

Priority and More

By “the people”

April 10, 2009

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. Furthermore, 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.

Because of its simplicity and efficiency on many computational problems, the greedy algorithm paradigm is one of most important tool in designing algorithms. Recently, Borodin, Nielsen and Rackoff, [3], 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 also 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. The paper of Allan Borodin, David Cashman, and Avner Magen [2] formalized a formal computational model that captures both the primal-dual and the local ratio approaches. This model is called a *stack algorithm*.

1.2 Related Work and Our Results

We present formal definition of priority algorithms and a relatively simple application to the Maximum Acyclic Subgraph (MAS) problem in section 2.

In section 3, lower bounds for the Minimum Steiner Tree (MST) problem are presented for some combinations of possible variations to either the input instances or the computation model. Davis and Impagliazzo [5] 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. Studying their cases further, we managed to improve their lower bound to $\frac{4}{3}$. See Theorem 3. We were also able to show arbitrarily high competitive ratio if a priority algorithm is not allowed to include the “implicit” edges in Theorem 5. Borodin, Cashman, Magen [2] proved a $\frac{4}{3}$ lower bound in stack model. See Theorem 6. We improve this result to $2 - \frac{2}{k}$ in Theorem 7. At last, we prove a separation between priority algorithm and the stack algorithm through two restricted versions of the MST problem.

In section 4, we suggest some possible directions for the future work.

2 Preliminaries

2.1 Greedy and Priority Algorithms

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 node, 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, Σ . This choice set, Σ , 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 *node model* or *edge model* can be used. The data items in the node model correspond to the nodes of the instance graph. More specifically, the data items contain certain information about the node in question. Depending on the specifics of the model, this information might include a node 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.

Two classes of priority algorithms are defined in [3], 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 description of adaptive priority model is shown in algorithm APA.

algorithm APA

<input>: $I = \{a_1, \dots, a_n\}$

<output>: $S = \{(a_i, d_i) | 1 \leq i \leq n\}$

begin

1: Loop: $i = 1, 2, \dots, n$

2: Assume the data items $\{a_{\pi_1}, a_{\pi_2}, \dots, a_{\pi_{i-1}}\}$ have been seen and processed. Based on the data items seen so far, choose a possible new ordering Π on all possible data items. Let a_{π_i} be the next data item according to this new order. Make an irrevocable decision d_{π_i} about the data item a_{π_i} .

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

4: end Loop

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

end algorithm

2.2 Priority Model as a Game and its application

The priority model can be described as a game between an adversary and an algorithm. At the beginning of the i^{th} iteration of the game, the algorithm has been given the partial instance $PI_{i-1} = \langle a_1, a_2, \dots, a_{i-1} \rangle$ and has committed to the partial solution $PS_{i-1} = \langle d_1, d_2, \dots, d_{i-1} \rangle$. 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 $I = \{a_1, a_2, \dots, a_{i-1}, a_i, \dots, a_n\}$, where a_1, \dots, a_{i-1} are as stated in the partial solution PI_{i-1} , a_i, \dots, 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 $PI_{i-1} = \langle a_1, \dots, a_{i-1} \rangle$, committed to the decisions $PS_{i-1} = \langle d_1, \dots, d_{i-1} \rangle$, 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_{π_i} . Hence, the game is simplified by allowing the algorithm to choose a_{π_i} from P_{i-1} and make a decision d_i on a_{π_i} . Knowing a_{π_i} and d_{π_i} , the adversary is allowed to narrow down P_{i-1} to P_i by removing some data items (including a_{π_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 $\langle a_{\pi_1}, a_{\pi_2}, \dots, a_{\pi_i} \rangle$ and made decisions $\langle d_1, \dots, d_i \rangle$ 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.

We will prove a relatively easy adaptive priority lower bound for demonstration purpose. 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. We have been unsuccessful to prove a matching lower bound of 2 both in the node model and in the edge model with degrees. Instead, we present a lower bound without degree information in edge model.

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

Proof of Theorem 1: 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 simple $(k+1)$ -complete graph. Let us assume the nodes are indexed from 1 to $k+1$. 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.

