optimal solution with x binary and $Proj_x(EPIDS) = conv(X)$.*

Proof. Proof of Theorem 3.4. It should be clear that the solution of the algorithm in Section 3.3.1 provides a feasible solution to BIP_{dummy} and thus LP_{dummy} . Recall that $a(i)$ is the set of node i 's neighbors in G_t and $n(i)$ is that in G . For x variables, if a node i is in the PIDS, $x_i^* = 1$. Otherwise, $x_i^* = 0$. To obtain y variables' values, if $x_i^* = 1$, set $y_{id}^* = 1$ for all d in $a(i)$. Then, for all j in $n(i)$, if $x_j^* = 0$, set $y_{dj}^* = 1$ where d is the dummy node inserted in between node i and node j . For the remaining undecided edges $\{i, d\}$, we set $y_{id}^* = 0$ and $y_{di}^* = 0$. Thus, we obtain a feasible solution for LP_{dummy} based on the solution returned by the DP algorithm. In this proof, we show that we can construct a dual feasible solution to DLP_{dummy} and this pair of primal and dual solutions satisfies the CS conditions as follows:

$$(x_i - y_{dj})u_{ij} = 0 \quad \forall i \in V \ \& \ \{i, d, j\} \in E_t \quad (3.61)$$

$$(y_{id} - x_i)v_{id} = 0 \quad \forall i \in V, d \in a(i) \quad (3.62)$$

$$(1 - y_{id} - y_{di})w_{id} = 0 \quad \forall \{i, d\} \in E_t \quad (3.63)$$

$$(g_i - \sum_{d \in n(i)} y_{di} - g_i x_i)z_i = 0 \quad \forall i \in V \quad (3.64)$$

$$(b_i - \sum_{\{i,d,j\} \in E_t} u_{ji} + \sum_{\{i,d\} \in E_t} v_{id} - g_i z_i)x_i = 0 \quad \forall i \in V \quad (3.65)$$

$$(v_{id} - w_{id})y_{id} = 0 \quad \forall i \in V, d \in a(i) \quad (3.66)$$

$$(-u_{ji} - w_{id} + z_i)y_{di} = 0 \quad \forall d \in D, i \in a(d) \quad (3.67)$$

After an execution of the algorithm in Section 3.3.1, we have a primal solution (x^*, y^*) , the compressed cost b' of non-leaf nodes (recall that for a leaf node i , it has

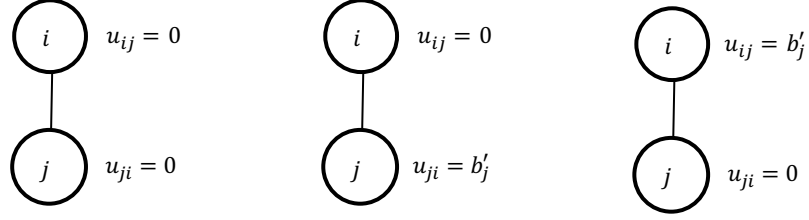


Figure 3.9: Node j is a (a) free child (b) expensive child (c) core child of node i for u variables.

$b'_i = b_i$), and the order of star subproblems in this execution. First of all, we always have $v_{id} = w_{id}$ for all $\{i, d\}$ in G_t to satisfy the dual constraint (3.56) and CS condition (3.66). Consequently, we can focus on the remaining CS conditions.

In DLP_{dummy} , only u variables interact between two regular nodes. If we fix their values first, we can isolate each regular node and assign v , w and z variables. Following the order in the execution of the DP algorithm, we first assign value for all u variables. Let node j be current node and node i be its parent node in the original tree G . We have three cases as shown in Figure 3.9. First, if node j is a free child of node i , let $u_{ji} = 0$ and $u_{ij} = 0$. Second, if node j is an expensive child of node i , $u_{ji} = b'_j$ and $u_{ij} = 0$. Third, if node j is a core child of node i , let $u_{ji} = 0$ and $u_{ij} = b'_j$. For the first case, CS condition (3.61) for the extended edge $\{i, d, j\}$ is satisfied because both u_{ji} and u_{ij} are zeros. But for the latter two cases, u variables can take positive value. Hence, when it happens, the corresponding primal constraint (3.48) for the extended edge $\{i, d, j\}$ should be binding. For the second case, we can focus on the node j because only u_{ji} could be positive by construction. If $x_j = 0$, then, the corresponding primal constraint (3.48) is binding. If $x_j = 1$, the primal constraint (3.48) is binding when the adjacent node i in the original graph is not selected (i.e., $x_i = 0$). Based on the DP algorithm in Section 3.3.1, if node i is selected, we would not select any nodes from its core nor expensive children. If any

of its core or expensive child node are selected, node i would not be selected. Thus, we have $x_i = 0$ and can set $y_{di} = 1$. Then, the primal constraint (3.48) is binding. For the last case, it is similar to the second case except that we focus on node i . Hence, when a u variable has positive value, the corresponding primal constraint (3.48) for the extended edge $\{i, d, j\}$ is binding. Therefore, CS condition (3.61) is satisfied.

To set the remaining dual variables, we consider three cases for a node i in V . Case 1: If $g_i = \text{deg}(i)$, primal constraint (3.49), (3.50) and (3.51) are always binding. It is because if node i is selected, $y_{id} = 1$ for all d in $a(i)$. Otherwise, $y_{di} = 1$ for all d in $a(i)$. Also, $y_{id} + y_{di} = 1$ and $\sum_{d \in n(i)} y_{di} + g_i x_i = g_i$ for both situations. So, CS condition (3.62), (3.63) and (3.64) are satisfied and v, w and z should be nonnegative. We set $z_i = \max\{u_{ji} : j \in n(i)\}$. For all d in $a(i)$, $w_{id} = z_i - u_{ji}$. Thus, we have dual constraints (3.57) binding and CS condition (3.67) is satisfied. Now, if this node i is a core child for its parent (i.e. $b'_i > 0$), the dual constraint (3.55) is binding. This is because LHS of (3.55) is equal to:

$$\sum_{j \in n(i)} (u_{ij} + u_{ji}) - \text{deg}(i)z_i + \text{deg}(i)z_i = \sum_{j \in L(i)} b'_j + b'_i = b_i$$

For the first term, what is because $v_{id} = w_{id} = z_i - u_{ji}$ for all d in $a(i)$ and $g_i = \text{deg}(i)$. From the first term to the second then to last one, follows from algebra and the way we obtain b'_i . Thus, CS condition (3.65) is satisfied. If this node i is a free child (i.e. $b'_i = 0$) for its parent node, we first calculate δ as the LHS of dual constraint (3.55) minus its RHS. δ is a positive value from the way we defined a free child ($b'_i < \pi_{g_i-1}$). Then, we pick a dummy node k in $a(i)$ and set $w_{ik} = w_{ik} + \delta$. So, its corresponding dual constraint (3.57) is satisfied. Also $y_{ki} = 0$ because $x_i = 1, y_{ik} = 1$ due to the fact that a free node is always

selected in the DP algorithm. Dual constraint (3.55) is binding by construction because LHS of (3.55) is equal to:

$$\sum_{j \in n(i)} (u_{ij} + u_{ji}) - \text{deg}(i)z_i + \text{deg}(i)z_i - \delta = \sum_{j \in L(i)} b'_j - \delta = b_i$$

So, CS condition (3.65) and (3.67) are satisfied. Hence, all CS conditions are satisfied.

In the remaining two cases, we have $g_i < \text{deg}(i)$. Case 2: If $x_i = 1$, primal constraint (3.49), (3.50) and (3.51) are binding. So, CS condition (3.62), (3.63) and (3.64) are satisfied and v , w and z are nonnegative. The dual constraint (3.55) needs to be binding. For a node j in $n(i)$, if $u_{ij} > 0$, it is a core child or the parent node. We first calculate $\delta = \sum_{j \in n(i)} u_{ij} - b_i$ and consider two situations. First, when $\delta \geq 0$, it means the LHS of the dual constraint (3.55) exceeds its RHS already and we need to reduce the LHS. We set $z_i = 0$ and $w_{id} = 0$ for all d in $a(i)$. Then, we pick a dummy node k in $a(i)$ and set $w_{ik} = w_{ik} + \delta$. In this way, dual constraint (3.57) is satisfied with $y_{ki} = 0$. Second, when $\delta < 0$, it means we need to increase the LHS of the dual constraint (3.55) and variable z_i can help here. Let $z_i = \max\{u_{ji} : j \in S_{g_i-1}\}$. For a node j in $n(i)$, if u_{ji} is smaller than z_i (i.e., it means $j \in \{S_{g_i-1} \cup p\}$), we set $w_{id} = z_i - u_{ji}$ and dual constraint (3.57) is binding. Otherwise, $w_{id} = 0$, dual constraint (3.57) is satisfied because $u_{ji} \geq z_i$. In both situations based on the value of δ , dual constraint (3.55) is binding. The former situation with $\delta \geq 0$ is obvious. For the latter situation with $\delta < 0$, we have LHS of (3.55) equal to:

$$\sum_{j \in S_{g_i-1} \cup p} (u_{ij} + u_{ij}) - g_i z_i + g_i z_i = \sum_{j \in S_{g_i-1}} b'_j + b'_i = b_i$$

For the first term, what is because if a node j is in $\{S_{g_i-1} \cup p\}$, $v_{id} = w_{id} = z_i - u_{ji}$.

Otherwise, $v_{id} = 0$. We go from the first term to the second one because $u_{ip} = b'_i$, $u_{pi} = 0$ and $v_{id} = w_{id} = z_i$. Going from the second term to the last one follows the algebra from the way we obtain b'_i . So, CS condition (3.65) and (3.67) are satisfied.

Case 3: If $x_i = 0$, because a free child is always selected and a core child i requires $g_i = \text{deg}(i)$, this node i is an expensive node for its parent p ($b'_i > 0$). Let R_i be the set of nodes who are selected and belong to $n(i)$. Primal constraint (3.49) is binding ($x_i = 0$ and $y_{id} = 0$) and the CS condition (3.62) is satisfied. Also, v_{id} should be nonnegative for all d in $a(i)$. If a node j is in R_i , primal constraint (3.50) is binding ($y_{di} = 1$) for the $\{i, d\}$ part of the extended edge $\{i, d, j\}$ and w_{id} should be nonnegative. Otherwise, it is not binding ($y_{id} + y_{di} = 0$) and w_{id} must be 0. When $|R_i| = g_i$, primal constraint (3.51) is binding and z_i must be nonnegative. When $|R_i| > g_i$, primal constraint (3.51) is not binding and z_i must be 0. We set $z_i = \max\{u_{ji} : j \in R_i\}$. Here, because when node j is a core child of node i , $u_{ji} = 0$. Also, because node i is an expensive child of node p , $u_{pi} = 0$. Thus, z_i takes positive value only when R_i contains expensive children. When that happens, the size of R_i must be g_i because we only select exactly the number of necessary (i.e., $g_i - |C(i)| - |F(i)|$) nodes from expensive children. Therefore, we have the CS condition (3.64) fulfilled. For a node j in $n(i)$, if u_{ji} is smaller than z_i (i.e. $j \in \{R_i\}$), let $w_{id} = z_i - u_{ji}$. Then, dual constraint (3.57) is binding. Otherwise, $w_{id} = 0$ and dual constraint (3.57) is satisfied because $u_{ji} \geq z_i$ and $y_{di} = 0$. So, the CS conditions (3.63) and (3.67) are satisfied.

Now we need to show that dual constraint (3.55) is satisfied given that $x_i = 0$. We have four situations based on R_i . For the first two situations, R_i does not contain any expensive children. First, when $p \in R_i$ and $E(i) \cap R_i = \emptyset$, LHS of (3.55) is equal to:

$$\sum_{j \in R_i} (u_{ij} + u_{ji}) = \sum_{j \in R_i \setminus p} b'_j + b'_i = \sum_{j \in C(i)} b'_j + b'_i = \sum_{j \in S_{g_i-1}} b'_i + b'_i = b_i$$

For the first term, what is because $z_i = 0$ and $v_{id} = w_{id} = 0$ for all d in $a(i)$. We can go from the first term to the second term, it is due to the way we assign the values for u variables and especially $u_{ip} + u_{pi} = b'_i$. We can go from the second term to the third and then to the fourth term, because a free child j has $b'_j = 0$. So, $\sum_{j \in S_{g_i-1}} b'_i = \sum_{j \in C(i)} b'_j = \sum_{j \in R_i \setminus p} b'_j$. The fourth term to the last one follows from the way we obtain b'_i .

Second, when $p \notin R_i$ and $E(i) \cap R_i = \emptyset$, LHS of (3.55) is equal to:

$$\sum_{j \in R_i} (u_{ij} + u_{ji}) = \sum_{j \in R_i} b'_j = \sum_{j \in C(i)} b'_j = \sum_{j \in S_{g_i}} b'_i \leq b_i$$

For the first term, what is because $z_i = 0$ and $v_{id} = w_{id} = 0$ for all d in $a(i)$. We can go from the first term to the second term, it is due to the way we assign the values for u variables and especially $u_{ip} + u_{pi} = b'_i$. We can go from the second term to the third and then to the fourth term, because a free child j has $b'_j = 0$. So, $\sum_{j \in S_{g_i}} b'_i = \sum_{j \in C(i)} b'_j = \sum_{j \in R_i} b'_j$. The last inequality is due the DP algorithm otherwise. Given that $x_i = 0$, the CS condition (3.65) is satisfied.

For the third and fourth situations, R_i contains expensive children and $|R_i| = g_i$.

Third, when $p \in R_i$ and $E(i) \cap R_i \neq \emptyset$, LHS of (3.55) is equal to:

$$\sum_{j \in S_{g_i-1} \cup p} u_{ij} + u_{ij} - (g_i - 1)z_i + g_i z_i = \sum_{j \in S_{g_i-1}} b'_j + b'_i - z_i + z_i = \sum_{j \in S_{g_i-1}} b'_i + b'_i = b_i$$

The first term follows because $v_{id} = w_{id} = z_i - u_{ji}$ if $j \in \{S_{g_i-1} \cup p\}$. Otherwise, it is 0. We can go from the first term to the second term because $u_{ip} = b'_i$, $u_{pi} = 0$ and

$v_{id} = w_{id} = z_i$. The second term to the third and then to the fourth term) follows from the way we obtain b'_i .

Fourth, when when $p \notin R_i$ and $E(i) \cap R_i \neq \emptyset$, LHS of (3.55) is equal to:

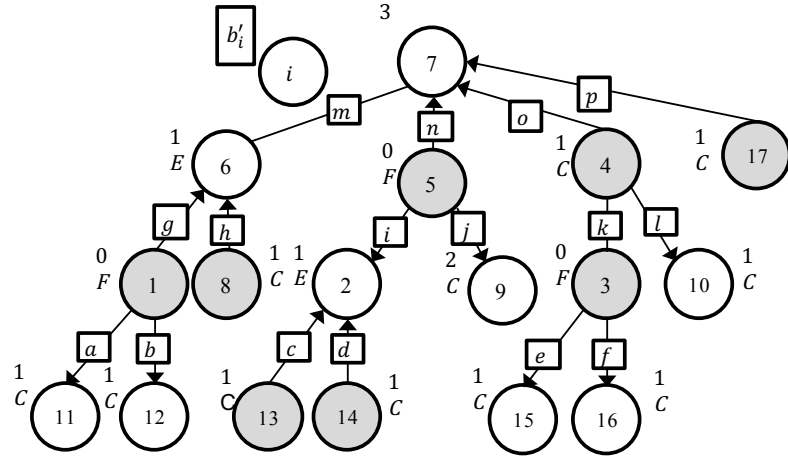
$$\begin{aligned} \sum_{j \in S_{g_i}} (u_{ij} + u_{ij}) - g_i z_i + g_i z_i &= \sum_{j \in S_{g_i}} b'_j + b'_i - z_i \\ &= \sum_{j \in S_{g_{i-1}}} b'_j + b'_{(g_i)} + b'_i - z_i = \sum_{j \in S_{g_{i-1}}} b'_j + b'_i = b_i \end{aligned}$$

The first term follows because $v_{id} = w_{id} = z_i - u_{ji}$ if $j \in \{R_i \cup p\}$. Otherwise, it is 0.

We can go from the first term to the second term because for the edge (i, d, p) , $u_{ip} = b'_i$, $u_{pi} = 0$ and $v_{id} = w_{id} = z_i$. We can go from the second term to the third and then to the fourth term because $z_i = b'_{(g_i)} = \max\{u_{ji} : j \in S_{g_i}\}$. The fourth term to the last one, follows from the way we obtain b'_i . Therefore, all CS conditions are satisfied.

Finally, for the root node (which has no parent node), if it is selected, we follow Case 2. Otherwise, it is similar to Case 3 although its dual constraint (3.55) is not binding.

□



(a)

$u_{1,11} = 1$	$u_{11,1} = 0$	$w_{1a} = 1$	$w_{6h} = 0$	$z_1 = 0$
$u_{1,12} = 1$	$u_{12,1} = 0$	$w_{1b} = 0$	$w_{6m} = 0$	$z_2 = 0$
$u_{1,6} = 0$	$u_{6,1} = 0$	$w_{1g} = 0$	$w_{7m} = 0$	$z_3 = 0$
$u_{2,13} = 1$	$u_{13,2} = 0$	$w_{2c} = 0$	$w_{7n} = 0$	$z_4 = 1$
$u_{2,14} = 1$	$u_{14,2} = 0$	$w_{2d} = 0$	$w_{7o} = 0$	$z_5 = 0$
$u_{2,5} = 1$	$u_{5,2} = 0$	$w_{2i} = 0$	$w_{7p} = 0$	$z_6 = 0$
$u_{3,15} = 1$	$u_{15,3} = 0$	$w_{3e} = 0$	$w_{8h} = 0$	$z_7 = 0$
$u_{3,16} = 1$	$u_{16,3} = 0$	$w_{3f} = 0$	$w_{9j} = 0$	$z_8 = 1$
$u_{3,4} = 0$	$u_{4,3} = 0$	$w_{3k} = 0$	$w_{10l} = 0$	$z_9 = 1$
$u_{4,10} = 1$	$u_{10,4} = 0$	$w_{4k} = 1$	$w_{11a} = 0$	$z_{10} = 1$
$u_{4,7} = 0$	$u_{7,4} = 1$	$w_{4l} = 1$	$w_{12b} = 0$	$z_{11} = 1$
$u_{5,9} = 2$	$u_{9,5} = 0$	$w_{4o} = 0$	$w_{13c} = 0$	$z_{12} = 1$
$u_{5,7} = 0$	$u_{11,1} = 0$	$w_{5i} = 1$	$w_{14d} = 0$	$z_{13} = 1$
$u_{6,8} = 1$	$u_{8,6} = 0$	$w_{5j} = 0$	$w_{15e} = 0$	$z_{14} = 1$
$u_{6,7} = 1$	$u_{7,6} = 0$	$w_{5n} = 0$	$w_{16f} = 0$	$z_{15} = 1$
$u_{7,17} = 1$	$u_{17,7} = 0$	$w_{6g} = 0$	$w_{17p} = 0$	$z_{16} = 1$
				$z_{17} = 1$

(b)

Figure 3.10: A PIDS problem instance for Theorem 3.4. and Dual variable values.

In Figure 3.10(a) is the transformed graph and its solution based on the instance in Figure 3.6. The number beside a node is its b' value and the letter "C", "E" and "F" represent the node is a core, expensive and free child of its parent node, respectively. On the right part of Figure 3.10, it has the dual variables' values. Because $w_{id} = v_{id}$, we only

show w_{id} here. In this instance, first of all, we set up all u variables. They are shown in the first two columns in Figure 3.10(b).

For Case 1, we have node 3, 4, 8, 9, 10, 11, 12, 13, 14, 15, 16, and 17. Following the procedure in Theorem 3.4, for node 3, $z_3 = \max\{u_{4,3}, u_{15,3}, u_{16,3}\} = \max\{0, 0, 0\} = 0$. Then, $w_{3e} = w_{3f} = w_{3k} = v_{3e} = v_{3f} = v_{3k} = 0$. Similarly, we can set up the dual variables for the rest Case 1 nodes. For Case 2, we have node 1 and 5. Using node 1 as the example, we first calculate $\delta = u_{1,11} + u_{1,12} + u_{1,6} - b_1 = 1 + 1 + 0 - 1 = 1$. Then, $z_1 = 0$ and $w_{1a} = w_{1b} = w_{1g} = 0$. Next, we pick the dummy node a and set $w_{1a} = w_{1a} + \delta = 1$. Thus, $w_{1a} = v_{1a} = 1$ and $w_{1b} = w_{1g} = v_{1b} = v_{1g} = 0$. It is similar for node 5. For Case 3, we have node 2, 6 and 7. For node 2, its $R_2 = \{5, 13, 14\}$. So, $z_2 = \max\{u_{5,2}, u_{13,2}, u_{14,2}\} = \max\{0, 0, 0\} = 0$. Then, we have $w_{2c} = w_{2d} = w_{2i} = v_{2c} = v_{2d} = v_{2i} = 0$. Similarly, we can apply the procedure to node 6 and 7. The objective value is 9 which is exactly the same as that of the solution obtained by the DP algorithm.

3.3.3 Projection of the Dummy Node Formulation

In this section, we project the dummy node formulation onto natural node space by projecting out arc variables. In other words, we obtain a formulation which only has the x variables by projecting out all y variables. Usually, there are two approaches for projecting out variables. One approach is Fourier-Motzkin elimination (which can easily be applied to project out the x variables). A more elegant method, proposed by Balas and Pulleyblank [1983], is based upon a theorem of the alternatives. We will follow this approach.

Given the dummy node formulation, we first replace constraint (3.50) by its equality form. It is easy to check that the LP relaxation of this new form is equivalent to that of the old one. Given a feasible solution (\mathbf{x}, \mathbf{y}) of the new form, it is a feasible solution for the old form already because the old form is a relaxation of the new one. Give a solution (\mathbf{x}, \mathbf{y}) for the LP relaxation of the old form, for an edges $\{i, d\}$ where $y_{id} + y_{di} < 1$, we can set $y_{id} = 1 - y_{di}$ to have this constraint binding. Thus, we obtain a solution for the LP relaxation of the formulation where constraint (3.50) is in its equality form and the objective value remains the same.

Next, we substitute out all y_{id} variables by $1 - y_{di}$ because $y_{id} + y_{di} = 1$. Then, we have the following formulation and denote its feasible region as P_{dummy} .

$$\text{Minimize} \quad \sum_{i \in V} b_i x_i \quad (3.68)$$

$$\text{Subject To } (u_{id}) \quad -y_{di} + x_j \geq 0 \quad \forall j \in V \ \& \ (j, d, i) \in E_t \quad (3.69)$$

$$(v_{id}) \quad -y_{di} - x_i \geq -1 \quad \forall i \in V, d \in a(i) \quad (3.70)$$

$$(w_i) \quad \sum_{d \in a(i)} y_{di} + g_i x_i \geq g_i \quad \forall i \in V \quad (3.71)$$

$$0 \leq x_i \leq 1, y_{di} \geq 0 \quad \forall i \in V \ \& \ d \in a(i) \quad (3.72)$$

Based on Theorem 2 in Balas and Pulleyblank [1983], the cone W is described by the following linear inequalities:

$$w_i - u_{id} - v_{id} \leq 0 \quad \forall i \in V \ \& \ d \in a(i) \quad (3.73)$$

$$w_i \geq 0, u_{id} \geq 0, v_{id} \geq 0 \quad \forall i \in V \ \& \ d \in a(i) \quad (3.74)$$

where \mathbf{u} , \mathbf{v} and \mathbf{w} are dual multipliers corresponding to constraints (3.69), (3.70) and (3.71)

respectively. If P_{dummy} can be represented as $\{A\mathbf{x} + G\mathbf{y} \geq \mathbf{b}\}$. Then, for any feasible vector $(\mathbf{w}, \mathbf{u}, \mathbf{v})$ to W , it defines a valid inequality: $(\mathbf{w}, \mathbf{u}, \mathbf{v})^T A\mathbf{x} \geq (\mathbf{w}, \mathbf{u}, \mathbf{v})^T \mathbf{b}$ in the space of the node (x) variables. Furthermore, the projection of P_{dummy} is defined by the valid inequalities defined by the extreme rays of W .

Theorem 3.5. *The vector $\mathbf{r} = (\mathbf{w}, \mathbf{u}, \mathbf{v}) \in W$ is extreme if and only if there exists a positive α such that it is in one of these three cases:*

1. $u_{id} = \alpha$ for one $\{i, d\} \in E_t$. Other $(\mathbf{w}, \mathbf{u}, \mathbf{v})$ are 0.
2. $v_{id} = \alpha$ for one $\{i, d\} \in E_t$. Other $(\mathbf{w}, \mathbf{u}, \mathbf{v})$ are 0.
3. $w_i = \alpha$ for one $i \in V$. Then for $d \in a(i)$, either $u_{id} = \alpha$ or $v_{id} = \alpha$. Other $(\mathbf{w}, \mathbf{u}, \mathbf{v})$ are 0.

Proof. Proof of Theorem 3.5. Recall that a polyhedral cone C is the intersection of a finite number of half-spaces through the origin, and a pointed cone is one of which the origin is an extreme point. A ray of a cone C is the set $R(\mathbf{y})$ of all non-negative multipliers of some $y \in C$, called the direction (vector) of $R(\mathbf{y})$. A vector $y \in C$ is extreme, if for any $y_1, y_2 \in C$, $y = \frac{1}{2}(y_1 + y_2)$ implies $y_1, y_2 \in R(\mathbf{y})$. A ray $R(\mathbf{y})$ is extreme if its direction vector y is extreme.

Sufficiency. Let $\mathbf{r} \in W$ be of the form Case 1 and assume that $\mathbf{r} = \frac{1}{2}(\mathbf{r}^1 + \mathbf{r}^2)$ for some $\mathbf{r}^1, \mathbf{r}^2 \in W$. Then, except u_{id}^1 and u_{id}^2 , all other directions are 0. Then, $\mathbf{r}^1, \mathbf{r}^2$ are in $R(\mathbf{r})$. So, r is extreme.

Case 2 is similar to Case 1.

For Case 3, let $\mathbf{r} \in W$ be of the form Case 3 and assume that $\mathbf{r} = \frac{1}{2}(\mathbf{r}^1 + \mathbf{r}^2)$ for

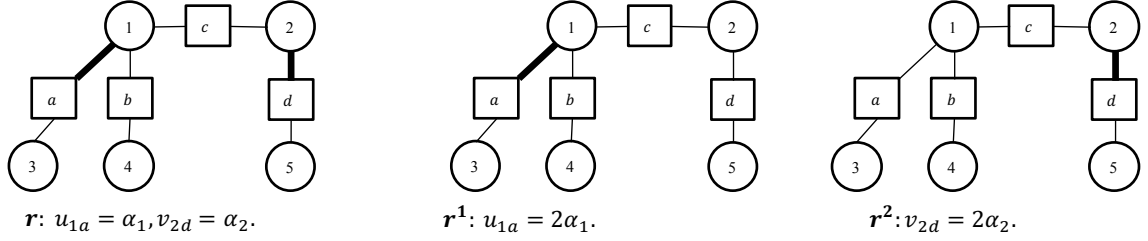


Figure 3.11: Theorem 3.5 necessity proof: $T = \emptyset$ and $|S^u| + |S^v| > 1$ example.

some $\mathbf{r}^1, \mathbf{r}^2 \in W$. So, for 0 components in \mathbf{r} , their corresponding components in \mathbf{r}^1 and \mathbf{r}^2 are also 0. Then, we have $w_i^1 + w_i^2 = 2\alpha$ and $p_{id}^1 + p_{id}^2 = 2\alpha$ where $p_{id}^k, k = 1, 2$, represent the positive component between u_{id}^k and $v_{id}^k, k = 1, 2$, for all $d \in a(i)$. Then, if there is a pair d_1 and d_2 , we have $p_{id_1}^1 > p_{id_2}^1$ if and only if $p_{id_1}^2 < p_{id_2}^2$. But the constraint (3.73) imposes that $w_i^k \leq p_{id}^k, k = 1, 2$. Hence, $p_{id_1}^k = p_{id_2}^k = \alpha_k, k = 1, 2$, for all $d_1, d_2 \in a(i)$. Otherwise, the constraint (3.73) would be violated. Therefore, $\mathbf{r}^1, \mathbf{r}^2$ are in $R(\mathbf{r})$. So, \mathbf{r} is extreme.

Necessity. Let \mathbf{r} be an extreme vector of W . Let $T = \{i \in V : w_i > 0\}$, $S^u = \{\{i, d\} \in E_t : u_{id} > 0\}$ and $S^v = \{\{i, d\} \in E_t : v_{id} > 0\}$ based on this \mathbf{r} . First, we consider the situation where $T = \emptyset$. If $|S^u| + |S^v| > 1$, we can have $\mathbf{r} = \frac{1}{2}(\mathbf{r}^1 + \mathbf{r}^2)$. Let \mathbf{r}^1 contain all but one positive components in \mathbf{r} and double their values. Let \mathbf{r}^2 contains the one positive component omitted by \mathbf{r}^1 and double its value. So, if $|S^u| + |S^v| > 1$, \mathbf{r} is not extreme, contrary to the assumption. We conclude that if $T = \emptyset$, then $|S^u| + |S^v| = 1$. Thus \mathbf{r} is either in Case 1 or in Case 2. In Figure 3.11, it shows an example for this situation. The bold line represents the positive u and v components in a vector r and the positive components are shown below the pictures.

Now consider the case when $T \neq \emptyset$. When $|T| > 1$, without loss of generality, let $i \in T$. Then, \mathbf{r}^1 has value $w_i^1 = 2w_i$ and $u_{id}^1 = 2u_{id}, v_{id}^1 = 2v_{id}$ for all $d \in a(i)$ and 0s

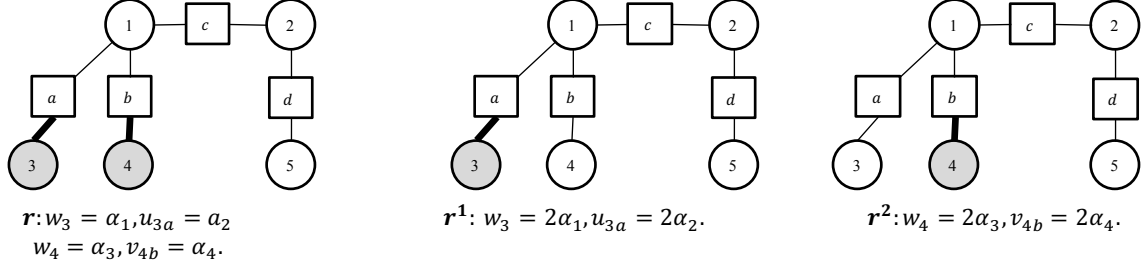


Figure 3.12: Theorem 3.5 necessity proof: $|T| > 1$ example.

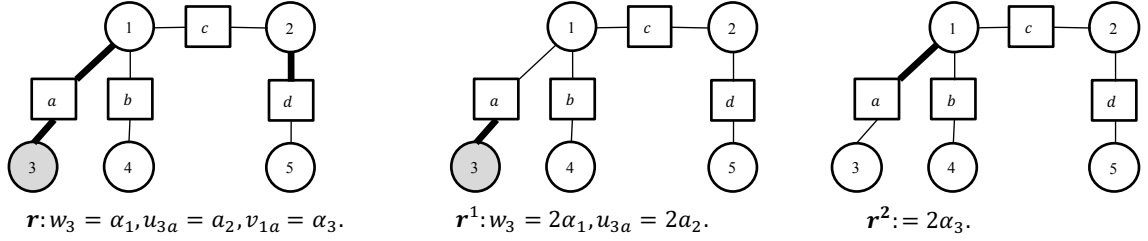


Figure 3.13: Theorem 3.5 necessity proof: $|T| = 1$ and $S_j \neq \emptyset$ example.

for other components. Then, let \mathbf{r}^2 be $2\mathbf{r} - \mathbf{r}^1$. Hence, we have $\mathbf{r} = \frac{1}{2}(\mathbf{r}^1 + \mathbf{r}^2)$. Also, r^2 does not have any positive components associated with node i . Thus, $\mathbf{r}^1, \mathbf{r}^2$ are different in at least one direction. So, if $|T| > 1$, \mathbf{r} is not extreme, contrary to the assumption. In Figure 3.12, it shows an example for this situation. The shaded nodes represent the positive w components in a vector r .

When $|T| = 1$ and $i \in T$, define $S_j = \{\{j, d\} \in E_t : p_{jd} > 0 \ \& \ j \in V \setminus i\}$. If $S_j \neq \emptyset$, let \mathbf{r}^1 have $w_i^1 = 2w_i$ and $u_{id}^1 = 2u_{id}, v_{id}^1 = 2v_{id}$ for all $d \in a(i)$ and 0s in other components. Then, let \mathbf{r}^2 be $2\mathbf{r} - \mathbf{r}^1$. Hence, we have $\mathbf{r} = \frac{1}{2}(\mathbf{r}^1 + \mathbf{r}^2)$. Also, r^2 has the positive components in S_j but r^1 does not. Thus, $\mathbf{r}^1, \mathbf{r}^2$ are different in at least one direction. So, if $|T| = 1$ and $S_j \neq \emptyset$, \mathbf{r} is not extreme, contrary to the assumption. In Figure 3.13, it shows an example for this situation and $S_j = \{\{1, a\}\}$.

