# Improved results on models of greedy and primal-dual algorithms

Hyukjoon Kwon

A thesis submitted to the Faculty of Graduate Studies
in partial fulfilment of the requirements
for the degree of

Master of Science

Graduate Program in Computer Science
York University
Toronto, Canada

May 2008

**IMPROVED RESULTS ON MODELS OF GREEDY AND PRIMAL-DUAL ALGORITHMS**

by **Hyukjoon Kwon**

a thesis submitted to the Faculty of Graduate Studies of York University in partial fulfilment of the requirements for the degree of

**MASTER OF SCIENCE**

**IMPROVED RESULTS ON MODELS OF GREEDY AND PRIMAL-DUAL ALGORITHMS**

by **Hyukjoon Kwon**



Examination Committee Members:

1. Jeff Edmonds
2. Suprakash Datta
3. Patrick Dymond
4. Stephen Watson

# Abstract

A class of priority algorithms that capture reasonable greedy-like algorithms was introduced by Borodin, Nielson, and Rackoff [7]. Later, Borodin, Cashman, and Magen [4] introduced the stack algorithms, advocating that they capture reasonable primal-dual and local-ratio algorithms. In this thesis, some $NP$-hard graph optimization problems - Maximum Acyclic Subgraph (MAS) problem and Minimum Steiner Tree (MST) problem - are studied in priority and stack models. First, a $2 - \frac{1}{k}$ priority lower bound in the edge model is shown for the MAS problem. Secondly, a $\frac{4}{3}$ priority lower bound in the edge model is presented for the MST problem, improving the result of Davis and Impagliazzo [10]. Making variations on input instances and the stack model, we show a $2 - \frac{2}{k}$ stack lower bound improving the $\frac{4}{3}$ stack lower bound in Borodin, Cashman, and Magen [4].

*To our parents and families*

# Acknowledgments

I am indebted to my teacher, Jeff Edmonds, for “irrevocably” accepting me as his student, for being an excellent supervisor, and for being a wonderful friend on this academic journey. He made it his priority to meet with me, and patiently helped me to understand problems at hand whenever he saw a frown on my face. His invariant emphasis on abstraction of research problems and his research style have highly influenced me over the last two years. I am grateful for his mentorship helping me understand both academic and non-academic cultures.

I thank to Patrick Dymond and Suprakash Datta for their suggestions and directions. They spotted errors and encouraged me to correct them with dignity. I thank my external committee member, Stephen Watson, for giving me appropriate feedback and suggestions. I also thank Parke Godfrey for warm conversations whenever I came by his office.

I thank Andrew Ryu and Jungmin Han for their mentorship, kindness, and friendship. I am grateful for Jaisingh Solanki’s encouragement and his organization of a series of theory talks. I also thank Erich Leung and Shakil Khan for suggesting improvements on the presentation of this thesis. Including Nassim Nasser, Romil Jain, Nelson Moniz, and Anton Belov, many graduate colleagues have been valuable on this unforgettable journey provid-

ing cheerfulness.

With all my heart, I would like to thank my mother, my wife, and my daughter. Kija Song, my mother, has consistently provided her unconditional love for three decades. Ye-unsil Lee, my wife, has supported my work with her love and cheer. Yeju, my daughter, has shown me how wonderful the gift of life is with her incredible smile. Their love became the profound basis for this thesis.

# Table of Contents

# List of Tables

# List of Figures

# 1 Introduction

## 1.1 Motivation

Many known efficient algorithms for computational problems can be classified into a few algorithm paradigms: greedy algorithms, divide-and-conquer, dynamic programming, linear programming and backtracking algorithms. An approach to evaluating different algorithm paradigms is to show approximability of a specific computational problem in each paradigm. A necessary component of such evaluation is to formalize a precise model for each algorithm paradigm.

Using the precise model, one can compare different algorithm paradigms. For a particular optimization problem, it may be the case that it can be proved that greedy algorithms captured by a specific model cannot be approximated beyond a certain ratio whereas algorithms within another model can. This provides not only the theoretical separation of these models but also practical guides to designing algorithms by ruling out possibilities. Moreover, the formal model gives us a better understanding of the structure of the computational problem at hand. Using the model, one can joyfully wander between lower and upper bound. When one is to explore a lower bound of a particular computational prob-

lem, the understanding of the structure of the problem can be deepened. On the other hand, when one studies an upper bound side one can devise/modify an algorithm using the gained understanding of the problem structure. This oscillating journey with the problem may continue until the gap between the upper bound and the lower bound is closed. At last, varying the model deepens one's understanding of what is important in the model.

Because of its simplicity and efficiency on many computational problems, the greedy algorithm paradigm is one of most important tool in designing algorithms. The terminology, *Greedy*, appears in the paper of Jack Edmonds [11]. The classical matroid and greedoids, studied during 1970s and 1980s, are limited in which algorithms they capture and deal only with optimal algorithms. Recently, Borodin, Nielsen and Rackoff, [7], introduced a model called ***priority algorithms*** which captures any reasonable greedy or greedy-like algorithms. Along with the idea of an optimal solution these algorithms provide the idea of an approximation algorithm.

The primal-dual and local ratio approaches for approximation algorithms are also two fundamental algorithm design paradigms. These two approaches usually encompass the greedy approach. That is, if one can prove arbitrary large competitive ratio in priority algorithm framework, one can hope to find a better approximation ratio using a more powerful algorithm paradigm such as primal-dual and local ratio design paradigms. These two paradigms have successfully tackled many computational problems and, for many of those problems, they currently hold the best approximation ratio. The paper of Allan Borodin, David Cashman, and Avner Magen, [4], starts to explore how much better these approaches can do. They formalized a formal computational model that captures both the primal-dual

and the local ratio approaches. This model is called a ***stack algorithm***. In this thesis, the class of priority algorithms and stack algorithms are studied in the context of $NP$-hard graph optimization problems.

## 1.2 Related Work

The priority algorithm model has the same decision characteristic as the on-line model. In both models, the irrevocable decisions are made on an input item based on its local information. However, a priority algorithm can order the input items, whereas an on-line algorithm processes input items in an order that an adversary presents. Hence, it is the ordering power that makes lower bound gap between these two models. Borodin, Nielson, and Rackoff [7] defines two types of priority algorithms - fixed priority algorithm and adaptive priority algorithm. In fixed priority the input order is initially specified by an algorithm and won't change throughout the game between an algorithm and an adversary. On the other hand, in an adaptive priority algorithm, algorithm can change the order based on the data items that it has processed so far. Clearly, the on-line algorithm is a special case of fixed priority algorithms and a fixed priority algorithm is a special case of adaptive priority algorithms.

Borodin, Nielson, and Rackoff [7] proved some upper and lower bound results for various scheduling problems. They separated the class of adaptive priority algorithm from the class of fixed priority algorithm for the interval scheduling problem on identical machines. They also made a separation between the class of deterministic and randomized

priority algorithms for the interval scheduling with arbitrary profits: they proved $\Delta$ lower bound in adaptive priority framework whereas a fixed priority algorithm with randomness can achieve $(\log \Delta)$-approximation. Here, $\Delta$ represents the ratio of the maximum to the minimum unit profit.

Shortly after, Angelopoulous and Borodin [2] proved that no adaptive algorithm can achieve an approximation ratio better than $\Omega(\log n)$. This bound is tight since there is an $(\log n)$-approximation algorithm which falls in the class of adaptive priority algorithm. They also found $\frac{4}{3}$ lower bound in adaptive priority framework for the uniform metric facility location problem.

Davis and Impagliazzo [10] modeled priority algorithms where input items are not independent. They separated the class of adaptive priority algorithms from the class of fixed priority algorithms for the shortest path problem: fixed priority algorithms cannot solve the problem with any approximation ratio whereas the adaptive priority can solve the problem optimally. In addition, they separated the class of adaptive priority algorithms from the memoryless priority algorithms for the weighted independent set problem on 2-regular graphs.

Borodin, Boyar, and Larsen [6] further extended priority algorithms for other graph optimization problems such as maximum independent set problem, unweighted vertex cover problem, and graph colouring problem. They presented a acceptance-first model and advocated that any memoryless algorithm for accept/reject problems can be simulated by the model. Horn [13] presented revocable acceptance priority model when a priority algorithm has the power to withdraw decisions on accepted items among the ones that has been

processed so far.

Borodin, Cashman, and Magen presented a model called *stack algorithm* that captures primal-dual schema and local ratio algorithms. They proved $\Omega(\log n)$ inapproximability result for the set cover problem, a $\frac{4}{3}$ inapproximability for minimum Steiner tree and a 0.913 inapproximability for interval scheduling problems. They advocated that stack algorithm with the corresponding LP input representation can show an inapproximability results for a suggested LP relaxation.

In recent work, Alekhnovich, Borodin, Buresh-Openheim, Impagliazzo, Magen, and Pitassi [1] proposed a decision-tree based model, BT algorithm, that captures properties of backtracking algorithms and simple dynamic programming algorithms. In this model, each node of the tree contains the input items processed so far and decisions on them. The width of a tree is not fixed since some branches are pruned off if the decision of the item conflicts with some constraints of a problem. This model allows us to think of a trade-off between the performance guarantee of the solution and the width of it. The width of a tree is defined as a maximum number of vertices among all depth levels. For example, 2-approximation is possible with 2-width of BT algorithm for knapsack problem whereas, in [1], the width of an optimal adaptive BT algorithm is at least $\binom{n/2}{n/4} = \Omega\left(2^{n/2}/\sqrt{n}\right)$. They also proved upper and lower bounds in the BT model for satisfiability and interval scheduling problems.

## 1.3 Summary and Organization of Thesis

This thesis is organized and summarized follows.

- In Chapter 2, we present the formal definition of priority algorithms. We also show how priority algorithms can be seen in context of a combinatorial game between an adversary and an algorithm.

- In Chapter 3, we show that no adaptive priority algorithm in the edge model can achieve $2 - \frac{1}{k}$ approximation for Maximum Acyclic Subgraph (MAS) problem.

- In Chapter 4, lower bounds for the Minimum Steiner Tree (MST) problem are presented for some combinations of possible variations to either the input instances or to the computation model. These results are summarized in Table 1.1. Each column of the table corresponds to way in the model may be varied. A *Y* in a column means that the variation considered is the one that makes solving the MST problem easier for an algorithm, correspondingly making it harder to prove a lower bound, and making the optimal competitive ratio possibly lower.