Case 1: If the algorithm accepts the edge $\langle 1, 2 \rangle$, then, the adversary reduces P_0 down to P_1 by removing all edges except $\langle 2, i \rangle$ and $\langle j, 1 \rangle$. In fact, the adversary declares at this point that this is the final input instance. For each $j \in [3, 4, \dots, k+1]$, 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 $j \in [3, 4, \dots, k+1]$ 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 $\mathcal{R} = \frac{OPT}{ALG} = \frac{2k-1}{k} = 2 - \frac{1}{k}$.

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 $\mathcal{R} = \frac{1}{0} \approx \infty$. ■

3 Minimum Steiner Tree (MST) Problem

We study Minimum Steiner Tree (MST) Problem in priority and stack algorithm frameworks. In Section 3.1, the problem background is explained. In Section 3.2, some modeling issues are discussed. In Section 3.3, we prove an arbitrary high and a $\frac{4}{3}$ priority lower bound for the MST problem under different variations. In Section 3.4, 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 [2]. Furthermore, we prove a $2 - \frac{2}{k}$ stack lower bound improving the result $\frac{4}{3}$ lower bound in [2]. At last we prove a separation between priority algorithm and stack algorithm considering two variations of the MST problem.

3.1 Basis for the MST problem

An instance of the MST problem consists of a graph $G = (V, E)$, a nonnegative real-valued cost $c : E \rightarrow R^+$ on the edges and a set $N \subseteq V$ of *terminal nodes*. A tree T of G is called a *Steiner tree* if it spans all the terminal nodes of G . The problem is to find a Steiner tree T whose cost $\sum_{ij \in E(T)} c(ij)$ is minimum. The $V \setminus N$ are called *Steiner nodes*. An edge $e = \{u, t\}$ is called *Steiner edge* if both u and t are Steiner nodes. An edge $e' = \{t_g, t_{g+1}\}$ is called a *terminal edge* if the two endpoints, t_g and t_{g+1} , are both terminal nodes. We denote *k-star* for k Steiner edges incident to the same Steiner node. At last, we denote $w(e)$ for the weight of an edge e .

The problem has practical applications such as routing VLSI layout, the design of communication networks, and accounting and billing for the use of telephone networks [9]. Unfortunately, the problem is *NP-hard*, therefore, designing an approximation algorithm seems a good and profitable direction. According to [5], the best approximation algorithm achieves 1.55 performance guarantee, but it neither fits into a priority model nor a stack model. The best known priority algorithm, shown in Vazirani [8], 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 nodes in the given instance graph.

3.2 Issues on Variations of Instances and Model

We consider a number of variations in the MST problem and the model in this section.

First, with unconstrained edge weights, we will prove unbounded inapproximability within priority framework. Hence, we assume the triangle inequality (Δ) to hold, namely $w(\{u, v\}) \leq w(\{u, w\}) + w(\{w, v\})$.

Our second variation is whether the adversary is allowed to vary the weights on the edges or must fix them (**FW**). 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.

Our third variation (**n**) is whether the lower bound is proved with a fixed instance size (having some small number of nodes) without giving the algorithm the instance size or whether a stronger result can be proved in arbitrarily large graphs or in a condition that the size information is given to the algorithm.

Our fourth variation is whether the input instance must be a complete graph or not (**CG**). If yes, then there must be an explicit data item in the instance, providing the weight between every possible pair of nodes (except those between Steiner nodes). 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. 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 because the final instance graph must be complete, it must either delete all the edges incident to the node 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 node u (or v)*.

3.3 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 [5] 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 [5]) *No adaptive priority algorithm in the edge model can achieve better than $\frac{5}{4}$ approximation for the MST problem with the following variations.*

Δ	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.*

Δ	Fixed Weight	n	Complete Graph	Stack	\mathcal{R} Bound
Y	Y	N	Y	N	$\frac{4}{3}$

Proof of Theorem 3: The adversary initially constructs a set P_0 shown in Figure 1.