When $|T| = 1$ and $i \in T$, define $S_1 = \{\{i, d\} \in E_t : u_{id} > 0 \oplus v_{id} > 0\}$ where only one of u and v variables for an edge $\{i, d\}$ is positive and $S_2 = \{\{i, d\} \in$

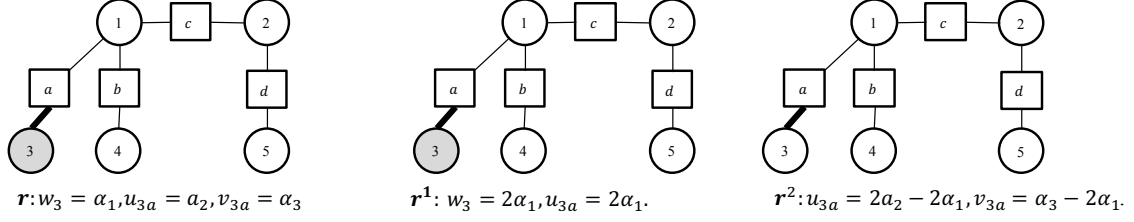


Figure 3.14: Theorem 3.5 necessity proof: $|T| = 1$ and $S_2 \neq \emptyset$ example.

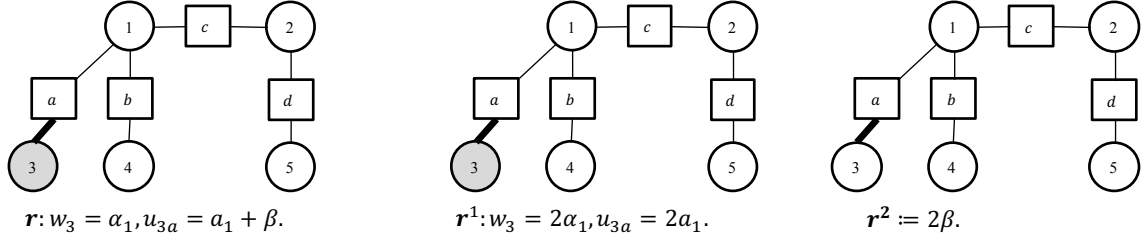


Figure 3.15: Theorem 3.5 necessity proof: $|T| = 1$ and $S^+ \neq \emptyset$ example.

$E_t : u_{id} > 0 \ \& \ v_{id} > 0$ where both u and v variables for an edge $\{i, d\}$ are positive. Suppose $S_2 \neq \emptyset$, then, we define $\alpha^1 = 2 \min\{w_i, u_{id}, v_{id} : \{i, d\} \in S_2\}$ and make \mathbf{r}^1 have $w_i^1 = \alpha^1$. For $\{i, d\} \in S_2$, we have $u_{id}^1 = \alpha^1$. Also, for $\{i, d\} \in S_1$, if $u_{id} > 0$, we have $u_{id}^1 = \alpha^1$. Otherwise, we have $v_{id}^1 = \alpha^1$. The rest components are 0s. Then, let \mathbf{r}^2 be $2\mathbf{r} - \mathbf{r}^1$. Hence, we have $\mathbf{r} = \frac{1}{2}(\mathbf{r}^1 + \mathbf{r}^2)$. Also, r^1 does not have any edges which have both u and v variables positive. Thus, $\mathbf{r}^1, \mathbf{r}^2$ are different in at least one direction. So, if $|T| = 1$ and $S_2 \neq \emptyset$, \mathbf{r} is not extreme, contrary to the assumption. Thus we must have $|S_1| = \deg(i)$ where $\deg(i)$ is node i 's degree number. Otherwise the constraint (3.73) would not be respected. In Figure 3.14, it shows an example for this situation. We have $S_2 = \{\{3, a\}\}$ and α_1 is the smallest value.

Next, if $|T| = 1$ and $|S_1| = \deg(i)$, let $w_i = \alpha$ and define $S^+ = \{\{i, d\} : p_{id} > \alpha\}$. When $S^+ \neq \emptyset$, without loss of generality, let $\{i, d\} \in S^+$ and $u_{id} > 0$, we can make \mathbf{r}^1 have $u_{id}^1 = 2(u_{id} - \alpha)$ and 0s in other components. Then, let \mathbf{r}^2 be $2\mathbf{r} - \mathbf{r}^1$. Hence, we have $\mathbf{r} = \frac{1}{2}(\mathbf{r}^1 + \mathbf{r}^2)$. Also, r^1 has only one positive component which is not w_i . Thus,

$\mathbf{r}^1, \mathbf{r}^2$ are different in at least one direction. So, if $|T| = 1$, $|S_1| = \text{deg}(i)$ and $S^+ \neq \emptyset$, \mathbf{r} is not extreme, contrary to the assumption. Then, we must have $S^+ = \emptyset$. In Figure 3.15, it shows an example for this situation. We have $S^+ = \{\{3, a\}\}$ and β is a positive value.

This proves that if $T \neq \emptyset$, we must have $|T| = 1$, $|S_1| = \text{deg}(i)$ and $S_2 = S_j = S^+ = \emptyset$. Thus, \mathbf{r} is in Case 3. \square

Applying Theorem 2 in Balas and Pulleyblank [1983], Case 1 and Case 2 extreme directions give the trivial constraints: $0 \leq x_i \leq 1$ for all $i \in V$. Case 3 extreme directions generate the following valid inequality in the original graph G :

$$(g_i - k)x_i + \sum_{j \in S} x_j \geq g_i - k \quad \forall i \in V \ \& \ k = 0, 1, \dots, \text{deg}(i) \ \& \ S \in C_i^{\text{deg}(i)-k}.$$

Here, we use $C_i^{\text{deg}(i)-k}$ to denote the set of all combinations with $\text{deg}(i) - k$ elements from node i 's neighbors and S is one combination picked from $C_i^{\text{deg}(i)-k}$. For a given i , if $k \geq g_i$, Case 3 extreme directions generate constraints that are redundant. Thus, the projection of P_{dummy} onto the \mathbf{x} space is the following one:

$$g_i x_i + \sum_{j \in n(i)} x_j \geq g_i \quad \forall i \in V \quad (3.75)$$

$$(g_i - k)x_i + \sum_{j \in S} x_j \geq g_i - k \quad \forall i \in V \ \& \ k = 1, 2, \dots, g_i - 1 \quad (3.76)$$

$$\ \& \ S \in C_i^{\text{deg}(i)-k}$$

$$0 \leq x_i \leq 1 \quad \forall i \in V \quad (3.77)$$

Constraint (3.75) is obtained when $k = 0$. We list it separately to emphasize that constraint (3.75) is same to constraint (3.34) in BIP_{saxena} . Also, constraint (3.76) is the

new one obtained from the projection. To illustrate this set of valid inequalities, using node 1 in Figure 3.7 as an example, we have $g_1 = 3$ and $n(i) = \{2, 3, 4\}$. So, when $k = 1$, $C_1^{3-1=2} = \{\{2, 3\}, \{2, 4\}, \{3, 4\}\}$. When $k = 2$, $C_1^{3-2=1} = \{\{2\}, \{3\}, \{4\}\}$. Except those lower and upper bound constraints, node 1's corresponding constraints in the projection are as follows:

$$3x_1 + x_2 + x_3 + x_4 \geq 3, \quad 2x_1 + x_2 + x_3 \geq 2, \quad 2x_1 + x_2 + x_4 \geq 2,$$

$$2x_1 + x_3 + x_4 \geq 2, \quad x_1 + x_2 \geq 1, \quad x_1 + x_3 \geq 1, \quad x_1 + x_4 \geq 1$$

Intuitively, for a node i and given the k value, the set of inequalities (3.76) can be interpreted as either node i is selected or at least $g_i - k$ nodes are selected among node i 's ($deg(i) - k$) neighbors. We recognize that the set of valid inequalities (3.76) is exponential many. The following proposition shows that the separation problem can be solved in polynomial time. Let δ denote the largest degree number among all nodes in V ($\delta = \max\{deg(i) : i \in V\}$).

Proposition 3.4. *The valid inequalities (3.76) can be separated in $O(|V|\delta \log \delta)$ time.*

Proof. Proof of Proposition 3.4. Given a fractional solution \mathbf{x}^* , a node i in V and a specific k where $k = 1, 2, \dots, g_i - 1$, the corresponding separation procedure of inequality (3.76) can be formulation as the following optimization problem:

$$\text{Minimize } (g_i - k)x_i^* + \sum_{j \in n(i)} x_j^* z_j \quad (3.78)$$

$$\text{Subject To } \sum_{j \in n(i)} z_j = deg(i) - k \quad (3.79)$$

$$z_j \in \{0, 1\} \quad \forall j \in n(i) \quad (3.80)$$

Algorithm 11 Separation algorithm for inequality set (3.76)

Require: A solution x^* and a PIDS instance.

```
1: for  $i \in V$  do
2:   Let  $S \leftarrow n(i)$ .
3:   for  $k = 1, 2, \dots, g_i - 1$  do
4:     let  $m_k = \arg \max\{x_i^* : i \in S\}$  and  $S \leftarrow S \setminus m_k$ .
5:     if  $(g_i - k)x_i^* + \sum_{j \in S} x_j^* < g_i - k$  then
6:       Add  $(g_i - k)x_i^* + \sum_{j \in S} x_j^* \geq g_i - k$ .
7:     else
8:       Break
9:     end if
10:  end for
11: end for
```

For each node i in V , if node i is selected, the binary variable z_i is 1. Otherwise, it is 0. If the objective value is smaller than $g_i - k$, we have a violated constraint. Otherwise, we either change the value of k or move to another node i . This optimization problem has one constraint and can be solved by taking the $\deg(i) - k$ smallest values of x_j^* among node i 's neighbors ($j \in n(i)$). We can use Algorithm 11 to separate the whole inequality set (3.76).

First, the solution x^* satisfies $g_i x_i^* + \sum_{j \in S} x_j^* \geq g_i$ for all i in V . For the inequality $(g_i - k)x_i^* + \sum_{j \in S} x_j^* < g_i - k$, as k increases by 1, the LHS decreases by $x_i^* + x_{m_k}^*$ where $x_{m_k}^*$ is the k th largest value of x_j^* for all j in $n(i)$ and the RHS decreases by 1. For the current iteration k_0 , if it is first time that $(g_i - k_0)x_i^* + \sum_{j \in S} x_j^* > g_i - k_0$. Then, it means $x_i^* + x_{m_{k_0}}^* < 1$. Then, in the future iteration, we will not find any violated constraint for this particular i because $x_i^* + x_{m_k}^* \leq x_i^* + x_{m_{k_0}}^*$ when $k > k_0$. Therefore, in Algorithm 11, we use **Break** in line 8. For each node, we sort its neighbors which takes at most $O(\delta \log \delta)$ steps and compare value at most δ times. The process is repeat for V nodes. So, the overall time complexity is $O(|V|\delta \log \delta)$. \square

3.3.4 Polyhedral Study of the PIDS Problem on General Graphs

In this section, we conduct a polyhedral study of the PIDS problem on general graphs based on the projection in Section 3.3.3. $\text{BIP}_{\text{dummy}}$ is not only a perfect formulation for trees, but also valid on general graphs as we shown in Proposition 3.3. Consequently, the following formulation based on the projection of $\text{BIP}_{\text{dummy}}$ is valid on general graphs as well.

$$(\text{BIP}_{\text{projection}}) \text{ Min} \quad \sum_{i \in V} b_i x_i \quad (3.81)$$

$$\text{S.T.} \quad (g_i - k)x_i + \sum_{j \in S} x_j \geq g_i - k \quad \forall i \in V \quad (3.82)$$

$$\& k = 0, 1, \dots, g_i - 1 \ \& S \in C_i^{\text{deg}(i)-k}$$

$$x_i \in \{0, 1\} \quad \forall i \in V \quad (3.83)$$

This formulation is referred to as $\text{BIP}_{\text{projection}}$ and its LP relaxation is referred to as $\text{LP}_{\text{projection}}$.

Let $P_{\text{projection}}$ denote the feasible area of $\text{LP}_{\text{projection}}$. Also, let $\mathbf{1} \in \mathbb{R}^{|V|}$ denote the vector that contains all 1s and $\mathbf{e}_i \in \mathbb{R}^{|V|}$ denote the vector that has 1 in i th position but 0s in all other positions. First of all, we show $P_{\text{projection}}$ has full dimensions.

Theorem 3.6. *$P_{\text{projection}}$ is full dimensional. In other words, the dimension of $P_{\text{projection}}$ is $|V|$.*

Proof. Proof of Theorem 3.6. We prove this by showing that there are $|V| + 1$ affinely independent points in $P_{\text{projection}}$. The following procedure is used to find these $|V| + 1$ points: The first point is $\mathbf{x}_0 = \mathbf{1}$. Then, we have another $|V|$ points: $\mathbf{x}_i = \mathbf{1} - \mathbf{e}_i$ for all i in V . Then, we obtain $|V|$ linearly independent points by $(\mathbf{x}_i - \mathbf{x}_0) = -\mathbf{e}_i$ for all

i in V . Hence, \mathbf{x}_0 and \mathbf{x}_i for all i in V are affinely independent. Furthermore, all these points are feasible in $P_{\text{projection}}$. First, \mathbf{x}_0 means we pick all nodes. Thus, it is feasible. Second, \mathbf{x}_i means we pick all nodes but node i . Then, node i 's threshold requirement is satisfied because all its neighbors are in the PIDS. Therefore, $P_{\text{projection}}$ has $|V| + 1$ affinely independent points and its dimension is $|V|$. \square

Knowing the dimension of $P_{\text{projection}}$, we study facet-defining conditions of the inequalities for $P_{\text{projection}}$. We start with the trivial constraints in $LP_{\text{projection}}$. We show that $x_i \leq 1$ for all i in V are facet defining and $x_i \geq 0$ for all i in V define faces for $P_{\text{projection}}$. Then, we present the conditions under which $x_i \geq 0$ for all i in V are facet defining for $P_{\text{projection}}$.

Proposition 3.5. *The trivial constraint $x_i \leq 1$ for all i in V are facet defining for $P_{\text{projection}}$.*

Proof. Proof of Proposition 3.5. Given a node i in V , when $x_i = 1$, we can find the following $|V|$ affinely independent points: The first one is $\mathbf{x}_0 = \mathbf{1}$. Then, we can have $|V|$ more as $\mathbf{x}_j = \mathbf{1} - \mathbf{e}_j$ for all j in $V \setminus i$. From these points, we can obtain $|V| - 1$ linearly independent points as $(\mathbf{x}_j - \mathbf{x}_0) = -\mathbf{e}_j$ for all j in $V \setminus i$. Hence, \mathbf{x}_0 and \mathbf{x}_j for all j in $V \setminus i$ are affinely independent. They are feasible points as showed in Theorem 3.6. Hence, $x_i \leq 1$ is a facet of $P_{\text{projection}}$. \square

Proposition 3.6. *For a node i in V , its corresponding trivial constraint $x_i \geq 0$ is a face of $P_{\text{projection}}$. Furthermore, $x_i \geq 0$ is facet defining for $P_{\text{projection}}$ if the following conditions are satisfied:*

1. $g_i \leq \text{deg}(i) - 1$.

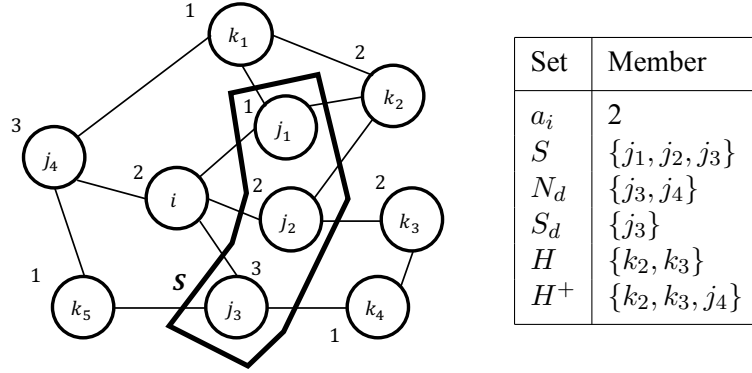


Figure 3.16: Illustration for notations in facet-defining proof of inequality (3.84).

2. For a node j in $n(i)$, if it does not share neighbors with node i ($n(j) \cap n(i) = \emptyset$), we need $g_j \leq \deg(j) - 1$. Otherwise, $g_j \leq \deg(j) - 2$

Proof. Proof of Proposition 3.6. $x_i \geq 0$ is satisfied with equality because we have the feasible point $\mathbf{x}_i = \mathbf{1} - \mathbf{e}_i$. Thus, $x_i \geq 0$ is a face of $\mathbf{P}_{\text{projection}}$. When Condition 1 and 2 of Proposition 3.6 are satisfied, we can show $x_i \geq 0$ is a facet of $\mathbf{P}_{\text{projection}}$. Let $F(i) = \{j \in n(i) : g_j \leq \deg(j) - 1 \text{ if } n(i) \cap n(j) = \emptyset \ \& \ g_j \leq \deg(j) - 2 \text{ if } n(i) \cap n(j) \neq \emptyset\}$. It is the subset of node i 's neighbors which satisfy Condition 2. Then, we can find the following points: $\mathbf{x}_i = \mathbf{1} - \mathbf{e}_i$. Then, $\mathbf{x}_j = \mathbf{x}_i - \mathbf{e}_j$ for all j in $F(i)$, and $\mathbf{x}_k = \mathbf{x}_i - \mathbf{e}_k$ for all k in $\{V \setminus n(i)^+\}$ where $n(i)^+ = n(i) \cup i$. First, if $g_i = \deg(i)$, $\mathbf{x}_j = \mathbf{x}_i - \mathbf{e}_j$ for all j in $F(i)$ are not feasible for $\mathbf{P}_{\text{projection}}$ because node i needs all its neighbors. But they are feasible when $g_i \leq \deg(i) - 1$. Second, \mathbf{x}_i and \mathbf{x}_k for all k in $\{V \setminus n(i)^+\}$ are feasible and distinct points in $\mathbf{P}_{\text{projection}}$. Therefore, we have $|V| - |n(i)| - 1 + |F(i)| + 1$ feasible points in $\mathbf{P}_{\text{projection}}$. Then, $\mathbf{x}_j - \mathbf{x}_i = -\mathbf{e}_j$ for all j in $V \setminus n(i)^+ \cup F(i)$ are linearly independent. When $|F(i)| = |n(i)|$, we have $|V|$ affinely independent points and $x_i \geq 0$ is facet defining for $\mathbf{P}_{\text{projection}}$. \square

Next, we study the valid inequality in the form of (3.82) and present the conditions

under which it is facet defining. Given an inequality in the form of (3.82) with the specific values of i and k , and the set S , we write it in the following form for the ease of explanation:

$$a_i x_i + \sum_{j \in S} x_j \geq a_i \quad (3.84)$$

where $a_i = g_i - k$, and $a_i \leq |S|$. In Figure 3.16, we have one example which is used to illustrate the notations we use. The number beside a node is its threshold value. Here, we have $a_i = 2$ and $S = \{j_1, j_2, j_3\}$. In order to show when it is facet defining, we first prove the following lemmas:

The first lemma states that if $x_i = 1$ and the given inequality is binding, for a node j in S , it must have at least g_j neighbors which are not in S . Thus, for a node j in S , $x_j = 0$ is allowed for a feasible point in $P_{\text{projection}}$. Hence, the valid inequality in the form of (3.84) can be binding when $x_i = 1$.

Lemma 3.1. *If the given inequality (3.84) is satisfied with equality when $x_i = 1$, we must have the following condition: $g_j \leq \deg(j) - |n(j) \cap S|$ for all j in S .*

Proof. Proof of Lemma 3.1. For a node j in S , given that $x_i = 1$ and the inequality is binding, we cannot select node j . Thus, to satisfy node j 's threshold requirement, g_j of node j 's neighbors should be selected (recall that $x_i = 1$ and node i is adjacent to node j). So, we need to make sure that among node j 's neighbors, those not in S are enough to satisfy node j 's threshold requirement ($g_j \leq \deg(j) - |n(j) \cap S|$). Otherwise, it means node j has not enough neighbors except those in S to satisfy its threshold requirement. Then, we must select some nodes from S to obtain a feasible point in $P_{\text{projection}}$. Then, the

inequality is not binding. A contradiction is found. \square

Next, we consider the situation that $x_i = 0$. The second lemma states that if $x_i = 0$ and the given inequality is binding, among the members in S , the number of those requiring all their neighbors is at most a_i . Let $N_d = \{j \in n(i) : g_j = \deg(j)\}$ and $S_d = N_d \cap N_d$. In Figure 3.16, we have $N_d = \{j_3, j_4\}$ and $S_d = \{j_3\}$.

Lemma 3.2. *If the given inequality is satisfied with equality by setting $x_i = 0$, we must have the following condition: $|S_d| \leq a_i$.*

Proof. Proof of Lemma 3.2. If this condition is violated, then, $|S_d| > a_i$. Thus, given that $x_i = 0$, we must have $x_j = 1$ for all j in S_d . Then, $a_i x_i + \sum_{j \in S} x_j \geq \sum_{j \in S_d} x_j > a_i$. It is a contradiction. \square

In Lemma 3.3, we show that both Lemma 3.1 and 3.2 are necessary.

Lemma 3.3. *If the given inequality (3.84) is facet defining, it must satisfy Lemma 3.1 and 3.2.*

Proof. Proof of Lemma 3.3. First, if the given inequality violates both Lemma 3.1 and 3.2, it cannot be facet defining because it would not satisfy inequality (3.84) with equality. Second, if it only satisfies Lemma 3.1 but not Lemma 3.2, we can only have the inequality binding when $x_i = 1$. Also, at most $|V| - |S|$ affinely independent points in $P_{\text{projection}}$ can be found because $x_j = 0$ for all j in S . Third, if it only satisfies Lemma 3.2 but not Lemma 3.1, we can only have the inequality binding when $x_i = 0$. then, at least one node in S , denoted by j , violates the condition that $g_j \leq \deg(j) - |n(j) \cup S|$. Among neighbors of node j , the number of those not in S is less than g_i . Thus, if node j is not selected, at

least one other nodes in S must be selected given that $x_i = 0$ and node i is adjacent to node j . Then, we can find at most $|V| - 1$ affinely independent points. \square

Given Lemma 3.3, we have the following two situations for a binding inequality (3.84): First, when $x_i = 1$, we have $x_j = 0$ for all j in S . Then, except node i , for a node k adjacent to a node j in S , if it has more than $deg(k) - g_k$ neighbors in S , we must select node k (i.e., $x_k = 1$) in a feasible point in $\mathbf{P}_{\text{projection}}$. We use H to denote the set of this kind of nodes. In Figure 3.16, for node k_2 , $deg(k_2) - g_{k_2} = 1$ and $|n(k_2) \cap S| = 2$. Also, for node k_3 , $deg(k_3) - g_{k_3} = 0$ and $|n(k_3) \cap S| = 1$. Thus, $H = \{k_2, k_3\}$. Hence, we can have $T_1 = |V| - |S| - |H| + 1$ points in $\mathbf{P}_{\text{projection}}$ in the following way: Let $\mathbf{x}_0 = \mathbf{1} - \sum_{j \in S} \mathbf{e}_j$. Then, let $\mathbf{x}_j = \mathbf{x}_0 - \mathbf{e}_j$ for all j in $\{V \setminus (S^+ \cup H)\}$.

Second, when $x_i = 0$, we have $x_j = 1$ for all j in N_d because they require all their neighbors but node i is not selected (recall that $N_d = \{j \in n(i) : g_j = deg(j)\}$). Then, we need to choose $a_i - |S_d|$ nodes from the set $S \setminus S_d$ to have this inequality satisfied with equality. Let $C_{S \setminus S_d}^{a_i - |S_d|}$ be the set of all combinations. Also, similarly to H , we define a set H^+ . For a node k in H^+ , it is adjacent to a node in S^+ but not in S^+ and it has more than $deg(k) - g_k$ neighbors in S^+ . Thus, nodes in H^+ must be selected if we do not selected any nodes in S^+ . In Figure 3.16, in addition to node k_2 and k_3 , node j_4 has $deg(j_4) - g_{k_3} = 0$ and $|n(j_3) \cap S^+| = 1$. Thus, $H^+ = \{k_2, k_3, j_4\}$. However, some nodes in H^+ do not have to be selected in a feasible point of in $\mathbf{P}_{\text{projection}}$. Given a C in $C_{S \setminus S_d}^{a_i - |S_d|}$, for a node k in H , we know all nodes in $\{C \cup S_d\}$ are selected. Therefore, if this node k has less than or equal to $deg(k) - g_k + |n(k) \cap \{C \cup S_d\}|$ neighbors in S , we do not have to select node k (i.e., $x_k = 1$) in a feasible point in $\mathbf{P}_{\text{projection}}$. We use $H(C)$ to denote the set of this kind of nodes. Then,

we can have following points in $\mathbf{P}_{\text{projection}}$: for a C in $C_{S \setminus S_d}^{a_i - |S_d|}$, $\mathbf{x}_{C0} = \mathbf{1} - \sum_{j \in S^+ \setminus \{C \cup S_d\}} \mathbf{e}_j$.

Next, let $\mathbf{x}_{Cj} = \mathbf{x}_{C0} - \mathbf{e}_j$ for all j in $\{H(C) \cup V \setminus (S^+ \cup H^+)\}$. Then, we can find

$$T_0 = \sum_{C \in C_{S \setminus S_d}^{a_i - |S_d|}} (|H(C)| + |V| - |S^+| - |H^+| + 1) \text{ points in } \mathbf{P}_{\text{projection}}.$$

If $T_0 + T_1 \geq V$, this given inequality (3.84) is facet defining if for $(\mu, \mu_0) = \lambda \mathbf{y}$, $\lambda \neq 0$ and \mathbf{y} has a_i in the i th position, 1 in the j th position for all j in S , a_i in the 0th position and 0s in all other positions is the only solution for the following equation system with $V + 1$ unknowns (μ, μ_0) :

$$\mu_i + \sum_{h \in V \setminus S^+} \mu_h = \mu_0 \quad (3.85)$$

$$\mu_i + \sum_{h \in V \setminus S^+} \mu_h - \mu_j = \mu_0 \quad \forall j \in \{V \setminus (S^+ \cup H)\} \quad (3.86)$$

$$\sum_{k \in C \cup S_d} \mu_k + \sum_{h \in V \setminus S^+} \mu_h = \mu_0 \quad \forall C \in C_{S \setminus S_d}^{a_i - |S_d|} \quad (3.87)$$

$$\sum_{k \in C \cup S_d} \mu_k + \sum_{h \in V \setminus S^+} \mu_h - \mu_j = \mu_0 \quad \forall C \in C_{S \setminus S_d}^{a_i - |S_d|}, \quad (3.88)$$

$$j \in \{H(C) \cup V \setminus (S^+ \cup H^+)\}$$

Lemma 3.4. *If the given inequality (3.84) is facet defining, we must have*

1. N_d is an empty set.
2. $a_i \leq |S| - 1$ unless $|S| = 1$.

Proof. Proof of Lemma 3.4. Now, assume for \mathbf{y} , we can have 0s for all positions in $V \setminus S^+$.

So, we must have $N_d \setminus S_d = \emptyset$. It means $N_d = S_d$. Then, we can simplify the equation system of (3.85) to (3.88) as:

$$\mu_i = \mu_0 \quad (3.89)$$

$$\sum_{k \in S_d} \mu_k + \sum_{k \in C} \mu_k = \mu_0 \quad \forall C \in C_{S \setminus S_d}^{a_i - |S_d|} \quad (3.90)$$

We know $C_{S \setminus S_d}^{a_i - |S_d|}$ be the set of all combinations for choosing $a_i - |S_d|$ nodes from the set $S \setminus S_d$. It is easy to see that $u_k = \frac{\mu_0}{a_i}$ for all k in S is a solution for (3.89) and (3.90). However, it is not the only solution. Depending on the value of $\sum_{k \in S_d} \mu_k$, we can have infinitely many solutions by setting $u_k = \frac{\mu_0 - \sum_{j \in S_d} \mu_j}{a_i - |S_d|}$ for all k in $S \setminus S_d$. In order to obtain the desired \mathbf{y} and ensure that it is the only solution. First, we need $S_d = \emptyset$. Thus, the solution does not depend on the value of $\sum_{k \in S_d} \mu_k$ anymore. Second, it must be the only solution. Thus, when $|S| \geq 2$, we need $a_i \leq |S| - 1$. Then, from $C_{S \setminus S_d}^{a_i - |S_d|}$ we can take any two combinations which have two different positions, say k_1 and k_2 , and minus one to the other. We will obtain $u_{k_1} = u_{k_2}$. Repeat this process, we have $u_{k_1} = u_{k_2}$ for any two distinct k_1 and k_2 in S . When $|S| = 1$, we have $u_k = \mu_0$ because $a_i = 1$. \square

For a node j in H , let $\beta_j = \max\{0, g_j - (deg(j) - |n(i) \cap S|)\}$. Here, $deg(j) - |n(i) \cap S|$ the number of node j 's neighbors not in S . Assume all these neighbors selected. Then, β_j is the least number of nodes it needs from S if node j is not selected.

Lemma 3.5. *If the given inequality (3.84) is facet defining, we must satisfy this condition:*

1. $\beta_j \leq a_i$ for each node j in H .

Consequently, we can have 0s in all positions in $V \setminus S^+$.

Proof. Proof of Lemma 3.5. Based on the equation system (3.85) to (3.88), take the equation (3.85) to minus the j th equation from the equation set (3.86), we have $u_j = 0$ for all j in $\{V \setminus (S^+ \cup H)\}$.

Similarly, for a given C in $C_{S \setminus S_d}^{a_i - |S_d|}$, we have $u_j = 0$ for all j in $\{V \cup H(C) \setminus (S^+ \cup H^+)\}$. Then, compared to those 0's positions based on equations (3.85) and (3.86),

we have additional 0 positions based on equations (3.87) and (3.88). Those additional 0 positions are in $H(C)$. Then, if $H \subseteq \cup_{C \in \mathcal{C}_{S \setminus S_d}^{a_i - |S_d|}} H(C)$, we can have all positions in H are 0s. Thus, we have 0s in all positions except these positions in $V \setminus S^+$. For a node j in H , if $\beta_j > a_i$, it cannot be any $H(C)$ because we can only select a_i nodes from S if $x_i = 0$. Otherwise, there must be one combination C which covers at least β_j node j 's neighbors. Thus, if it holds for all nodes in H , we have $H \subseteq \cup_{C \in \mathcal{C}_{S \setminus S_d}^{a_i - |S_d|}} H(C)$. \square

Combining Lemma 3.4 and 3.5, we can have the desired vector \mathbf{y} as the only solution for the linear equation system. Thus, from the analysis above, the following theorem is proposed:

Theorem 3.7. *The given inequality (3.84) is facet defining if and only if the following four conditions are satisfied:*

1. $g_j \leq \deg(j) - |n(j) \cap S|$ for all j in S (Lemma 3.1).
2. N_d is an empty set (Lemma 3.4).
3. $a_i \leq S - 1$ unless $S = 1$ (Lemma 3.4).
4. $\beta_j \leq a_i$ for each node j in H recall that $\beta_j = \max\{0, g_j - (\deg(j) - |n(i) \cap S|)\}$ (Lemma 3.5).

Proof. Proof of Theorem 3.7. *Sufficiency.* First, we show that we are able to obtain $|V|$ feasible points and the linear equations system as (3.85) to (3.88) with these three Lemmas. When Condition 1 and 2 (i.e., Lemma 3.1 and Condition 1 in Lemma 3.4) are satisfied, we can find $T_1 = |V| - |S| - |H| + 1$ feasible points by setting $x_i = 1$ and $T_0 =$

$\sum_{C \in C_{S \setminus S_d}^{a_i - |S_d|}} |H(C)| + |C_{S \setminus S_d}^{a_i - |S_d|}|(|V| - |S^+| - |H^+| + 1)$ feasible points by setting $x_i = 0$.

Then,

$$T_0 + T_1 = |V| - |S| - |H| + 1 + \sum_{C \in C_{S \setminus S_d}^{a_i - |S_d|}} |H(C)| + |C_{S \setminus S_d}^{a_i - |S_d|}|(|V| - |S^+| - |H^+| + 1) \quad (3.91)$$

$$\geq |V| - |S| + |C_{S \setminus S_d}^{a_i - |S_d|}|(|V| - |S| - |H^+|) \quad (3.92)$$

$$\geq |V| - |S| + |C_{S \setminus S_d}^{a_i - |S_d|}| = |V| - |S| + |C_S^{a_i}| \quad (3.93)$$

$$\geq |V| \quad (3.94)$$

We can go from (3.91) to (3.92) because of Condition 3 (i.e., Condition 2 in Lemma 3.4).

Then, $H \subseteq \cup_{C \in C_{S \setminus S_d}^{a_i - |S_d|}} H(C)$. Hence, $\sum_{C \in C_{S \setminus S_d}^{a_i - |S_d|}} |H(C)| \geq |H|$. Also, $|S^+| = |S| + 1$.

Then, we can go from (3.92) to (3.93) due to the fact that $|V| - |S| - |H^+| \geq 1$ because at least node i is left. Lastly, we know $C_{S \setminus S_d}^{a_i - |S_d|}$ is the set of all combinations for choosing a_i nodes from the set S because of Condition 2 (i.e., Condition 1 in Lemma 3.4), $S_d = \emptyset$. Thus, $|C_S^{a_i}| \geq |S|$ because we have Condition 3 (i.e., Condition 2 in Lemma 3.4), $1 \leq a_i \leq |S| - 1$ unless $|S| = 1$, and it holds when $|S| = 1$ as well.

Now, Condition 2 satisfies Lemma 3.2 as well. Also, Condition 2, 3 and 4 (i.e., Lemmas 3.4 and 3.5) ensure that $(\mu, \mu_0) = \lambda \mathbf{y}$ where $\lambda \neq 0$ and \mathbf{y} which has a_i in the i th position, 1 in the j th position for all j in S , a_i in the 0th position and 0s in all other positions is the only solution for the linear equations system as (3.85) to (3.88). Thus, the given inequality is facet defining for $P_{\text{projection}}$ when these conditions are satisfied.

Necessity. It is already proved in Lemma 3.1, 3.3, 3.4 and 3.5. \square

3.3.5 Computational Experiments

In this section, we conduct computational experiments to show that solving the LP relaxation of the dummy node formulation can obtain a tighter dual bound compared to other formulations. Furthermore, we solve random generated instances to compare the performance between $\text{BIP}_{\text{saxena}}$, $\text{BIP}_{\text{baïou}}$, $\text{BIP}_{\text{dummy}}$ and $\text{BIP}_{\text{projection}}$.