  Davis and Impagliazzo [10] gave the first priority lower bound for this problem, giving a competitive ratio of $\mathcal{R} = \frac{ALG}{OPT} = \frac{5}{4}$, where $ALG$ represents the cost of an algorithm's solution and $OPT$ represents the cost of an optimal solution. See Theorem 2.

  A quick exploration of the limits of their technique gives their result can easily be improved to an arbitrarily high competitive ratio, but this requires giving the algorithm instances where the triangle inequality does not hold. See Theorem 4. The first column ($\triangle$) of our table indicates whether the result assumes the triangle inequality. All the other results does.

| Thm | Sec | $\triangle$ | FW | $n$ | CG | Stack | $\mathcal{R}$ Bound |
|---|---|---|---|---|---|---|---|
| 2 | [10] | Y | N | N | Y | N | $\frac{5}{4}$ |
| 4 | 4.3 | N | Y | Y | Y | Y | $\infty$ |
| 3 | 4.3 | Y | Y | N | Y | N | $\frac{4}{3}$ |
| 5 | 4.4 | Y | Y | Y/N | Y | N | $1 + O(\frac{1}{k})$? |
| Open | | Y | N | Y | Y | N | ? |
| 6 | 4.4 | Y | Y | Y | N/N | N | $\infty$ |
| 7 | [4] | Y | Y | Y | N/N | Y | $\frac{4}{3}$ |
| 8 | 4.4 | Y | Y | Y | N/Y | Y | $2 - \frac{2}{k}$ |

Table 1.1: Summary of the known results and results presented in this paper for the MST problem: $\triangle$ stands for triangle inequality; FW stands for fixed weight on edges; CG stands for complete graphs; $n$ is the size of the connected instances; and $\mathcal{R}$ is a competitive ratio. See Section 4.2 for clearer definitions. A *Y* means easier for an algorithm and harder for an adversary, resulting in a potentially lower competitive ratio.

Playing with their cases, we managed to improve their lower bound to $\frac{4}{3}$. See Theorem 3. Besides getting a slightly better bound, an interesting thing is that unlike Davis and Impagliazzo's proof, ours assumes that the edge weights are fixed and known to an algorithm. This assumption greatly simplifies the proof. The second column (FW) of our table indicates whether the edges are fixed.

A concern we had about the result of Davis and Impagliazzo is that it only considers graphs with at most four vertices. With this counter example, they concluded that the algorithm cannot achieve $\mathcal{R} = \frac{ALG}{OPT} \leq \frac{5}{4}$. However, because the lower bound does not consider arbitrarily large graphs, it does not prove that this error is multiplicative. Hence, their result does not rule out the possibility that $ALG \leq 1 \cdot OPT + c$ for some constant $c$ depending on the maximum weight in the graph but not on the number of vertices in it. Whether $n$ is bounded, is indicated in the third column variation that we consider of the table. Note that *Y* in this column indicates that the instance is not only arbitrarily large but also its size, the number of vertices, is known to the algorithm, while *Y/N* indicates that the instance large but unknown size.

Given these ideas, our next goal was to prove a lower bound with fixed weights and arbitrarily large $n$. However, it turns out that this gives too much power to the algorithm. Instead, we conjecture that there is an algorithm with competitive ratio $1 + O(\frac{1}{n})$. This is not a reasonable algorithm. It plays tricks to learn the input instance and then uses its unbounded computational power to give a near optimal solution. See Theorem 5.

Given this result that the algorithm unfairly uses the fact that the edge weights are fixed, we tried hard to prove a lower bound with arbitrarily large $n$ by varying the weights. See the fifth row in the table. We were unable to do this. This is still an interesting open problem.

The result of Davis and Impagliazzo assumes that the input instance explicitly provides the edge weight for every possible pairs of vertices (except possibly those between Steiner vertices). With this requirement, the only way to change the size of the instance is to change the number of vertices. At first, we were under the impression that we had to do the same. Later, Borodin reassured us that it was acceptable to instead consider incomplete graphs as input instances. The algorithm is not allowed directly ask for and learn the edges that are not explicitly mentioned in the instance. Our fourth column (CG) indicates whether a *complete graph* is required. *N/N* indicates Borodin's initial model in which the algorithm is not allowed to include the implicit edges in the MST problem, while *N/Y* indicates our new model in which the algorithm is allowed. The reason for this differentiation is that if a priority algorithm is not allowed to include the implicit edges, then we were able to show arbitrarily high competitive ratio. See Theorem 6. This differentiation appears not to be an issue in the *stack* model in which Borodin intended it.

Borodin, Cashman, Magen [4] strengthened the priority model by defining a stack model in which an algorithm does not need to commit to an edge when it first sees it, but instead pushes the edge on a stack with the possibility that the edge will get

rejected when it is popped. Borodin et. al. advocate that this stack model captures reasonable primal-dual schema and local-ratio algorithms. They proved a $\frac{4}{3}$ stack lower bound in this model. See Theorem 7. We improve this result to $2 - \frac{2}{k}$. See Theorem 8.

- Finally, in chapter 5, we summarize our results and suggest some possible directions for the future work.

# 2 Preliminaries

In this chapter some definitions and properties for priority model are shown in section 2.1. Then, the game between algorithm and adversary is described in section 2.2.

## 2.1 Greedy and Priority Algorithms

According to many undergraduate textbooks, a greedy algorithm chooses and irrevocably commits or rejects to one of objects in the set of the objects specified in the input instance because, according to some simple criteria, it seems to be the best locally. With these characteristics, Borodin, Nielsen, and Rackoff [7] introduced the class, ***priority algorithms***, which capture any reasonable greedy algorithm.

An instance to a priority algorithm consists of a set of data items. Let $T$ be the type of a data item. It could be a vertex, an edge, an interval, or some other object. A priority algorithm chooses a decision $d$ on each data item $a_i \in I$, where $d$ is a choice from set of options, $\Sigma$. This choice set, $\Sigma$, varies from problem to problem, however, we deal only with $\Sigma = \{accept,\ reject\}$ in this paper. A solution for an instance is a set of tuple, $\{(a_i, d_i)|a_i \in I\}$. For example, for MAX3SAT problem, the priority algorithm assigns 0 (false) or 1 (true) to each variable $x_i \in I$. Therefore, $\Sigma = \{0, 1\}$ and $T$ is defined in context

of boolean variables.

When the instance of a problem is a graph G, either a ***vertex model*** or ***edge model*** can be used. The data items in the vertex model correspond to the vertices of the instance graph. More specifically, the data items contain certain information about the vertex in question. Depending on the specifics of the model, this information might include a vertex name, a weight, a list of edge names, and/or a list of neighbours. On the other hand, data items in an ***edge model*** are the edges in a given graph. Again depending on the model, these could contain the names of the two endpoints of an edge, weights of vertices, an edge label, an edge weight, and/or degrees of endpoints. Note that proving a priority lower bound is more difficult when the data items provide more information to the algorithm.

Two classes of priority algorithms are defined in [7], depending on whether or not the ordering of data items is fixed. A ***fixed*** priority algorithm orders the data items once at the start. On the other hand, an ***adaptive*** priority algorithm may reorder the unprocessed data items after it processes each data item. Therefore, the fixed priority algorithm is a special case of the adaptive priority algorithm. Moreover, because the on-line algorithm makes decision on the data item without ever ordering the data items, the on-line algorithm is a special case of a fixed priority algorithm. The algorithmic descriptions are shown in Algorithm 1 and Algorithm 2.

**Algorithm 1** Fixed Priority Algorithm

**Input:** $I = \{a_1, \ldots, a_n\}$

**Output:** $S = \{(a_i, d_i) | \ 1 \leq i \leq n\}$

1: Choose an ordering $\Pi$ on all possible data items.

2: Sort the data items, $\{a_1, \ldots, a_n\}$, based on the chosen order.

3: **for** $i = 1$ to $n$ **do**

4: Make an irrevocable decision $d_{\pi_i}$ about data item $a_{\pi_i}$.

5: Update the solution: $S = S \cup \{(a_{\pi_i}, d_{\pi_i})\}$.

6: **end for**

7: Output $S = \{(a_i, d_i) | \ 1 \leq i \leq n\}$.

**Algorithm 2** Adaptive Priority Algorithm

**Input:** $I = \{a_1, \ldots, a_n\}$

**Output:** $S = \{(a_i, d_i) | \ 1 \leq i \leq n\}$

**for** $i = 1$ to $n$ **do**

2: Assume the data items $\{a_{\pi_1}, a_{\pi_2}, \ldots, a_{\pi_{i-1}}\}$ have been seen and processed. Based on the data items seen so far, choose a possible new ordering $\Pi$ on all possible data items. Let $a_{\pi_i}$ be the next data item according to this new order.

Make an irrevocable decision $d_{\pi_i}$ about the data item $a_{\pi_i}$.

4: Update the solution: $S = S \cup \{(a_{\pi_i}, d_{\pi_i})\}$.

**end for**

6: Output $S = \{(a_i, d_i) | \ 1 \leq i \leq n\}$.

## 2.2 Priority Lower Bound via Combinatorial Game Between an Algorithm and an Adversary

In this section, a game between an adversary and an algorithm is described.

It is far too hard to prove bounds on the computational powers of an algorithm. Hence, we avoid this problem by assuming that the algorithm has an unbounded computational power. The priority lower bound scheme described in this section is information theoretic. Just like with lower bounds for on-line algorithms, we would like to assume that an algorithm does not "know" about any unprocessed item in detail.

Unlike, on-line algorithms, however, a priority algorithm has some knowledge and control over the unprocessed data items because it gets to have them be sorted. For example, if the algorithm has sorted data items based on the weight of data items and has been given item $a_{i-1}$ with weight 10, then the algorithm knows that any later item has a weight greater or equal to 10.