We generate networks using the method proposed by Watts and Strogatz [1998]. As indicated in the Stanford large network dataset collection [Leskovec, 2011], real social networks are sparse. We took this into account as follows: For sparsity, we generated network with average degree number 8. We chose the rewiring probability p as 0.3 (loosely, it is the probability that an edge is reconnected to a uniformly chosen node after initializing a ring with pre-specified average degree), because Watts and Strogatz [1998] showed this corresponds most closely to the social networks they studied. We randomly generated node type g_i from a discrete uniform distribution between $[1, \text{deg}(i)]$ and cost b_i from a discrete uniform distribution between $[1, 50]$. We used CPLEX 12.6 with Python API and ran our tests on a machine with the following specifications: Intel i5 3.40GHz, 24 GB ram, Ubuntu 14.04.

In our first set of experiments we study the strength of the LP relaxation. For this, ten 200-node instances are generated. First, we compare the strength of LP relaxations. Recall that z_{dummy} , z_{saxena} and $z_{\text{baïou}}$ denote the optimal objective values for LP_{dummy} , $\text{LP}_{\text{saxena}}$ and $\text{LP}_{\text{baïou}}$ respectively. According to Theorem 3.4, $z_{\text{saxena}} = z_{\text{baïou}}$. Thus, we only show z_{saxena} here. In Table 3.1, the first two columns are the objective values of $\text{LP}_{\text{saxena}}$ and LP_{dummy} . The third column has the optimal value, denoted by z^* , obtained by solving the formulation

	z_{saxena}	z_{dummy}	z^*	z_{saxena}/z^*	z_{dummy}/z^*
1	1913.61	2379.86	2653	72.13%	89.70%
2	1724.29	2185.48	2439	70.70%	89.61%
3	1768.73	2251.48	2558	69.14%	88.02%
4	1922.12	2410.15	2712	70.87%	88.87%
5	1599.83	2029.32	2252	71.04%	90.11%
6	1800.61	2274.33	2569	70.09%	88.53%
7	1888.10	2393.09	2663	70.90%	89.86%
8	1639.24	2140.79	2329	70.38%	91.92%
9	1699.09	2108.43	2277	74.62%	92.60%
10	1832.95	2278.45	2505	73.17%	90.96%

Table 3.1: LP Relaxations of $\text{BIP}_{\text{saxena}}$ and $\text{BIP}_{\text{dummy}}$ with 200-node instances.

with binary constraints. The last two columns are the relative dual bounds of $\text{LP}_{\text{saxena}}$ and LP_{dummy} . The dummy node formulation can improve the dual bound significantly. On average, the dual bound of $\text{BIP}_{\text{saxena}}$ is about 71% of the optimal value. However, LP_{dummy} is about 90%.

In the second experiment, we use all four formulations to solve these ten 200-node instances. In this experiment, a five minutes time limit is imposed for each instance. Also, the CPLEX default setting is used except that all CPLEX cuts are disabled so that the performance of these four formulations can be compared directly. Table 3.2 contains running time in seconds for all instances. If an instance hit the time limit, "o.o.t" is presented. The $\text{BIP}_{\text{saxena}}$ formulation cannot solve any instances to optimality within the time limit. The $\text{BIP}_{\text{baïou}}$ formulation is able to solve one instance in about 176 seconds. However, $\text{BIP}_{\text{dummy}}$ and $\text{BIP}_{\text{projection}}$ can solve all ten instances. In the last column, the number is in **boldface** if the running time is improved. Furthermore, $\text{BIP}_{\text{projection}}$ improves the running

	BIP _{saxena}	BIP _{baiou}	BIP _{dummy}	BIP _{projection}
1	o.o.t	o.o.t	140.27	32.53
2	o.o.t	o.o.t	66.42	27.55
3	o.o.t	o.o.t	100.67	71.74
4	o.o.t	o.o.t	41.91	37.78
5	o.o.t	o.o.t	12.96	15.83
6	o.o.t	o.o.t	123.53	71.58
7	o.o.t	175.47	5.96	9.91
8	o.o.t	o.o.t	16.37	11.36
9	o.o.t	o.o.t	10.40	7.14
10	o.o.t	o.o.t	17.15	9.40

Table 3.2: running time of four formulations in seconds with 200-node instances

time for 8 out of 10 instances and the average running time is reduced from 53.56 seconds to 29.48 seconds when we switch from BIP_{dummy} to BIP_{projection}. The improvement mostly comes from those hard instances. Instance 1, 3, and 6 are reduced from 140.27, 100.67, and 123.53 seconds to 32.53, 71.74 and 71.58 seconds respectively. Although compared BIP_{dummy} instance 5 and 7 need more running time when BIP_{projection} is used, they are easy ones and the increment is not significant. The increment in running time is because in the BIP_{projection} formulation, we need to separate and add violated inequalities dynamically and this procedure can be relatively more expensive for these easy instances.

	1000			2000			5000		
	z_{saxena}	z_{dummy}	Rel.Imp.	z_{saxena}	z_{dummy}	Rel.Imp.	z_{saxena}	z_{dummy}	Rel.Imp.
1	9063.41	11306.83	24.75%	18145.53	22608.60	24.60%	45059.02	56325.24	25.00%
2	8682.95	10814.24	24.55%	17761.77	22287.07	25.48%	44599.94	55480.87	24.40%
3	9099.06	11357.96	24.83%	17771.79	22349.04	25.76%	44743.56	56221.35	25.65%
4	8947.80	11151.28	24.63%	17959.77	22590.05	25.78%	44724.15	56272.48	25.82%
5	9147.79	11381.91	24.42%	17763.57	22355.18	25.85%	44929.47	56097.74	24.86%
6	8930.28	11236.03	25.82%	17955.23	22396.24	24.73%	44910.13	56297.54	25.36%
7	9144.19	11393.59	24.60%	18017.03	22443.43	24.57%	45133.55	56479.87	25.14%
8	9155.03	11481.79	25.42%	18160.80	22582.92	24.35%	45191.27	56515.55	25.06%
9	8668.83	10894.40	25.67%	17757.40	22565.41	27.08%	45016.29	56489.13	25.49%
10	8775.68	11110.74	26.61%	17562.47	22114.51	25.92%	45822.99	57353.89	25.16%

Table 3.3: Linear Programming Relaxation Comparison. (Rel.Imp. = $\frac{z_{\text{dummy}} - z_{\text{saxena}}}{z_{\text{saxena}}}$)

Lastly, in the third experiment, we test LP_{saxena} and LP_{dummy} on larger instances. The purpose of this experiment is to show that the dummy node formulation can improve the LP relaxation significantly. We set the number of nodes to 1000, 2000 and 5000 and generate instances by the method described above. Table 3.3 contains the results. For each size of instances, it has three columns. The first two columns are for the LP relaxation values. The third column is the relative improvement of LP_{dummy} compared to LP_{saxena} which is calculated as $\frac{z_{\text{dummy}} - z_{\text{saxena}}}{z_{\text{saxena}}}$. The improvement is stable in between 24% and 27% in our experiment. In terms of running time, for 1000-node instances, LP_{saxena} takes about 0.1 seconds and LP_{dummy} needs about 10 seconds on average. For 2000-node instances, they increase to 0.5 seconds and 54 seconds on average, respectively. For 5000-node instances, they further increase to 8 seconds and 380 seconds on average, respectively. Therefore, the proposed formulation is able to improve the linear relaxation greatly but we pay a price in running time.

3.4 Conclusions

In this chapter we studied the PIDS and TPIDS problems. We show that both problems on trees can be solved in $O(|V|)$ time. More importantly, for the TPIDS problem, we show that the natural adaptations of those formulations in Bouchakour et al. [2008], Saxena [2004], Baïou and Barahona [2014] describes the polytope on trees. For the PIDS problem on trees, a novel tight and compact extended formulation is presented. This leads to our third contribution. We project the extended formulation onto the natural node variables space and prove that it is stronger than the natural adaptations of the formulations

in Bouchakour et al. [2008], Saxena [2004], Baïou and Barahona [2014]. From there, we derive a new set of valid inequalities for the PIDS problem and provide a polynomial time separation procedure for it. Based on our computational experiment, our formulation provides stronger linear relaxation bounds. Our work is a building block for developing exact approaches for these two problems.

Chapter 4: Tailored Incentives and Least Cost Influence Maximization on Social Networks

4.1 Introduction

The model studied by earlier researchers suffers from a significant practical shortcoming. It restricts the marketer to interventions where those selected for targeting receive the product gratis. Motivated by practical considerations we consider a version of this viral marketing problem where an individual can be partially influenced by the use of monetary inducements (i.e., coupons that reduce the price of a product instead of receiving the product for free). We believe this is a crucial aspect, and the use of tailored (i.e., partial) incentives which allows for differentiated targeting is more natural in a marketing setting. The problem we study is in a deterministic setting, and seeks to minimize the cost of tailored incentives provided to individuals in a social network while ensuring a given fraction of the network is "influenced". We refer to this problem as the *Least Cost Influence Problem* (LCIP). Günneç [2012] and Günneç and Raghavan [2016] first describe the LCIP, where it arose in a product design setting that took into account social network effects. Subsequently, and parallel to this work, there are only two other papers [Demaine et al., 2014, Cordasco et al., 2015] in this area that discuss partial incentives in the viral

marketing context.

4.1.1 Problem Definition

Consider a social network represented as an undirected graph $G = (V, E)$, where node set $V = \{1, 2, \dots, n\}$ denotes the set of people in the network and edge set E shows the connections between people on the social network. Following a well-studied linear threshold model on the diffusion of innovations [Granovetter, 1978], we will use the term that a node is active if it has adopted the product and the term that a node is inactive if it has not adopted the product. In the threshold model, each inactive node $i \in V$ is influenced by an amount d_{ij} (referred to as the *influence factor*) by its neighbor node j (i.e., there is an edge in the graph between nodes i and j) if node j is active (i.e., has already adopted the product). For each node in the network, $i \in V$, there is a threshold, denoted by b_i . This threshold represents how easily a node can be influenced. We permit a payment p_i which is the tailored incentives for a node $i \in V$. Also, α is given as the desired penetration rate taking values between 0 and 1 ($0 \leq \alpha \leq 1$).

All nodes are inactive initially. Then, we decide the tailored incentives p_i for each node $i \in V$. Now, a node i becomes active immediately if $p_i \geq b_i$ (i.e., if the payment is greater than the threshold). After that, in each step, we update the states of nodes by the following rule: an inactive node i becomes active if the sum of the tailored incentive p_i and the total influence coming from its active neighbors is at least b_i . The process continues until there is no change in the state of the network (i.e., no additional nodes are becoming active). The goal is to find the minimum total payment (e.g., $\sum_{i \in V} p_i$) while ensuring that

at least $\alpha|V|$ nodes are active by the end of this activation process.

Note that the assumption that all nodes are inactive is without loss of generality. If some nodes were active at the outset we can propagate their influence and reduce the problem to a smaller one where all nodes are inactive initially. We also note that in a deterministic setting there is no benefit to delaying the payment of the tailored incentive. Hence, all incentives paid to a node i may be viewed as being paid at the outset of the process.

A simple integer programming model for the LCIP introduces the notion of time periods $t = 1, 2, \dots, T$ when nodes can become active. This is to capture the order in which nodes become active in the social network. Binary variable y_{it} denotes whether node i is active in time period t (note that in the diffusion model once a node becomes active it remains active), and so $y_{iT} = 1$ indicates node i is active at the end of the diffusion process. Let $a(i)$ be the neighbor set of node i . The formulation is as follows:

$$(MIP4.1) \quad \min \quad \sum_{i \in V} p_i \quad (4.1)$$

$$\text{s.t.} \quad y_{i0} = 0 \quad \forall i \in V \quad (4.2)$$

$$p_i + \sum_{j \in a(i)} d_{ij} y_{j(t-1)} \geq b_i y_{it} \quad \forall i \in V, t = 1, \dots, T \quad (4.3)$$

$$\sum_{i \in V} y_{iT} \geq \alpha|V| \quad (4.4)$$

$$y_{it} \in \{0, 1\}, p_i \geq 0 \quad \forall i \in V, t = 1, 2, \dots, T. \quad (4.5)$$

Here, the objective (4.1) is to minimize the sum of the incentives given over the network. Constraint set (4.2) models the initial condition that all nodes are inactive initially. Constraint set (4.3) models the diffusion process---an inactive node i becomes active if the

sum of the tailored incentive p_i and the total influence coming from its active neighbors is at least b_i . Constraint set (4.4) ensures that the desired penetration rate is achieved at the end of the diffusion process. Note that since there are $|V|$ nodes, the number of time indices required for the diffusion process to complete should be less than or equal to $|V|$. However, since we do not know a priori how quickly the diffusion process terminates we set $T = |V|$.

4.1.2 Related Literature

As mentioned earlier Kempe et al. [2003] considered a budgeted version of the problem in a randomized setting that we will refer to as the *Influence Maximization Problem* (IMP). They make a particular assumption (for technical convenience and on which the submodularity property their results critically depend upon) on the distribution of the threshold values b_i . Roughly, this can be interpreted as the thresholds b_i being distributed uniformly in the range $[0, L]$ where $L = \max_{i \in V} \{\max\{b_i, \sum_{j \in a(i)} d_{ij}\}\}$.¹ Further, their objective is to maximize the number of people influenced given that only k individuals are allowed to be fully influenced (i.e., provided the product for free).

Recently, and subsequent to Günneç [2012], Demaine et al. [2014] presented a fractional version of the IMP considered by Kempe et al. [2003]. In their model, identically to the LCIP, nodes can be partially influenced via a payment, and the goal is to maximize the number of nodes influenced for a given budget. They retain the same technical assumption as Kempe et al. [2003] on the uniform distribution of thresholds and consider

¹In their notation the threshold values lie between 0 and 1. The data in our setting can be converted to theirs and vice versa by dividing or multiplying all data values by L respectively.

the budgeted version of the problem with a budget of kL . In theory, they showed that the fractional version of the IMP has the same computational complexity as the IMP. That is the submodularity property of the objective function holds (which is again due to the particular randomized assumption on the uniform distribution of thresholds) yielding the same $(1 - \frac{1}{e})$ -greedy approximation algorithm for the problem. In practice, they showed that the solutions of the two versions can be significantly different: the fractional allocation can improve the influence greatly (i.e., with a budget of k , where $L = 1$ in their setting, the fractional allocation model can influence a larger number of nodes).

Recently, Cordasco et al. [2015] studied a problem which corresponds to a specialized version of the LCIP where the influence factor is the same over the whole network (i.e., $d_i = d_j \forall i, j \in V$) and provide a polynomial time algorithm for complete graphs and trees. As we will see, our results will provide a trivial algorithm for the problem on trees (they provide a non-trivial algorithm for the problem on trees).

4.1.3 Our Contributions

We first study the complexity of the LCIP problem. We show that the LCIP is NP-complete. We then consider several special conditions including (1) equal influence from neighbors, (2) 100% adoption, and (3) restricting the problem to trees. Specifically, we show that the LCIP is NP-complete even on bipartite graphs, when all neighbors exert equal influence, and we do not require 100% adoption. When we require 100% adoption the problem remains NP-complete (and is in fact APX-hard). For trees, when neighbors exert *unequal* influence and we require 100% adoption, the problem remains

NP-complete.

Then, we focus on the case when neighbors exert equal influence and 100% adoption is required. We study the LCIP on trees. Our contributions in this regard are three-fold. First, we propose two polynomial algorithms for the LCIP on trees. We describe a greedy algorithm which has $O(|V|\log|V|)$ running time. Second, we show a dynamic programming (DP) algorithm that has a better $O(|V|)$ running time. The DP algorithm decomposes a tree into several "star" subproblems. For each star subproblem, it finds at most two solution candidates. After all subproblems are examined, a backtracking procedure is used to determine the final solution. More importantly, the DP algorithm also works in the unequal influence case, although the running time is no longer polynomial (it is dependent on that of the mixed 0-1 knapsack problem).

Third, we present two strong formulations. One is a totally unimodular (TUM) formulation for the LCIP on trees. This TUM formulation is built on the influence propagation network, i.e., influence traveling over arcs, and makes use of special structures about the amount of influence passing along an arc. The other is an extended formulation making use of the natural payment variables and the directed influence arc variables. For the latter extended formulation, we project it onto the natural payment variable space and give a complete description for its polytope.

This leads to our contribution on general graphs. Using the observation that the influence propagation network must be a directed acyclic graph (DAG), the TUM formulation for trees can be embedded into a formulation on general graphs, where an additional exponentially sized set of constraints is added to ensure that the arcs selected form a DAG. We use this to design and implement a branch-and-cut approach for the LCIP on general

graphs. In our computational study, we are able to obtain high quality solutions for random graph instances with up to 10,000 nodes and 20,000 edges (40,000 arcs) within a reasonable amount of time.

The rest of the chapter is organized as follows. In the next section, we focus on problem complexity. Section 4.3 discusses the LCIP on trees. Section 4.4 shows how to adapt the TUM formulation for the LCIP on trees to general graphs in a branch-and-cut framework. Section 4.5 presents our computational experience applying the branch-and-cut approach. Section 4.6 provides concluding remarks.

4.2 Problem Complexity

In this section we show that the LCIP is NP-complete. We then consider several special conditions including (1) equal influence from neighbors, (2) 100% adoption, and (3) restricting the problem to trees. Specifically, we show that the LCIP is NP-complete even on bipartite graphs, when all neighbors exert equal influence, and we do not require 100% adoption. When we require 100% adoption the problem remains NP-complete (and is in fact APX-hard). For trees, when neighbors exert *unequal* influence and we require 100% adoption the problem remains NP-complete. On the other hand, when neighbors exert equal influence and we require 100% adoption, we will show in Section 4.3 the

$0 < \alpha < 1$			$\alpha = 1$		
	Equal Inf.	Unequal Inf.		Equal Inf.	Unequal Inf.
General Graphs	NP-hard	NP-hard	General Graphs	APX-hard	APX-hard
Bipartite Graphs	NP-hard	NP-hard	Trees	P	NP-hard

Table 4.1: Summary of complexity results.

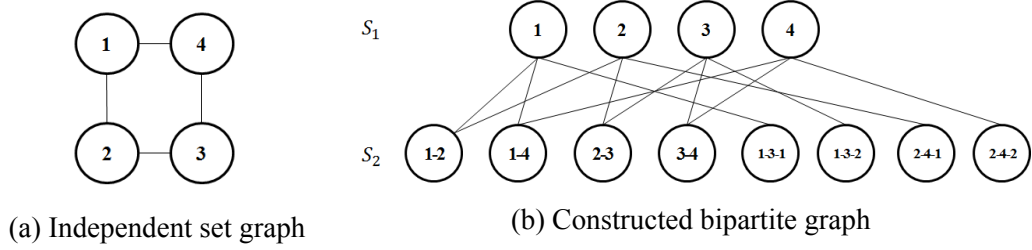


Figure 4.1: Illustration of the reduction from independent set

problem is polynomially solvable on trees. Table 4.1 summarizes our results.

Consider the decision version of the LCIP. Given an instance of the LCIP with $(H, \mathbf{b}, \mathbf{d}, \alpha, k)$, where $H = (V_H, E_H)$ is the graph, \mathbf{b} the threshold vector, \mathbf{d} the influence factor vector, k the budget, and α the desired penetration rate; does there exist a payment/inducement vector \mathbf{p} satisfying $\sum_{i \in V_H} p_i \leq k$ that achieves the desired market penetration (i.e., the number of nodes that are influenced is at least $\alpha|V_H|$)?

Theorem 4.1. *The LCIP is NP-complete.*

Proof. Proof of Theorem 4.1. Given a graph $G = (V, E)$ and a number t the decision version of the independent set problem asks if there is an independent set in G of cardinality t . We will transform an instance of the independent set problem to an instance of the LCIP. To do so, we construct a bipartite graph $H = (S_1 \cup S_2, E_H)$ as follows. For each node $i \in V$, we create a node in S_1 . Then, we create nodes in S_2 . For each edge (i, j) in E , we create a node denoted by $i-j$ in S_2 and connect nodes i and j in S_1 to it. For example, as shown in Figure 4.1, for edge $(1, 2)$ we create node $1-2$ in S_2 and connect it with nodes 1 and 2 in S_1 . For any edge (i, j) not in E (i.e., a possible edge that does not exist in the graph), we create two nodes in S_2 , denoted by $i-j-1$ and $i-j-2$. We connect node i in S_1 to node $i-j-1$ in S_2 and node j in S_1 to node $i-j-2$. In Figure 4.1(a), edge $(1, 3)$ is not

present in G . We add nodes (1-3-1) and (1-3-2) to S_2 and connect them to nodes 1 and 3, respectively. The number of nodes in H is $|V| + |E| + 2(\frac{|V|^2 - |V|}{2} - |E|) = |V|^2 - |E|$, and the number of edges in E is $|V|(|V| - 1)$. Next, for each node $i \in S_1 \cup S_2$ we set b_i as 1. For the influence factors, if a node $i \in S_1$ we set $d_{ij} = 0 \forall j \in a(i)$, and if a node $i \in S_2$ we set $d_{ij} = 1 \forall j \in a(i)$. We set $\alpha = \frac{k|V|}{|V|^2 - |E|}$ and $k = t$.

We claim that there exists a payment vector \mathbf{p} that satisfies the budget (i.e., $\sum_{i \in V_H} p_i \leq k$) and meets the desired penetration rate (i.e., the number of nodes influenced is at least $k|V|$) if and only if G has an independent set of size t . Notice, in the constructed LCIP instance it suffices to only make payments to nodes in S_1 . Since all b_i values are 1 and d_{ij} values are 0 for nodes $i \in S_1$ and 1 for nodes $i \in S_2$, if we ever make a payment to a node j in S_2 (i.e., $p_j = 1$) we can get (at least) the same market penetration rate by reallocating this payment to its neighbor in S_1 . Hence, we can focus on solutions which only make payments to nodes in S_1 . Each node in S_1 has exactly $|V| - 1$ neighbors and two nodes in S_1 share one neighbor if and only if they are neighbors in G . So, when $S \subseteq S_1$ is the set of nodes receiving unit payment the number of influenced nodes in V_H can be calculated as $|V||S| - |S_c|$ where $S_c = \{\{i, j\} \in E : i, j \in S\}$. Thus, we can see that for a given budget k , we can influence at least $k|V|$ nodes in V_H if and only if $|S| = k$ and $S_c = \emptyset$. Looking at the definition of S_c it implies S is an independent set of size $t (= k)$ in G . \square

Corollary 4.1. *The LCIP is NP-complete even when all neighbors of a node exert equal influence (i.e., $d_{ij} = d_i$ for all $i \in V$).*

Proof. Proof of Corollary 4.1. Notice that in the transformation in the proof of Theorem 4.1 all neighbors of a node exert equal influence. \square

Corollary 4.2. *The LCIP is NP-complete on bipartite graphs.*

Proof. Proof of Corollary 4.2. The proof follows from the bipartite graph instance constructed in Theorem 4.1. □

Consider the penetration rate $\alpha = \frac{k|V|}{|V|^2 - |E|}$ obtained from the transformation. This can be written as $\alpha = \frac{k}{|V| - \frac{\Delta}{2}}$ where Δ represents the average degree number in a graph. It is easy to see that by varying the size of the graph, the average degree number Δ , and k , a continuum of values strictly between 0 and 1 may be obtained for α .² This naturally raises questions about the complexity of the problem when we require 100% penetration. We answer this question below.

Theorem 4.2. *Unless $NP \subseteq DTIME(|V|^{polylog(|V|)})$, the LCIP with $\alpha = 1$ cannot be approximated within $O(2^{\log^{1-\epsilon}|V|})$ for any fixed constant $\epsilon > 0$.*

Proof. Proof of Theorem 4.2. We will prove the theorem by a reduction from the TSS problem. Consider a TSS problem instance on an undirected graph $G_t = (V_t, E_t)$ where each node i in V_t has threshold value g_i (recall for the TSS these are the number of nodes of a neighbor that must adopt before node i is influenced). We construct an LCIP instance based on the given TSS instance and show that the two problem instances have optimal solutions with identical costs. Since Chen [2009] proves that the TSS problem cannot be approximated within a ratio of $O(2^{\log^{1-\epsilon}|V_t|})$ for any fixed constant $\epsilon > 0$, unless $NP \subseteq DTIME(|V_t|^{polylog(|V_t|)})$, the same result for the LCIP follows.

²If the instance for the independent set problem has at least one edge we will never select all nodes in S_1 and thus 100 percent adoption is not feasible. It is easy to see then if $\alpha \geq 1$, it means the specific t value is infeasible for the independent set problem. Or $k \leq |V| - \frac{\Delta}{2}$. Or an upper bound on the size of an independent set in the graph G is given by $\min\{\lfloor |V| - \frac{\Delta}{2} \rfloor, \lceil |V| - \frac{\Delta}{2} - 1 \rceil\}$.

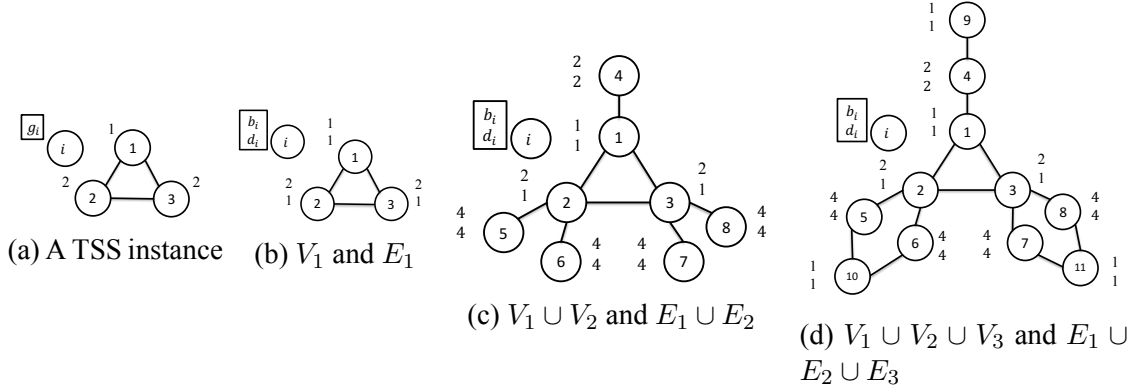


Figure 4.2: An Illustration of the reduction from target set selection

From G_t for the TSS instance, we construct a new graph $G = (V, E)$ to create an LCIP instance as follows. First, we copy the entire graph G_t into G . We denote these nodes and edges as V_1 and E_1 respectively (they are identical to V_t and E_t), and for each node i in V_1 , set its $b_i = g_i$ and $d_i = 1$. This is illustrated in Figure 4.2(b). Next, for each node i in V_1 , we add g_i nodes and connect all of them to node i . These new nodes and edges are denoted by V_{i2} and E_{i2} , respectively. For each of these nodes $j \in V_{i2}$ set $b_j = d_j = 2b_i$. Let $V_2 = \cup_{i \in V_1} V_{i2}$ and $E_2 = \cup_{i \in V_1} E_{i2}$. This is illustrated in Figure 4.2(c). Finally, for each i in V_1 , one more node is added and connected to all nodes in V_{i2} . v_{i3} and E_{i3} are used to denote this new node and its associated edges. Also, let $V_3 = \cup_{i \in V_1} v_{i3}$ and $E_3 = \cup_{i \in V_1} E_{i3}$. For each node i in V_3 , set $b_i = 1$ and $d_i = 1$. This is illustrated in Figure 4.2(d). We consider the LCIP instance (with $\alpha = 1$) on $G = (V_1 \cup V_2 \cup V_3, E_1 \cup E_2 \cup E_3)$. Notice in the LCIP instance created if a node in $i \in V_1$ is activated, it will activate all nodes in V_{i2} , which will activate v_{i3} . Similarly, if node v_{i3} is activated, it will activate all nodes in V_{i2} which will activate node i .

If the TSS instance (on G_t) has an optimal target set S with size k , then, we can find a payment vector with total amount k for the LCIP instance to activate the entire graph (G)

in the following way. For each node i in S , we find its corresponding node in V_1 (continue denoting this node as i) and pay node v_{i3} an amount 1. By the preceding arguments this will activate all nodes corresponding to S in V_1 . But since G_1 and G_t are identical all nodes in V_1 will become active as S is a feasible target set, which again by the preceding arguments will activate the rest of the graph G .

If the LCIP instance has an optimal payment vector with total amount k , we can find a feasible solution for the TSS instance with size k . It is easy to observe in the LCIP instance created any non-zero payment to a node (in an optimal solution) must be at least 1. Hence, based on the preceding arguments, it suffices to focus on solutions to the LCIP that only make non-zero payments to nodes in V_3 (because any payment of 1 to a node v_{i3} will activate all nodes in V_{i2} which will activate node i). Consider such an optimal solution. For each node v_{i3} in V_3 that receives a payment (of 1) we add the corresponding node i (in V) to the target set S . The cardinality of S is k and by the preceding arguments it should be clear that S is a feasible target set for G_t (because G_1 and G_t are identical). \square

4.2.1 Unequal Influence Factors

We prove that when a node i receives unequal influence from its neighbors, the LCIP problem is NP-complete on stars. Recall that for each node i , it has a threshold value b_i . Also, we use d_{ij} to denote the influence factor which captures how much node j influences node i if node j has become active before node i .

Theorem 4.3. *The LCIP problem with unequal influence is NP-complete on stars.*

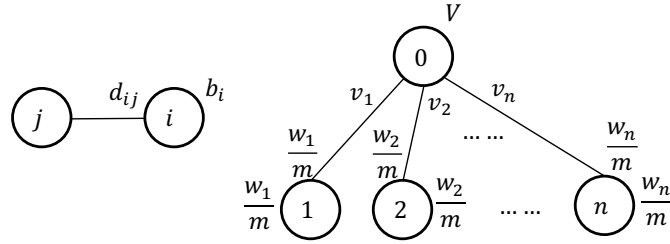


Figure 4.3: Transforming a 0-1 knapsack problem to the LCIP problem with unequal influence on stars

Proof. Proof of Theorem 4.3. Without loss of generality, we assume all input data are positive integers. The decision version of the 0-1 knapsack problem is defined as follows: Given a set N of n items numbered from 1 up to n , each item i with a weight w_i and a value v_i , along with a maximum weight capacity W , can we select a subset of these items such that a value of at least V will be achieved without exceeding the weight W ?

We construct a star network from this 0-1 knapsack problem. For each item i , we put a node i in the graph as a leaf node. After that, we add one extra node and label it as node 0 which is the central node. All leaf nodes connect to the central node but do not connect to each other. Thus, there are $n + 1$ nodes numbered from 0 up to n in the graph. Next, we find a value $m = \max\{w_i + 1 : i \in N\}$. Then, for each leaf node i , we have $b_i = d_{i0} = \frac{w_i}{m}$. For the central node 0, we have $d_{0i} = v_i$ for all $i \in a(0)$ and $b_0 = V$. The constructed star is shown in Figure 4.3. The decision question is: Can we find a payment vector without its total cost exceeding $\frac{W}{m}$ to activate the whole network? In the constructed LCIP instance, each leaf has weight strictly less than 1 and provides an integer amount of influence to the central node 0. Given that V is integer, it is easy to see that we should never pay the central node 0 any incentives. So, it is equivalent to ask: Can we select a subset of leaf nodes such that the incoming influence of the central node is at

least V and the total cost of those selected leaf nodes does not exceed $\frac{W}{m}$? Therefore, if the answer is "Yes", those selected leaf nodes also solve the 0-1 knapsack problem. \square

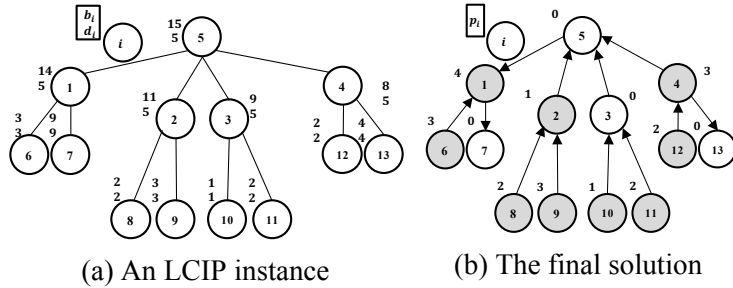
4.3 LCIP on Trees

From now on, for the LCIP, we assume that all neighbors of a node exert equal influence and we require 100% adoption ($\alpha = 1$). In this section, we show that the LCIP is polynomially solvable on trees. Consequently, each node i in the network has associated with it two parameters, b_i and d_i , that represent the threshold and the influence factor for that node.

Without loss of generality, we assume $d_i \leq b_i \leq \text{deg}(i)d_i$. If $b_i \leq d_i$ we would either pay node i the full amount b_i to activate it, or it will become active from the influence of a single neighbor (i.e., either $p_i = 0$ or $p_i = b_i$). Consequently, we can simply update $d_i = b_i$ in cases where $b_i < d_i$. If $b_i > \text{deg}(i)d_i$ node i must be paid a minimum of $b_i - \text{deg}(i)d_i$. This cost can be taken care of in preprocessing and b_i updated to equal $\text{deg}(i)d_i$.

4.3.1 Greedy Algorithm

We now describe an $O(|V|\log|V|)$ greedy algorithm. In each step, from among the inactive nodes, we select the node with the smallest value of d_i (ties can be broken arbitrarily) and pay it the amount of its threshold b_i . Next, we carry out the propagation process from this newly activated node. We update the thresholds by lowering the value of b_i by the amount of (incoming) influence. After that all active nodes are removed from



Step	Node Selected	Payment
1	10	1
2	8	2
3	11	2
4	12	2
5	6	3
6	9	3
7	2	1
8	4	3
9	1	4

Figure 4.4: Greedy Algorithm for LCIP on a Tree

the graph.

Figure 4.4 illustrates the Greedy Algorithm. Figure 4.4(a) shows the instance of the LCIP on a tree. There are 13 nodes and the numbers next to each node show the threshold and the influence factor for that node. Node 10 has the smallest influence factor ($d_{10}=1$), so it is selected and paid $b_{10} = 1$. Newly activated node 10 sends influence to node 3, causing its threshold to be lowered to $b_3 = 4$. Since, $d_3 > b_3$ now we also update $d_3 = b_3 = 4$. Next, node 8 is selected since it has the lowest influence factor value ($d_8 = 2$, recall ties are broken arbitrarily) and paid $b_8 = 2$. Its influence causes the threshold on node 2 to be lowered to 6. Then node 11 is selected and paid $b_{11} = 2$, which causes node 3 to become active. This propagates to node 5 and reduces its threshold to 10. After that, node 12 is selected and paid $b_{12} = 2$, which causes the threshold and influence factor of node 4 to be updated to 3. Then, node 6 is selected and paid $b_6 = 3$, which causes the threshold of

node 1 to be updated to 9. Next, node 9 is selected and paid $b_9 = 3$ which causes node 2's threshold and influence factor to be updated to 1. Then, node 2 is selected and paid $b_2 = 1$ which causes node 5's threshold to be updated to 5. Next, node 4 is selected and paid $b_4 = 3$ which causes nodes 5 and 13 to become active. This influence propagates from node 5 to 1 causing node 1's threshold and influence factor to be updated to 4. Finally, node 1 is selected and paid $b_1 = 4$ which causes node 7 to become active resulting in all nodes being active.

We show the correctness of the greedy algorithm by proving that there exists an optimal solution to the LCIP on a tree where node $k = \arg \min\{d_i : i \in V\}$ is paid its full threshold value b_k . The greedy algorithm recursively applies this property to obtain an optimal solution.

Proposition 4.1. *Given an instance of the LCIP on a tree, there exists an optimal solution where node $k = \arg \min\{d_i : i \in V\}$ is paid its full threshold value b_k .*

Proof. Proof of Proposition 4.1. In the LCIP to initialize the influence propagation process at least one node must be paid its full threshold. Such a node will propagate influence out (i.e., it will not receive influence). Consider an optimal solution P^* where node k is not paid its full threshold value b_k . That means it must receive influence from at least one of its neighbors (say node l). This node l must either be paid its full threshold value (b_l) in P^* or it must receive influence from at least one of its neighbors (different from node k). Repeating this process, we can identify a node j that is paid its full threshold value and from whom influence propagates to node k via a directed path. Furthermore, along this path from node j to node k all nodes other than node j are receiving payments strictly

less than their full threshold values. We can now change the solution as follows. We will reverse the propagation of influence on this path from node k to node j . This means the payment to node j decreases by an amount of d_j its influence factor (and it is no longer paid its full threshold value) and the payment to node k increases by at most d_k its influence factor. The payments for all other nodes remain the same. Since $d_k \leq d_j$ the cost of this solution does not increase. Repeating this argument until node k no longer receives any influence from one of its neighbors proves the claim. \square

Theorem 4.4. *The greedy algorithm solves the LCIP on a tree optimally in $O(|V| \log |V|)$.*

Proof. Proof of Theorem 4.4. The greedy algorithm is based on repeatedly applying Claim 4.1. In the first step the node with the smallest influence factor is paid its full threshold value. Once influence is propagated from the node that has just received its full payment and been removed from the graph and the solution is updated, we are left with smaller LCIPs on separate trees. Claim 4.1 holds separately to each one of these trees, and so in the next step the node with the smallest influence factor can be selected and paid its full threshold value (in this updated graph). It takes $O(|V| \log |V|)$ time to initially sort the nodes based on their d_i values. After that in each step updating the sorted list (when the d_i value of a node changes) takes $O(\log |V|)$ time; and there are at most $|E| = |V| - 1$ updates. \square

4.3.2 Dynamic Programming Algorithm.

We now describe a dynamic programming (DP) algorithm with a better running time of $O(|V|)$. Another advantage of the DP algorithm over the greedy algorithm is the fact that the DP algorithm can be applied when neighbors of a node have unequal influence (the running time is dependent on that of the mixed 0-1 Knapsack problem and is no longer polynomial) whereas the greedy algorithm is no longer an exact algorithm.

The dynamic programming algorithm decomposes the tree into subproblems. Each subproblem is used to find the most promising solution candidates (at most two) where one of them will be part of the final solution of the tree. A subproblem is defined on a star network which has a single central node and (possibly) multiple leaf nodes. By solving the subproblem, we have one solution candidate for the case where there is influence coming into the central node along the link which connects the star to the rest of the tree and one solution candidate for the case where influence goes out of the central node on this link (to its parent). Next, the star is compressed into one single leaf node for the next star network. This process is repeated until we are left with a single (last) star with its central node as the root node of the tree. After we exhaust all subproblems, a backtracking method is used to combine the solutions from star subproblems and obtain the final solution (set of nodes that are paid incentives along with the incentive amounts) for the tree. The pseudocode of the proposed algorithm is shown in Algorithm 12.

We now discuss how to solve the LCIP on a star. Let c denote the central node of a star (all the other nodes are leaf nodes) and refer to this star as star c . To select which nodes to give incentives to on a star, we focus on the central node c . Any leaf node i

Algorithm 12 DP Algorithm for the LCIP on trees

- 1: Arbitrarily pick a node as the root node of the tree and let $Z = 0$.
 - 2: Define the order of star problems based on the bottom-up traversal of the tree.
 - 3: **for** each star subproblem **do**
 - 4: StarHandling
 - 5: **end for**
 - 6: SolutionBacktrack
-

with $b_i \geq d_c$ can be neglected (recall for all leaf nodes their thresholds are equal to their influence factors). When $b_i \geq d_c$, giving b_i units of incentives to a leaf node i sends d_c units of influence to node c , thus the decrease in threshold is less than or equal to what we spend ($d_c \leq b_i$). We are no worse off giving the incentive directly to the central node c , and never use such leaf nodes (this can also be seen by invoking the greedy algorithm on the star). We collect the nodes with thresholds less than d_c in set S and sort them in increasing order of their thresholds. The nodes in S are candidates to provide incentives to (in addition to the central node). Let $g_c = \lceil \frac{b_c}{d_c} \rceil$ be the number of active neighbors required to activate node c if no incentives are paid to it. The cost of the solution to the LCIP on a star depends on the size of the set S . When $|S| \geq g_c$ (i.e., there are more than enough leaf nodes), the solution is to pay the first $(g_c - 1)$ nodes in S an amount equal to their threshold and then to compare the threshold of the g_c -th leaf node in S (b_{g_c}) against the remaining threshold $(b_c - (g_c - 1)d_c)$ needed to activate the central node c . If $b_{g_c} < (b_c - (g_c - 1)d_c)$ we pay the g_c -th leaf node b_{g_c} ; else we pay the central node $(b_c - (g_c - 1)d_c)$. If $|S| < g_c$, then all nodes in the set S are paid incentives equal to their thresholds and the remaining amount of the threshold of the central node $(b_c - |S|d_c)$ is paid directly to the central node. Here, we assumed there is no influence coming into the central node from its parent. For the situation where the central node receives influence from its parent, we simply reduce

b_c to $b_c - d_c$ on the star, and accordingly update $d_c = \min\{b_c, d_c\}$, $g_c = \lceil \frac{b_c}{d_c} \rceil$, and solve the problem on the star.

After we determine the two solution candidates for the current star subproblem, the star is compressed into a single node for the next star subproblem. If in the optimal solution the central node c receives influence from its parent the cost of the solution is denoted by C_I^c . If in the optimal solution the central node c sends influence to its parent the cost of the solution on the star is denoted by C_{NI}^c . Thus the amount C_I^c (which is smaller) must be incurred for the star at a minimum in the optimal solution. The incremental amount $C_{NI}^c - C_I^c$ must be paid if the star sends influence to its parent in the optimal solution. Thus, when we compress the star into a single node for the next star subproblem the threshold for the compressed star is $C_{NI}^c - C_I^c$. Since the compressed star is a leaf node for the next star its influence factor is also set to $C_{NI}^c - C_I^c$. This DP calculation procedure is repeated until we arrive at the root node of the tree. Here, since there is no possibility of external influence to the root node there is only one solution candidate.

Algorithm 13 provides the pseudocode associated with this calculation procedure. At its core is the function `SolveStar` that finds the optimal solution for a given star. The function returns X (set of leaf nodes that are given incentives), \mathbf{p} (vector of partial incentives given, i.e., $\{p_i | i \in V\}$) and C (total cost of the star). In Algorithm 13 the subscripts NI and I and superscript c (for X , \mathbf{p} and C) represent the outputs in the cases of no influence, influence and star c , respectively. Also $L(c)$ denotes the set of leaf nodes for star c .

After we obtain the solution of the last star which has the root node as its central node, we invoke a backtracking procedure to obtain the final solution for this tree (if we

Algorithm 13 StarHandling

Require: star c

- 1: $(X_{NI}^c, \mathbf{p}_{NI}^c, C_{NI}^c) \leftarrow \text{SolveStar}(\text{star } c, \text{no-influence})$.
 - 2: **if** star c is the last star **then**
 - 3: $Z = Z + C_{NI}^c$.
 - 4: **else**
 - 5: $(X_I^c, \mathbf{p}_I^c, C_I^c) \leftarrow \text{SolveStar}(\text{star } c, \text{with-influence})$.
 - 6: The compressed node's threshold is $C_{NI}^c - C_I^c$.
 - 7: $Z = Z + C_I^c$.
 - 8: **end if**
 - 9: **function** SolveStar(a star c , flag)
 - 10: **if** flag is with-influence **then**
 - 11: $b_c = b_c - d_c$ and $d_c = \min\{b_c, d_c\}$.
 - 12: **end if**
 - 13: Let $g_c = \lceil \frac{b_c}{d_c} \rceil$ and $S = \{i \mid b_i < d_c, i \in L(c)\}$.
 - 14: **if** $|S| \geq g_c$ **then**
 - 15: Let S_{g_c} and S_{g_c-1} be the sets of the first g_c and $(g_c - 1)$ nodes in S , respectively.
 - 16: **if** $\sum_{i \in S_{g_c}} b_i \leq \sum_{i \in S_{g_c-1}} b_i + b_c - (g_c - 1)d_c$ **then**
 - 17: $X \leftarrow S_{g_c}, p_i = b_i$ for $i \in X$.
 - 18: **else**
 - 19: $X \leftarrow S_{g_c-1}, p_i = b_i$ for $i \in X$, and let $p_c = b_c - (g_c - 1)d_c, X \leftarrow X \cup c$.
 - 20: **end if**
 - 21: **else**
 - 22: $X \leftarrow S, p_i = b_i$ for $i \in S$, and let $p_c = b_c - |S|d_c, X \leftarrow X \cup c$.
 - 23: **end if**
 - 24: $C = \sum_{i \in X} p_i$.
 - 25: **return** X, \mathbf{p}, C .
 - 26: **end function**
-

are simply interested in the cost of the optimal solution no backtracking is necessary as global variable Z contains the cost of the optimal solution). Let r denote the root of the tree (as determined in Algorithm 12) and NL denote the set of non-leaf nodes in the tree. After solving the last star subproblem, we know (from Algorithm 13) that leaf nodes in X_{NI}^r do not receive influence from their parent (the root) but the remaining leaf nodes do. With this information we can proceed down the tree, incorporating partial solutions at each node based on whether it receives influence from its parent or not. Algorithm 14 describes this backtracking procedure. It contains two recursive functions: with-influence

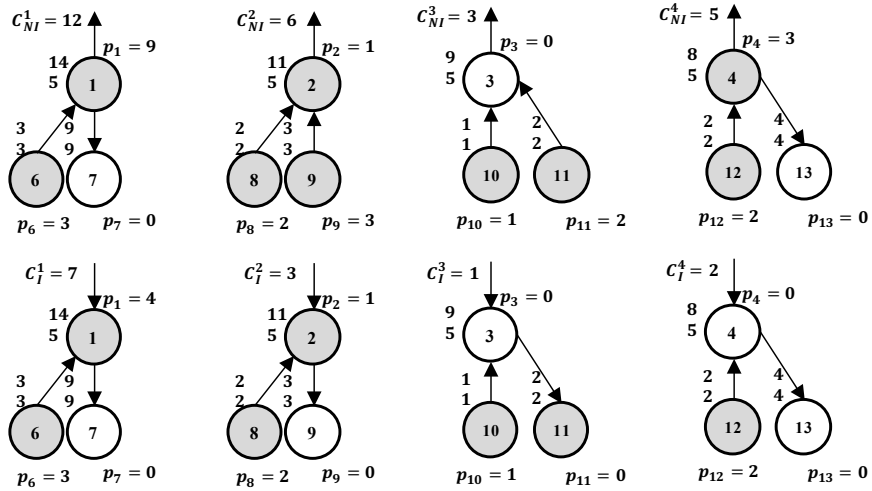
Algorithm 14 SolutionBacktrack

Require: the last star r and its solution

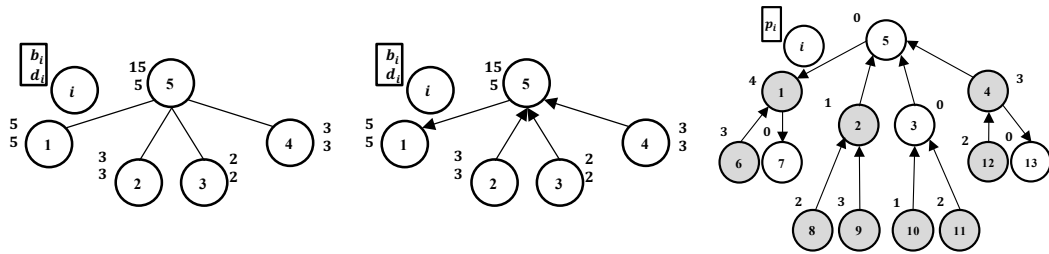
- 1: Let $P \leftarrow P_{NI}^r$.
 - 2: $\forall l \in \{L(r) \cap X_{NI}^r \cap NL\}$ call No-Influence(l, P, X).
 - 3: $\forall l \in \{L(r) \setminus X_{NI}^r \cap NL\}$ call With-Influence(l, P, X).
 - 4: **return** P, X .
 - 5: **function** With-Influence(c, P, X)
 - 6: $X \leftarrow (X \setminus c) \cup X_I^c$, update $p_c = 0$, and $P = P + P_I^c$.
 - 7: $\forall l \in \{L(c) \cap X_I^c \cap NL\}$ call No-Influence(l, P, X).
 - 8: $\forall l \in \{L(c) \setminus X_I^c \cap NL\}$ call With-Influence(l, P, X).
 - 9: **return** P, X .
 - 10: **end function**
 - 11: **function** No-Influence(c, P, X)
 - 12: $X \leftarrow (X \setminus c) \cup X_{NI}^c$, update $p_c = 0$, and $P = P + P_{NI}^c$.
 - 13: $\forall l \in \{L(c) \cap X_{NI}^c \cap NL\}$ call No-Influence(l, P, X).
 - 14: $\forall l \in \{L(c) \setminus X_{NI}^c \cap NL\}$ call With-Influence(l, P, X).
 - 15: **return** P, X .
 - 16: **end function**
-

for the case where a central node receives influence from its parent and no-influence for the case where a central node does not receive influence from its parent.

Figure 4.5 illustrates the DP algorithm for the instance shown in Figure 4.4(a). Figure 4.5(a) shows solutions for the star subproblems in the DP algorithm. The first row of Figure 4.5(a) displays the solutions for the no-influence case (i.e., no influence from parent node) and the second row displays the solutions for the with-influence case (i.e., influence from parent node). Figure 4.5(b), shows the final star at the root node (after all other stars have been compressed). From Figure 4.5(a) one can see that star 1 has cost 12 and 7 for the no- and with-influence solutions, respectively. Thus, the threshold (and influence factor) for compressed node 1 is 5 in this final star. Similarly, the thresholds are 3, 2, and 3 for compressed node 2, 3, and 4, respectively in this final star. In Figure 4.5(c), the influence propagation directions are displayed for the optimal solution of the final star. This identifies which stars receive influence from their parents and which



(a) Star subproblems for the DP algorithm



(b) The final star

(c) Influence in the final star

(d) The final solution

Figure 4.5: The DP Algorithm for LCIP on a Tree

ones do not. Thus, star 1 uses the with-influence solution; and stars 2, 3 and 4 use the no-influence solution. Figure 4.5(d) provides this final solution, which is identical to the one found by the greedy algorithm.

Theorem 4.5. *The DP algorithm solves the LCIP on trees optimally in $O(|V|)$ time.*

Proof. Proof of Theorem 4.5. The bottleneck is the calculation to solve each star subproblem. There are at most $|V|$ stars in a tree. For each star, we need to find g_i cheapest children and it takes $O(deg(i))$ time. Finding the g_i th order statistics can be done in $O(deg(i))$ time by the Quickselect method in Chapter 9 of [Stein et al., 2009], thus it takes $O(deg(i))$ time to go through the list to collect the g_i cheapest children. For the whole tree, this sum is

bounded by $O(|V|)$ (since $\sum_{i \in V} \text{deg}(i) = 2|E| = 2|V| - 2$). \square

In addition to a better time complexity, another advantage of the DP algorithm is the fact that it also applies for the LCIP on trees with unequal influence factors. The DP recursion remains the same. However, as we now explain, the solution to the no-influence and with influence cases correspond to a mixed 0-1 knapsack problem [Marchand and Wolsey, 1999]. Given an LCIP on stars with unequal influence factors, let the central node c have b_c and d_{cj} for all j in $L(c)$. Each leaf node j in $L(c)$ has $b_j = d_j$ given that it only has one neighbor. Then, finding the optimal solution for this star is equivalent to solving the following problem:

$$\text{(MixedKP) Min } p_c + \sum_{j \in L(c)} b_j x_j \quad (4.6)$$

$$\text{Subject to } p_c + \sum_{j \in L(c)} d_{cj} x_j \geq b_c \quad (4.7)$$

$$p_c \geq 0, x_j \in \{0, 1\} \quad \forall j \in V, i \in L(c) \quad (4.8)$$

where p_c represent the incentive we give to node c . For a node j in $L(c)$, binary variable x_j decides if it receives payment. Note that for a leaf node, it either receives full payment or no payment in an optimal solution. For the with-influence solution, we update b_c as $b_c - d_{cp}$ where d_{cp} is the influence from node c 's parent. Thus, we need to solve two mixed 0-1 knapsack problems. Although the running time is no longer polynomial, mixed 0-1 knapsack problem can be solved quite efficiently in practice.

4.3.3 Totally Unimodular Formulation

The MIP model in Section 4.1.1 for the LCIP tracks influence propagation by creating artificial time periods. In this section, we propose a different MIP formulation for the LCIP on trees (recall we have equal influence and 100% adoption) that uses the directed influence propagation network (i.e., the direction influence travels over edges in the network). First, consider the following formulation (MIP4.2).

$$\text{(MIP4.2) Min} \quad \sum_{i \in V} p_i \quad (4.9)$$

$$\text{Subject to} \quad y_{ij} + y_{ji} = 1 \quad \forall \{i, j\} \in E \quad (4.10)$$

$$p_i + \sum_{j \in a(i)} d_i y_{ji} \geq b_i \quad \forall i \in V \quad (4.11)$$

$$p_i \geq 0 \quad \forall i \in V \quad (4.12)$$

$$y_{ji} \in \{0, 1\} \quad \forall j \in V, i \in a(j) \quad (4.13)$$

In MIP4.2, p_i is a continuous variable denoting the amount of incentive paid to a node i and y_{ij} is a binary variable that tells us whether or not node i influences node j (it is 1 if node i influences node j). Constraint set (4.10) says that for each edge $\{i, j\}$ in the network either node i influences node j or node j influences node i . Constraint set (4.11) ensures that for each node i in V , the total of the incoming influence and the payment it receives is greater than or equal to its threshold. Although this model is much smaller than MIP4.1, its linear relaxation does not provide integral solutions on trees (nor does MIP4.1).

We will build upon MIP4.2 to derive a TUM formulation for trees. Observe that if constraint set (4.11) always held at equality we could replace p_i by $b_i - \sum_{j \in a(i)} d_i y_{ji}$ in

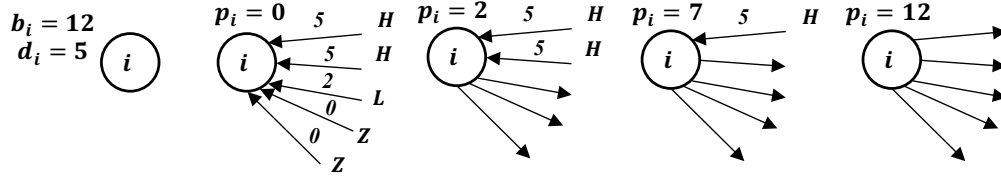


Figure 4.6: Categorization of incoming influence when and $g_i \geq 2$

the objective function and eliminate the payment variables from the model. In that case we are left with constraint set (4.10) which is TUM. Unfortunately constraint set (4.11) does not necessarily hold at equality since a node may have g_i or greater incoming arcs in a feasible solution (i.e., it may receive influence from g_i or more neighbors). Our model will instead categorize the incoming influence to node i on an arc into three types: H with incoming influence d_i , L with incoming influence $l_i = b_i - (g_i - 1)d_i$ and Z with incoming influence 0, so that the incoming influence is exactly equal to the difference between its threshold b_i and its payment p_i .

We first consider the situation where $g_i \geq 2$ for node i and explain this categorization. Consider the example in Figure 4.6. Here $b_i = 12$ and $d_i = 5$. Thus $g_i = 3$ and $l_i = 2$. Observe that there are $g_i + 1 = 4$ possible scenarios. Either the node receives no payment (i.e., $p_i = 0$) which means it must receive incoming influence of type H on $g_i - 1 = 2$ arcs, and an incoming influence of type L on one arc. Any remaining incoming arcs are of type Z and provide an incoming influence of 0. Or, the node receives a payment of $l_i + \lambda d_i$ (where $\lambda = 0, \dots, g_i - 1$) and has exactly $g_i - 1 - \lambda$ incoming arcs of type H . Figure 4.6 shows these scenarios with $p_i = 0, 2, 7, 12$ respectively.

We now consider the situation where $g_i = 1$ for node i and explain this categorization. In the example in Figure 4.7, $b_i = d_i = 4$. Thus $g_i = 1$ (recall without loss of

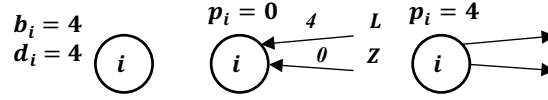


Figure 4.7: Categorization of incoming influence when $g_i = 1$

generality when $g_i = 1$, $b_i = d_i$) and $l_i = d_i$. There are $g_i + 1 = 2$ possible scenarios. Either the node receives no payment which means it must receive an incoming influence of type L on one arc. Any remaining incoming arcs are of type Z and provide an incoming influence of 0. Or, the node receives a payment of $p_i = l_i$ and has no incoming arcs. Figure 4.7 shows these scenarios with $p_i = 0, 4$ respectively.

Taken together, we observe that when there is no payment to a node there are exactly $(g_i - 1)$ arcs with incoming influence of type H and one arc with incoming influence of type L . When there is a payment (notice the payment set is discrete) the incoming arcs can only provide influence of type H , and there are at most $(g_i - 1)$ of them. Finally the payment at a node is easily recovered by subtracting the sum of incoming influences from its threshold b_i .

We use these observations and develop our third formulation MIP4.3 (a pure 0-1 integer program). Essentially, the binary variable y_{ji} in MIP4.2 is decomposed into three binary variables x_{ji}^H , x_{ji}^L , and x_{ji}^Z to represent the type of incoming influence. In other words x_{ji}^H , x_{ji}^L , and x_{ji}^Z are set to 1, when node i receives incoming influence of type H , L , and Z respectively from node j , and is 0 otherwise.

$$(BIP4.3) \quad \text{Max} \quad \sum_{i \in V} \sum_{j \in a(i)} \sum_{k \in \{H, L, Z\}} c_i^k x_{ji}^k \quad (4.14)$$

$$\text{Subject to} \quad \sum_{k \in \{H, L, Z\}} (x_{ij}^k + x_{ji}^k) = 1 \quad \forall \{i, j\} \in E \quad (4.15)$$

$$\sum_{j \in a(i)} x_{ji}^H \leq g_i - 1 \quad \forall i \in V \quad (4.16)$$

$$\sum_{j \in a(i)} x_{ji}^L \leq 1 \quad \forall i \in V \quad (4.17)$$

$$x_{ji}^k \in \{0, 1\} \quad \forall i \in V, j \in a(i), k \in \{H, L, Z\} \quad (4.18)$$

Constraint set (4.15) is the analog of constraint set (4.10). It specifies that for each edge $\{i, j\}$ in the network either node i influences node j or node j influences node i ; and the amount of influence may only be one of the three types. Constraint set (4.16) ensures that a node i has no more than $g_i - 1$ of its incoming arcs having type H influence. Constraint set (4.17) ensures that a node i has at most one incoming arc with type L influence. The objective is to maximize the influence propagation on the network. The objective coefficient $c_i^H = d_i$ provides the amount of incoming influence into node i when it receives type H influence on an arc, $c_i^L = l_i$ provides the amount of incoming influence into node i when it receives type L influence on an arc, and $c_i^Z = 0$ provides the amount of incoming influence into node i when it receives type Z influence on an arc. Because our model ensures the total sum of incoming influences never exceeds the threshold, minimizing the sum of the payments ($\sum_{i \in V} p_i$) is the same as minimizing the sum of the thresholds minus the incoming influences at each node ($\sum_{i \in V} (b_i - \sum_{j \in a(i)} \sum_{k \in \{H, L, Z\}} c_i^k x_{ji}^k)$). However $\sum_{i \in V} b_i$ is a constant, and so minimizing the sum of the payments is equivalent to $\sum_{j \in a(i)} \sum_{k \in \{H, L, Z\}} c_i^k x_{ji}^k$, the total incoming influence over the network.

Theorem 4.6. *The constraint matrix of MIP4.3 is totally unimodular.*

Proof. Proof of Theorem 4.8. Let A denote the constraint matrix of MIP4.3---composed of constraint sets (4.15), (4.16), and (4.17)---and a_{ij} denote its elements. A is a 0 – 1 matrix that has at most two nonzero elements in each column. Observe we can partition

the rows of A into two subsets Q_1 containing constraint set (4.15) and Q_2 containing constraint sets (4.16) and (4.17). With this columns that have two nonzero elements have one of the nonzero coefficients in Q_1 and one of the nonzero coefficients in Q_2 . Thus, from Corollary 2.8 in Nemhauser and Wolsey [1988, p. 544] A is a TUM. \square

Since the right hand sides of the constraint sets are integer, the linear relaxation of MIP4.3 (we will refer to this linear relaxation where constraint set (4.18) is replaced by nonnegativity constraints as LP3) provides integral solutions to the LCIP on trees.

4.3.4 A Tight and Compact Extended Formulation

In this section, we present another good formulation for the LCIP on trees. It has the same variables as defined for MIP4.1 in section 4.1.1. Compared to MIP4.3, the meaning of variables is more intuitive. This formulation is referred to as MIP4.4 and is given below:

$$(MIP4.4) \quad \text{Min} \quad \sum_{i \in V} p_i \quad (4.19)$$

$$\text{Subject to} \quad y_{ij} + y_{ji} \leq 1 \quad \forall \{i, j\} \in E \quad (4.20)$$

$$p_i + \sum_{j \in a(i)} d_i y_{ji} \geq b_i \quad \forall i \in V \quad (4.21)$$

$$p_i + \sum_{j \in a(i)} l_i y_{ji} \geq l_i g_i \quad \forall i \in V \quad (4.22)$$

$$p_i \geq 0 \quad \forall i \in V \quad (4.23)$$

$$y_{ji} \in \{0, 1\} \quad \forall j \in V, i \in n(j) \quad (4.24)$$

Constraint (4.22) is added to MIP4.1 to obtain this new formulation. The linear relaxation of MIP4.4 is the following linear programming problem:

$$(LP4.4) \quad \text{Min} \quad \sum_{i \in V} p_i \quad (4.25)$$

$$\text{Subject to } (u_{\{ij\}}) \quad -y_{ij} - y_{ji} \geq -1 \quad \forall \{i, j\} \in E \quad (4.26)$$

$$(v_i) \quad p_i + \sum_{j \in a(i)} d_i y_{ji} \geq b_i \quad \forall i \in V \quad (4.27)$$

$$(w_i) \quad p_i + \sum_{j \in a(i)} l_i y_{ji} \geq l_i g_i \quad \forall i \in V \quad (4.28)$$

$$p_i \geq 0 \quad \forall i \in V \quad (4.29)$$

$$y_{ji} \geq 0 \quad \forall j \in V, i \in n(j) \quad (4.30)$$

We refer to this linear programming problem as LP4.4. The dual to LP4.4 is as follows:

$$(DLP4.4) \quad \text{Max} \quad \sum_{i \in V} b_i v_i + \sum_{i \in V} l_i g_i w_i - \sum_{\{i,j\} \in E} u_{\{ij\}} \quad (4.31)$$

$$\text{Subject to } (y_{ji}) \quad -u_{\{ij\}} + d_i v_i + l_i w_i \leq 0 \quad \forall j \in V, i \in n(j) \quad (4.32)$$

$$(p_i) \quad v_i + w_i \leq 1 \quad \forall i \in V \quad (4.33)$$

$$u_{\{ij\}} \geq 0 \quad \forall \{i, j\} \in E \quad (4.34)$$

$$v_i, w_i \geq 0 \quad \forall i \in V \quad (4.35)$$

We have u_{ij} , v_i , and w_i as dual variables for constraint sets (4.20), (4.21), and (4.22), respectively. We refer to the dual linear problem as DLP4.4. Let $\text{conv}(P)$ denote the convex hull of feasible payment vectors \mathbf{p} , and let $P_{LP4.4}$ denote the feasible region of LP4.4.

Theorem 4.7. *Given a tree, LP4.4 has optimal solutions with a valid payment vector p .*

Also, and $\text{Proj}_{\mathbf{p}}(P_{LP4.4}) = \text{conv}(P)$

Proof. Proof of Theorem 4.7. The complementary slackness conditions:

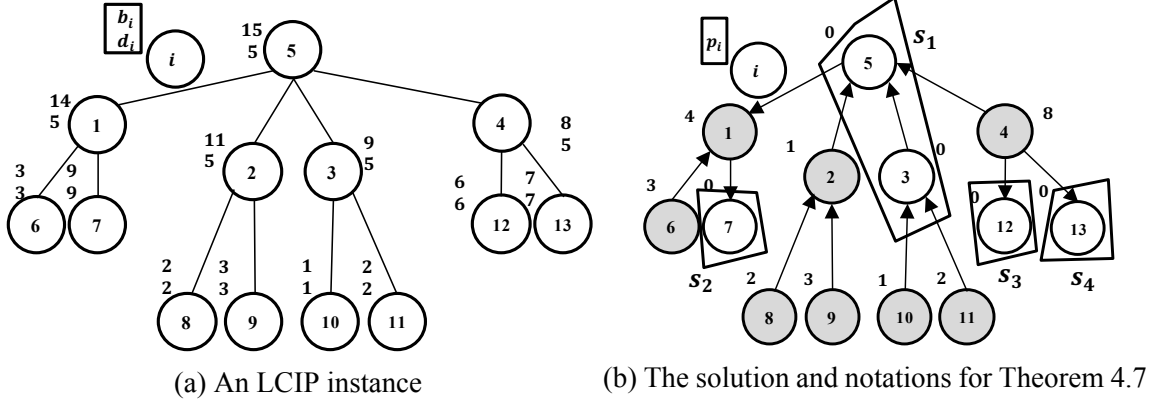


Figure 4.8: Illustration for Theorem 4.7

$$(y_{ij} + y_{ji} - 1)u_{\{ij\}} = 0 \quad \forall \{i, j\} \in E \quad (4.36)$$

$$(p_i + \sum_{j \in a(i)} d_i y_{ji} - b_i)v_i = 0 \quad \forall i \in V \quad (4.37)$$

$$(p_i + \sum_{j \in a(i)} l_i y_{ji} - l_i g_i)w_i = 0 \quad \forall i \in V \quad (4.38)$$

$$(-u_{\{ij\}} + d_i v_i + l_i w_i)y_{ji} = 0 \quad \forall j \in V, i \in n(j) \quad (4.39)$$

$$(v_i + w_i - 1)p_i = 0 \quad \forall i \in V \quad (4.40)$$

Following the greedy algorithm, we have an optimal primal solution of the MIP4.4. The values of the p variables are set as the payment found in the greedy algorithm. The values of the y variables are determined by the influence propagation process discovered in the greedy algorithm. Hence, we have a feasible primal solution for the LP4.4. Then, based on this payment vector p , we detect connected components which are induced by nodes with payment zeros. The whole set of connected components are denoted by S_Z and a component in it is denoted by S . For a component S , we defined a value k_S which takes value as $\min\{l_i = b_i - (g_i - 1)d_i : y_{ij} = 1, i \in V, j \in S\}$. In Figure 4.8(a), it has an LCIP instance and its solution found by the greedy algorithm is shown in Figure 4.8(b).