At the beginning of the $i^{th}$ iteration of the game, the algorithm has been given the partial instance $PI_{i-1} = \langle a_1, a_2, \ldots, a_{i-1} \rangle$ and has committed to the partial solution $PS_{i-1} = \langle d_1, d_2, \ldots, d_{i-1} \rangle$. The adversary has chosen the set $P_{i-1}$ of the data items from which future data items are chosen. More formally, the actual input instance will be $I = \{a_1, a_2, \ldots, a_{i-1}, a_i, \ldots, a_n\}$, where $a, \ldots, a_{i-1}$ are as stated in the partial solution $PI_{i-1}$, $a_{i+1}, \ldots, a_n$ are from the restricted set of unseen data items $P_{i-1}$ and $n \geq i-1$ is the yet unknown number of items in the instance. The loop invariant that the adversary maintains is that given any instance $I$ of this form the algorithm would have seen the partial instance $PI_{i-1} =$

$\langle a_1, \ldots, a_{i-1} \rangle$, committed to the decisions $PS_{i-1} = \langle d_1, \ldots, d_{i-1} \rangle$, and know nothing about the future items except that they come from $P_{i-1}$.

The adversary initializes the game by narrowing down the entire universe of possible data items to $P_0$. At the same time, $PI_0$ and $PS_0$ are defined to be empty sets since the algorithm has done nothing at this stage.

During the $i^{th}$ iteration, $P_i$, $PI_i$, and $PS_i$ that maintain this loop invariant are chosen as follows. The algorithm being adaptive is allowed to reorder the data items in $P_{i-1}$. The adversary promises that he will make whichever data item $a_k \in P_{i-1}$ is the algorithm's favorite be $a_{\pi_i}$. Note that we don't really care about the complete order since we only care about which data item will be $a_{\pi_i}$. Hence, the game is simplified by allowing the algorithm to choose $a_{\pi_i}$ from $P_{i-1}$ and make a decision $d_i$ on $a_{\pi_i}$. Knowing $a_{\pi_i}$ and $d_{\pi_i}$, the adversary is allowed to narrow down $P_{i-1}$ to $P_i$ by removing some data items (including $a_{\pi_i}$) that would make the algorithm's task "easy". This ends $i^{th}$ iteration and opens $(i+1)^{th}$ iteration of the game, that is, the algorithm has seen $\langle a_{\pi_1}, a_{\pi_2}, \ldots, a_{\pi_i} \rangle$ and made decisions $\langle d_1, \ldots, d_i \rangle$ on them and knows the future items are from $P_i$.

Generally, the adversary is allowed to dynamically choose how many data items there will be in the actual instance. Hence, the adversary has the power at the beginning of the $i^{th}$ iteration to declare that the game ends. Then, the actual instance is $I = PI_{i-1} \cup P_{i-1}$ and by the loop invariant $S = PS_{i-1}$ would be the algorithm's solution. The adversary wins if $I$ is a valid instance and $S$ is not an optimal solution for it. If there is an adaptive priority algorithm for this problem then such an algorithm is able to win the game against any adversary. The contrapositive is that if we can show an adversarial strategy for this

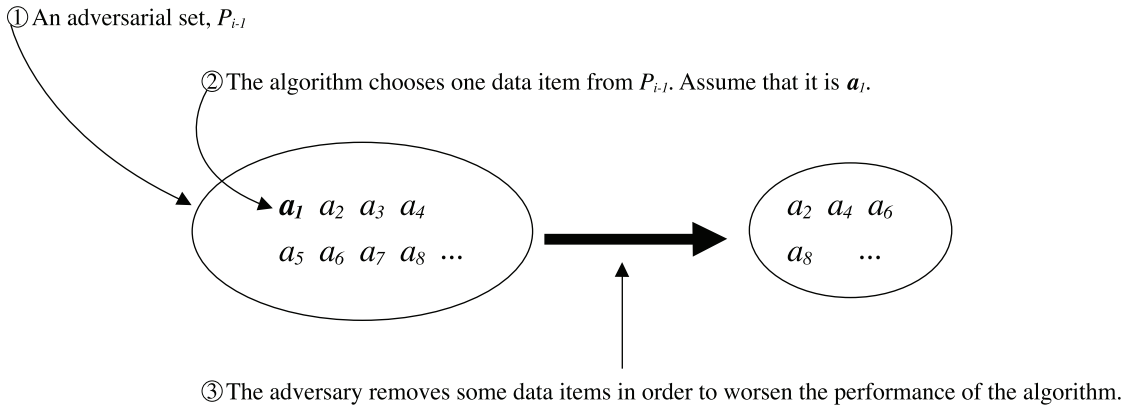


Figure 2.1: The description of the $i^{th}$ round of the game

game then there is no adaptive algorithm for this problem .

# 3 Maximum Acyclic Subgraph problem

As a warm up, we will prove a relatively easy adaptive priority lower bound. The problem considered is Maximum Acyclic Subgraph (MAS) problem. An instance of the problem is a simple directed graph $G = (V, E)$. The problem is to find a maximum sized subset of the edges that is acyclic. Two 2-approximation algorithms are known and are described in Vazirani [16] and in Berger and Shor [3]. The first of these algorithms can be interpreted as a priority algorithm that keeps two solution sets: the algorithm arbitrarily orders the vertices; the algorithm then puts the forward edges in the first solution set and puts the backward edges in the second solution set. When all edges are processed, the algorithm returns either the set of forward edges or the set of backward edges, depending on which set has greater cardinality. Clearly, both of these two solutions are acyclic. One of the two sets must have at least $|E(G)|/2$ edges. Hence, the value achieved by the algorithm, $ALG$, is at least $|E(G)|/2$, which is at least $OPT/2$, giving the ratio $\mathcal{R} = \frac{OPT}{ALG} = 2$.

The problem with the above algorithm is that it does not commit to accepting or rejecting an edge until it has seen all the edges. Borodin et. al. [1] extended the definition of the priority algorithm to allow it to keep some polynomial number of solutions in parallel: the extended model is called the *backtracking model*. The number of parallel solutions that

the above algorithm needs is 2. Can we design a priority algorithm that keeps only one solution for the MAS problem? This question is answered in [3]. Their algorithm first arbitrarily orders the vertices. Then, for each vertex $v$ in the chosen order, it considers all incident edges to $v$ that have not been considered before. Of these, it accepts either all incoming edges or all outgoing edges, depending on which has greater cardinality and rejects the others. After the algorithm processes all vertices, it returns the set of accepted edges as its solution. Note that this solution is acyclic as follows. Consider any cycle $C$ in the graph. Let $v$ denote the first vertex in the cycle that the algorithm considers. Let $\langle u, v\rangle$ and $\langle v, w\rangle$ be the edges in the cycle coming into and leaving $v$. Because $v$ is the first edge considered in the cycle, neither of these edges has been considered yet. Hence, when the algorithm reject either all of $v$'s unprocessed incoming edges or all of $v$'s unprocessed outgoing edges, either $\langle u, v\rangle$ or $\langle v, w\rangle$ will be rejected. Either way this breaks the cycle. Because this is true for every cycle in the graph the solution produced by the algorithm is valid. With the similar reason described for the first algorithm in [16], it is easily shown that the performance guarantee by this algorithm is at least half of the optimal solution. Their algorithm can be fit in as either a vertex or an edge model. Fitting into the vertex model, the data item is a vertex with a list of its neighbours. One difference from the usual vertex model is that the decisions are made on the edges. Denote $deg^-(u)$ to be the number of directed edges coming into $u$ and $deg^-(u)$ to be the number of directed edges going out from $u$. Fitting in the edge model, the data item is an edge $\langle u, v\rangle$ with the in-degree of $u$ and out-degree of $u$, namely $(u, v, deg^-(u), deg^+(u))$. Since the algorithm remembers all accepted items and deleted items, when it considers a data item $(u, v, deg^-(u), deg^+(u))$,

it will make a decision based on the comparison of $deg^-(u) - x$ and $deg^+(u) - y$, where $x$ is the number of either accepted or rejected incoming edges to $u$ so far and $y$ is the number of either accepted or rejected outgoing edges from $u$ so far. If $deg^-(u) - x \geq deg^+(u) - y$ is the case, the algorithm rejects the considered data item. Subsequently, using its ordering power, it rejects all remaining unseen outgoing edges from $u$ and then accepts all remaining unseen incoming edges to $u$. Otherwise, the algorithm accepts the data item and all remaining unseen outgoing edges from $u$ and rejects all remaining unseen incoming edges to $u$.

We tried, but failed to prove a matching lower bound of 2 both in the vertex model and in the edge model with degrees. These remain open problems. We did however manage to prove a lower bound in the edge model where the degrees are not included.

**Theorem 1.** *No adaptive priority algorithm in the edge model can achieve better than* $2-\frac{1}{k}$ *approximation for the MAS problem.*

*Proof.* A data item, a directed edge, is represented as an ordered pair $\langle u, v\rangle$, where $u$ is the tail and $v$ is the head. The set of decisions is $\Sigma = \{accepted, rejected\}$. Initially, an adversary narrows down the entire universe of possible data items down to $P_0$, which consists of the edges in a $(k+1)$-complete graph. Because of the symmetry in the set of edges $P_0$, we can assume without loss of generality that the algorithm processes the edge $\langle 1, 2\rangle$ first. This edge has been made bold in Figure 3.1.

- **Case 1:** If the algorithm accepts the edge $\langle 1, 2\rangle$, then, the adversary reduces $P_0$ down to $P_1$ as shown in Figure 3.1. In fact, the adversary declares at this point that this is

the final input instance. For each $j \in [3, 4, \ldots, k+1]$, the algorithm can either take the edge $\langle 2, j\rangle$ or the edge $\langle j, 1\rangle$ but not both. Otherwise, this will create a cycle with the edge $\langle 1, 2\rangle$ already committed to. The optimal solution on the other hand can take both $\langle 2, j\rangle$ and $\langle j, 1\rangle$ for each $j \in [3, 4, \ldots, k+1]$ and take $\langle 2, 1\rangle$. This gives that the value of the algorithm's solution is at most $k$ whereas the cost of the optimal solution is $2k-1$. Hence, the approximation ratio is $\mathcal{R} = \frac{OPT}{ALG} = \frac{2k-1}{k} = 2 - \frac{1}{k}$.

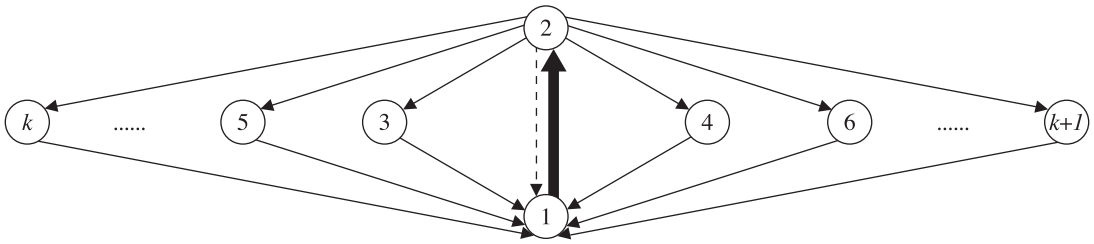


Figure 3.1: A possible adversarial set $P_1 \subseteq P_0$ based on the algorithm's first choice.

- **Case 2:** If the algorithm rejects the bold edge $\langle 1, 2\rangle$, the adversary reduces $P_0$ to $P_1$ by deleting all edges in $P_0$ except the single edge $\langle 1, 2\rangle$ just rejected by the algorithm. The approximation ratio is then $\mathcal{R} = \frac{1}{0} \approx \infty$.

Based on the two cases, no priority algorithm can ever achieve $2-\frac{1}{k}$ performance guarantee.

□

The complement of the MAS problem is the Minimum Feedback Arc-set (MFA) problem. The instance of the problem is a simple directed graph $G = (V, E)$. The problem is to remove the minimum number of edges that leaves the remaining graph as acyclic. Let us briefly mention that *no adaptive priority algorithm in the edge model can solve the MFA problem with any approximation ratio.* The proof is very similar to that of the MAS

problem except that the value of the solution is the number of edges deleted, which is the number of edges in the graph minus the number kept. In case one, $\mathcal{R} = \frac{ALG}{OPT} = \frac{k}{1} = k$ and, in case two, $\mathcal{R} = \frac{1}{0} \approx \infty$.

# 4 Minimum Steiner Tree (MST) Problem

We study Minimum Steiner Tree (MST) Problem in priority and stack algorithm frameworks. In Section 4.1, specifications, applications, and definitions of the problem are stated. In Section 4.2, some modeling issues are discussed such as triangle inequality, complete graph instances, the size of the connected graph instances, and fixed edge weights. In Section 4.3, we prove an arbitrary high and a $\frac{4}{3}$ priority lower bound for the MST problem under different variations. In Section 4.4, with another variation of the model, we conjecture a strange upper bound, $1 + O(\frac{1}{k})$. We in turn show that an arbitrary high priority lower bound can be achieved under the model appeared in Borodin, Cashman, and Magen [4]. Finally, we prove a $2 - \frac{2}{k}$ stack lower bound improving the result $\frac{4}{3}$ lower bound in [4].

## 4.1 Basis for the MST problem

An instance of the MST problem consists of a graph $G = (V, E)$, a nonnegative real-valued cost $c : \ E \longrightarrow R^+$ on the edges and a set $N \subseteq V$ of *terminal vertices*. A tree $T$ of $G$ is called a *Steiner tree* if it spans all the terminal vertices of $G$. The problem is to find a Steiner tree $T$ whose cost $\sum_{ij \in E(T)} c(ij)$ is minimum.

The problem has practical applications: routing VLSI layout; the design of communica-

tion networks; accounting and billing for the use of telephone networks; and the phylogeny problem in biology. Unfortunately, the problem is *NP-hard*, therefore, it is likely that any algorithm for finding a minimal Steiner tree is inefficient. Hence, designing an approximation algorithm, which has a polynomial running time, seems a good and profitable direction for this problem. The best approximation algorithm appears in Robins and Zelikovsky [15] and achieves 1.55 performance guarantee. However, it neither fits into a priority model nor a stack model. The best known priority algorithm, shown in Vazirani [16], achieves the 2-approximation ratio for the metric MST problem, simply by running the minimum spanning tree algorithm on the induced subgraph of the terminal vertices in the given instance graph. Clearly, this algorithm being fixed priority greedy, fits into the priority model.

Before we go further, several terms need to be defined. The Steiner vertices $V \setminus N$ are called ***Steiner*** vertices. An edge $e = (u, v)$ is called ***Steiner edge*** if $u$ is a Steiner vertex and $v$ is a terminal vertex, or vice versa. An edge $e = (t_g, t_{g+1})$ is called a ***terminal edge*** if the two endpoints, $t_1$ and $t_2$, are both terminal vertices. A ***quasi-bipartite*** graph $Q$ is a bipartite graph with some edges between terminal vertices. A ***complete*** quasi-bipartite graph denoted by $Q_{K,H}$ is a complete bipartite graph $G_{K,H}$ where terminal vertices in $K$ form clique. We denote ***k-star*** for $k$ Steiner edges incident to the same Steiner vertex. At last, we denote ***w(e)*** for the weight of an edge $e$.

## 4.2 Issues on Variations of Instances and Model

Section 1.3 outlined a number of variations in the MST problem and the model. These variations are described in more detail here.

**Triangle Inequality ($\Delta$):** With unconstrained edge weights, we will prove that *no adaptive priority algorithm can solve this version of the MST problem with any approximation ratio*. Hence, we consider the metric version of the MST problem which requires the triangle inequality to hold, namely $w((u,v)) \leq w((u,w)) + w((w,v))$.

**Fixed Edge Weights vs. Varying Edge Weights (FW)** : Our second variation is whether the adversary is allowed to vary the weights on the edges or must fix them. The motivation of the latter is that having the adversary change the weight of unseen edges complicates the proof. In fact, one needs to be careful about how one models this. The adversary cannot change these weights arbitrarily. The algorithm may have some very complex criteria that it uses to sort the data items. If the weights change even slightly, the total order may change, changing which data item the algorithm would have seen first. Instead, this is modeled by the adversary placing multiple data items $\langle u, v, w(u,v)\rangle$ between the same two vertices $\langle u, v\rangle$ in the set $P_i$ of possible future data items, each with distinct edge weights $w(u,v)$. The adversary can throw out any copies of an edge $\langle u, v\rangle$ that have weights $w(u,v)$ that he does not want to give the algorithm. Then when algorithm sorts possible data items in $P_i$, it states which of these edge $\langle u, v, w(u,v)\rangle$ it wants next. When the adversary gives the algorithm this edge, it needs to immediately delete from $P_{i+1}$ any additional copies

of this same edge $\langle u, v \rangle$ that have different weights. This ensures that the actual input instance only has at most one edge between any pair of vertices.

Davis and Impagliazzo [10] proved a $\frac{5}{4}$ priority lower bound with the varying edge weight approach. See Theorem 2. In Section 4.3, we improve this to $\frac{4}{3}$ but with the fixed edge weights approach. In Theorem 5, however, we see that in general, having fixed weights gives too much power to the algorithm.

**Not Known but Fixed Instance Size vs. Arbitrary but Known Instance Size $n$** : Our t-hird variation is whether the lower bound is proved with a fixed instance size (having some small number of vertices) without giving the algorithm the instance size or whether a stronger result can be proved in the following two ways: first, it is stronger because it applies to arbitrarily large graphs; secondly, it is stronger because it gives the size information to the algorithm.

For many computational problems, the size difference is not crucial because we can achieve the same lower bound by creating a final instance, consisting of multiple copies from the class of small instances. However, this causes the generated instance graph to be disconnected. For the MST problem, the graph needs to be connected. This is why we had this concern about Davis and Impagliazzo's result. Because it only considers graphs with at most four vertices, it is not clear whether this proves the multiplicative constant $\frac{ALG}{OPT} \leq \frac{5}{4}$ or only an additive one $ALG \leq 1 \cdot OPT + c$ for some constant $c$ depending on the maximum weight in the graph but not on the number of vertices in it. In fact, Theorem 5 shows how the latter is the case when

edge weights are fixed.

Providing the size of instances at the beginning restricts the adversarial reduction strategies from $P_{i-1}$ to $P_i$ because it cannot delete all the edges incident to a vertex $u$. Hence, this variation favours for an upper bound. In Section 4.4, we prove a priority and a stack lower bound letting the algorithm "know" the instance size.

**Incomplete vs. Complete Instances:** Our fourth variation is whether the input instance must be a complete graph or not. If yes, then there must be an explicit data item in the instance, providing the weight between every possible pair of vertices (expect those between Steiner vertices). If no, edges that are not explicitly mentioned in the instance have weights imposed by the triangle inequality, (i.e. the weight of a missing edge $(u, v)$ is the length of the shortest path from $u$ to $v$ amongst the included edges).

Allowing a non-complete instance expands the set of possible input instances and, therefore, gives more power to the adversary. It gives the adversary complete power to restrict the set $P_i$ of possible future edges from the current set $P_{i-1}$ as defined in the section 2.2. We call this *the adversary deleting edges*. In contrast, for the complete graph model, the adversary is not able to delete edges an arbitrary way. When restricting the data item set $P_i$ from $P_{i-1}$ , it still has the power to delete an edge $(u, v)$, but, in order to keep the final instance graph complete, it must either delete all the edges incident to the vertex $u$ or all those incident to $v$. Although such a step $P_i \subseteq P_{i-1}$ is really a reduction in the set of edges, we call this *the adversary deleting the vertex $u$ (or $v$)*.

## 4.3 A $\frac{4}{3}$ Priority Lower Bound

In this section, we prove the arbitrary large priority lower bound and the $\frac{4}{3}$ priority lower bound. Davis and Impagliazzo [10] proved the first priority lower bound for the MST problem, where $\mathcal{R} = \frac{5}{4}$. See Theorem 2. A quick exploration of the limits of their technique gives their result can easily be improved to an arbitrarily high competitive ratio, but this requires giving the algorithm instances where the triangle inequality does not hold. See Theorem 4. Playing with their cases further, we managed to improve their lower bound to $\frac{4}{3}$. Another small improvement of our result is that we do it in the model in which each edge weight in an instance is fixed. This strengthens the result because the algorithm has more knowledge and the adversary has less power. In addition, the fixed weight variation significantly simplifies the proof.