We have four components formed by nodes with payment zeros. $S_1 = \{3, 5\}$, $S_2 = \{7\}$, $S_3 = \{12\}$ and $S_4 = \{13\}$. For S_1 , $k_{S_1} = \{l_2, l_3, l_4, l_5, l_{10}, l_{11}\} = \{1, 4, 3, 5, 1, 2\} = 1$. Similarly, $k_{S_2} = 4$, $k_{S_3} = 3$, and $k_{S_4} = 3$

Now, we are ready to construct a dual solution and show this pair of primal and dual solutions satisfies the complementary slackness conditions. First, we show how to decide $u_{\{ij\}}$ variables. Let $E^+(S)$ denote the set of edges which have their head in S from the primal solution, $E^+(S) = \{\{i, j\} : y_{ij} = 1, i \in V, j \in S\}$. Then, for a S in S_Z , we set $u_{\{i,j\}} = k_S$ for all $\{i, j\}$ in $E^+(S)$. In Figure 4.8(b), We have $E^+(S_1) = \{\{2, 5\}, \{3, 5\}, \{4, 5\}, \{10, 3\}, \{11, 3\}\}$, $E^+(S_2) = \{\{1, 7\}\}$, $E^+(S_3) = \{\{4, 12\}\}$ and $E^+(S_4) = \{\{4, 13\}\}$. So, $u_{ij} = 1$ for all $\{i, j\}$ in $E^+(S_1)$, $u_{17} = 4$, $u_{4,12} = 3$ and $u_{4,13} = 3$.

For the remaining $u_{\{ij\}}$, their corresponding edges become arcs whose heads are nodes given positive payment (i.e., $p > 0$). We set $u_{\{ij\}} = \min\{p_j, d_j\}y_{ij} + \min\{p_i, d_i\}y_{ji}$. Thus, if $y_{ij} = 1$, it means $u_{\{ij\}} = \min\{p_j, d_j\}y_{ij}$. Otherwise, we have $y_{ji} = 1$ and it means $u_{\{ij\}} = \min\{p_i, d_i\}$. Note that for a node i the only positive p_i value could be smaller than d_i is l_i . We observe that in the primal solution, constraint set (4.26) is always binding because each edge is given an influence direction. Thus, condition (4.36) is satisfied. In Figure 4.8(b), $u_{61} = 4$, $u_{51} = 4$, $u_{82} = 1$ and $u_{92} = 1$.

Then, we have three cases for a node i . First, if $p_i = 0$ and i is in a S which is in S_Z , we have $v_i = 0$ and $w_i = k_S/l_i \leq 1$ because $k_S \leq l_i$. Condition (4.37) is satisfied because $v_i = 0$. Condition (4.38) is satisfied because constraint (4.28) is binding because node i has exactly g_i incoming arcs when $p_i = 0$ as proved in Proposition 4.1. Next, for condition (4.39), when both node i and j are in S , constraint (4.32) is binding. When

node j is not in S and $y_{ji} = 1$, constraint (4.32) is binding because $l_i w_i = k_S = u_{\{ij\}}$. Also, node j is not in S and when $y_{ji} = 0$, constraint (4.32) is satisfied because $d_j \geq k_S$. Hence, condition (4.39) is satisfied. Lastly, condition (4.40) is satisfied because $p_i = 0$. For node 3, 5, 7, 12 and 13 in Figure 4.8(b), $v_3 = 0, v_5 = 0, v_7 = 0, v_{12} = 0, v_{13} = 0$ and $w_3 = \frac{1}{4}, w_5 = \frac{1}{5}, w_7 = \frac{1}{9}, w_{12} = \frac{1}{6}, w_{13} = \frac{1}{7}$.

Second, if $p_i = l_i$, we set $v_i = 0$ and $w_i = 1$. Condition (4.37) is satisfied because when $v_i = 0$. Condition (4.38) is satisfied because constraint (4.28) is binding given that $p_i = l_i$ only happens when the number of incoming arcs is $g_i - 1$. Then, regarding condition (4.39), when $y_{ji} = 1$, we have $u_{\{ij\}} = l_i$ and constraint (4.32) is binding. When $y_{ji} = 0$, it means $y_{ij} = 1$, we have $u_{\{ij\}} \geq l_i$ because node i is selected by the greedy algorithm before node j and constraint (4.32) is satisfied. So, condition (4.39) is satisfied. Lastly, condition (4.40) is satisfied because constraint (4.33) is binding. For node 1, 2, 6, 8, 9, 10, and 11 in Figure 4.8(b), $v_1 = 0, v_2 = 0, v_6 = 0, v_8 = 0, v_9 = 0, v_{10} = 0, v_{11} = 0$ and $w_1 = 1, w_2 = 1, w_6 = 1, w_8 = 1, w_9 = 1, w_{10} = 1, w_{11} = 1$.

Third, if $p_i > l_i$, we set $v_i = 1$ and $w_i = 0$. Condition (4.37) is satisfied because constraint (4.27) is binding when $p_i > 0$. Condition (4.38) is satisfied because $w_i = 0$. Then, regarding condition (4.39), when $y_{ji} = 1$, we have $u_{\{ij\}} = d_i$ and constraint (4.32) is binding. When $y_{ji} = 0$, it means $y_{ij} = 1$, we have $u_{\{ij\}} \geq d_i$ because node i is selected by the greedy algorithm before node j and constraint (4.32) is satisfied. So, condition (4.39) is satisfied. Lastly, condition (4.40) is satisfied because constraint (4.33) is binding. For node 4 in Figure 4.8(b), $v_4 = 1$ and $w_4 = 0$. □

4.3.5 Polytope of the LCIP on Trees

In this section, we derive the polytope of the LCIP problem on trees. The extended formulation is projected onto the space of the payment (i.e., p) variables by projecting out all arc (i.e., y) variables. Usually, there are two approaches for projecting out variables. One approach is Fourier-Motzkin elimination (which can easily be applied to project out the x variables). A more elegant method, proposed by Balas and Pulleyblank [1983], is based upon a theorem of the alternatives. We will follow this approach.

Based on Theorem 2 in Balas and Pulleyblank [1983], the projection cone W is described by the following linear inequalities:

$$-u_{\{ij\}} + d_i v_i + l_i w_i \leq 0 \quad \forall i \in V \quad (4.41)$$

$$u_{\{ij\}} \geq 0 \quad \forall \{i, j\} \in E \quad (4.42)$$

$$v_i \geq 0, w_i \geq 0 \quad \forall i \in V \quad (4.43)$$

where $u_{\{ij\}}$, v_i and w_i are dual multipliers corresponding to constraints (4.26), (4.27) and (4.28) respectively. If P_{LP1} can be represented as $\{A\mathbf{p} + G\mathbf{y} \geq \mathbf{b}\}$. Then, any feasible vector $(\mathbf{w}, \mathbf{u}, \mathbf{v})$ to W defines a valid inequality: $(\mathbf{w}, \mathbf{u}, \mathbf{v})^T A\mathbf{p} \geq (\mathbf{w}, \mathbf{u}, \mathbf{v})^T \mathbf{b}$ in the space of the payment (p) variables. Furthermore, the projection of P_{LP1} is defined by the valid inequalities defined by the extreme rays of W . Let $S \subseteq V$ and S is connected in the graph G . Also, define $E^+(S)$ as the set of edges which have at least one end in S , $E^+(S) = \{\{i, j\} \in E : i \in S \text{ or } j \in S\}$.

Theorem 4.8. *The vector $\mathbf{r} = (\mathbf{u}, \mathbf{v}, \mathbf{w}) \in W$ is extreme if and only if there exists a positive α such that one of the following two cases holds true:*

1. $u_{\{ij\}} = \alpha$ for one $\{i, j\} \in E$. All other u, v, w are 0.
2. Given a $S \subseteq V$ and S is connected, $u_{\{ij\}} = \alpha$ for all $\{i, j\} \in E^+(S)$. In addition, either $v_i = \frac{\alpha}{d_i}$ or $w_i = \frac{\alpha}{l_i}$ for all $i \in S$. All other u, v, w are 0.

Proof. Proof of Theorem 4.8. Recall that a polyhedral cone C is the intersection of a finite number of half-spaces through the origin, and a pointed cone is one in which the origin is an extreme point. A ray of a cone C is the set $R(\mathbf{y})$ of all non-negative multipliers of some $\mathbf{y} \in C$, called the direction (vector) of $R(\mathbf{y})$. A vector $\mathbf{y} \in C$ is extreme, if for any $\mathbf{y}^1, \mathbf{y}^2 \in C$, $\mathbf{y} = 1/2(\mathbf{y}^1 + \mathbf{y}^2)$ implies $\mathbf{y}^1, \mathbf{y}^2 \in R(\mathbf{y})$. A ray $R(\mathbf{y})$ is extreme if its direction vector \mathbf{y} is extreme. Also, if for two distinct direction $\mathbf{y}^1 \neq \mathbf{y}^2 \in C$ and a $\beta \in [0, 1]$, $\mathbf{y} = \beta\mathbf{y}^1 + (1 - \beta)\mathbf{y}^2$ implies \mathbf{y} is not extreme.

Sufficiency. Let $\mathbf{r} \in W$ be of the form Case 1 and assume that $\mathbf{r} = \frac{1}{2}(\mathbf{r}^1 + \mathbf{r}^2)$ for some $\mathbf{r}^1, \mathbf{r}^2 \in W$. Then, except $u_{\{ij\}}^1$ and $u_{\{ij\}}^2$, all other directions are 0. Then, $\mathbf{r}^1, \mathbf{r}^2$ are in $R(\mathbf{r})$. So, \mathbf{r} is extreme.

Let $\mathbf{r} \in W$ be of the form Case 2 and assume that $\mathbf{r} = \frac{1}{2}(\mathbf{r}^1 + \mathbf{r}^2)$ for some $\mathbf{r}^1, \mathbf{r}^2 \in W$. So, for any component in \mathbf{r} with value 0, their corresponding components in \mathbf{r}^1 and \mathbf{r}^2 are also 0. Given i , let q_i^k , $k = 1, 2$, represent the positive component v_i^k or w_i^k , $k = 1, 2$. Then, we have $u_{\{ij\}}^1 + u_{\{ij\}}^2 = 2\alpha$ for all $\{i, j\} \in E^+(S)$ and $c_i p_i^1 + c_i p_i^2 = 2\alpha$ where $c_i = d_i$ if $v_i > 0$ and $c_i = l_i$ if $w_i > 0$ for all $i \in S$. Then, if there is a pair i and j , we have $c_i q_i^1 > c_j p_j^1$ if and only if $c_i p_i^2 < c_j q_j^2$. But constraint (4.41) imposes that $u_{\{ij\}}^k \geq c_i q_i^k$, $k = 1, 2$. Hence, $c_i q_i^k = c_j q_j^k = \alpha_k$, $k = 1, 2$, for all $i \in S$. Otherwise, either constraint (4.41) would be violated or $u_{\{ij\}}^1 + u_{\{ij\}}^2 > 2\alpha$ because $u_{\{ij\}}^k$ would take the larger value between $c_i q_i^k$ and $c_j q_j^k$, $k = 1, 2$. Thus, $\mathbf{r}^1, \mathbf{r}^2$ are in $R(\mathbf{r})$. Therefore, \mathbf{r} is

extreme.

Necessity. Let \mathbf{r} be an extreme vector of W . Based on this \mathbf{r} , let $C_{\mathbf{r}} = \{S \subseteq V : v_i > 0 \text{ or } w_i > 0 \forall i \in S \text{ \& } S \text{ is connected in the tree graph } G \text{ and maximal}\}$ and $S_{\{ij\}} = \{\{i, j\} \in E : u_{\{ij\}} > 0\}$. First, we consider the situation where $C_{\mathbf{r}} = \emptyset$ and assume $|S_{\{ij\}}| > 1$. Let \mathbf{r}^1 contain all but one of the positive components in \mathbf{r} with double their values. Let \mathbf{r}^2 contain the one positive component omitted by \mathbf{r}^1 in \mathbf{r} with double its value. Then, $\mathbf{r}^1, \mathbf{r}^2 \in W$ and $\mathbf{r} = \frac{1}{2}(\mathbf{r}^1 + \mathbf{r}^2)$. So, if $|S_{\{ij\}}| > 1$, \mathbf{r} is not extreme, contrary to the assumption. We conclude that if $C_{\mathbf{r}} = \emptyset$, then $|S_{\{ij\}}| = 1$ and thus \mathbf{r} is in the form of Case 1.

Now consider the situation when $C_{\mathbf{r}} \neq \emptyset$. When $|C_{\mathbf{r}}| > 1$, it means there is more than one connected component. Consider any set $S \in C_{\mathbf{r}}$. Then, \mathbf{r}^1 has values $u_{\{ij\}}^1 = 2u_{\{ij\}}, v_i^1 = 2v_i$ for all $i \in S$ and $w_i^1 = 2w_i$ for all $\{i, j\} \in E^+(S)$ and 0s in other components. Then, let \mathbf{r}^2 be $2\mathbf{r} - \mathbf{r}^1$. Hence, we have $\mathbf{r} = \frac{1}{2}(\mathbf{r}^1 + \mathbf{r}^2)$ and $\mathbf{r}^1, \mathbf{r}^2$ are different in at least one direction. So, if $|C_{\mathbf{r}}| > 1$, \mathbf{r} is not extreme, contrary to the assumption.

When $|C_{\mathbf{r}}| = 1$ (so the set of nodes with $v_i > 0$ or $w_i > 0$ in the original tree G form a single connected component) and $S \in C_{\mathbf{r}}$, define $S_1 = \{\{i, j\} \in E \setminus E^+(S) : u_{\{ij\}} > 0\}$. If $S_1 \neq \emptyset$, let \mathbf{r}^1 have values $u_{\{ij\}}^1 = 2u_{\{ij\}}, v_i^1 = 2v_i$ and $w_i^1 = 2w_i$ for all $\{i, j\} \in E^+(S)$ and all $i \in S$ and 0s in other components. Then, let \mathbf{r}^2 be $2\mathbf{r} - \mathbf{r}^1$. \mathbf{r}^2 is feasible and it has at least one positive component because S_1 is not empty and its corresponding components are not in \mathbf{r}^1 . Hence, we have $\mathbf{r} = \frac{1}{2}(\mathbf{r}^1 + \mathbf{r}^2)$ and $\mathbf{r}^1, \mathbf{r}^2$ are different in at least one direction. So, if $|C_{\mathbf{r}}| = 1$ and $S_1 \neq \emptyset$, \mathbf{r} is not extreme, contrary to the assumption.

When $|C_{\mathbf{r}}| = 1$, define $S_2 = \{i \in S : v_i > 0 \text{ \& } w_i > 0\}$. If $S_2 \neq \emptyset$ and let $i \in S_2$, then, we have $d_i v_i + l_i w_i \leq u_{\{ij\}}$ for all $j \in a(i)$. We make \mathbf{r}^1 have same values as \mathbf{r}

except that $v_i^1 = \frac{d_i v_i + l_i w_i}{d_i}$ and $w_i^1 = 0$. Similarly, let \mathbf{r}^2 have same values as \mathbf{r} except that $v_i^2 = 0$ and $w_i^2 = \frac{d_i v_i + l_i w_i}{l_i}$. Then, $\mathbf{r}^1 \neq \mathbf{r}^2$ and $\mathbf{r} = \beta \mathbf{r}^1 + (1 - \beta) \mathbf{r}^2$ where $\beta = \frac{d_i v_i}{d_i v_i + l_i w_i}$. Thus we must have $|S_2| = \emptyset$.

Recall that we use q_i to generically represent either of the positive components v_i or w_i and use $c_i q_i$ for $d_i v_i$ or $l_i w_i$. Next, if $|C_{\mathbf{r}}| = 1$, $|S_1| = \emptyset$ and $|S_2| = \emptyset$, consider an edge $\{i, j\} \in E^+(S)$. Define $S_3 = \{\{i, j\} \in E^+(S) : u_{\{ij\}} > c_i q_i \ \& \ u_{\{ij\}} > c_j q_j\}$. When $S_3 \neq \emptyset$, consider an edge $\{i, j\} \in S_3$. We can make \mathbf{r}^1 have $u_{\{ij\}}^1 = 2 \times (u_{\{ij\}} - \max\{c_i q_i, c_j q_j\})$ and 0s in the other components. Then, let \mathbf{r}^2 be $2\mathbf{r} - \mathbf{r}^1$ (note that \mathbf{r}^2 is feasible). Hence, we have $\mathbf{r} = \frac{1}{2}(\mathbf{r}^1 + \mathbf{r}^2)$ and $\mathbf{r}^1, \mathbf{r}^2$ are different in at least one direction because \mathbf{r}^1 only has one positive component and \mathbf{r}^2 has at least two positive components. Thus, if $|C_{\mathbf{r}}| = 1$ then $S_3 = \emptyset$.

Now, define $S_4 = \{\{i, j\} \in E^+(S) : i \neq j \in S \ \& \ u_{\{ij\}} > c_i q_i \oplus u_{\{ij\}} > c_j q_j\}$. If $S_4 \neq \emptyset$, consider an edge $\{i, j\} \in S_4$ such that its $u_{\{ij\}}$ has the smallest value among all edges in S_4 . Without loss of generality, let $u_{\{ij\}} = \gamma = c_j q_j > c_i q_i$ and v_i take positive value. Note that for a edge $\{i, k\}$, $c_k q_k$ is either bigger than or equal to $c_i q_i$ because if it is smaller than $c_i q_i$, $u_{\{ij\}}$ is not the smallest value among all edges in S_4 . We have two situations. First, when $u_{ik} > c_i q_i$ for all k in $a(i)$, We can make \mathbf{r}^1 have the same values as \mathbf{r} except that $v_i^1 = v_i + \epsilon$ such that $d_i v_i^1 \leq u_{ij}$. Then, we can make \mathbf{r}^2 have the same values as \mathbf{r} except that $v_i^2 = v_i - \epsilon$. Hence, we have $\mathbf{r} = \frac{1}{2}(\mathbf{r}^1 + \mathbf{r}^2)$ and $\mathbf{r}^1, \mathbf{r}^2$ are different in v_i component.

Second, when $u_{ik} = c_i q_i$ for some k in $a(i)$, we obtain a subgraph T_S in this way: let node i be the root of the tree induced by $E^+(S)$ and keep a branch $\{i, k\}$ if $u_{\{ik\}} = c_i q_i = c_k q_k$ for all k in $a(i)$. Then, we can make \mathbf{r}^1 and \mathbf{r}^2 in the following way: For all h

in $T_S \cap S$, if $c_h q_h \leq c_i q_i$, we have $q_h^1 = q_h \frac{\gamma}{c_i q_i}$ and $q_h^2 = 0$. Otherwise, we let $q_h^1 = q_h$ and $q_h^2 = q_h$. Next, for an edge $\{e, f\} \in T_S$, if $u_{\{ef\}} \leq c_i q_i$, $u_{\{ef\}}^1 = u_{\{ef\}} \frac{\gamma}{c_i q_i}$ and $u_{\{ef\}}^2 = 0$. Otherwise, $u_{\{ef\}}^1 = u_{\{ef\}}$ and $u_{\{ef\}}^2 = u_{\{ef\}}$. Other components are the same as \mathbf{r} . Then, $\mathbf{r}^1 \neq \mathbf{r}^2$ and $\mathbf{r} = \beta \mathbf{r}^1 + (1 - \beta) \mathbf{r}^2$ where $\beta = \frac{c_i q_i}{\gamma}$. Thus we must have $|S_4| = \emptyset$.

Consider an edge $\{i, j\}$ in the graph G where both $i, j \in S$, we have $u_{\{ij\}} = c_i q_i = c_j q_j$ because $S_3 = \emptyset$ and $S_4 = \emptyset$. Given S is a connected component, we have $c_i q_i = c_j q_j$ for $i \neq j$ in S ; and so $u_{\{ij\}} = \alpha$ for $\{i, j\} \in E^+(S)$. This proves that if $C_r \neq \emptyset$, then \mathbf{r} must be in the form of Case 2. \square

Applying Theorem 2 in Balas and Pulleyblank [1983]---Case 1 extreme directions generate the valid inequality $0 \geq -1$ which is not very useful. Case 2 extreme directions generate the following valid inequality in the graph G :

$$|E^+(S)| + \sum_{i \in S: v_i > 0} \frac{p_i}{d_i} + \sum_{i \in S: w_i > 0} \frac{p_i}{l_i} \geq \sum_{i \in S: v_i > 0} \frac{b_i}{d_i} + \sum_{i \in S: w_i > 0} g_i \quad \forall S \subseteq V \text{ \& } S \text{ is connected.}$$

Noting that given a set S , there are many extreme rays \mathbf{r} satisfying Case 2. We can partition S into two sets, S_v^S and S_w^S . They satisfy that $S_v^S \cap S_w^S = \emptyset$ and $S_v^S \cup S_w^S = S$. Then, let $P_{LP4.4}$ denote the feasible region of **LP1**, we can write the projection of $P_{LP4.4}$ onto the \mathbf{p} space as:

$$|E^+(S)| + \sum_{i \in S_v^S} \frac{p_i}{d_i} + \sum_{i \in S_w^S} \frac{p_i}{l_i} \geq \sum_{i \in S_v^S} \frac{b_i}{d_i} + \sum_{i \in S_w^S} g_i \quad (4.44)$$

$$\forall S \subseteq V \text{ \& } S \text{ is connected \& } S_v^S \cap S_w^S = \emptyset \text{ \& } S_v^S \cup S_w^S = S$$

$$p_i \geq 0 \quad \forall i \in V \quad (4.45)$$

Proposition 4.2. *The valid inequalities (4.44) can be separated in $O(|V|^3)$ time.*

Proof. Proof of Proposition 4.2. When the size of S is h , the separation procedure of inequalities (4.44) can be stated as the following optimization problem: Given a tree $G = (V, E)$ and an integer $h \leq |V|$, each node i in V has a weight which takes value from $\{c_{ij} : j = 0, 1, \dots, \deg(i)\}$ based on the number of neighbors not in the selected component, denoted by j . The goal is to find a connected component S with h nodes whose weight $\sum_{i \in S} c_{ij}$ is the minimum.

In Blum [2007], a dynamic programming algorithm is proposed for a generalized k -minimum spanning (k -MST) problem when the given graph is a tree. Its definition is as follows: Given a tree $G_B = (V_B, E_B)$ and an integer $k < |V_B|$, each node i in V_B has a weight w_i and each edge in E_B has a weight e_{ij} , the goal is to find a subtree T of k edges whose weight $\sum_{i \in T} w_i + \sum_{(i,j) \in T} e_{ij}$ is the minimum.

Given an instance of the separation problem with the cardinality of S specified to be h , we show that it is equivalent to the generalized k -MST problem on trees. From the input graph G , we transform it into graph G_t by adding one dummy node to each edge in G . For each edge $\{i, j\} \in E$, insert a dummy node d . Let D denotes the set of dummy nodes. Since the dummy nodes have effectively split each edge into two in the original graph, we replace each of the original edges $\{i, j\} \in E$ by two edges $\{i, d\}$ and $\{d, j\}$ in the new graph G_t . Let E_t denote the set of edges in G_t . Thus, $G_t = (V \cup D, E_t)$. Recall that we have a solution \mathbf{p} . For all i in V , set its weight $w_i = \min\{\frac{p_i - b_i}{d_i}, \frac{p_i}{l_i} - g_i\} - 1$. Then, for all d in D , set its weight $w_d = 0$. Lastly, for each (i, d) in E_t , we have edge weight $e_{id} = 1$. We also set $k = 2(h - 1)$ as the target cardinality. Let L be the number of nodes with degree one in T . The objective value of T is: $\sum_{i \in T \cap V} (\min\{\frac{p_i - b_i}{d_i}, \frac{p_i}{l_i} - g_i\} - 1) + \sum_{(i,d) \in T} 1 = \sum_{i \in T \cap V} \min\{\frac{p_i - b_i}{d_i}, \frac{p_i}{l_i} - g_i\} - h + 2h - 1 + L = \sum_{i \in T \cap V} \min\{\frac{p_i - b_i}{d_i}, \frac{p_i}{l_i} - g_i\} + h - 1 + L$.

Let \bar{T} denote $T \cap V$. In the original graph, $h - 1 + L$ is equal to $|E^+(S)|$. Thus, the objective value is equal to $\sum_{i \in T \cap V} \min\{\frac{p_i - b_i}{d_i}, \frac{p_i}{l_i} - g_i\} + |E^+(S)|$. This is exactly the constraint (4.44) with $S = \bar{T}$ after we move $\sum_{i \in S_v^S} \frac{b_i}{d_i} + \sum_{i \in S_w^S} g_i$ to the left hand side and consider it as a minimization problem by selecting nodes. If the objective value is smaller than 0, we have found a violated inequality. Otherwise, there is no violated inequality for sets S with cardinality h .

The time complexity of Blum's algorithm is $O(k^2|V_B|)$. It returns the values of the best l -cardinality trees in G_B for all l values in the range $0 \leq l \leq k$. Thus, we only need to run it once by setting $k = |V_B| - 1$ ($k = |V_B|$ is trivial). Hence, in the worst case, the time complexity is $O(|V|^3)$ because k is bounded by $|V_B|$ and $|V_B| = 2|V|$. \square

4.4 From Trees to General Graphs: A Branch-and-Cut Approach

We now discuss how the TUM formulation MIP4.3 can be applied to general graphs. The key idea is that the influence propagation process can be modeled as a directed acyclic graph (DAG). As it stands MIP4.3 is not valid for graphs that contain cycles. Figure 4.9 explains the problem caused by a cycle. In this figure, we have a directed influence cycle consisting of nodes 1, 2 and 3. Each of them receives influence equal to their threshold from one incoming arc and so no payments are required to influence all 3 nodes. For instance, node 1 receives 5 unit influence from node 3 ($x_{31}^L = 5$), node 2 receives 7 units of influence from node 1 ($x_{12}^L = 7$), and node 3 receives 9 units of influence from node 2 ($x_{23}^L = 9$). Clearly, this is not a feasible solution. One node must have been paid incentives equal to its threshold (in the optimal solution it is cheapest to pay node 1 its threshold 5)

to start the influence propagation process.

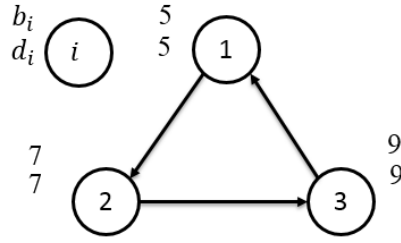


Figure 4.9: A pathological example of a cycle in influence propagation

Our formulation for general graphs will model the fact that the influence propagation network in a feasible solution to the LCIP should be acyclic. For this, we bring back the binary variables y_{ij} that tells us whether or not node i influences node j from MIP4.2. To model the fact that the influence propagation network should be acyclic we add a set of constraints which ensures that the directed graph formed by \mathbf{y} , denoted by $G(\mathbf{y})$, has to be a DAG is needed as well. With this in hand, we have the following formulation that we refer to as BIP4.6:

$$(BIP4.6) \quad \text{Max} \quad \sum_{i \in V} \sum_{j \in a(i)} \sum_{k \in \{H, L, Z\}} c_i^k x_{ji}^k \quad (4.46)$$

$$\text{Subject to} \quad y_{ji} = \sum_{k \in \{H, L, Z\}} x_{ji}^k \quad \forall i \in V, j \in a(i) \quad (4.47)$$

$$\sum_{(i,j) \in C} y_{ij} \leq |C| - 1 \quad \forall \text{ dicycles } C \text{ in } G(\mathbf{y}) \quad (4.48)$$

$$y_{ij} + y_{ji} = 1 \quad \forall (i, j) \in E \quad (4.49)$$

$$\sum_{j \in a(i)} x_{ji}^H \leq g_i - 1 \quad \forall i \in V \quad (4.50)$$

$$\sum_{j \in a(i)} x_{ji}^L \leq 1 \quad \forall i \in V \quad (4.51)$$

$$x_{ij}^k \in \{0, 1\} \quad \forall i \in V, j \in a(i), k \in \{H, L, Z\} \quad (4.52)$$

$$y_{ij} \in \{0, 1\} \quad \forall i \in V, j \in a(i) \quad (4.53)$$

We have three new constraint sets. Constraint set (4.47) is a linking constraint which connects x and y variables. Constraint set (4.48), called k -dicycle inequalities, says that the directed influence propagation network formed by y_{ij} , denoted by $G(\mathbf{y})$, must be a DAG. Constraint set (4.53) enforces the binary constraint on variable y_{ij} . The objective function and the rest of the constraint sets are the same as BIP1. We refer to this new formulation as BIP4.6. Similarly, we can obtain another formulation based on MIP4.4 for general graphs. This one is referred to as MIP4.7 and is given below:

$$(MIP4.7) \quad \text{Min} \quad \sum_{i \in V} p_i \quad (4.54)$$

$$\text{Subject to} \quad \sum_{(i,j) \in C} y_{ij} \leq |C| - 1 \quad \forall \text{ dicycles } C \text{ in } G(\mathbf{y}) \quad (4.55)$$

$$y_{ij} + y_{ji} = 1 \quad \forall \{i, j\} \in E \quad (4.56)$$

$$p_i + \sum_{j \in a(i)} d_i y_{ji} \geq b_i \quad \forall i \in V \quad (4.57)$$

$$p_i + \sum_{j \in a(i)} l_i y_{ji} \geq l_i g_i \quad \forall i \in V \quad (4.58)$$

$$p_i \geq 0 \quad \forall i \in V \quad (4.59)$$

$$y_{ji} \in \{0, 1\} \quad \forall j \in V, i \in a(j) \quad (4.60)$$

The natural question is that which formulation is better on general graph? We answer this question theoretically by the following theorem.

Theorem 4.9. *BIP4.6 and MIP4.7 are equivalent to each other in terms of the strength of their LP relaxations.*

Proof. Proof of Theorem 4.9. Let BLP6 and MLP7 be the LP relaxations of BIP4.6 and MIP4.7 respectively. We want to show that these two LP relaxations are equivalent. First, given an optimal solution of BLP6, $(\bar{\mathbf{y}}, \bar{\mathbf{x}})$, we can construct a feasible solution $(\mathbf{y}^m, \mathbf{p}^m)$ for

MLP7 with the same total payment. Set $\mathbf{y}^m = \bar{\mathbf{y}}$ and $p_i^m = b_i - \sum_{j \in a(i)} \sum_{k \in t(i)} c_i^k \bar{x}_{ji}^k$ for all i in V . In this way, $p_i^m \geq 0$ because $\sum_{j \in a(i)} \sum_{k \in t(i)} c_i^k \bar{x}_{ji}^k \leq b_i$ due to constraint (4.50) and (4.51). Constraint (4.55) and (4.56) are satisfied because $\bar{\mathbf{y}}$ satisfies constraint (4.48) and (4.49). Also, Constraint (4.57) is satisfied because $p_i^m = b_i - \sum_{j \in a(i)} \sum_{k \in t(i)} c_i^k \bar{x}_{ji}^k \geq b_i - d_i \sum_{j \in a(i)} \sum_{k \in t(i)} \bar{x}_{ji}^k = b_i - d_i \sum_{j \in a(i)} y_{ji}^m$ for all i in V . Lastly, for constraint (4.58), we have three cases. If $\sum_{j \in a(i)} y_{ji}^m \geq g_i$, we just need $p_i^m \geq 0$ which is already satisfied. If $\sum_{j \in a(i)} y_{ji}^m \leq g_i - 1$, we have $p_i^m = b_i - d_i \sum_{j \in a(i)} y_{ji}^m = l_i + d_i(g_i - 1 - \sum_{j \in a(i)} y_{ji}^m) \geq l_i + l_i(g_i - 1 - \sum_{j \in a(i)} y_{ji}^m) = g_i l_i - \sum_{j \in a(i)} l_i y_{ji}^m$. If $g_i - 1 < \sum_{j \in a(i)} y_{ji}^m < g_i$, we use the fact that $\sum_{j \in a(i)} \bar{x}_{ji}^H = g_i - 1$ and $\sum_{j \in a(i)} \bar{x}_{ji}^L = \sum_{j \in a(i)} y_{ji}^m - \sum_{j \in a(i)} \bar{x}_{ji}^H$ because this is a maximization problem and increasing the value of $\sum_{j \in a(i)} \bar{x}_{ji}^H$ first improves the objective value. Thus, $p_i^m = b_i - \sum_{j \in a(i)} \sum_{k \in t(i)} c_i^k \bar{x}_{ji}^k = b_i - d_i \sum_{j \in a(i)} \bar{x}_{ji}^H - l_i \sum_{j \in a(i)} \bar{x}_{ji}^L = l_i - l_i \sum_{j \in a(i)} \bar{x}_{ji}^L = l_i - l_i(\sum_{j \in a(i)} y_{ji}^m - g_i + 1) = l_i g_i - \sum_{j \in a(i)} l_i y_{ji}^m$.

Second, given an optimal solution of MLP7, $(\mathbf{y}^m, \mathbf{p}^m)$, a feasible solution $(\bar{\mathbf{y}}, \bar{\mathbf{x}})$ is constructed for BLP6 with the same total payment. Set $\bar{\mathbf{y}} = \mathbf{y}^m$ first. Then, for a node i in V , based on its influence type, we set up its associated $\bar{\mathbf{x}}$ variables in the following way: First, if $\sum_{j \in a(i)} \bar{y}_{ji} \geq g_i$, set $\bar{x}_{ji}^H = \frac{g_i - 1}{\sum_{j \in a(i)} \bar{y}_{ji}} \bar{y}_{ji}$, $\bar{x}_{ji}^L = \frac{1}{\sum_{j \in a(i)} \bar{y}_{ji}} \bar{y}_{ji}$, and $\bar{x}_{ji}^Z = \frac{\sum_{j \in a(i)} \bar{y}_{ji} - g_i}{\sum_{j \in a(i)} \bar{y}_{ji}} \bar{y}_{ji}$. Constraint (4.58) ensures that for a node i , when $\sum_{j \in a(i)} \bar{y}_{ji} \geq g_i$, we have $p_i^m = 0$. So, by construction, constraint (4.50) and (4.51) are binding. Thus, p_i^b is 0. Second, if $\sum_{j \in a(i)} \bar{y}_{ji} \leq g_i - 1$, set $\bar{x}_{ji}^H = \bar{y}_{ji}$, $\bar{x}_{ji}^L = 0$ and $\bar{x}_{ji}^Z = 0$. Because $p_i^m = b_i - \sum_{j \in a(i)} d_i \bar{y}_{ji}$ and $\bar{p}_i = b_i - \sum_{j \in a(i)} d_i \bar{x}_{ji}^H$, thus, $p_i^m = \bar{p}_i$. Lastly, if $g_i - 1 < \sum_{j \in a(i)} \bar{y}_{ji} < g_i$, set $\bar{x}_{ji}^H = \frac{g_i - 1}{\sum_{j \in a(i)} \bar{y}_{ji}} \bar{y}_{ji}$, $\bar{x}_{ji}^L = \frac{\sum_{j \in a(i)} \bar{y}_{ji} - g_i + 1}{\sum_{j \in a(i)} \bar{y}_{ji}} \bar{y}_{ji}$ and $\bar{x}_{ji}^Z = 0$. Here, we have $p_i^m = l_i g_i - \sum_{j \in a(i)} l_i \bar{y}_{ji} \geq b_i - \sum_{j \in a(i)} d_i \bar{y}_{ji}$ and $\bar{p}_i = b_i - (g_i - 1)d_i - \sum_{j \in a(i)} l_i \bar{x}_{ji}^L$, therefore, $p_i^m = \bar{p}_i$. Note that in the latter two cases, by construction, constraint (4.50)

and (4.51) are satisfied. □

Although their LP relaxations are equivalent on general graphs, the TUM only has binary variables and the coefficients are either 0 or 1. This makes it more computational efficient in our experiments. Therefore, based on the TUM formulation, we propose a branch-and-cut approach for solving the LCIP on general networks. The main tasks in our branch-and-cut approach are identifying strong valid inequalities, separating these violated valid inequalities, handling symmetry and using a specialized branching rule, which we now discuss.