**Theorem 2.** *(Davis and Impagliazzo [10]) No adaptive priority algorithm in the edge model can achieve better than $\frac{5}{4}$ approximation for the MST problem with the following variations.*

| $\triangle$ | *Fixed Weight* | $n$ | *Complete Graph* | *Stack* | $\mathcal{R}$ *Bound* |
|---|---|---|---|---|---|
| *Y* | *N* | *N* | *Y* | *N* | $\frac{5}{4}$ |

**Theorem 3.** *No adaptive priority algorithm in an edge model can achieve better than $\frac{4}{3}$ approximation for the MST problem with the following variations.*

| $\triangle$ | *Fixed Weight* | $n$ | *Complete Graph* | *Stack* | $\mathcal{R}$ *Bound* |
|---|---|---|---|---|---|
| *Y* | *Y* | *N* | *Y* | *N* | $\frac{4}{3}$ |

*Proof.* The adversary initially constructs a set $P_0$, consisting of the edges within the complete quasi-bipartite graph $Q_{3,1}$ shown on the left of Figure 4.1.

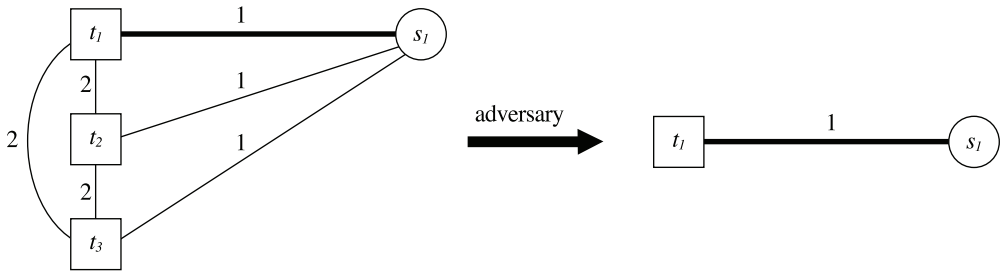


Figure 4.1: The left is an initial adversarial set $P_0$ and the bold edge is assumed to be considered first by the algorithm. The right is the second adversarial set $P_1 \subseteq P_0$, reduced from $P_0$ by the adversary. The squared vertices are terminals and the circled vertices are Steiners.

- **Case 1:** Since the three Steiner edges are identical initially, without loss of generality, assume that it considers considers the edge $a_{\pi_1} = (r_1, s_1)$, made in bold in Figure 4.1.
  - **Case 1.1:** If the algorithm accepts the edge, the adversary responds by removing all the remaining edges as shown on the right of Figure 4.1. The remaining graph will be the final input instance. It is a complete graph as promised. The algorithm should not have taken the bold edge. The approximation ratio is $\mathcal{R} = \frac{ALG}{OPT} = \frac{1}{0} \approx \infty$.
  - **Case 1.2:** If the algorithm rejects the bold edge, the adversary gives the entire graph as the instance shown in Figure 4.2. Again this is complete. No matter what future decisions are made, the best cost for the algorithm is 4 whereas the optimal cost is by taking the three Steiner edges. Therefore, the approximation

ratio is $\mathcal{R} = \frac{4}{3}$.

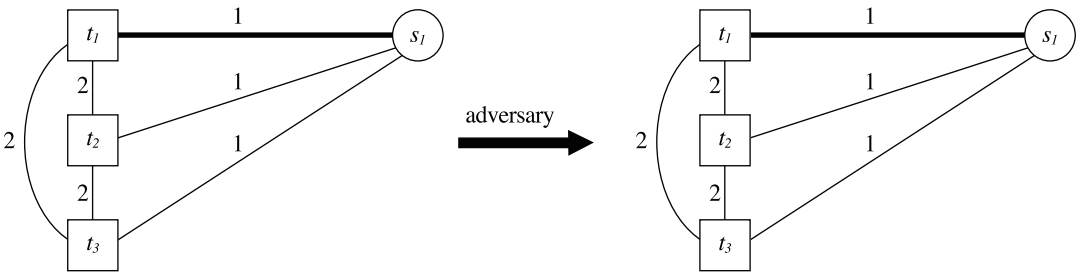


Figure 4.2: A possible adversarial set $P_1$ (the right graph) reduced from $P_0$ (the left graph) by the adversary.

- **Case 2:** We now consider the second case where the algorithm takes a terminal edge. Since three terminal edges are identical, without loss of generality, we assume that the algorithm considers the edge $a_{\pi_1} = (r_1, r_2)$ made the bold in Figure 4.3.
  - **Case 2.1:** If the algorithm accepts the edge, the adversary does nothing. No matter what future decisions are made, the best cost for the algorithm is 4 whereas the optimal cost is just 3 using the three Steiner edges. The approximation ratio is yet $\mathcal{R} = \frac{4}{3}$.
  - **Case 2.2:** If the algorithm rejects the edge, the adversary removes all the remaining edges as shown in Figure 4.4. In this case, the algorithm even fails to find a valid Steiner tree.

Based on the two cases, the best competitive ratio $\mathcal{R}$ that the algorithm can achieve is $\frac{4}{3}$.

□

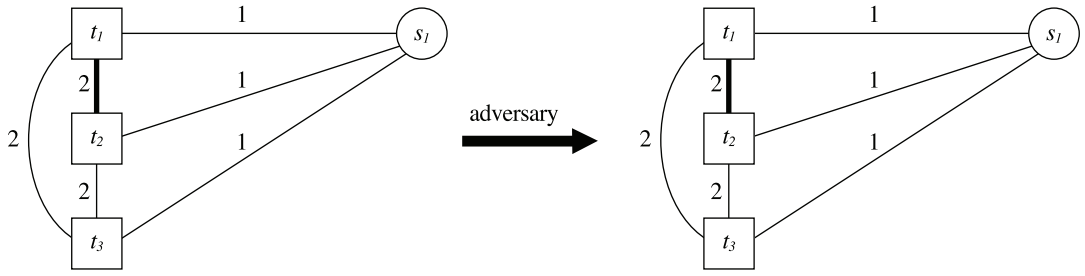


Figure 4.3: A possible adversarial set $P_1$ (the right graph) reduced from $P_0$ (the left graph) by the adversary when the algorithm decides to accept the bold terminal edge.

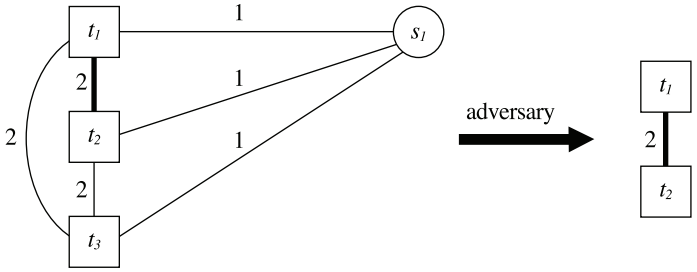


Figure 4.4: A possible adversarial set $P_1$ (the right graph) reduced from $P_0$ (the left graph) by the adversary when the algorithm decides to reject the bold terminal edge.

The reader may wonder if one can get a better lower bound by having much bigger weights on the terminal edges. This is the case, however, it violates the triangle inequality. We demonstrate this with the following theorem.

**Theorem 4.** *If we do not need to satisfy the triangle inequality then we can obtain an arbitrarily large lower bound. No adaptive priority algorithm in the edge model can solve the MST problem with any approximation ratio with the following variations.*

| $\triangle$ | *Fixed Weight* | $n$ | *Complete Graph* | *Stack* | $\mathcal{R}$ *Bound* |
|---|---|---|---|---|---|
| *N* | *Y* | *N* | *Y* | *N* | $\infty$ |

*Proof.* The graph being considered is simply a triangle with two terminal verticies $t_1$ and $t_2$ and one Steiner vertex $s$. The two Steiner edges still have weight one, but now we increase the weight of the terminal edge to $r$. This graph is $P_0$.

- **Case 1:** Since the two steiner edges are symmetric, assume $(t_1, s)$ is considered by an algorithm.
  - **Case 1.1:** If the algorithm accepts the edge, the adversary removes the other two edges. The cost of the algorithm's solution is one whereas the cost of the optimal solution is zero because if the instance contains only one isolated terminal vertex, then the solution requires no edges. Hence, the ratio $\frac{1}{0}$ is arbitrarily large.
  - **Case 1.2:** If the algorithm rejects the edge, the adversary does nothing. Having, rejected one of the Steiner edges, the algorithm is forced to take the terminal edge with weight $r$. The optimal solution still is $1 + 1$. This gives the ratio $\mathcal{R}$ is $\frac{r}{2}$, which becomes arbitrarily large as the weight $r$ increases.
- **Case 2:** Algorithm consider the edge $e_3$.
  - **Case 2.1:** If the algorithm accepts the edge, the adversary does nothing. The ratio is arbitrarily large as in Case 1.2.
  - **Case 2.2:** If the algorithm rejects the edge, the adversary removes the two Steiner edges (and the Steiner vertices). Having rejected the only edge in the input instance, the algorithm fails to come up with a valid solution.

Based on the two cases, the competitive ratio $\mathcal{R}$ is arbitrarily large.

□

With this motivation, we return now to graphs with the triangle inequality. A concern with our above $\frac{4}{3}$ result and Davis and Impagliazzo [10]'s $\frac{5}{4}$ result is that only small graphs, instances with at most four vertices, are considered. Because the lower bound does not consider arbitrarily large graphs, it does not prove that this error is multiplicative. Hence, these results do not rule out the possibility $\mathcal{R} \leq 1 \cdot OPT + c$ for some constant $c$, depending on the maximum weight in the graph but not on the number of vertices in it. This motivates us to want to prove a lower bound with fixed weights, arbitrarily large $n$, and a complete graph.

## 4.4 A $2 - \frac{2}{k}$ Stack Lower Bound

Proving a lower bound with arbitrarily large $n$ turned out to be harder than we expected. In line with what we did in Section 4.3, we first try to prove the result using fixed edge weights. However, it turns out that this gives too much power to the algorithm. At the end of this section we outline a strange and unfair algorithm that uses tricks to learn the input instance and then uses its unbounded computational power to obtain what we conjecture to be a near optimal solution.

**Conjecture 5.** *A priority algorithm in the edge model can achieve a* $1 + O(\frac{1}{k})$ *upper bound for the MST problem with the following variations.*

| $\triangle$ | *Fixed Weight* | $n$ | *Complete Graph* | *Stack* | $\mathcal{R}$ *Bound* |
|---|---|---|---|---|---|
| *Y* | *Y* | *Y/N* | *Y* | *N* | $1 + O(\frac{1}{k})$ |

*The **Y/N** variation for $n$ means that the instance size is arbitrarily large but the exact size is not known to the algorithm.*

This conjecture forced us to attempt to prove a lower bound by varying the weights. However, we were unsuccessful. See the discussion about this in Section 4.2. This remains as an interesting open problem. See Chapter 5.

Borodin, Cashman, and Magen [4] get around the problems of having fixed weights in an arbitrarily large graph by considering incomplete graphs. We follow their lead. In this model, some edges are not included *explicitly* in the input instance and, therefore, an algorithm cannot make decision on them. The edges not included will be called *implicit edges*. Their weights, as dictated by the triangle inequality, will be the length of the shortest explicit path from one of the edge's endpoint to the other.

We, however, found that before the *stack* is introduced, this model has a small glitch making it too favorable for a lower bound. Even if the information about the size of the instances and the weights on edges are provided, an arbitrarily large competitive ratio can be still obtained for the priority model.

**Theorem 6.** *If the priority algorithm is not allowed to accept or reject the implicit edges then we can prove an arbitrarily large lower bound.*

| $\triangle$ | *Fixed Weight* | $n$ | *Complete Graph* | *Stack* | $\mathcal{R}$ *Bound* |
|---|---|---|---|---|---|
| *Y* | *Y* | *Y* | *N/N* | *N* | $\infty$ |

*The **N/N** variation for Complete Graph means that implicit edges cannot be accepted or rejected.*

This result is proved after the proof the the main result for this section. Though this glitch goes away when the stack is introduced, the glitch still motivated us to vary the model yet one more time. In this new version, the algorithm can accept or reject an implicit edge $(u, v)$ any time. This decision however will have to be made without explicitly learning the weight of the edge. This weight can be implied from the transitive closure of the explicit edges, once their weights are learned. The key thing that such an algorithm cannot do is to initially learn about $u$ and $v$ by saying "Give me the most expensive implicit edge and I will reject it." Note that this is exactly what the $1 + O(\frac{1}{k})$ algorithm of Conjecture 5 does. Using this model, we were able to prove a $2 - \frac{2}{k}$ priority lower bound. We, however, will not present this result because we were able to prove another that subsumes it.

Borodin, Cashman, and Magen [4] define a new model referred to as the *adaptive stack model*. They advocate that their model captures reasonable primal-dual algorithms and local-ratio algorithms. The weakness of the adaptive priority model is that the algorithm must make an irrevocable decision on the chosen data item, that is, $d_i \in \sum = \{\textit{accept}, \textit{reject}\}$. The stack model gives the algorithm a second phase in which it can reject previously "accepted" data items.

The ***adaptive stack model*** allows the algorithm to have two phases. The first phase is the same as the adaptive priority algorithms except that the stack algorithm has a stack and a choice set $\Sigma = \{\textit{reject}, \textit{ stack}\}$. The rejected data items are permanently discarded by the algorithm. Instead of accepting a data item, the data items for which $d_i = \textit{stack}$ are pushed on the top of the stack. These data items are reconsidered in the second phase.

The data items that are placed in the stack are processed during the second phase, which

is called the *pop* phase. Each data item is popped in the reverse of the processed order in the first phase. Each time an item is popped, the algorithm *is forced to discard* it if the data items that have been previously been popped and accepted together with those remaining in the stack create a feasible solution. Otherwise, the popped item *is forced to be included* in the algorithm's solution.

Note that the stack model is carefully designed to give the algorithm no control over the second phase. Giving the algorithm this control would allow it to solve even the halting problem in linear time. Recall that, as explained in Section 2.2, the algorithm has unlimited computational power. The only hope for the lower bound is to limit the information the algorithm has before it is forced to make an irrevocable decision. If the algorithm had control over the second phase, then it could read all of the data items during the first phase and push them all onto the stack. Then, knowing the entire input instance, it can use its unlimited power to compute the optimal solution. During the second phase, it could simply accept and reject the data items corresponding to this solution.

Using trick of incomplete graphs and fixed edge weights, Borodin, Cashman, and Magen [4] were able to prove a $\frac{4}{3}$ lower bound within this stack model.

**Theorem 7.** *[4] No stack algorithm in the edge model achieves better than $\frac{4}{3}$ lower bound for the MST problem with the following variations.*

| $\triangle$ | *Fixed Weight* | $n$ | *Complete Graph* | *Stack* | $\mathcal{R}$ *Bound* |
|---|---|---|---|---|---|
| *Y* | *Y* | *Y* | *N/N* | *Y* | $\frac{4}{3}$ |

We were able to tighten their result from $\frac{4}{3}$ to the tight competitive ratio $2 - \frac{2}{k}$.

**Theorem 8.** *No adaptive stack algorithm in the edge model achieves better than* $2 - \frac{2}{k}$ *approximation for the MST problem with the following variations.*

| $\triangle$ | *Fixed Weight* | $n$ | *Complete Graph* | *Stack* | $\mathcal{R}$ *Bound* |
|---|---|---|---|---|---|
| *Y* | *Y* | *Y* | *N/Y* | *Y* | $2 - \frac{2}{k}$ |

*Proof.* Recall that, in the stack model, a popped item is forced to be discarded if the data items that have been previously popped and accepted together with those remaining in the stack create a feasible solution. Hence, our goal is to force the algorithm to stack an expensive solution followed by a cheap solution. When the cheap solution is popped, it is forcefully discarded because there is still a feasible solution, namely the expensive solution, on the stack. Later, on the other hand, the expensive solution will be kept because there is either no more valid solution left on the stack or no previously accepted solution.

A valid solution for the MST problem consists of a set of terminal and Steiner edges that connect the terminal vertices together. Wanting such a solution to be pushed on the stack, the adversary will monitor how well the edges stacked by the algorithm so far connect the terminal vertices. To do this, the adversary partitions the terminal vertices into components that are connected via the stacked edges. Note that $k - c$ "edges" are needed to connect $k$ terminal vertices into $c$ components. (For example, if $c = k$ then $k - c = 0$ edges are needed to keep the $k$ terminal in $k$ isolated components, while if $c = 1$, then $k - 1$ edges are needed to connect them all together.)

Wanting this solution on the stack to be expensive, the adversary wants it to cost the algorithm at least 2 for each "edge" connecting two terminal vertices. There are four types of such connecting "edges". The first type of "edge" is simply a terminal edge $\langle t_g, t_{g'} \rangle$,

which as required costs 2. The second type of "edge" consists of two Steiner edges $\langle t_g, s_q \rangle$ and $\langle s_q, t_{g'} \rangle$, which also costs $1 + 1 = 2$. The third type of "edge" needs to be avoided because it is too inexpensive. This consists of a $k'$-star from one Steiner vertex $s_j$, fanning out with explicit Steiner edges to some $k'$ of the terminal vertices. This acts as $k'$-1 connecting "edges" but only costs $k'$, as explicit Steiner edges only cost 1 each. The adversary will avoid allowing the algorithm to push such stars onto the stack by making the rest of unseen edges incident to $s_j$ implicit as soon as the algorithm pushes two explicit edges incident to $s_j$. The final type of "edge" involves the algorithm accepting an implicit Steiner edge. However, these edges cost 3. We will let $r, b,$ and $b'$ denote the number of "edges" of the first, second, and fourth type. The loop invariant maintains that $r + b + b' \geq k - c$ which reconfirms that the number of these "edges" is at least the $k - c$ needed to connect the terminal vertices into $c$ connected components. Having enough intuition, we are now ready to prove the theorem.

There will be $k$ terminal vertices and $h \geq k$ Steiner vertices for some constant $a$. These informations are known to an algorithm. Each explicit Steiner edge has weight 1. Each implicit Steiner edge has weight 3. And, each terminal edge will be always explicit and will have weight 2. An adversary initially constructs $P_0$ narrowing down the universe of all possible data items to include all possible terminal edges and all possible explicit Steiner edges $\langle t_g, s_j, 1 \rangle$ for each terminal vertex $t_g$ and each Steiner vertex $s_j$. If, later, the adversary deletes $\langle t_g, s_j, 1 \rangle$ from $P_i$, then the deleted explicit edge becomes an implicit edge. The initial adversarial set $P_0$ is shown in Figure 4.5.

Recall that $P_{i-1}$ is the set of edges that the adversary still considers to be part of the

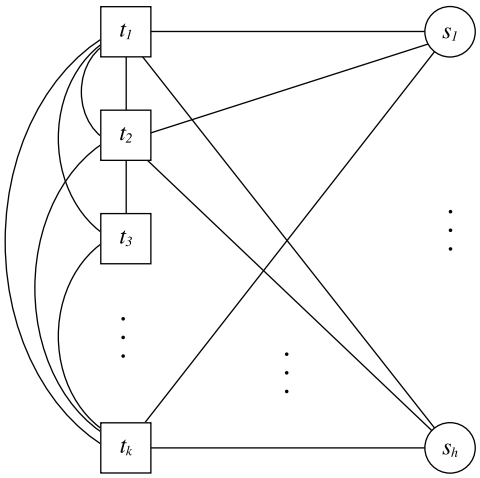


Figure 4.5: A possible situation at the beginning of the $i^{th}$ iteration.

input instance. We call a stacked edge *necessary* if it does not create a cycle with previously stacked edges. Otherwise, it is called an *unnecessary* edge. In order to prove the theorem, we first prove that the adversary is able to maintain the following loop invariants.

$LI_1$: Either $S_1$ or $S_2$ applies to each Steiner vertex $s$ in $P_{i-1}$.

- $S_1$: At most one necessary explicit Steiner edge incident to $s$ has been stacked by the algorithm.
- $S_2$: Exactly two necessary explicit Steiner edges are incident to $s$ and each of the rest of edges incident to $s$ has either been rejected, stacked as unnecessary edge, or removed by the adversary causing it to be implicit.

$LI_2$: A relation $r + b + b' \geq k - c$ holds. (The $r$ denotes the number of necessary stacked terminal edges. The $b$ denotes the number of Steiner vertices of $S_2$ type. The $b'$ denotes the number of accepted necessary implicit Steiner edges. And,$C_1, \ldots, C_c$ are

partitions of the $k$ terminal vertices into connected components where two terminal vertices $t_g$ and $t_{g'}$ are connected if the stacked terminal and Steiner edges form a path between them. The $c$ denotes the number of these components.