In a solution of the LCIP, by the definition of the DAG, the arc set of the directed influence propagation network, $G(\mathbf{y})$ contains no dicycles. Therefore, there are at most $|C| - 1$ members of the arc set of a dicycle C in $G(\mathbf{y})$. Then, for a dicycle C , referred to as a k -dicycle inequality, $\sum_{(i,j) \in C} y_{ij} \leq |C| - 1$, is a valid inequality for the LCIP. We use the separation procedure described by Grötschel et al. [1985] for the k -dicycle inequalities. For example, for the cycle in Figure 4.9, we can use $y_{12} + y_{23} + y_{31} \leq 2$ to cut off that cycle.

Further, we make an important observation that in BIP4.6: it suffices to define the y variables as binary (i.e., integrality can be relaxed on the x variables). This can be helpful in terms of branching.

Proposition 4.3. *BIP4.6 needs binary constraint only on the y variables.*

Proof. Proof of Proposition 4.3. Once the the y variables are fixed to either 0 or 1, the remaining constraint matrix is totally unimodular. This is easily seen by observing that the rows of the remaining constraints can be partitioned into two subsets Q_1 containing

constraint set (4.47) and Q_2 containing constraint sets (4.16) and (4.17). With this we have a 0-1 constraint matrix with no more than 2 nonzero entries per column. Further, columns that have two nonzero elements have one of the nonzero coefficients in Q_1 and one of the nonzero coefficients in Q_2 . □

We take advantage of Proposition 4.3 in implementing a branching rule. In BIP4.6, the y variables are given higher branching priority. It means that whenever it is possible, the branching rule is to select the y variables first. We call this **priority branching**.

An MIP is symmetric if its variables can be permuted without changing the structure of the problem. This could cause trouble in a computational sense (i.e., the branch-and-cut procedure can take a long time) because the search procedure wastes effort and time on eliminating symmetric solution. In our problem, there are 2 types of symmetry which lead to multiple optimal solutions. We use perturbations intelligently to reduce symmetry. Figures 4.10 and 4.11 show examples of these two kinds of symmetries.

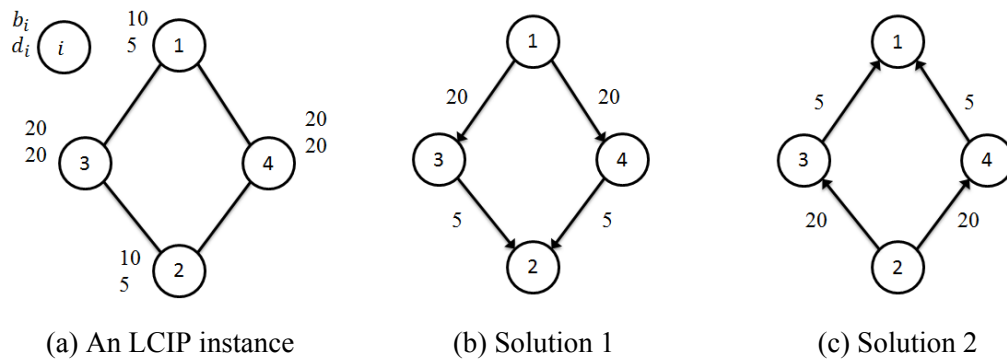


Figure 4.10: Influence Direction Symmetry

Influence direction symmetry (y perturbations): Given the LCIP problem in Figure 4.10(a), we have two solutions with the same objective value, 50, but different influence directions as shown in Figure 4.10(b) and (c). To reduce this kind of symmetry, we

perturb (subtract) in the amount of θ_{ij} , from the cost coefficient of y_{ij} where the value of θ_{ij} should satisfy the following conditions:

1. θ is non-negative, $\theta_{ij} \geq 0 \forall (i, j) \in G(\mathbf{y})$.
2. The sum of all θ is smaller than 1, $\sum_{\forall (i,j) \in G(\mathbf{y})} \theta_{ij} < 1$.
3. $\theta_{ij} > 0$ where $j > i$ and $\theta_{ij} = 0$ otherwise.
4. θ_{ij} is distinct on different y_{ij} where $j > i$.

Proposition 4.4. *If θ_{ij} satisfies conditions 1 and 2, among those optimal solutions without the perturbation, at least one of them will remain optimal after θ_{ij} is subtracted.*

Proof. Proof of Proposition 4.4. Condition 1 ensures that the objective value of a solution is non-increasing after the perturbations are subtracted. Otherwise, a non-optimal solution without the perturbation could increase its objective value after subtracting the perturbation and outperform an optimal solution. Condition 2 ensures that the decrement in the objective value of an optimal solution will not make the solution inferior to any non-optimal solutions. We have the assumption that all input data are integers. So, before we subtract the perturbation, the objective value of an optimal solution is at least 1 unit better than that of a non-optimal solution. These two conditions together ensure those optimal solutions remain superior compared to non-optimal solutions. Among these optimal solutions, the one with the smallest value of perturbations becomes an optimal solution after θ_{ij} is subtracted. □

Condition 3 gives higher preference to direction from node i to node j where $i > j$ and eliminates the symmetry between these two directions. It also fixes the value of θ_{ij}

to zero when $i > j$ and allows us to focus on assigning values to the remaining θ_{ij} where $j > i$. However, when conditions 1, 2 and 3 are satisfied, they are still not enough to handle the symmetry in Figure 4.10(b) and 4.10(c). If all θ_{ij} , where $j > i$, have the same value, these two solutions will have the same objective value after we subtract the perturbations. Condition 4 further reduces symmetry in this case. Therefore, condition 4 is needed. For the example in Figure 4.10, we set $\theta_{13} = 0.05, \theta_{14} = 0.10, \theta_{23} = 0.15, \theta_{24} = 0.20$ and other $\theta = 0$. Solution 1 becomes the unique optimal solution.

In order to satisfy these four conditions of y perturbations, we suggest a way to set up the θ as follows:

Algorithm 15 Setting up θ

Require: the network $G = (V, E)$, the number of edges $|E|$
 Set $\tau = 1/|E|^2, k = 0$ and all $\theta_{ij} = 0$
for $(i, j) \in E$ **do**
 $k = k + 1$
 $\theta_{ij} = \tau k$ where $i < j$
end for

It is easy to see that conditions 1, 3 and 4 are satisfied. For condition 2, the maximum decrement caused by θ in the objective value is $\frac{1}{|E|^2}(1 + 2 + \dots + |E|) = \frac{1}{|E|^2} \frac{|E|(|E|+1)}{2} = \frac{|E|+1}{2|E|} < 1$ since $|E| > 1$. Therefore, condition 2 is satisfied as well.

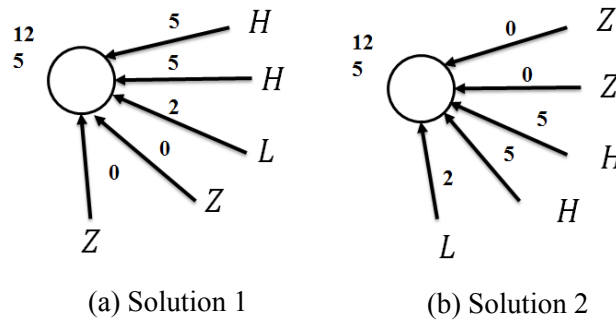


Figure 4.11: Influence Allocation Symmetry

Influence allocation symmetry (x perturbations): Figure 4.11 gives an example with a node which allocates influences as described in Section 4.3.3. We don't pay this node any incentives. There are multiple ways to allocate the influence to its incoming arcs. Figure 4.11(a) and Figure 4.11(b) are two of them. We perturb (subtract) the cost coefficient of x_{ij}^L by ϵ_{ij} and the cost coefficient of x_{ij}^H by δ_{ij} . For a node i , if it allocates influences as described in Section 4.3.3, the value of ϵ_{ij} and δ_{ij} should satisfy the following conditions:

1. These perturbations are non-negative, $\epsilon_{ij} \geq 0, \delta_{ij} \geq 0 \forall j \in a(i)$.
2. $\epsilon_{im} + \sum_{j \in M} \delta_{ij} < 1/|V|$ where $m = \arg \max_j \{\epsilon_{ij}\}$ is the index of the largest ϵ_{ij} and M is the set of indexes of the first $(g_i - 1)$ largest δ_{ij} .
3. For a fixed j , ϵ_{ij} is much bigger than δ_{ij} , $\epsilon_{ij} \gg \delta_{ij} \forall j \in a(i)$.
4. ϵ_{ij} and δ_{ij} are distinct on different arcs.

Proposition 4.5. *If ϵ_{ij} and δ_{ij} satisfy conditions 1 and 2, among those optimal solutions without the perturbations, at least one of them will remain optimal after ϵ_{ij} and δ_{ij} are subtracted.*

Proof. Proof. Condition 1 is used to ensure that non-optimal solutions will not have better objective values compared to optimal solutions after ϵ_{ij} and δ_{ij} are subtracted. If influences for node i are allocated as in described in Section 4.3.3, at most $(g_i - 1)$ arcs are of type H and at most one arc is of type L . Also, we have the assumption that all input data are integers. Therefore, condition 2 ensures that the amount of incentives paid to a node

will not be affected after ϵ_{ij} and δ_{ij} are subtracted. These two conditions together ensure at least one optimal solution remain optimal after ϵ_{ij} and δ_{ij} are subtracted. \square

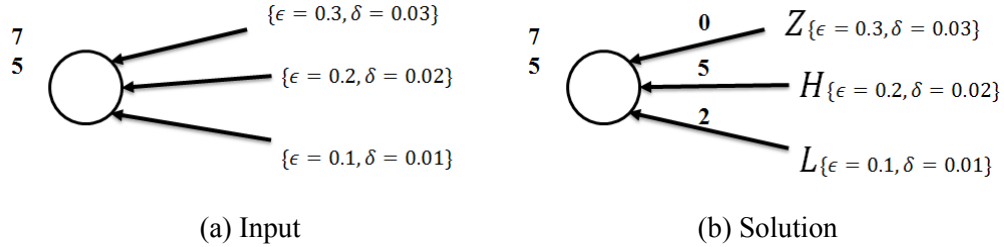


Figure 4.12: Illustration of Conditions 3 and 4

Figure 4.12(a) shows a situation where we need to allocate one H type, one L type and one Z type influence to three arcs. Condition 4 enforces that ϵ_{ij} and δ_{ij} are distinct. So, the Z type influence will go to the arc with the highest values of ϵ_{ij} and δ_{ij} because this type influence does not have perturbation assigned to it. In this example, the top arc is assigned the Z type influence. Condition 3 ensures $\epsilon_{ij} \gg \delta_{ij}$. So, combining with condition 4, L type influence is allocated to the one with the smallest ϵ value. In this example, it goes to the bottom arc. Then, the H type influence goes to the middle one. Figure 4.12(b) shows the unique optimal allocation with the objective value 6.88.

For the example in Figure 4.11, we increase the value of ϵ and δ by the counter clockwise orientation. The specific values are from 0.1 to 0.5 for ϵ and 0.01 to 0.05 for δ . Although these two solutions in Figure 4.11 have the same objective value without perturbations, solution 2 in Figure 4.11(b) with objective value 11.85 is the unique optimal solution after perturbation.

In order to satisfy these four conditions of x perturbations, using nodes that allocate influences as in Section 4.3.3, we suggest a way to set up the ϵ and δ as follows:

Algorithm 16 Setting up ϵ and δ

Require: a node i , the set of neighbors $a(i)$, the node type g_i and $deg(i) > g_i \geq 2$
Set $\tau = 1/|V|(deg(i) + 1)$, $\omega = 1/10|V|deg(i)^2$ and $k = 0$
for $j \in a(i)$ **do**
 $k = k + 1$
 $\epsilon_{ij} = \tau k$
 $\delta_{ij} = \omega k$
end for

It is easy to see that conditions 1 and 4 are satisfied. Condition 2 is satisfied because the maximum decrement caused by l and h in the objective value is $deg(i)/|V|(deg(i) + 1) + \{deg(i) + (deg(i) - 1) + (deg(i) - 2) + \dots + (deg(i) - g_i + 1)\}/10|V|deg(i)^2 < deg(i)/(deg(i) + 1) + (1 + 2 + \dots + deg(i))/10|V|deg(i)^2 = deg(i)/(deg(i) + 1) + (deg(i) + 1)/20|V|deg(i) < 1/|V|$ because we have $deg(i) > 1$. For a fixed j , $\epsilon_{ij}/\delta_{ij} = 10deg(i)^2/(deg(i) + 1) \approx 10deg(i)$. Thus, $\epsilon_{ij} \gg \delta_{ij}$ and condition 3 is satisfied. Therefore, these 4 conditions are all satisfied.

If we want to handle these two types of symmetry simultaneously, we need to divide all perturbation values by two.

After adding these perturbations, although it reduces the symmetry, it also introduces many solutions which have small difference in the objective value especially when the solutions are close to the optimal one. Then, the search procedure wastes time on identifying the optimal solution with perturbations even though it has found the optimal one before the perturbations are applied. Thus, given that the total amount of the perturbations applied to a solution is less than one, we propose the following **termination rule**. Let z^* denote the incumbent objective value and u denote the current dual bound. If $\lceil z^* \rceil \geq u$, we stop and return the incumbent solution as the optimal solution.

The next component is called **conservative separation procedure**. We realize that the separation procedure is expensive. Therefore, we modify the branch-and-cut procedure in the following way. In the root node, we do our best to find violated inequalities in order to achieve a better dual bound, (i.e., we focus on the cutting plane method). However, once we enter the branching phase, the separation procedure is only invoked when the integral constraints are satisfied at a node rather than at every node. The reason is that even if the integral constraints are satisfied, the solution could still be infeasible because it may contain a cycle.

Lastly, in the formulation BIP4.6, we use additional variables to model the influence propagation process. Thus, the MIP search process spends a significant amount of time on these variables because of the binary requirement on these variables. However, we actually are only interested in the payment vector (i.e., the natural node (\mathbf{p}) variables which can be easily decided from x and y variables). Consequently, for a given \mathbf{p} , we consider the feasibility of the solution and determine the set of nodes that can be activated using this payment vector. We call this process as **Feasibility Lift**. If all nodes are activated by this payment vector, we have obtained a feasible solution and the current node of the branch-and-bound tree can be fathomed. Otherwise, we can continue the branch-and-cut search as usual. One advantage of Feasibility Lift is that we can focus the separation procedure on the subgraph induced by the inactive nodes (because there must be some cycles to help them satisfy their thresholds). We refer to this subgraph as the **Inactive Induced Graph**. In this way, we have a smaller supporting graph and can add fewer violated inequalities in the branch-and-cut procedure.

4.5 Computational Experiments

We now discuss our computational experience with the branch-and-cut approach. We generated networks using the method proposed by Watts and Strogatz [1998] in their pioneering work on social network analytics. As indicated in the Stanford large network dataset collection [Leskovec, 2011], real social networks are sparse. We took this into account as follows. For sparsity, we generated network with average degree number 4. We chose the rewiring probability p as 0.3 (loosely, it is the probability that an edge is reconnected to a uniformly chosen node after initializing a ring with pre-specified average degree), because Watts and Strogatz [1998] showed this corresponds most closely to the social networks they studied.

For influence factor and threshold associated with a node, we first randomly generated node type g_i from a discrete uniform distribution between $[1, deg(i)]$ and influence factor d_i from a discrete uniform distribution between $[1, 50]$. The threshold for a node i was calculated as $b_i = d_i(g_i - 1) + s$, where s is generated from a discrete uniform distribution between $[1, d_i]$. By this method, we ensure that if all neighbors of a node are active, this node will become active as well.

We used CPLEX 12.6, Python API, coded our separation routines in C and ran our tests on a machine with the following specifications: Intel i5 3.40GHz, 24 GB ram, Ubuntu 14.04. In our implementation, two preprocessing steps are applied to the formulation on general networks. The first one is that we removed the constraint $y_{ij} + y_{ji} = 1$ and used only variables y_{ij} where $i < j$ (so y_{ji} is replace by $1 - y_{ij}$ in the model). In this way, we reduced the size of the model by $|E|$ constraints and $|E|$ variables. The second one is that

in addition to the constraints in the formulation, we also added all 3-dicycle constraints in the third experiment.

ID	LP4.7			LP4.6			BIP4.6			
	Obj. Value	Time (S)	LP4.7/BIP4.6 (%)	Obj. Value	Time (S)	LP4.6/BIP4.6 (%)	Obj. Value	Time (S)	Obj. Value	Time (S)
1	1624.88	0.24	54.40	2919	0.59	97.72	2987	6.28	2987	6.28
2	1597.88	0.21	55.16	2781	0.25	96.00	2897	10.36	2897	10.36
3	1558.10	0.33	53.69	2797	0.81	96.38	2902	7.92	2902	7.92
4	1212.50	0.31	47.92	2448	0.63	96.76	2530	9.54	2530	9.54
5	1421.38	0.23	51.07	2686	0.59	96.51	2783	9.97	2783	9.97
6	1838.72	0.20	52.28	3426	0.26	97.41	3517	4.84	3517	4.84
7	1559.34	0.22	51.81	2923	0.64	97.11	3010	9.48	3010	9.48
8	1741.08	0.31	53.46	3150	0.62	96.71	3257	8.40	3257	8.40
9	2015.51	0.20	57.62	3386	0.66	96.80	3498	10.62	3498	10.62
10	1845.59	0.27	58.39	3032	0.71	95.92	3161	14.20	3161	14.20

Table 4.2: LP Relaxation of MIP4.7 and BIP4.6 Formulation for the LCIP with 1000-Node Instances.

In our first set of experiments we study the strength of the LP relaxation of BIP4.6 (LP4.6). For this, ten instances with 1000 nodes and 2000 edges are generated. Note that we can obtain a valid formulation from MIP4.7 by including k -dicycle constraints. We refer to the resulting formulation as MIP4.7 from now on. The objective value of LP4.6 is compared against the LP relaxation of MIP4.7, denoted by LP4.7. Table 4.2 shows the results. The first column provides the instance number. ``LP4.7" and ``LP4.6" have the results for the LP4.7 and LP4.6 respectively. For each of them, we present objective value (``Obj. Value"), running time (``Time") and the linear relaxation bound (``LP4.7/BIP4.6" and ``LP4.6/BIP4.6") compared to the optimal value showed in ``BIP4.6". The value of the optimal solution that is obtained by solving BIP4.6 (with the best setting for branch-and-cut which will be discussed later in this section). For LP4.7, its LP relaxation can be solved within 0.25 seconds on average but provides a much worse LP bound than that of the BIP4.6 formulation. The LP bound of IP7 is about 53% on average. However, LP4.6 only needs about 0.57 seconds and provides an excellent LP bound. The LP bound of BIP4.6 is about 97%. If we look at the optimality gap which is calculated as $(1 - \text{LP bound})$, the optimality gap of LP4.6 is only 3 % which is less than one tenth of that obtained by the MIP4.7. Overall, the proposed model BIP4.6 is a much better formulation. Here, we do not include the MIP4.1. The reason is that for this formulation, we analytically show that it is not a strong formulation. For an LCIP instance, let the graph be a complete graph with n nodes. For each node i in V , b_i is $n - 1$ and d_i is 1. If we apply the LP relaxation of the MIP4.1 to this instance, a feasible solution has $p_i = \frac{n-1}{n}$, $v_{it} = \frac{1}{n}$, and $y_{it} = \frac{t}{n}$ for all i in V and $t \in \{1, 2, \dots, n\}$ (note that $T = n$ and $x_i = 1$ for all i in V). The objective value of this fractional solution is $n - 1$. For the integer optimal solution, initially a node must

be picked to start the influence propagation process. Then, a complete graph with $(n - 1)$ nodes is left with $b_i = n - 1$ for all nodes. Therefore, we can repeat this procedure until we only have two nodes left and we have to pick one of them with cost 1. This optimal solution has objective value $\frac{n(n-1)}{2}$. Then, given that we only use a feasible solution for the LP relaxation, the LP bound is no better than $\frac{2}{n}$ which decreases to 0 in the limit as the size of the graph increases.

In order to demonstrate the effects of the components in our branch-and-cut procedure, we conducted a set of experiments with the branch-and-cut procedure on ten instances with 10,000 nodes and 20,000 edges (after bidirecting these edges we have 40,000 arcs). We construct an initial feasible solution using a greedy heuristic. We run two algorithms. One is motivated the greedy algorithm on trees which we call Tree Greedy which calculates $m_i = \min\{b_i, d_i\}$ for a node i in V first and keeps selecting the node with the smallest m value among the inactive nodes. The other one is a Cost Greedy heuristic which repeatedly selects the node with smallest cost among those inactive ones (pays the remaining threshold as an incentive to activate it) and applies the propagation rule until a solution is found. For both heuristics, if there is a tie, we break it by picking the node with the highest degree number in the remaining graph. If a tie still exists, it is broken arbitrarily. Then, we choose the better solution from Tree Greedy and Cost Greedy as the initial solution for the Branch-and-Cut approach. All runs are capped to a 10-minute time limit.

Table 4.3 contains the results for 10000-node instances. We have two settings: ``BIP4.6" is the straight implementation of BIP4.6 formulation with k -dicycle separation. ``All" adds all other tricks to the ``BIP4.6" setting. Although we can relax integrality on

Optimality Gap (%)			Summary		
ID	BIP4.6	All		BIP4.6	All
1	3.11	0.33	Avg Gap (%)	2.88	0.26
2	2.56	0.17	Max Gap (%)	3.44	0.41
3	2.77	0.32	# of Optimal Solutions	0.00	1.00
4	2.77	0.30	Avg User Sep Time	587.27	135.85
5	3.39	0.00	Min User Sep Time	583.06	115.58
6	2.22	0.41	Max User Sep Time	592.16	161.93
7	2.70	0.12	Avg # of Nodes Explored	51.50	23729.80
8	2.95	0.31	Avg Payment Value	31802.90	31279.40
9	2.89	0.29			
10	3.44	0.34			

Table 4.3: Comparison of Brand-and-Cut Settings for the LCIP on 10000-Node Instances.

the x variables, they are kept as binary in all settings to get a better dual bound at the root node because we allow CPLEX to add its own cuts. In the left part of this table, we give detailed information regarding the optimality gap for these ten instances. In the right part of this table, summary measures (averaged over the ten instances) can be found. Although none of the instances can be solved in ten minutes for the BIP2 setting, the average gap is reduced from 2.88% to 0.26% and the maximum gap is reduced from 3.44% to 0.41%. More importantly, two instances are solved to optimality after applying all tricks. Next, we show average, minimum and maximum user separation time (in seconds) which is the time used for separating k -dicyle inequalities. Incorporating Inactive Induced Graph and Conservative Separation significantly reduces the separation time from about 587 seconds to 136 seconds on average. The same story can be found for minimum and maximum user separation time. "Avg # of Nodes Explored" is the average number of nodes in the search tree. "Avg Payment Value" shows the average payment value (calculated as $\sum_{i \in V} b_i - \sum_{i \in V} \sum_{j \in a(i)} \sum_{k \in t(i)} c_i^k x_{ji}^k$) of those best feasible solutions found in each setting. Based on the above experiment, we conclude that the setting "All" has improved

the performance significantly.

	Nodes	Avg Degree	1	2	3	4	5	6	7	8	9	10	Avg	Max
Gap (%)	10000	4	0.33	0.17	0.32	0.30	0.00	0.41	0.12	0.31	0.29	0.34	0.26	0.41
	5000	8	9.04	9.95	11.00	9.01	10.94	10.38	10.42	10.32	10.41	8.71	10.02	11.00
	2500	16	14.57	15.96	15.68	15.17	16.10	15.90	16.33	16.11	18.96	18.96	16.37	18.96

Table 4.4: Analyzing the effect of graph density on the branch-and-cut procedure for the LCIP.

		Random Adoption Percentage		Cost Greedy Adoption Percentage		Tree Greedy Percentage	
Edges	# of Instances	Avg (%)	Max (%)	Avg (%)	Max (%)	Avg (%)	Max (%)
400	10	48.45	57.50	97.35	99.00	98.75	99.00
600	10	35.55	48.00	95.70	97.00	97.60	99.00
800	10	36.30	49.50	95.60	97.50	97.25	99.00
1000	10	41.70	63.00	92.00	98.00	96.60	99.00
1200	10	34.10	39.50	89.45	94.00	95.65	99.00

Table 4.5: Heuristic for the LCIP Fixed Budget on 200-Node Instances.

Next, we investigate the role graph density plays in the difficulty solving a problem. For this, in addition to the 10,000 node instances we generated for the previous experiment; we generate ten instances with 5,000 nodes and average degree number 8 and ten instances with 2,500 nodes and average degree number 16. They all have 20,000 edges while different levels of graph density. We give 10-minutes time limits to all instances. We use the "All" setting. The results are shown in Table 4.4. "Gap" is the optimality gap (at termination). We also show their average and maximum values in the last two columns. The denser the graph, the bigger the gap. The average gap increases from 0.26% to 10.02% when the number of nodes decreases from 10,000 to 5,000. Then, it increases to 16.37% when the number of nodes becomes 2,500. Given that the optimality gap is increased considerably, the LCIP problem becomes much harder when the graph density increases.

Now, we focus on a set of experiments on 200 nodes where we evaluate the cost of the optimal solution against differing measures. We varied graph density and created instances where the number of edges takes values {400, 600, 800, 1000, 1200}. For each setting, 10 instances are generated and there are 50 instances in total. These instances are solved to optimality. Table 4.5 displays the results. In this experiment, we used two different measures to assess the quality of the solutions obtained by the branch-and-cut approach. The first measure tries to evaluate the benefit of optimal targeting by using the optimal payment value as a budget (upper bound) for a heuristic payment vector and evaluate the fraction of the graph that is influenced by this heuristic payment vector. We consider three budget constrained heuristic payment vectors. The first one evaluates the benefit of optimal payment vector against a random solution, the last two evaluate the

greedy heuristics. In all three cases, the budget is exceeded but the set of nodes must satisfy the following property: if any node is removed from this set, then the budget is respected. For the budget constrained payment vector that is generated randomly, we find 100 random solutions and select the one with the highest adoption (fraction of graph influenced). In the greedy heuristics, we follow the tree greedy and cost greedy respectively until the total payment violates the bound. From the column "Random" in Table 4.5, we can see that the random selection strategy does not perform well. The average adoption rate increases from 34% to 48% when the number of edges increases from 400 to 1200. But even the best one among the 5,000 samples only has 63% adoption. On the other hand the greedy heuristics have much better performance. Column "Cost Greedy" and "Tree Greedy" show the results for cost greedy and tree greedy respectively. For cost greedy, on average the adoption rate is around 94% with the best at 99%. The tree greedy has the best results. On average the adoption rate is around 97% with the best at 99%

Edges	# of Instances	TPI to Opt Gap		TPI-G to Opt Gap		Cost Greedy to Opt Gap		Tree Greedy to Opt Gap	
		Avg (%)	Max (%)	Avg (%)	Max (%)	Avg (%)	Max (%)	Avg (%)	Max (%)
400	10	73.68	97.43	94.67	181.67	7.32	11.51	2.84	6.43
600	10	123.41	169.26	117.90	140.04	14.77	20.47	7.22	12.95
800	10	184.58	260.14	148.07	217.29	13.69	20.55	7.64	14.95
1000	10	212.30	328.63	141.77	191.47	18.83	35.29	7.38	14.99
1200	10	278.72	450.51	199.69	231.60	27.03	35.97	5.99	9.61

Table 4.6: Optimality Gaps for Heuristics for 100% Adoption for the LCIP on 200-Node Instances.

The second measure evaluates the cost of optimal payment vector by comparing the optimal solution against heuristic solutions that have 100% adoption. We consider four heuristics. The first two are based on the TPI algorithm in Cordasco et al. [2015]. The TPI algorithm is designed for a special case of the LCIP where for a node i , b_i takes a value between 1 and its degree number and d_i is 1. They remove nodes based on a measure and make payment when they find a node's total possible incoming influence is not enough for its threshold in the remaining graph. In the "TPI" version, the measure is $\frac{b_i(b_i+1)}{deg(i)(deg(i)+1)}$. In the "TPI-G" version, the measure is changed to $\frac{g_i(g_i+1)}{deg(i)(deg(i)+1)}$. The last two heuristics are the Tree Greedy and Cost Greedy algorithm. Table 4.6 summarizes the results providing the gap between the heuristic solution and the optimal solution value (the gap is calculated as the difference between the heuristic solution and the optimal solution, then divid it by the optimal solution). The column "Tree Greedy to Opt Gap" shows that the solutions obtained by the Tree Greedy algorithm have the smallest gap to the optimal ones. Furthermore, this gap increases with graph density generally. Both active TPI algorithms perform badly in this more general LCIP setting.

Edges	# of Instances	WTSS % of Total Cost		LCIP % of Total Cost		% of Saving	
		Avg (%)	Max (%)	Avg (%)	Max (%)	Avg (%)	Max (%)
400	10	10.71	11.67	6.70	7.31	37.38	41.78
600	10	7.83	9.95	4.00	5.79	49.44	57.25
800	10	5.23	7.07	2.19	2.89	57.90	68.15
1000	10	5.35	6.85	1.98	3.05	63.63	71.22
1200	10	4.99	7.50	1.79	2.63	64.43	77.34

Table 4.7: Advantage of Partial Incentives of the LCIP with 200-Node Instances

In the next experiment, we show that partial incentives are able to provide significant saving in cost. In Raghavan and Zhang [2015], a set of WTSS instances is presented and solved to optimality. Each instance has 200 nodes. They varied graph density and created instances where the number of edges takes values $\{400, 600, 800, 1000, 1200\}$. For each setting, 10 instances are generated and there are 50 instances in total. We take those instances and solve their equivalent form of the LCIP. To obtain the equivalent LCIP, for each node i , its cost is divided by its threshold value for the influence factor, $d_i = b_i/g_i$. We want to point out that the influence factor can take bigger value between b_i/g_i and $b_i/(g_i - 1)$ (i.e. $b_i/g_i < d_i < b_i/(g_i - 1)$) and maintain the g_i value for node i . But we use the smallest value of influence factor for the following reason: The selected value for the influence factor is the most conservative one for evaluating the effects of partial incentives. Choosing a bigger influence factor setting will benefit the LCIP and result in smaller payment values because given a feasible solution under the current setting, the underlying influence propagation is still feasible and the payment to each node could be reduced due to the fact that it receives more influence. Therefore, we chose to set d_i as b_i/g_i . These instances are given ten minutes to run and those best feasible solutions found by the time limit are used for comparison here. Note that although the size of instances is not huge, they are not trivial instances because the WTSS problem is APX-hard. Furthermore, we need the optimal solution of the WTSS problem to show that the LCIP setting is able to provide savings.

Table 4.7 displays the results. The first column shows the number of edges and the number of instances for each setting. Then, in the second column "WTSS % of Total Cost", it contains the average and the maximum percentage of the total cost results in

100% adoption in the WTSS problem. The optimal objective value of the WTSS instances is compared to the sum of all node costs, $\sum_{i \in V} b_i$. Similarly, we compare the optimal payment value of the LCIP instances to the sum of all node costs. The results are shown in the "LCIP % of Total Cost" column of Table 4.7. We can see that on average, for graphs with 400 edges about 7% of the total cost is needed for the LCIP and 11% for the WTSS problem, and it decreases to less than 2% for the LCIP and 5% for the WTSS problem when the number of edges increases to 1200 edges. In the last column "% of Saving", we show the average and the maximum percentage of saving by considering partial incentives in the LCIP. The saving is calculated as $(Z_{wtss} - Z_{lcip})/Z_{wtss}$ where Z_{wtss} and Z_{LCIP} are the objective value for the WTSS problem and the payment value of the LCIP respectively. The saving increases as the density of the graph increases. On average, the saving is about 38% for graphs with 400 edges and it increases to about 65% for graphs with 1200 edges. The same story can be found for the maximum saving. Thus, it is safely to conclude that partial incentives are worthing considering in the influence maximization context and are able to reduce the cost dramatically.

4.6 Conclusions

We defined and studied the LCIP of finding the least expensive way of maximizing influence over an entire social network. We identified a polynomially solvable case of this NP-hard problem which is the LCIP on trees with equal influences from one's neighbors when the spread reaches the whole network. We proposed a greedy algorithm to solves the problem to optimality. We also showed a dynamic programming algorithm which has

a better time complexity and is able to handle the situation that the assumption that equal influences come from one's neighbors is relaxed.