We prove the loop invariants are maintained as follows. Initially, no edge has been seen. Therefore, $S_1$ applies to all Steiner vertices, and $r = b = b' = 0$ and $c = k$, which results in $r + b + b' \geq k - c$. Therefore, the base case clearly holds.

Now assume that the loop invariants hold at the beginning of $i^{th}$ iteration. The Figure 4.6 depicts a possible adversarial set $P_{i-1}$.

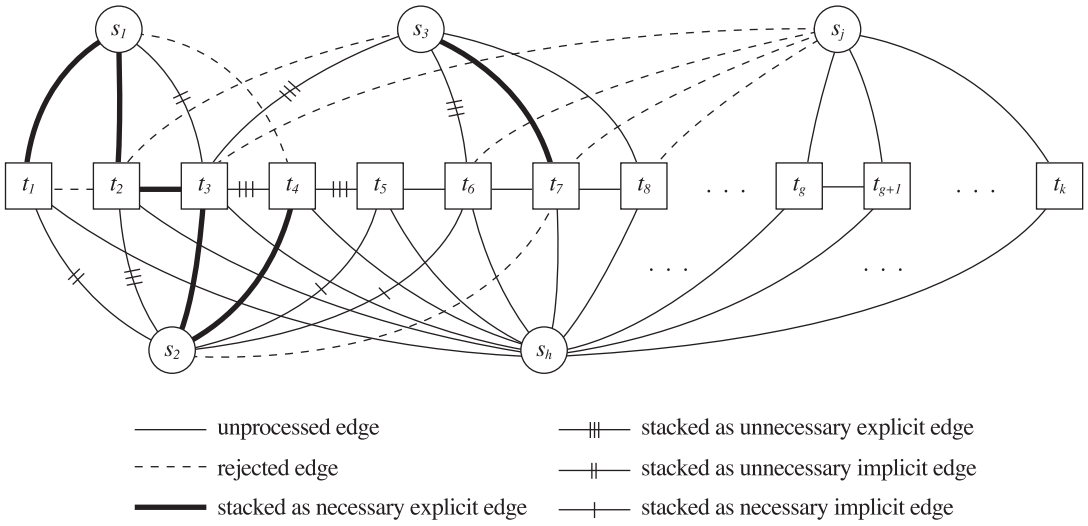


Figure 4.6: A possible situation at the beginning of the $i^{th}$ iteration.

Imagine that an algorithm considers an edge. No matter what type it is if it is rejected by the algorithm nothing changes. Hence, we consider only acceptance cases.

- **Case 1:** The algorithm stacks a necessary terminal edge $a_{\pi_i} = (t_g, t'_g)$. The adversary, in turn, does nothing, yielding $P_i = P_{i-1}$. Since $a_{\pi_i}$ is a terminal edge, it does

not affect $S_1$ or $S_2$. The $c$ decreases by one, $r$ increases by one, and $b$ and $b'$ are unchanged. Hence, the relation $r + b + b' \geq k - c$ still holds.

- **Case 2:** The algorithm stacks the explicit Steiner edge $a_{\pi_i} = (t_g, s_j)$, which is the first stacked edge incident to $s_j$. The adversary does nothing, yielding $P_i = P_{i-1}$. Because there was an unseen explicit Steiner edge incident to $s_j$, $s_j$ must have been of type $S_1$ and must remain of type $S_1$. Hence $b$ is unchanged. Because this new edge is the first stacked edge incident to $s_j$, it cannot create a cycle and hence is necessary. Also, it cannot merge two connected components and hence $c$ is also unchanged. In addition, $r$ and $b'$ are unchanged. Therefore, the relation $r + b + b' \geq k - 1$ holds.
- **Case 3:** The algorithm stacks the second necessary and explicit Steiner edge $a_{\pi_i} = (t_g, s_j)$ incident to Steiner vertex $s_j$. The adversary, in turn, removes the rest of the unseen edges incident to $s_j$, yielding $P_i \subset P_{i-1}$. These removed edges become implicit edges. Moreover, $S_2$ now applies to $s_j$ instead of $S_1$, increasing $b$ by one. Since $a_{\pi_i}$ is necessary, it does not create a cycle with previously stacked edges and hence $c$ decreases by 1. The $r$ and $b'$ are unchanged. The relation $r + b + b' \geq k - c$ is true.
- **Case 4:** The algorithm considers an edge which creates a cycle with stacked edges. By definition, such an edge is called unnecessary. Whether the algorithm decide to stack or to reject it, the adversary does nothing, yielding $P_i = P_{i-1}$. Note $b$ does not change because the definition of $S_1$ and $S_2$ considers only necessary edges. The $c$ does not change because the new edge creates a cycle. The $r$ or $b'$ may increase by

one. Hence, the relation $r + b + b' \geq k - c$ still holds.

- **Case 5:** The algorithm stacks a necessary implicit Steiner edge $(t_{g'}, s_j)$ without actually seeing it. If this edge is still in $P_{i-1}$ then the adversary can be kind by informing the algorithm that actually this edge is explicit in the actual input instance. It is just that the algorithm has not yet read this part of the input. This case can be handled by Case 2, 3, or 4. If the edge being accepted is not in $P_{i-1}$, the adversary kindly informs the algorithm that this is in fact an implicit edge and its weight is 3. In latter case, the adversary in turn does nothing, yielding $P_i = P_{i-1}$. Since the edge is implicit, it does not affect $S_1$ vertices. The $c$ decreases by one, $b'$ increases by one, and $r$ and $b$ are unchanged. Hence, the relation is still true.

Based on the five cases above, we conclude that the loop invariants are maintained.

When $c$ becomes one, the adversary knows that the terminal vertices are connected by the stacked edges into one component. Hence, he knows that there is an expensive solution pushed on the stack. The adversary at this point stops this first phase of the game. He declares to the algorithm that the actual input instance consists of the edges seen by the algorithm so far together with all the edges remaining in $P_i$. The next task is to prove that this instance contains a cheap optimal solution. The first step to accomplish this is to note that at most $k - 1$ Steiner vertices of type $S_2$ definitely connect the $k$ terminal vertices into one component and hence, we know that the number of such Steiner vertices cannot exceed this number. Because there are $k$ Steiner vertices, the loop invariant ensures us that there is at least one Steiner vertex remaining of type $S_1$. All the edges incident to such a vertex

have not ever been deleted from $P_i$ and hence all appear explicitly in the input. This forms a $k$-star which in itself is an optimal solution of weight $k$.

At the end of this first stage of the game, the algorithm still must make decisions about the remaining data items in the input instance (which according to the adversary will be those edges remaining in $P_i$). The algorithm may decide to stack them or decide to reject them. It does not actually matter to the adversary, because either way these edges will be rejected at the very beginning of the pop stage of the algorithm. The reason for this is that the adversary was careful that there is an expensive solution previously pushed on the stack and hence these remaining edges in $P_i$ are not needed.

As the rest of the pop stage of the algorithm proceeds, we need to prove which of the algorithm's stacked edges get forcefully accepted and which get forcefully rejected. This happens in reverse of the pushed order. Any edge labeled as unnecessary will be rejected when it is popped. It formed a cycle with the edges already pushed on the stack when it got pushed on and hence it will form the same cycle with the edges still on the stack when it is popped. The one necessary Steiner edge incident to a Steiner vertex of type $S_1$ will also be rejected. All other edges incident to this Steiner vertex either was initially rejected and not stacked or has already been popped and rejected by the pop stage. Hence, this edge is not a part of a remaining solution. What will get forcefully accepted when popped will be the "edges" forming the expensive solution. As they got stacked by the algorithm these were needed to merge the connected components of the terminal vertices together and hence are all needed for the solution. By definition, $r$ denotes the number of these necessary stacked terminal edges, $b$ denotes the number of Steiner vertices of $S_2$ type, and the $b'$ denotes the

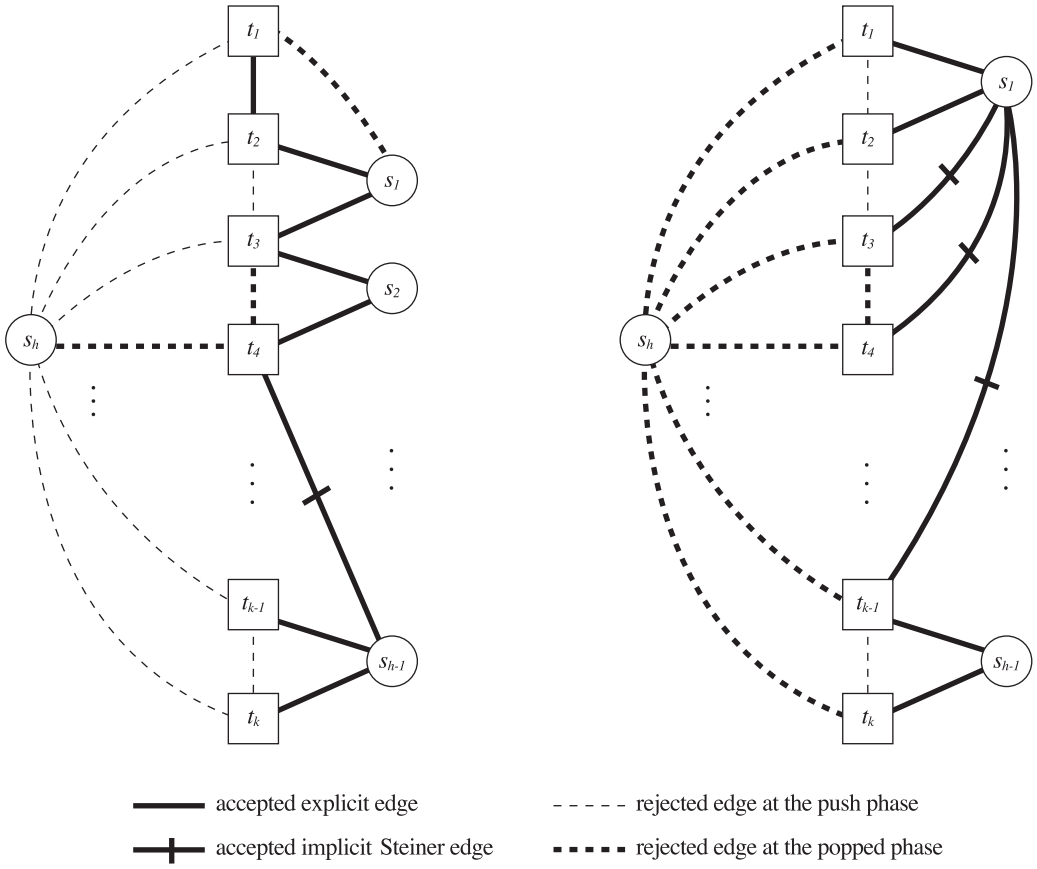


Figure 4.7: Two possible final instances of the newly defined game. The optimal solution is the $k$-star incident to $s_h$. Since there is another valid solution stacked beneath the $k$-star, the algorithm cannot accept the $k$-star.

number of accepted necessary implicit Steiner edges. Each terminal edge has weight 2. There are two Steiner edges incident to each Steiner vertex of type $S_2$, each with weight 1. Each implicit Steiner edges incident has weight 3. Hence the cost of this popped and accepted solution is $2 \cdot r + (1 + 1) \cdot b + 3 \cdot b' \geq 2(r + b + b')$ which by the loop invariant is at least $2(k - c)$ and by the termination condition is $2(k - 1)$. Above we showed how the cost of the optimal solution is only $k$. Hence, we can summarize the competitive ratio

to be $\mathcal{R} = \frac{ALG}{OPT} \geq \frac{2(k-1)}{k} = 2 - \frac{2}{k}$.

□

Before we close this section, two issues are discussed: 1) what problems arise when the implicit edges cannot be accepted and 2) what problems arise when all the edges in the complete graph must be explicit and the edge weights are fixed.

*Proof of Theorem 6.* Not allowing the implicit edges to be accepted was referred above as a glitch. When the model has a stack, this does not seem to be an issue. However, without a stack and without this ability, we can prove an arbitrarily large lower bound. The main changes to the proof of Theorem 8 is that all the terminal edges are implicit and hence can't be accepted. (Which many may say is unfair.) Not being able to accept implicit edges, the algorithm must find a solution using only Steiner edges, ideally the optimal solution consisting of the $k$-star. The second change to the proof is that the number $h$ of Steiner vertices will be increased to be much larger than $k$. To find the $k$-star it must consider each of these Steiner vertices, accepting one of their incident edges. Because the algorithm does not have a stack, we no longer need that these initial accepted edges form a valid solution. Hence, as soon as the algorithm accept one Steiner edge incident to $s_j$, the adversary deletes the rest of unseen edges incident to $s_j$. This causes the degree of $s_j$ in the actual input instance to be one. Accepting this one edge to $s_j$ was a costly mistake for the algorithm because this edge goes no where. The game terminates when the algorithm has done this for all but one of the Steiner vertices. Then the adversary allows the algorithm to accept the $k$-star incident to the remaining Steiner vertex. Hence, the competitive ratio is

$\mathcal{R} = \frac{ALG}{OPT} = \frac{(h-1)+k}{k}$ which can be arbitrarily large by making $h$ arbitrary large.

Having many Steiner vertices is not a problem when algorithm is allowed to accept the terminal edges, because it can get a competitive ratio of 2 simply by finding a minimum spanning tree on the terminal edges.

Not allowing the algorithm to accept or reject the implicit edges does not seem to be a problem when the algorithm has a stack, because all of these Steiner edges that lead no where will get rejected when they are popped off the stack.

□

We now consider what problems arise when all the edges in the complete graph must be explicit and the edge weights are fixed.

*Proof Sketch for Conjecture 5* Above we conjectured that a priority algorithm in the edge model can achieve a $1 + O(\frac{1}{k})$ upper bound for the MST problem in this case. Before discussing how this algorithm might work for a general input instance, let us see how an algorithm can trivially find the optimal $k$-star in the input instance used in the proof of Theorem 8. Suppose that the algorithm does not know either the number of terminal or Steiner vertices and does not know the weights of specific edges, but does know that the terminal edges have weight 2 and that Steiner edges either have weight 1 or 3. The algorithm with this information will start by asking for and rejecting all edges with weight 3. After doing this, it completely knows the input instance. If as in Theorem 8 one of the Steiner vertices is the root of a star with edges of weight 1 to each of the terminal vertices, then the algorithm will know which Steiner vertex $s_j$ this is because it is the one for which no incident edges have been rejected. Even if the optimal solution is not so

obvious because it is some complex subset of the terminal edges and the Steiner edges of weight 1, the algorithm can uses its unbounded computational power to obtain the optimal solution. A difficulty arises if the input instance has edges with many different weights and if the optimal solution contains a few of the more expensive weighted edges. The algorithm would not want to learn the what the graph is by asking for and rejecting the expensive edges only to learn that some of these rejected edges are needed for the optimal solution. However, to learn that a vertex is in the input instance, the algorithm only needs to see one edge incident to it. This can be the edge's most expensive incident edge. An algorithm certainly is able to ask for and reject the most expensive edge incident to each vertex. One has to be a little careful the one does not disconnect the graph. But that aside, we conjecture that there exists and nearly optimal solution not containing any of these rejected edges. The algorithm uses its unbounded computational power to find such a near optimal solution.

□

# 5 Summary and Future Directions

This chapter summarizes the results and states open problems. Modeling the priority algorithm and the stack algorithm is a recent achievement. Many $NP$-hard optimization problems still remain open to be formalized. The MAS problem and the MST problem are in the class MAX-SNP which is defined in Papadimitriou and Yannakakis [14], therefore, no polynomial-time algorithm achieves $1 + \epsilon$ approximation ratio unless $P = NP$.

We proved a $2 - \frac{2}{k}$ priority lower bound for the Maximum Acyclic Subgraph problem in the edge model. We also showed that the Minimum Feedback Arc-set problem has no priority algorithm that achieves any approximation ratio. It is worth noting that the complement problem of a particular one is harder in context of greedy or greedy-like algorithm paradigms.

For the MST problem, we started by improving the $\frac{5}{4}$ priority lower bound of Davis and Impagliazzo [10] to $\frac{4}{3}$ for the MST problem when the graph is small and the edge weights are known. See Section 4.3. We were unable to generalize this lower bound for an arbitrarily large graphs. In fact, we conjecture that if the weights on edges are known then there is a $1 + O(\frac{1}{k})$ upper bound. On the other hand, we were able to prove a $2 - \frac{2}{k}$ lower bound by weakening the result by considering incomplete graphs. This result was

strengthened by extending it to the stack model of Borodin, Cashman, and Magen [4]. See Section 4.4.

Though our $2 - \frac{2}{k}$ lower bound for the Maximum Acyclic Subgraph problem matches the competitive ratio of 2 achieved by the upper bound, there is a disconnection. Our lower bound is within the edge model that does not let the algorithm have degree information, while the upper bound requires such degree information. It is open to prove the same lower bound either in the vertex model or in the edge model in which the data items have vertex degree and the edge information. It is also possible that a lower bound higher than 2 could be proved when the algorithm does not have this degree information. See below table.

| | Upper Bound | Lower Bound |
|---|---|---|
| Edge Model without degree | ? | 2 |
| Edge Model with degree | 2 | ? |
| Vertex Model | 2 | ? |

For the Minimum Steiner Tree problem, it is completely open to prove a lower bound when the input instance is arbitrarily large and the graph is complete. We have seen that this would required varying the edge weights. This will ensure that the error of the algorithm's solution is indeed multiplicative as size of the instance grows.

In general, one can continue the oscillating journey between an upper and lower bound by tightening the gap or strengthening the model.

# Bibliography

[1] M. ALEKHNOVICH, A. BORODIN, J. BURESH-OPPENHEIM, R. IMPAGLIAZZO, A. MAGEN, AND T. PITASSI (2005) Toward a model for backtracking and dynamic programming, *IEEE Conference on Computational Complexity*.

[2] S. ANGELOPOULOS AND A. BORODIN (2002) On the power of priority algorithms for facility location and set cover, *5th International Workshop on Approximation Algorithms for Combinatorial Optimization*.

[3] B. BERGER AND P. SHOR, (1997) Tight bounds for the maximum acyclic subgraph problem, Journal of Algorithms 25(1), 1–18.

[4] A. BORODIN, D. CASHMAN, AND A. MAGEN, (2005) How well can primal-dual and local-ratio algorithms perform?, *International Colloquium on Automata, Languages and Programming (ICALP)*.

[5] A. BORODIN AND R. EL-YANIV (1998). *Online Computation and Competitive Analysis*. Cambridge University Press.

[6] A. BORODIN, J. BOYAR, AND K. LARSEN, (2002) Priority algorithms for graph optimization problems, *Proceedings of the 2nd Workshop on Approximation and Online Algorithms*.

[7] A. BORODIN, M. NIELSEN, AND C. RACKOFF, (2002) (Incremental) Priority algorithms, *Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms*.

[8] D. CASHMAN, (2005) Approximate Truthful Mechanisms for the knapsack problem, and negative results using a stack model for local ratio algorithms, MSc Thesis, University of Toronto.

[9] S. DAVIS, (2008) On the Power of the Basic Algorithmic Design Paradigms, Ph.D. Dissertation, University of California, San Diego.

[10] S. DAVIS AND R. IMPAGLIAZZO (2004) Models of greedy algorithms for graph problems, *Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms*.

[11] JACK EDMONDS, (1971) Matroids and the greedy algorithm, Mathematical Programming 1, 127–136.

[12] JEFF EDMONDS (2008). *How to Think About Algorithms*. Cambridge University Press.

[13] S. L. HORN, (2004) One-pass algorithms with revocable acceptances for job interval selection, MSc Thesis, University of Toronto.

[14] C. PAPADIMITRIOU AND M. YANNAKAKIS, (1991) Optimization, approximation and complexity classes, Journal of Computer and System Science 43, 770–779.

[15] G. ROBINS AND A. ZELIKOVSKY (2000) Improved Steiner tree approximation in graphs, *Proceedings of the 11th Annual ACM-SIAM Symposium on Discrete Algorithms*.

[16] V. VAZIRANI (2001). *Approximation Algorithms*. Springer.

[17] P. WINTER, Steiner problem in networks: a survey, *Networks 17*, 129–167.

[18] C. GRÖPL, S. HOUGARDY, T. NIERHOFF, AND H. PRÖMEL, Approximation Algorithms for the Steiner Tree Problems in Graphs, *Stiner Trees in Industries*, editors: D. Z. Du and X. Cheng, Kluwer.