The cascading propagation behavior of influence in the LCIP can be modeled as a directed network. Using this directed network of influence propagation, we provide a stronger formulation for the LCIP. When the underlying graphs are trees, the constraint matrix of this influence formulation is a TU matrix and an optimal integral solution can be obtained by solving its linear relaxation.

The TUM formulation can be used as a building block for developing efficient approaches for a general network. However, one important observation is that the directed network of influence propagation must be a directed acyclic graph. This requires additional constraints to be involved in a general network. Based on this, we design and implement a branch-and-cut approach for the LCIP on general networks. Our computational results show that we are able to find near optimal solutions for large randomly generated networks in a reasonable amount of time.

Chapter 5: The One Time Period Least Cost Influence Problem

5.1 Introduction

In this Chapter, we study an influence maximization problem which is similar to the PIDS problem but with partial incentives. The One Time Period Least Cost Influence Problem (1TPLCIP) considers partial incentives and an unit time restriction at the same time. In other words, the 1TPLCIP is identical to the LCIP except that we restrict the number of time periods that the diffusion takes place over to be one.

5.1.1 Problem Definition

We define the One Time Period Least Cost Influence Problem (1TPLCIP) as follows: Consider a social network represented as an undirected graph $G = (V, E)$, where node set $V = \{1, 2, \dots, n\}$ denotes the set of people in the network and edge set E shows the connections between people on the social network. In the threshold model, each inactive node $i \in V$ is influenced by an amount d_i (referred to as the *influence factor*) by it's neighbor node j (i.e., there is an edge in the graph between nodes i and j) if node j is active (i.e., has already adopted the product). For each node in the network, $i \in V$, there is a threshold, denoted by b_i . Further, a payment p_i represents the tailored incentives for

a node $i \in V$. All nodes are inactive initially. Then, we decide the tailored incentives p_i for each node $i \in V$. Now, a node i becomes active immediately if $p_i \geq b_i$ (i.e., if the payment is greater than the threshold). After that, the influence diffusion process is allowed for **one** time period. Then, we update the states of nodes by the following rule: an inactive node i becomes active if the sum of the tailored incentive p_i and the total influence coming from its active neighbors is at least b_i .

The goal is to find the minimum total payment (e.g., $\sum_{i \in V} p_i$) while ensuring that all nodes are active by the end of this one time period activation process.

A simple integer programming model for the 1TPLCIP is introduced here. Binary variable x_i denotes whether node i is activated by receiving full payment (b_i). Non-negative variable p_i represents the amount of incentives node i receives if it is not activated by full payment. Let $n(i)$ denote the set of node i 's neighbors. The formulation is as follows:

$$(MIP5.1) \quad \text{Min} \quad \sum_{i \in V} p_i + \sum_{i \in V} b_i x_i \quad (5.1)$$

$$\text{S.T.} \quad p_i + b_i x_i + d_i \sum_{j \in n(i)} x_j \geq b_i \quad \forall i \in V \quad (5.2)$$

$$x_i \in \{0, 1\}, p_i \geq 0 \quad \forall i \in V \quad (5.3)$$

Here, the objective (5.1) is to minimize the sum of the incentives given over the network. Constraint set (5.2) models the diffusion process---an inactive node i becomes active if it is selected for initial activation (i.e., paid b_i) or the sum of the tailored incentive (p_i) and the total influence coming from its active neighbors is at least b_i . Constraint set (5.3) ensures binary and non-negative requirements for the x and p variables, respectively. Note that

we do not enforce an upper bound on p variables although for a node i in V , p_i is at most $b_i - d_i$. The reason is that in this minimization problem, when $b_i - d_i < p_i < b_i$, it can be reduced to $b_i - d_i$ resulting in a better solution. If $p_i = b_i$, we can construct a solution that has $x_i = 1$ and $p_i = 0$. The new solution is no worse than the current one because the neighbors of node i can also receive influence from node i given $x_i = 1$. Hence, we do not need an upper bound on p variables and can focus on solutions which have at most one of x_i and p_i take a positive value for any node i in V .

5.1.2 Our Contributions

We first study the 1TPLCIP on trees. We propose a polynomial algorithm for the 1TPLCIP on trees. The algorithm uses a dynamic programming approach and decomposes a tree into several "star" subproblems. For each star subproblem, it finds at most two solution candidates. After all subproblems are examined, a backtracking procedure is used to determine the final solution. More importantly, we present a tight and compact extended formulation for the 1TPLCIP on trees. The key idea in this formulation is the addition of a dummy node on each edge in the graph that cannot be selected for receiving payment. Also, a dummy node's influence factor and threshold are equal to each other. Thus, if any of its neighbors are active, this dummy node would become active. Although we allow two time periods for the diffusion process after adding dummy nodes, the solution can be converted back to a solution with one time period for the diffusion process on the original graph.

5.2 The 1TPLCIP on Trees

5.2.1 Dynamic Programming Algorithm.

Our dynamic programming algorithm fragments the tree into star subnetworks iteratively, and solves the 1TPLCIP over each star. For each star, the 1TPLCIP is solved with and without external influence (whether or not the parent node of the star is paid in full) to determine the solution for that star. Next, the star is compressed into a single node and becomes a leaf node for the next star. This process is repeated until we are left with a single (last) star with its central node as the root of the original tree. After we exhaust all subproblems, a backtracking method is used to combine the solutions from stars and identify a final solution (set of nodes with incentives, and the incentive amounts paid to them) for the tree. The pseudocode of the proposed algorithm is shown in Algorithm 17.

For ease of exposition, we use the same notation and do not use indices to differentiate each star. The central node is represented by c and we refer to this star as star c . Also, for each star c , it has many children nodes connected to the central node. These children are categorized into two kinds. One is called normal children, denoted by $N(c)$, and it contains those children that are not leaf nodes in the original tree. So, these children nodes are central nodes of stars before the current star c and are obtained by compression.

Algorithm 17 Algorithm for the 1TPLCIP problem on trees

- 1: Arbitrarily pick a node as the root node of the tree
 - 2: Define the order of star problems based on the bottom-up traversal of the tree
 - 3: **for** each star subproblem **do**
 - 4: StarHandling
 - 5: **end for**
 - 6: SolutionBacktrack
-

The other kind of children are called core children, denoted by $M(c)$, and it has those children which are leaf nodes in the original tree. The key distinction between normal children and core children is that if the node c is not fully paid, those core children must be paid full incentives. This is not true for normal children because they could be covered by the influence from their children. Furthermore, let $L(c) = N(c) \cup M(c)$ denote the set of all children of star c . In a star, if the central node is fully paid, all nodes become active (since a leaf node only requires one active neighbor to adopt the product). When the central node does not receive full payment, we can make the central node active by giving it partial incentives or no incentives when some leaf nodes are paid in full payment first and influence the central node. The presence of the external influence from the rest of the tree affects which solution is cheaper.

In the StarHandling part of the dynamic programming (if the current star is not the last star) we consider these two cases (no-influence, with-influence). Denote their cost as C_{NI}^c and C_I^c , respectively. Although the solution is cheaper for the current star when external influence is present, it might be better for the final solution to have the star send influence to the rest of the tree (i.e. fully pay the node c and have it send influence to its parent). The purpose of compressing the current star into a single leaf node for the next star is to keep that possibility. The threshold for the compressed star is the difference between the cost of the central node c and that of the solution with external influence, $b_c - C_I^c$, because if the current star sends influence to the rest of the tree, then, we have to node c its full threshold value. From the parent node's point of view, there are only two possibilities. It receives full payment or not. If the parent node does, it sends influence to node c . However, in the other case, we must pay the difference between b_c and C_I^c .

The influence factor of this new node is set as infinity to reflect that it is a normal child and does not need be fully paid when the central node c is not be fully paid. For the last star, there is only one solution because there is no possibility of an external influence. Therefore, for each star, we find solution candidates to activate the central node c and its core children at the minimum cost.

SolveStar: This function returns X (set of leaf nodes that are given incentives), P (a payment vector, i.e., $\{p_i | i \in V\}$), C (total cost of the star) and B (set of nodes paid full incentive). In Algorithm 18 the subscripts NI and I and superscript c (for X , P , C and B) represent the outputs in the cases of no-influence, with-influence and star c , respectively. Recall that $N(c)$ is the set of normal children and $M(c)$ is the set of core children. Let $g_c = \lceil \frac{b_c}{d_c} \rceil$ be the required number of adopted neighbors for node c to became active, p_i be the partial incentives given to node i (initially set to $0 \forall i \in V$), and $l_c = b_c - (g_c - 1)d_c$.

First, consider the situation with no influence from the parent of node c , we compare the central node with all its children. If the cost of the central node is not bigger than the total cost of all its children, we pay the central node its full payment.

Next, we try to find the best solution without paying node c in full payment. First, we have to pay all core children. Then, to select which normal children to give incentives to, we focus on the central node c 's influence factor. Any leaf node i with $b_i \geq d_c$ can be neglected. When $b_i \geq d_c$, giving b_i units of incentives to a leaf node i sends d_c influence to node c , thus the decrease in threshold is less than what we spend. We could be better off by giving the incentive directly to the central node, and never use such normal children. Therefore, we collect the nodes with thresholds less than d_c in set S , thus providing incentives to nodes in S is advantageous in terms of decreasing the threshold of the central

Algorithm 18 StarHandling

Require: star c

1: $(X_{NI}^c, P_{NI}^c, C_{NI}^c, B_{NI}^c) \leftarrow \text{SolveStar}(\text{star } c, \text{no-influence}).$

2: **if** star c is not the last star **then**

3: $(X_I^c, P_I^c, C_I^c, B_I^c) \leftarrow \text{SolveStar}(\text{star } c, \text{with-influence}).$

4: The compressed node's threshold is $b'_c = b_c - C_I^c.$

5: **end if**

6: **function** SolveStar(a star c , flag)

7: Let $g_c = \lceil \frac{b_c}{d_c} \rceil$. Recall that $N(c)$ is the set of normal children, $M(c)$ is the set of core children and $L(c) = N(c) \cup M(c).$

8: **if** $b_c \leq \sum_{i \in L(c)} b_i$ **then**

9: Let $p_c = b_c$ and $X \leftarrow c, B \leftarrow c$ and $C = b_c.$

10: **else**

11: **if** flag is with-influence **then**

12: $g_c = g_c - 1.$

13: **end if**

14: $X \leftarrow M(c), B \leftarrow M(c), p_i = b_i$ for $i \in M(c).$

15: **if** $|M(c)| < g_c$ **then**

16: Let $S = \{i \mid b'_i < d_c, i \in N(c)\}, l_c = b_c - (g_c - 1)d_c$ and $k = g_c - |M(c)|.$

17: **if** $|S| \geq k$ **then**

18: Let S_k and S_{k-1} be the sets of the cheapest k and $(k - 1)$ nodes in $S,$
respectively.

19: **if** $\sum_{i \in S_k} b'_i \leq \sum_{i \in S_{k-1}} b'_i + l_c$ **then**

20: $X \leftarrow X \cup S_k, B \leftarrow B \cup S_k, p_i = b_i$ for $i \in X.$

21: **else**

22: $X \leftarrow X \cup S_{k-1}, B \leftarrow B \cup S_{k-1}, p_i = b_i$ for $i \in X,$ and let $p_c \leftarrow l_c,$

$X \leftarrow X \cup c.$

23: **end if**

24: **else**

25: $X \leftarrow X \cup S, B \leftarrow B \cup S, p_i = b_i$ for $i \in S,$ and let $p_c = b_c - |S|d_c,$

$X \leftarrow X \cup c.$

26: **end if**

27: **end if**

28: $C = \sum_{i \in X} p_i.$

29: **end if**

30: **return** $X, P, C, B.$

31: **end function**

Algorithm 19 SolutionBacktrack

Require: the last star and its solution X and P

```
1:  $X^* \leftarrow X, p_i^* \leftarrow p_i$  for all  $i \in X$ .
2: if  $X$  is  $r$  then
3:    $\forall i \in N(r)$  call Piecing( $i, X^*$ , with-influence).
4: else
5:    $\forall i \in N(r) \setminus B_{NI}^r$  call Piecing( $i, X^*$ , no-influence).
6:   for  $i \in B_{NI}^r$  do
7:      $\forall j \in N(i)$  call Piecing( $j, X^*$ , with-influence).
8:   end for
9: end if
10:  $C^* = \sum_{i \in X^*} p_i$ 
11: return  $C^*, X^*, P^*$ .
```

node. The cost of the solution depends on the size of the set S . Let $k = g_c - |M(c)|$. When $|S| \geq k$ (i.e., there are more than enough children nodes), the solution is to pay the first k nodes in S if the threshold for the k -th node ($b_{(k)}$) is less than or equal to l_c . Otherwise, it is less costly to pay the first $(g_c - 1)$ nodes and the remaining threshold of the central node is covered by direct incentives to c . If $|S| < k$, then all nodes in the set S are given incentives and the remaining amount of the threshold of the central node is covered by incentives given directly to central node.

Thus, we obtain the no-influence solution. For the situation with external influence, we reduce g_c to $g_c - 1$ accordingly. Then, we follow the same procedure as above for the with-influence solution.

In Algorithm 18, we described how to solve 1TPLCIP on a single star. In Algorithm 19, we describe how to combine these solutions with a backtracking procedure to obtain a final solution for the tree.

SolutionBacktrack: After we obtain the solution of the last star which has the root node as its central node, we invoke a backtracking procedure to choose the solution from the

Algorithm 20 Recursive Functions

```
1: function Piecing( $c, X, Flag$ )
2:   if  $Flag = \text{with-influence}$  then
3:      $X' \leftarrow X_I^c, P' = P_I^c.$ 
4:   else
5:      $X' \leftarrow X_{NI}^c, P' = P_{NI}^c.$ 
6:   end if
7:    $X \leftarrow X \cup X'.$ 
8:   if  $X' = c$  then
9:      $\forall i \in N(c)$  call Piecing( $i, X, \text{with-influence}$ ).
10:  else
11:     $\forall i \in N(c) \setminus B_{NI}^c$  call Piecing( $i, X, \text{no-influence}$ ).
12:    for  $i \in B_{NI}^c$  do
13:       $\forall j \in N(i)$  call Piecing( $j, X, \text{with-influence}$ ).
14:    end for
15:  end if
16:  return  $X, P.$ 
17: end function
```

candidates for each star subproblem and piece them together to obtain the final solution for this tree. After the last star subproblem is solved, for each child node in this star, we know if it is fully paid or not and if its parent node is fully paid or not. For instance, if the central node is fully paid, then, the central node sends influence to all its children. Then, all stars with central node in $N(c)$ will pick the candidate with influence. Otherwise, first, if a node i in $N(c)$ is fully paid, we can proceed to nodes in $L(i)$ and pick the candidate where the parent node is selected. Second, if a node i in $N(c)$ is not fully paid, they will pick the candidate where the parent node is not selected. With this information we can now proceed down the tree, incorporating the partial solution at a node based on the solution of its parent star. This backtracking procedure is described in Algorithm 19 SolutionBacktrack. Let r denote the root of the tree (as determined by Algorithm 17), X^* denote the final solution of the tree, C^* its cost, and P^* its payment vector.

In this algorithm, we have the recursive functions: Piecing. It chooses the solution

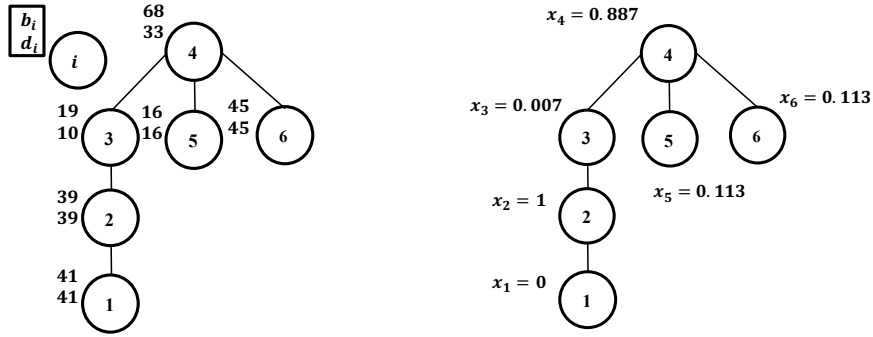


Figure 5.1: (a) A 1TPLCIP instance (b) A fractional optimal solution

for a star c and recursively choose solutions for stars whose central nodes are leaf nodes of the star c . Algorithm 20 provides a detailed description about them. Although it is possible to prove the correctness of this algorithm directly, we defer the proof until the next section. There we will provide a tight and compact extended formulation for the 1TPLCIP problem, and use linear programming duality to prove its correctness.

Proposition 5.1. *The 1TPLCIP problem on trees can be solved in $O(|V|)$ time.*

Proof. Proof of Proposition 5.1. There are at most $|V|$ stars. For each star, we need to find g_i cheapest children and it takes $O(deg(i))$ time. Finding the g_i th order statistics can be done in $O(deg(i))$ time by the Quickselect method in Chapter 9 of Stein et al. [2009] thus it takes $O(deg(i))$ time to go through the list to collect the g_i cheapest children. For the whole tree, this is bounded by $O(|V|)$ time. In the backtracking procedure, we pick the final solution for each node which takes $O(|V|)$ time over the tree. Therefore, the running time for the dynamic algorithm is linear with respect to the number of nodes. \square

5.2.2 A Tight and Extended Formulation

Although MIP5.1 is a valid formulation for the 1TPLCIP, we note that this formulation is weak and Figure 5.1 shows its linear programming (LP) relaxation provides fractional solutions for instances on trees. Figure 5.1(a) provides the 1TPLCIP instance, and Figure 5.1(b) describes a fractional optimal solution to the LP relaxation of MIP5.1. It has $x_1 = 0$, $x_2 = 1$, $x_3 = 0.006849$, $x_4 = 0.886986$, $x_5 = 0.113014$, $x_6 = 0.113014$. All other decision variables are zero. The objective value is 106.339041.

We now present a tight and compact extended formulation on trees. From the input graph G , we create a new graph G_t by adding one dummy node to each edge in G . For each edge $\{i, j\} \in E$, insert a dummy node d . Let D denotes the set of dummy nodes. Since the dummy nodes have effectively split each edge into two in the original graph, we replace each of the original edges $\{i, j\} \in E$ by two edges $\{i, d\}$ and $\{d, j\}$ in the new graph G_t . Let E_t denote the set of edges in G_t ($G_t = (V \cup D, E_t)$). We define an extended edge as $\{i, d, j\}$ in G_t for an edge $\{i, j\}$ in G . The dummy nodes cannot be selected for receiving payment (they can be viewed as having large costs), and all have their influence factor equal to their threshold (thus if one of it's neighbors is activated the dummy node will become active and propagate the influence to the other neighbor). Note that now the propagation is allowed to have two time periods in the transformed graph G_t , but as will become evident later, it is still a one time period propagation in the original graph G .

For each node $i \in V$, binary decision variable x_i denotes whether node i receives full payment. Also, a continuous decision variable p_i denotes the amount of payment node i receives if it is not paid in full. For each edge $\{i, d\} \in E_t$, where $i \in V$ and $d \in D$

(notice G_t is bipartite and E_t only contains edges between the nodes in V and D), create two binary arc variables y_{id} and y_{di} to represent the direction of influence propagation. If node i sends influence to node d , y_{id} is 1 and 0 otherwise. For any node $i \in V \cup D$, $a(i)$ denotes the set of node i 's neighbors in the transformed G_t . Recall that for any node $i \in V$, we use $n(i)$ denoting the set of node i 's neighbors in the original graph G . Furthermore, $g_i = \lceil \frac{b_i}{d_i} \rceil$ and $l_i = b_i - d_i(g_i - 1)$. We can now write the following compact extended formulation on trees:

$$\text{(MIP5.2) Min} \quad \sum_{i \in V} p_i + \sum_{i \in V} b_i x_i \quad (5.4)$$

$$\text{Subject to} \quad y_{id} + y_{di} = 1 \quad \forall \{i, d\} \in E_t \quad (5.5)$$

$$x_i \geq y_{dj} \quad \forall i \in V, j \in n(i) \quad (5.6)$$

$$x_i \leq y_{id} \quad \forall i \in V, d \in a(i) \quad (5.7)$$

$$p_i + b_i x_i + d_i \sum_{d \in a(i)} y_{di} \geq b_i \quad \forall i \in V \quad (5.8)$$

$$p_i + l_i g_i x_i + l_i \sum_{d \in a(i)} y_{di} \geq l_i g_i \quad \forall i \in V \quad (5.9)$$

$$p_i \geq 0, x_i \in \{0, 1\} \quad \forall i \in V \quad (5.10)$$

$$y_{id}, y_{di} \in \{0, 1\} \quad \forall \{i, d\} \in E_t \quad (5.11)$$

We refer to the above formulation as MIP5.2. The objective function (5.4) minimizes the total cost. Constraint (5.5) makes sure on each edge influence is only propagated in one direction. Constraint (5.6) says that if node i is selected, then node d can send influence to node j for an extended edge $\{i, d, j\}$ in G_t . Constraint (5.7) means that if node i is selected, it sends influence to all its neighbors. Constraint (5.8) says for a node i in V , the sum of the payment it receives and the influence from its neighbors is at least

b_i . Constraint (5.9) says for a node i in V , the sum of the payment it receives and product of the number of incoming arcs and l_i is at least $l_i g_i$. The rest are non-negative constraint and binary constraints. We do not enforce an upper bound on p variables for the same reason stated earlier for MIP5.1.

The linear relaxation of MIP5.2 is the following linear programming problem:

$$(LP5.2) \text{ Min} \quad \sum_{i \in V} p_i + \sum_{i \in V} b_i x_i \quad (5.12)$$

$$\text{Subject to } (s_{id}) \quad -y_{id} - y_{di} = -1 \quad \forall \{i, d\} \in E_t \quad (5.13)$$

$$(t_{ij}) \quad x_i - y_{dj} \geq 0 \quad \forall i \in V, j \in n(i) \quad (5.14)$$

$$(u_{id}) \quad y_{id} - x_i \geq 0 \quad \forall i \in V, d \in a(i) \quad (5.15)$$

$$(v_i) \quad p_i + b_i x_i + \sum_{d \in a(i)} d_i y_{di} \geq b_i \quad \forall i \in V \quad (5.16)$$

$$(w_i) \quad p_i + l_i g_i x_i + \sum_{d \in a(i)} l_i y_{di} \geq l_i g_i \quad \forall i \in V \quad (5.17)$$

$$p_i, x_i \geq 0 \quad \forall i \in V \quad (5.18)$$

$$y_{id}, y_{di} \geq 0 \quad \forall \{i, d\} \in E_t \quad (5.19)$$

We refer to this linear programming problem as LP5.2. The dual to LP5.2 is as follows:

$$(DLP5.2) \text{ Max} \quad \sum_{i \in V} b_i v_i + \sum_{i \in V} l_i g_i w_i - \sum_{i \in V} \sum_{d \in a(i)} s_{id} \quad (5.20)$$

$$\text{S.T. } (y_{id}) \quad -s_{id} + u_{id} \leq 0 \quad \forall i \in V, d \in a(i) \quad (5.21)$$

$$(y_{di}) \quad -s_{id} - t_{ji} + d_i v_i + l_i w_i \leq 0 \quad \forall d \in D, i \in a(d) \quad (5.22)$$

$$(x_i) \quad \sum_{j \in n(i)} t_{ij} - \sum_{d \in a(i)} u_{id} + b_i v_i + l_i g_i w_i \leq b_i \quad \forall i \in V \quad (5.23)$$

$$(p_i) \quad v_i + w_i \leq 1 \quad \forall i \in V \quad (5.24)$$

$$s_{id} \text{ free} \quad \forall \{i, d\} \in E_t \quad (5.25)$$

$$t_{ij} \geq 0 \quad \forall i \in V, j \in n(i) \quad (5.26)$$

$$u_{id} \geq 0 \quad \forall i \in V, d \in a(i) \quad (5.27)$$

$$v_i, w_i \geq 0 \quad \forall i \in V \quad (5.28)$$

We have s_{id} , t_{ij} , u_{id} , v_i , and w_i as dual variables for constraint sets (5.13), (5.14), (5.15), (5.16), and (5.17) respectively. We refer to the dual linear problem as DLP5.2. Let $\text{conv}(P)$ denote the convex hull of payment vectors (\mathbf{p}, \mathbf{x}) , and let E1TPLCIP denote the feasible region of LP5.2.

Theorem 5.1. *Given a tree, LP5.2 has optimal solutions with x binary and $\text{Proj}_{(\mathbf{p}, \mathbf{x})}(\text{E1TPLCIP}) = \text{conv}(P)$.*

Proof. Proof of Theorem 5.1. First, based on this pair of primal and dual problems, we have the complementary slackness conditions:

$$(y_{id} + y_{di} - 1)s_{id} = 0 \quad \forall \{i, d\} \in E_t \quad (5.29)$$

$$(x_i - y_{dj})t_{ij} = 0 \quad \forall i \in V, j \in n(i) \quad (5.30)$$

$$(y_{id} - x_i)u_{id} = 0 \quad \forall i \in V, d \in a(i) \quad (5.31)$$

$$(p_i + b_i x_i + \sum_{d \in a(i)} d_i y_{di} - b_i)v_i = 0 \quad \forall i \in V \quad (5.32)$$

$$(p_i + l_i g_i x_i + \sum_{d \in a(i)} l_i y_{di} - l_i g_i)w_i = 0 \quad \forall i \in V \quad (5.33)$$

$$(-s_{id} + u_{id})y_{id} = 0 \quad \forall i \in V, d \in a(i) \quad (5.34)$$

$$(-s_{id} - t_{ji} + d_i v_i + l_i w_i)y_{di} = 0 \quad \forall d \in D, i \in a(d) \quad (5.35)$$

$$\left(\sum_{j \in n(i)} t_{ij} - \sum_{d \in a(i)} u_{id} + b_i v_i + l_i g_i w_i - b_i\right) x_i = 0 \quad \forall i \in V \quad (5.36)$$

$$(v_i + w_i - 1) p_i = 0 \quad \forall i \in V \quad (5.37)$$

The payment vector p^* obtained in the DP algorithm can be transfer into a feasible solution for LP5.2. Given the payment vector, if a node i receives full payment, set $x_i = 1$ and $p_i = 0$. Then, set $y_{di} = 0$ and $y_{id} = 1$ for all d in $a(i)$. If not, set $x_i = 0$ and $p_i = p_i^*$. Next, based on the solution candidate selected in the backtracking procedure, we assign values for the remaining y variables. Let S denote the set of children are selected in star i 's final solution (With or Without external influence). Then, if the ``with influence'' solution is picked, then, let $S = S \cup h$ where h is node i 's parent node in the original graph. Let S_d denote the set of dummy nodes adjacent to node i and nodes in S , thus, $S_d = \{d \in a(i) : d \in a(j) \quad \forall j \in S\}$. Finally, for all $d \in S_d$, set $y_{di} = 1$ and $y_{id} = 0$. For all $d \in a(i) \setminus S_d$, set $y_{di} = 0$ and $y_{id} = 1$. Note that, each regular node has at most g_i incoming arcs unless the size of its core children set is bigger than g_i .

With this primal feasible solution in hand, a dual feasible solution will be constructed in the following proof and show that CS conditions are satisfied by this pair of primal and dual solutions. Throughout the proof, we always have $u_{id} = s_{id}$. So, CS condition (5.34) is satisfied. Then, we can focus on other CS conditions.

First, we assign values to all dual variables associated with leaf nodes. For a leaf node l , set $v_l = 1$, $w_l = 0$, $t_{li} = 0$, $t_{il} = b_i$, $s_{ld} = 0$ and $u_{ld} = 0$ where i is in $n(l)$ and d is in $a(l)$. For a leaf node, $b_i = d_i$. Then, constraint (5.22), (5.23) and (5.24) are always binding. Thus, CS condition (5.35), (5.36), and (5.37) are satisfied. Furthermore, CS condition (5.29), (5.30), (5.31), (5.33) are satisfied because $s_{ld} = 0$, $t_{li} = 0$, $u_{ld} = 0$

and $w_l = 0$ respectively. Lastly, CS condition (5.32) is respected because constraint (5.16) is always binding regardless of the value of x_i .

Next, following the bottom-up order in the DP algorithm, for each non-leaf node, we have three cases depending on the amount of incentives it receives in the solution obtained by the DP algorithm. Let h denote node i 's parent node in the original tree following the DP algorithm and $deg(i)$ be node i 's degree number. Given that bottom-top order, the dual variable t associated with node i 's children are set already. So we only need to assign value to t_{ih} and t_{hi} .

Case one: For a node i , if $x_i = 1$, it means $p_i = 0$, $y_{id} = 1$ and $y_{di} = 0$ for all d in $a(i)$. Set $v_i = w_i = 0$, $t_{ih} = b'_i$, $t_{hi} = 0$, and $u_{id} = s_{id} = (\sum_{j \in n(i)} t_{ij} - b_i)/deg(i)$. Then, CS condition (5.29), (5.31), (5.32), (5.33) are satisfied because those corresponding primal constraints are binding. CS condition (5.30) is satisfied as follows: when $x_j = 0$, $x_i = y_{dj} = 1$ and when $x_j = 1$, we must have $t_{ih} = b'_i = 0$. Otherwise, x_i would be zero in the DP algorithm. Constraint (5.22) and (5.24) are satisfied as inequalities. Thus, conditions (5.35), and (5.37) are satisfied because $y_{di} = 1$ and $p_i = 0$. Lastly, conditions (5.36) is satisfied because constraint (5.23) is binding by construction.

Case two: For a node i , if $x_i = 0$ and $p_i > 0$, it means there are less than g_i incoming arcs for this node. Let I^e be the set of the dummy nodes such that $I^e = \{d \in a(i) : y_{di} = 1\}$. Also, for ease of explanation, we make $C_d(i)$ contain those dummy nodes adjacent to node i 's core children. Set $t_{ih} = 0$ and $t_{hi} = b'_i$. If $p_i = l_i$, set $v_i = 0$ and $w_i = 1$. Otherwise, set $v_i = 1$ and $w_i = 0$. Let $k_i = d_i v_i + l_i w_i$. Next, for d in $I^e \setminus C_d(i)$, set $u_{id} = s_{id} = k_i - t_{ji}$ where j is the other regular node adjacent to the dummy node d . For d in $C_d(i)$, set $u_{id} = s_{id} = k_i$. Then, for d in $a(i) \setminus I^e$, set $u_{id} = s_{id} = 0$. CS

condition (5.29) is satisfied because when d is in I^e , constraint (5.13) is binding and when d is in $a(i) \setminus I^e$, $s_{id} = 0$. CS condition (5.30),(5.31), (5.32) are satisfied because constraint (5.14) ($x_i = y_{dj} = 0$), (5.15) ($x_i = y_{id} = 0$) and (5.16) are binding respectively. CS condition (5.33) is satisfied because constraint (5.17) is binding when $p_i = l_i$ and $w_i = 0$ when $p_i \neq l_i$. CS condition (5.35) is satisfied. This can be seen as follows: For d in I^e , constraint (5.22) is binding by construction. For d in $a(i) \setminus I^e$, constraint (5.22) is satisfied ($t_{ji} = b'_j \geq k_i$, Otherwise $x_j = 1$ by the DP algorithm) and $y_{di} = 0$. CS condition (5.36) is satisfied because constraint (5.23) is satisfied and $x_i = 0$. Lastly, CS condition (5.37) is satisfied because constraint (5.24) is binding.

Case three: For a node i , if $x_i = 0$ and $p_i = 0$, it means there are at least g_i incoming arcs for this node. Let I^o be the set of the regular nodes such that $I^o = \{j \in n(i) : y_{jd} = 1, y_{di} = 1\}$. Depending on the size of $C_d(i)$, we consider two situations. First, if $|C_d(i)| \geq g_i$, set $v_i = w_i = 0$, $t_{ih} = b'_i$ and $t_{hi} = 0$. Also, $u_{id} = s_{id} = 0$ for all d in $a(i)$. CS condition (5.29), (5.31), (5.32), (5.33) are satisfied because $s_{id} = 0$, $u_{id} = 0$, $v_i = 0$ and $w_i = 0$ respectively. CS condition (5.30) is satisfied because constraint (5.14) ($x_i = y_{dj} = 0$) is binding. For CS condition (5.35), for d in $C_d(i)$, constraint (5.22) is binding by construction ($t_{ji} = 0$). For d in $a(i) \setminus C_d(i)$, constraint (5.22) is satisfied as an inequality ($t_{ji} \geq 0$) and $y_{di} = 0$. CS condition (5.36) and (5.37) are satisfied because constraint (5.22) and constraint (5.23) are satisfied and $x_i = 0$ and $p_i = 0$ respectively.

Second, if $|C_d(i)| \leq g_i - 1$, then, node i has exactly g_i incoming arcs. We set $t_{ih} = b'_i$, $t_{hi} = 0$, $v_i = 0$, $w_i = \frac{b'_m}{l_i}$ where $m = \arg \max\{b_j : j \in I^o \cup \{i\} \setminus (C(i) \cup \{h\})\}$. Let $k_i = l_i w_i$. Next, for d in $I^e \setminus C_d(i)$, set $u_{id} = s_{id} = k_i - t_{ji}$ where j is the other regular node adjacent to the dummy node d . For d in $C_d(i)$, set $u_{id} = s_{id} = k_i$. Then, for

d in $a(i) \setminus I^e$, set $u_{id} = s_{id} = 0$. CS condition (5.29) is satisfied because when d is in I^e , constraint (5.13) is binding and when d is in $a(i) \setminus I^e$, $s_{id} = 0$. CS condition (5.30) and (5.33) are satisfied because constraint (5.14) ($x_i = y_{dj} = 0$) and (5.17) are binding respectively. CS condition (5.31) is satisfied because constraint (5.15) ($x_i = y_{id} = 0$) when d is in I^e and $s_{id} = 0$ when d is in $a(i) \setminus I^e$. CS condition (5.32) is satisfied because $v_i = 0$. CS condition (5.35) is satisfied. This can be seen as follows: For d in I^e , constraint (5.22) is binding by construction. For d in $a(i) \setminus I^e$, constraint (5.22) is satisfied ($t_{ji} = b'_j \geq k_i$, Otherwise $x_j = 1$ by the DP algorithm) and $y_{di} = 0$. CS condition (5.36) is satisfied because constraint (5.23) is satisfied and $x_i = 0$. Lastly, CS condition (5.37) is satisfied because constraint (5.24) is satisfied and $p_i = 0$. \square

5.2.3 Polytope of the 1TPLCIP on Trees

Next, we derive the polytope of the 1TPLCIP on trees. The extended formulation is projected onto the space of the payment (i.e., p and x) variables by projecting out all arc (i.e., y) variables.

We first substitute out all y_{id} variables by $1 - y_{di}$ because $y_{id} + y_{di} = 1$. Then, we have the following formulation and denote its feasible region as P_d .

$$(LP3) \text{ Min} \quad \sum_{i \in V} p_i + \sum_{i \in V} b_i x_i \quad (5.38)$$

$$\text{Subject to } (t_{di}) \quad x_j - y_{di} \geq 0 \quad \forall j \in V, i \in n(j) \quad (5.39)$$

$$(u_{id}) \quad -x_i - y_{di} \geq -1 \quad \forall i \in V, d \in a(i) \quad (5.40)$$

$$(v_i) \quad p_i + b_i x_i + \sum_{d \in a(i)} d_i y_{di} \geq b_i \quad \forall i \in V \quad (5.41)$$

$$(w_i) \quad p_i + l_i g_i x_i + \sum_{d \in a(i)} l_i y_{di} \geq l_i g_i \quad \forall i \in V \quad (5.42)$$

$$p_i, x_i \geq 0 \quad \forall i \in V \quad (5.43)$$

$$y_{di} \geq 0 \quad \forall \{i, d\} \in E_t \quad (5.44)$$

Based on Theorem 2 in Balas and Pulleyblank [1983], the projection cone W is described by the following linear inequalities:

$$-t_{di} - u_{id} + d_i v_i + l_i w_i \leq 0 \quad \forall i \in V, d \in a(i) \quad (5.45)$$

$$t_{di}, u_{id}, v_i, w_i \geq 0 \quad \forall i \in V, d \in a(i) \quad (5.46)$$

where \mathbf{t} , \mathbf{u} , \mathbf{v} and \mathbf{w} are dual multipliers corresponding to constraints (5.39), (5.40), (5.41) and (5.42) respectively.

Theorem 5.2. *The vector $\mathbf{r} = (\mathbf{t}, \mathbf{w}, \mathbf{u}, \mathbf{v}) \in W$ is extreme if and only if there exists a positive α such that it is in one of these four cases:*

1. $t_{di} = \alpha$ for one $\{i, d\} \in E_t$. Other $(\mathbf{t}, \mathbf{w}, \mathbf{u}, \mathbf{v})$ are 0.
2. $u_{di} = \alpha$ for one $\{i, d\} \in E_t$. Other $(\mathbf{t}, \mathbf{w}, \mathbf{u}, \mathbf{v})$ are 0.
3. $v_i = \alpha$ for one $i \in V$. Then for $d \in a(i)$, either $t_{di} = d_i \alpha$ or $u_{id} = d_i \alpha$. Other $(\mathbf{t}, \mathbf{w}, \mathbf{u}, \mathbf{v})$ are 0.
4. $w_i = \alpha$ for one $i \in V$. Then for $d \in a(i)$, either $t_{di} = l_i \alpha$ or $u_{id} = l_i \alpha$. Other $(\mathbf{t}, \mathbf{w}, \mathbf{u}, \mathbf{v})$ are 0.

Proof. Proof of Theorem 5.2. *Sufficiency.* Let $\mathbf{r} \in W$ be of the form Case 1 and assume that $\mathbf{r} = \frac{1}{2}(\mathbf{r}^1 + \mathbf{r}^2)$ for some $\mathbf{r}^1, \mathbf{r}^2 \in W$. Then, except t_{di}^1 and t_{di}^2 , all other directions are 0. Then, $\mathbf{r}^1, \mathbf{r}^2$ are in $R(\mathbf{r})$. So, r is extreme.

For Case 2, it is similar to Case 1.

For Case 3, let $\mathbf{r} \in W$ be of the form Case 3 and assume that $\mathbf{r} = \frac{1}{2}(\mathbf{r}^1 + \mathbf{r}^2)$ for some $\mathbf{r}^1, \mathbf{r}^2 \in W$. So, for 0 components in \mathbf{r} , their corresponding components in \mathbf{r}^1 and \mathbf{r}^2 are also 0. Then, we have $v_i^1 + v_i^2 = 2\alpha$ and $q_{id}^1 + q_{id}^2 = 2d_i\alpha$ where $q_{id}^k, k = 1, 2$, represent the positive component between t_{id}^k and $u_{di}^k, k = 1, 2$, for all $d \in a(i)$. Then, if there is a pair d_1 and d_2 , we have $q_{id_1}^1 > q_{id_2}^1$ if and only if $q_{id_1}^2 < q_{id_1}^2$. But the constraint (5.45) imposes that $d_i v_i^k \leq q_{id}^k, k = 1, 2$. Hence, $d_i v_i^k$ takes value as $\min\{q_{id_1}^k, q_{id_2}^k\}$ for $k = 1, 2$. Thus, $q_{id_1}^k = q_{id_2}^k = d_i \alpha_k, k = 1, 2$, for all $d_1, d_2 \in a(i)$. Otherwise, the constraint (5.45) would be violated or we could only have $v_i^1 + v_i^2 < 2\alpha$. Either of them are undesired. Therefore, $\mathbf{r}^1, \mathbf{r}^2$ are in $R(\mathbf{r})$. So, \mathbf{r} is extreme.

For Case 4, it is similar to Case 3.

Necessity. Let \mathbf{r} be an extreme vector of W . Let $S^t = \{\{i, d\} \in E_t : t_{di} > 0\}$, $S^u = \{\{i, d\} \in E_t : v_{id} > 0\}$, $S^v = \{i \in V : v_i > 0\}$, and $S^w = \{i \in V : w_i > 0\}$ based on this \mathbf{r} . First, we consider the situation where $S^v \cup S^w = \emptyset$. If $|S^t| + |S^u| > 1$, we can have $\mathbf{r} = \frac{1}{2}(\mathbf{r}^1 + \mathbf{r}^2)$. Let \mathbf{r}^1 contains all but one positive components in \mathbf{r} and double their values. Let \mathbf{r}^2 contains the one positive component omitted by \mathbf{r}^1 and double its value. So, if $|S^t| + |S^u| > 1$, \mathbf{r} is not extreme, contrary to the assumption. We conclude that if $S^v \cup S^w = \emptyset$, then $|S^t| + |S^u| = 1$. Thus \mathbf{r} is either in Case 1 or in Case 2.

Now consider the case when $S^v \cup S^w \neq \emptyset$. When $|S^v \cup S^w| > 1$, without loss of generality, let $i \in S^v \cup S^w$. Then, \mathbf{r}^1 have value $v_i^1 = 2v_i, w_i^1 = 2w_i, t_{di}^1 = 2t_{di}, u_{id}^1 = 2u_{id}$ for all $d \in a(i)$ and 0s for other components. Then, let \mathbf{r}^2 be $2\mathbf{r} - \mathbf{r}^1$. Hence, we have $\mathbf{r} = \frac{1}{2}(\mathbf{r}^1 + \mathbf{r}^2)$ and $\mathbf{r}^1, \mathbf{r}^2$ are different in at least one direction. So, if $|S^v \cup S^w| > 1$, \mathbf{r} is not extreme, contrary to the assumption.

Next, When $|S^v \cup S^w| = 1$ and let $i \in S^v \cup S^w$, consider the case when $S^v \cap S^w \neq \emptyset$. It means $i \in S^v$ and $i \in S^w$. Then, \mathbf{r}^1 have value $v_i^1 = 2v_i$, $t_{di}^1 = 2 \min\{t_{di}, d_i v_i\}$, $u_{id}^1 = 2(d_i v_i - t_{di}^1)$ for all $d \in a(i)$ and 0s for other components. Then, let \mathbf{r}^2 be $2\mathbf{r} - \mathbf{r}^1$. Hence, we have $\mathbf{r} = \frac{1}{2}(\mathbf{r}^1 + \mathbf{r}^2)$ and $\mathbf{r}^1, \mathbf{r}^2$ are different in at least one direction. So, if $|S^v \cup S^w| = 1$ and $S^v \cap S^w \neq \emptyset$, \mathbf{r} is not extreme, contrary to the assumption.

When $|S^v \cup S^w| = 1$, $|S^v \cap S^w| = 0$, and let $i \in S^v \cup S^w$, define $S_q = \{\{j, d\} \in E_t : q_{jd} > 0 \ \& \ j \in V \setminus i\}$. If $S_q \neq \emptyset$, let \mathbf{r}^1 have $v_i^1 = 2v_i$, $w_i^1 = 2w_i$ and $t_{di}^1 = 2t_{di}$, $u_{id}^1 = 2u_{id}$ for all $d \in a(i)$ and 0s in other components. Then, let \mathbf{r}^2 be $2\mathbf{r} - \mathbf{r}^1$. Hence, we have $\mathbf{r} = \frac{1}{2}(\mathbf{r}^1 + \mathbf{r}^2)$ and $\mathbf{r}^1, \mathbf{r}^2$ are different in at least one direction. So, if $|S^v \cup S^w| = 1$, $|S^v \cap S^w| = 0$, and $S_q \neq \emptyset$, \mathbf{r} is not extreme, contrary to the assumption.

When $|S^v \cup S^w| = 1$ and $|S^v \cap S^w| = 0$, without loss of generality, let $i \in S^v$ and $S^w = \emptyset$, define $S_1 = \{\{i, d\} \in E_t : t_{di} > 0 \oplus u_{id} > 0\}$ (exactly one of t_{di} and u_{id} takes positive value) and $S_2 = \{\{i, d\} \in E_t : t_{di} > 0 \ \& \ u_{id} > 0\}$ (both t_{di} and u_{id} take positive values). If $S_2 \neq \emptyset$ and let $\{i, d\} \in S_2$, then, we define $\alpha^1 = 2 \min\{v_i, \frac{t_{di}}{d_i}, \frac{u_{id}}{d_i} : \{i, d\} \in S_2\}$ and make \mathbf{r}^1 have $v_i^1 = \alpha^1$. For $\{i, d\} \in S_2$, we have $t_{di}^1 = d_i \alpha_i^1$. Also, for $\{i, d\} \in S_1$, if $t_{di} > 0$, we have $t_{di}^1 = d_i \alpha^1$. Otherwise, we have $u_{id}^1 = d_i \alpha^1$. The rest components are 0s. Then, let \mathbf{r}^2 be $2\mathbf{r} - \mathbf{r}^1$. Hence, we have $\mathbf{r} = \frac{1}{2}(\mathbf{r}^1 + \mathbf{r}^2)$ and $\mathbf{r}^1, \mathbf{r}^2$ are different in at least one direction. So, if $|S^v \cup S^w| = 1$, $|S^v \cap S^w| = 0$ and $S_2 \neq \emptyset$, \mathbf{r} is not extreme, contrary to the assumption. Thus we must have $|S_1| = \deg(i)$ where $\deg(i)$ is node i 's degree number otherwise the constraint (5.45) would not be respected.

Next, if $|S^v \cup S^w| = 1$, $|S^v \cap S^w| = 0$ and $|S_1| = \deg(i)$, without loss of generality, let $i \in S^v$ and $S^w = \emptyset$, let $v_i = \alpha$ and define $S^+ = \{\{i, d\} : q_{id} > d_i \alpha\}$. When $S^+ \neq \emptyset$, without loss of generality, let $\{i, d\} \in S^+$ and $t_{di} > 0$, we can make \mathbf{r}^1 have

$u_{di}^1 = 2(u_{di} - d_i\alpha)$ and 0s in other components. Then, let \mathbf{r}^2 be $2\mathbf{r} - \mathbf{r}^1$. Hence, we have $\mathbf{r} = \frac{1}{2}(\mathbf{r}^1 + \mathbf{r}^2)$ and $\mathbf{r}^1, \mathbf{r}^2$ are different in at least one direction. So, if $|S^v \cup S^w| = 1$, $|S^v \cap S^w| = 0$, $|S_1| = \text{deg}(i)$ and $S^+ \neq \emptyset$, \mathbf{r} is not extreme, contrary to the assumption. Then, we must have $S^+ = \emptyset$. This proves that if $S^v \cup S^w \neq \emptyset$, we must have $|S^v \cup S^w| = 1$, $|S^v \cap S^w| = 0$, $|S_1| = \text{deg}(i)$ and $S_2 = S_q = S^+ = \emptyset$. Thus, \mathbf{r} is either in Case 3 or in Case 4. \square

Applying Theorem 2 in Balas and Pulleyblank [1983], Case 1 and Case 2 extreme directions give the trivial constraints: $0 \leq x_i \leq 1$ for all $i \in V$. For an extreme direction of the form Case 3, recall that $S^t = \{\{i, d\} \in E_t : t_{di} > 0\}$, $S^u = \{\{i, d\} \in E_t : v_{id} > 0\}$ based on this extreme direction. Let $V^t = \{j \in V : \{i, d, j\} \in E_t \ \& \ \{i, d\} \in S^t\}$. It generates the following valid inequality in the original graph G :

$$p_i + (b_i - |S^u|d_i)x_i + d_i \sum_{j \in V^t} x_j \geq b_i - |S^u|d_i.$$

Similarly, an extreme direction of the form Case 4 generates the following valid inequality in the original graph G :

$$p_i + (l_i g_i - |S^u|l_i)x_i + l_i \sum_{j \in V^t} x_j \geq l_i g_i - |S^u|l_i.$$

Here, we use $C_i^{\text{deg}(i)-k}$ to denote the set of all combinations with $\text{deg}(i) - k$ elements from node i 's neighbors and S is one combination picked from $C_i^{\text{deg}(i)-k}$. For a given i , if $k \geq g_i$, Case 3 and Case 4 extreme directions generated constraints are redundant. Thus, the projection of P_d onto (\mathbf{p}, \mathbf{x}) space is the following one:

$$p_i + b_i x_i + d_i \sum_{j \in n(i)} x_j \geq b_i \quad \forall i \in V \quad (5.47)$$

$$p_i + (b_i - kd_i)x_i + d_i \sum_{j \in S} x_j \geq b_i - kd_i \quad \forall i \in V \ \& \ k = 1, 2, \dots, g_i - 1 \quad (5.48)$$

$$\ \& \ S \in C_i^{deg(i)-k}$$

$$p_i + l_i g_i x_i + l_i \sum_{j \in n(i)} x_j \geq l_i g_i \quad \forall i \in V \quad (5.49)$$

$$p_i + (l_i g_i - kl_i)x_i + l_i \sum_{j \in S} x_j \geq l_i g_i - kl_i \quad \forall i \in V \ \& \ k = 1, 2, \dots, g_i - 1 \quad (5.50)$$

$$\ \& \ S \in C_i^{deg(i)-k}$$

$$1 \geq x_i \geq 0, \ p_i \geq 0 \quad \forall i \in V \quad (5.51)$$

Proposition 5.2. *The valid inequalities (5.48) and (5.50) can be separated in $O(|V|\delta \log \delta)$*

where δ is the maximum degree number among all nodes ($\delta = \max\{deg(i) : i \in V\}$).

Proof. Proof of Proposition 5.2. Given a fractional solution $(\mathbf{p}^*, \mathbf{x}^*)$, a node i in V and a specific k where $k = 1, 2, \dots, g_i - 1$, the corresponding separation procedure of inequality (5.48) can be formulation as the following optimization problem:

$$\text{Minimize } p_i^* + (b_i - kd_i)x_i^* + d_i \sum_{j \in n(i)} x_j^* z_j \quad (5.52)$$

$$\text{Subject To } \sum_{j \in n(i)} z_j = deg(i) - k \quad (5.53)$$

$$z_j \in \{0, 1\} \quad \forall j \in n(i) \quad (5.54)$$

For each node i in V , the binary variable z_i is 1 if node i is selected. Otherwise, it is 0. If the objective value is smaller than $b_i - kd_i$, we have a violated constraint. Otherwise, we either change the value of k or move to another node i . This optimization problem has one constraint and can be solved by taking the $deg(i) - k$ smallest values of x_j^* among node i 's neighbors ($j \in n(i)$). We can use the following algorithm to separate the whole

inequality set (5.48):

Algorithm 21 Separation algorithm for inequality set (5.48)

Require: A solution x^* and a 1TPLCIP instance.

```

1: for  $i \in V$  do
2:   Let  $S \leftarrow n(i)$ .
3:   for  $k = 1, 2, \dots, g_i - 1$  do
4:     let  $m_k = \arg \max\{x_i^* : i \in S\}$  and  $S \leftarrow S \setminus m_k$ .
5:     if  $p_i^* + (b_i - kd_i)x_i^* + d_i \sum_{j \in S} x_j^* < b_i - kd_i$  then
6:       Add  $p_i^* + (b_i - kd_i)x_i^* + d_i \sum_{j \in S} x_j^* \geq b_i - kd_i$ .
7:     else
8:       Break
9:     end if
10:  end for
11: end for

```

First, the solution x^* satisfied $p_i + b_i x_i^* + d_i \sum_{j \in S} x_j^* \geq b_i$ for all i in V . For the inequality $p_i + (b_i - kd_i)x_i + d_i \sum_{j \in S} x_j \geq b_i - kd_i$, as k increases by 1, the LHS decreases by $d_i(x_i^* + x_{m_k}^*)$ and the RHS decreases by d_i . For the current iteration k_0 , if it is the first time that $p_i + (b_i - k_0 d_i)x_i + d_i \sum_{j \in S} x_j \geq b_i - k_0 d_i$. Then, it means $x_i^* + x_{m_{k_0}}^* < 1$. Then, in the future iteration, we will not find any violated constraint for this particular i because $x_i^* + x_{m_k}^* \leq x_i^* + x_{m_{k_0}}^*$ when $k > k_0$. Therefore, in Algorithm 21, we use **Break** in line 8. Let δ denote the largest degree number among all nodes in V ($\delta = \max\{deg(i) : i \in V\}$). For each node, we sort its neighbors which takes at most $O(\delta \log \delta)$ steps and compare value at most δ times. The process is repeat for V nodes. So, the overall time complexity is $O(|V|\delta \log \delta)$. It is similar to separate inequalities (5.50). The difference is that the objective function of the optimization problem should be

$$\text{Minimize } p_i^* + (l_i g_i - kl_i)x_i^* + l_i \sum_{j \in n(i)} x_j^* z_j. \quad (5.55)$$

If the objective value is smaller than $l_i g_i - kl_i$, we have a violated constraint. \square

For the instance in Figure 5.1, the fractional solution in Figure 5.1(b) violates the constraint $p_3 + 18x_3 + 9(x_4 + x_2) \geq 18$ because $0 + 18 * 0.007 + 9 * (0.887 + 1) < 18$ given $p_3 = 0$, $x_3 = 1$, $x_3 = 0.007$ and $x_4 = 0.887$. Adding this constraint, resolving the LP relaxation of MIP5.1 gives an optimal solution with $x_2 = 1$ and $x_4 = 1$. Other variables are zeros. The objective value is 107.

Chapter 6: Conclusions and Future Work

With the widespread use of online social networks there is significant interest in solving problems dealing with viral marketing and influence maximization on social networks. There has been a flurry of activity by researchers in the computer science community on algorithmic aspects of problems in this arena. However, there seems to be little in terms of mathematical programming models on these important research problems. Our research is motivated by the desire to rectify this deficiency.

In this thesis, we study influence maximization problems on social networks from a mathematical programming perspective. Specifically, we study four combinatorial optimization problems. First we consider the Weighted Target Set Selection (WTSS) problem. Then, we study a problem called the Positive Influence Dominating Set (PIDS) problem which can be considered as the WTSS problem with a one time period restriction for influence diffusion process. After that, we incorporate partial incentives to obtain another problem called the Least Cost Influence Problem (LCIP). Lastly, the fourth problem called the One Time Period Least Cost Influence Problem is identical to the LCIP but only one time period is allowed for the diffusion process to take place.

Although almost all previous work involving influence maximization problems has focused on approximation algorithm and heuristics, our research demonstrates that mathe-

mational programming approaches are able to provide a better understanding of these problems. Our work fills in the gap for exact methods for influence maximization problems. We are able to find good quality solutions for random graphs with the size of 10,000 nodes and 20,000 edges within a reasonable amount of time. Our results can be used as meaningful benchmarks to evaluate the heuristics and approximation algorithms proposed for these problems. In the event the social networks analyzed are of smaller size our approach may be used to solve these smaller problems to optimality.

6.1 Future Work

In this section, we discuss four extensions to our current work. They consider proportion requirements, marked targets, latency constraints and combinatorial games.

6.1.1 Proportion Requirements

The first extension is to consider a less than 100% adoption of our problems. This has been mentioned in the general setting of the LCIP in Section 4.1.1. Instead of having $\alpha = 1$, now, α can take any value between 0 and 1. This generalization increases the difficulty level of our problems. Even for trees, we suspect that they are NP-hard. They are similar to the relationship between the Minimum Spanning Tree (MST) problem and the k -Minimum Spanning Tree (k -MST) problem. Given a graph with weighted edges, the MST problem looks for a spanning tree with weight less than or equal to the weight of every other spanning tree. The k -MST problem is given an additional input integer k . Then, it wants to find a minimum weighted tree with k nodes. It is well known that the

MST problem can be solved polynomially. But the k -MST problem is NP-hard (see Ravi et al. [1996]). This less than 100% adoption version is worth considering in real-world because it might be difficult to persuade all people to adopt a product or technique. After we solve these problem with $\alpha = 1$, it may turn out that we do not have enough money to spend. Then, it is natural to ask that how much money do we need if we want to cover 90% of the network? In terms of modeling, if we incorporate this requirement, those two good formulations for the LCIP and those extended formulations for the WTSS problem, the PIDS problem and the 1TPLCIP are no longer valid and needed to be modified.

6.1.2 Marked Targets

In this extension, in the input data, we have a group of special nodes. They are called "Marked Targets". We must have them activated in our solution. Following the argument in Section 6.1.1, it is possible to ask how much money do we need if we want these specific nodes activated? In terms of hardness, we think it is harder than the $\alpha = 1$ version which serves as a special case where all nodes are marked targets. But it should be easier than the one in Section 6.1.1. Again, if we incorporate this requirement, those two good formulations for the LCIP and those extended formulations for the WTSS problem, the PIDS problem and the 1TPLCIP are no longer valid and needed to be modified.

6.1.3 Latency Constraints

In previous models, we do not include time factor in the diffusion process for the LCIP and the WTSS problem. It is implicitly assumed the we can have as many time

periods as it needs, although the maximum number of time periods is bounded by the number of nodes in the graph. However, for a graph with n nodes, it is possible that a solution with minimum cost could take n time periods to have all nodes adopted.

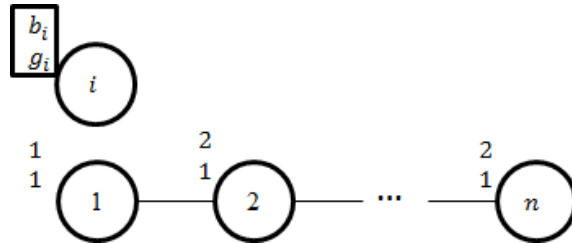


Figure 6.1: Time Period Example.

An example based on the WTSS problem is shown in Figure 6.1. The input graph is a line and has n nodes. For simplicity, each node has threshold $g_i = 1$. Node 1 has cost 1 and all other nodes have cost 2. The optimal solution is to pay node 1 at cost 1. Then, all nodes will become active at the end of the diffusion process. However, the whole process takes $n - 1$ time periods. In practice, this process may not be desirable when time plays a critical role. For instance, in Cicalese et al. [2014], in viral marketing, the speed of information spreading is important. Also, humans tend to make decisions based on recent events and information. Furthermore, decision makers prefer knowing the estimated maximum amount of adoption that can be achieved within a prefixed time periods over simply knowing that eventually (but may take a long time) the whole market might be covered.

Another interesting observation is that if time periods are considered, the PIDS problem can be seen as the problem where the diffusion takes place and all nodes should become adopters in one time period. Also, the WTSS problem is a case where n time periods are allowed. Similarly, the 1TPLCIP can be considered as a special case of the LCIP. Fur-

thermore, a generalized case can be as follows. We require one additional input integer k and study the problem that all nodes must become active within only k time periods.

6.1.4 Combinatorial Games

Although these diffusion models are considered in viral marketing setting, they can also be applied in the epidemiological setting (see Günneç [2012]). We have a social network where each person has a risk to be affected by a kind of disease. For a person i , let r_i be the safe risk level for it. Suppose that e_{ji} denotes the risk factors of an untreated neighbor j on node i (e.g., if δ_{ji} denotes the probability of node i getting infected by an untreated neighbor j , then $e_{ji} = -\log(1 - \delta_{ji})$). Let f_i denote the reduction of the risk on node i if one of its neighbor j is treated so that its risk level is less than or equal to a threshold risk level r_j . We would like to ensure that the sum of all $e_{ji} - f_i$ for node i minus the intervention or treatment strategy z_i reduces the overall risk of node i below the threshold risk level r_i . This may be equivalently modeled by the LCIP in the marketing setting with $b_i = \sum_{j \in N(i)} e_{ji} - r_i$ and $d_i = f_i$, with a discrete set intervention or treatment strategy choices at each node (e.g. $z_i = p_i$).

There is a key difference between the marketing setting and the epidemiological one. In the marketing setting, there is a centralizer (a company) who want to promote a new product. Hence, they are willing to pay the extra incentives to each potential customer. The cost can be considered as a marketing campaign cost. In the epidemiological setting, each person wants to reduce his/her risk level. Then, the cost is afforded by the person who receives the treatment. However, there are many free riders.

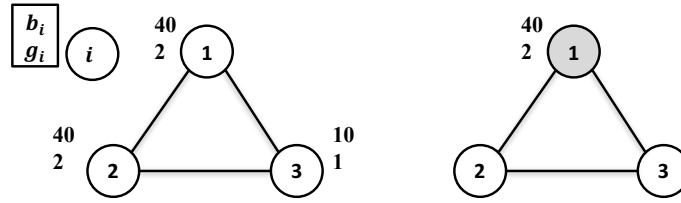


Figure 6.2: A WTSS problem instance and its optimal solution

For instance, in Figure 6.2, person 1 and person 3 are free riders in the small example because they have not received any treatment but their risk level are reduced due the fact that only person 2 receives treatment. In the small example, the cost is 40. Hence, naturally, this leads to the question that how should we allocate the cost? Is it fair that only person 2 pays the cost? To answer above questions, we would like to combine cooperative game theory with these diffusion models. Cooperative game theory is concerned with situations in which at least two decision makers can benefit by cooperation (see Nisan [2007]). We would like to ask these questions: How can we apply cooperative game theory to our model in the epidemiological setting?

Bibliography

- E. Ackerman, O. Ben-Zwi, and G. Wolfowitz. Combinatorial model and bounds for target set selection. *Theoretical Computer Science*, 411(44):4017--4022, 2010.
- M. Baïou and F. Barahona. The dominating set polytope via facility location. In *Combinatorial Optimization*, pages 38--49. Springer, 2014.
- E. Balas and W. Pulleyblank. The perfectly matchable subgraph polytope of a bipartite graph. *Networks*, 13(4):495--516, 1983.
- O. Ben-Zwi, D. Hermelin, D. Lokshtanov, and I. Newman. Treewidth governs the complexity of target set selection. *Discrete Optimization*, 8(1):87--96, 2011.
- C. Blum. Revisiting dynamic programming for finding optimal subtrees in trees. *European Journal of Operational Research*, 177(1):102--115, 2007.
- M. Bouchakour, T. Contenza, C. Lee, and A. R. Mahjoub. On the dominating set polytope. *European Journal of Combinatorics*, 29(3):652--661, 2008.
- N. Chen. On the approximability of influence in social networks. *SIAM Journal on Discrete Mathematics*, 23(3):1400--1415, 2009.
- W. Chen, C. Castillo, and L. V. Lakshmanan. *Information and Influence Propagation in Social Networks*. Morgan & Claypool Publishers, 2013.
- C. Chiang, L. Huang, B. Li, J. Wu, and H. Yeh. Some results on the target set selection problem. *Journal of Combinatorial Optimization*, 25(4):702--715, 2013.
- F. Cicalese, M. Milanič, and U. Vaccaro. Hardness, approximability, and exact algorithms for vector domination and total vector domination in graphs. In *Fundamentals of Computation Theory*, pages 288--297. Springer, 2011.
- F. Cicalese, G. Cordasco, L. Gargano, M. Milanič, and U. Vaccaro. Latency-bounded target set selection in social networks. *Theoretical Computer Science*, 535:1--15, 2014.
- G. Cordasco, L. Gargano, A. A. Rescigno, and U. Vaccaro. Optimizing spread of influence in social networks via partial incentives. In *Structural Information and Communication Complexity*, pages 119--134. Springer, 2015.

- E. D. Demaine, M. Hajiaghayi, H. Mahini, D. L. Malec, S. Raghavan, A. Sawant, and M. Zadimoghadam. How to influence people with partial incentives. In *Proceedings of the 23rd international conference on World wide web*, pages 937--948. International World Wide Web Conferences Steering Committee, 2014.
- T. N. Dinh, Y. Shen, D. T. Nguyen, and M. T. Thai. On the approximability of positive influence dominating set in social networks. *Journal of Combinatorial Optimization*, 27(3):487--503, 2014.
- P. Domingos and M. Richardson. Mining the network value of customers. In *Proceedings of the seventh ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 57--66. ACM, 2001.
- S. Goel, A. Anderson, J. Hofman, and D. J. Watts. The structural virality of online diffusion. *Management Science*, 62(1):180--196, 2015.
- M. Granovetter. Threshold models of collective behavior. *American Journal of Sociology*, 83(6):1420--1443, 1978.
- M. S. Granovetter. The strength of weak ties. *American Journal of Sociology*, 78(6):1360--1380, 1973.
- M. Grötschel, M. Jünger, and G. Reinelt. On the acyclic subgraph polytope. *Mathematical Programming*, 33(1):28--42, 1985.
- D. Günneç. Integrating social network effects in product design. Dissertation, 2012.
- D. Günneç and S. Raghavan. Integrating social network effects in the share-of-choice problem. *Decision Science*, 2016.
- T. W. Haynes, S. Hedetniemi, and P. Slater. *Fundamentals of domination in graphs*. CRC Press, 1998a.
- T. W. Haynes, S. T. Hedetniemi, and P. J. Slater. *Domination in graphs: advanced topics*. Marcel Dekker, 1998b.
- D. Kempe, J. Kleinberg, and É. Tardos. Maximizing the spread of influence through a social network. In *Proceedings of the ninth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, pages 137--146, 2003.
- J. Leskovec. Stanford large network dataset collection. URL <http://snap.stanford.edu/data/index.html>, 2011.
- H. Marchand and L. Wolsey. The 0-1 knapsack problem with a single continuous variable. *Mathematical Programming*, 85(1):15--33, 1999.
- G. L. Nemhauser and L. A. Wolsey. *Integer and combinatorial optimization*, volume 18. Wiley New York, 1988.
- N. Nisan. *Algorithmic game theory*. Cambridge University Press, 2007.

- H. Raei, N. Yazdani, and M. Asadpour. A new algorithm for positive influence dominating set in social networks. In *Proceedings of the 2012 International Conference on Advances in Social Networks Analysis and Mining (ASONAM 2012)*, pages 253--257. IEEE Computer Society, 2012.
- S. Raghavan and R. Zhang. Weighted target set selection on social networks. Working paper, University of Maryland, 2015.
- R. Ravi, R. Sundaram, M. V. Marathe, D. J. Rosenkrantz, and S. S. Ravi. Spanning trees-short or small. *SIAM Journal on Discrete Mathematics*, 9(2):178--200, 1996.
- A. Saxena. On the dominating set polytope of a cycle. *Working Paper*, 2004.
- P. Shakarian, S. Eyre, and D. Paulo. A scalable heuristic for viral marketing under the tipping model. *Social Network Analysis and Mining*, 3(4):1225--1248, 2013.
- G. Spencer and R. Howarth. Maximizing the spread of stable influence: Leveraging norm-driven moral-motivation for green behavior change in networks. *Working paper*, 2015.
- C. Stein, T. Cormen, R. Rivest, and C. Leiserson. *Introduction to algorithms*. MIT Press Cambridge, MA, 2009.
- F. Wang, E. Camacho, and K. Xu. Positive influence dominating set in online social networks. In *Combinatorial Optimization and Applications*, pages 313--321. Springer, 2009.
- F. Wang, H. Du, E. Camacho, K. Xu, W. Lee, Y. Shi, and S. Shan. On positive influence dominating sets in social networks. *Theoretical Computer Science*, 412(3):265--269, 2011.
- G. Wang, H. Wang, X. Tao, and J. Zhang. A self-stabilizing algorithm for finding a minimal positive influence dominating set in social networks. In *Proceedings of the Twenty-Fourth Australasian Database Conference-Volume 137*, pages 93--99. Australian Computer Society, Inc., 2013.
- D. Watts and S. Strogatz. Collective dynamics of "small-world" networks. *Nature*, 393(6684):440--442, 1998.
- W. Zhang, W. Wu, F. Wang, and K. Xu. Positive influence dominating sets in power-law graphs. *Social Network Analysis and Mining*, 2(1):31--37, 2012.
- X. Zhu, J. Yu, W. Lee, D. Kim, S. Shan, and D.-Z. Du. New dominating sets in social networks. *Journal of Global Optimization*, 48(4):633--642, 2010.
- F. Zou, Z. Zhang, and W. Wu. Latency-bounded minimum influential node selection in social networks. In *Wireless Algorithms, Systems, and Applications*, pages 519--526. Springer, 2009.