Case 1: Since the three Steiner edges are identical initially, without loss of generality, assume that it considers considers the edge $a_{\pi_1} = \{t_1, s_1\}$, made bold in Figure 1. If the algorithm accepts the edge, the adversary responds by removing all the remaining edges, shown as P_0^1 in Figure ???. The remaining graph will be the final input instance. It is a complete graph

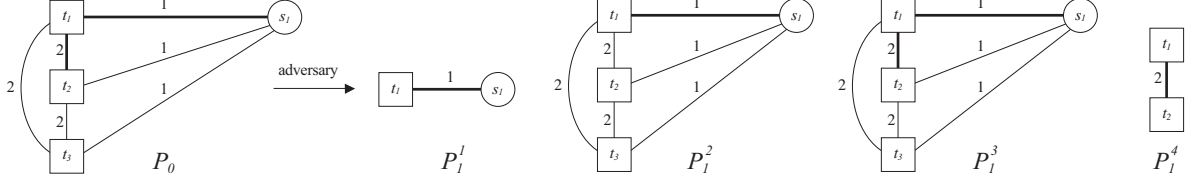


Figure 1: The adversarial set P_0 and a bold edge is assumed to be considered first by the algorithm. Each right of P_0 is the second adversarial set $P_1 \subseteq P_0$ reduced by the adversary. The squared nodes are terminals and the circled ones are Steiners.

as promised. The algorithm should not have taken the bold edge. The approximation ratio is $\mathcal{R} = \frac{ALG}{OPT} = \frac{1}{0} \approx \infty$. If the algorithm rejects the bold edge, the adversary gives the entire graph as the instance, shown as P_1^2 in Figure 1. 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 $\mathcal{R} = \frac{4}{3}$.

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 $a_{\pi_1} = \{t_1, t_2\}$ in Figure 1. If the algorithm accepts the edge, the adversary does nothing as shown P_1^3 in Figure 1. 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 $\mathcal{R} = \frac{4}{3}$. If the algorithm rejects the edge, the adversary removes all the remaining edges, shown as P_1^4 in Figure 1. In this case, the algorithm even fails to find a valid Steiner tree. ■

The reader may wonder if one can get a better lower bound by having much bigger weights on the terminal edges. This, however, 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	∞

Proof of Theorem 4: The graph being considered is simply a triangle with two terminal vertices t_1 and t_2 and one Steiner node 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 node, 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 nodes). Having rejected the only edge in the input instance, the algorithm fails to come up with a valid solution. ■

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 [5]’s $\frac{5}{4}$ result is that only small graphs, instances with at most four nodes, 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 $\mathcal{R} \leq 1 \cdot \text{OPT} + c$ for some constant c , depending on the maximum weight in the graph but not on the number of nodes in it. This motivates us to want to prove a lower bound with fixed weights, arbitrarily large n , and a complete graph.

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

Proving a lower bound with arbitrarily large n turned out to be harder than we expected. It seems 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 1 *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.

Borodin, Cashman, and Magen [2] get around the problems of having fixed weights in an arbitrarily large graph by considering incomplete graphs. In their 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. However, we found that the model is too favorable to lower bound.

Theorem 5 *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	∞

The **N/N** variation for Complete Graph means that neither the instances are complete graphs nor the implicit edges are visible to algorithm.

This result is proved after the proof the the main result for this section. Though this issue goes away when the stack is introduced, it 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 1 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. See Theorem 7.

Borodin, Cashman, and Magen [2] 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, $d_i \in \Sigma = \{\text{accept}, \text{reject}\}$. 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 $\Sigma = \{\text{reject}, \text{stack}\}$. The rejected data items are permanently discarded by the algorithm, whereas stacked items 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* or *clean up* 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. 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 [2] were able to prove a $\frac{4}{3}$ lower bound within this stack model. See Theorem 6. We improve this ratio to $2 - \frac{2}{k}$ in Theorem 7.

Theorem 6 [2] *No stack algorithm in the edge model achieves better than $\frac{4}{3}$ lower bound for the MST problem with the following variations.*

Δ	Fixed Weight	n	Complete Graph	Stack	\mathcal{R} Bound
Y	Y	Y	N/N	Y	$\frac{4}{3}$

Theorem 7 *No adaptive stack algorithm in the edge model achieves better than $2 - \frac{2}{k}$ approximation for the MST problem with the following variations.*

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

Proof of Theorem 7: 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 edges that connect the terminal nodes 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 nodes. To do this, the adversary partitions the terminal nodes into components that are connected via the stacked edges. Note that $k - c$ “edges” are needed to connect k terminal nodes into c components.

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 nodes. 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 l -star from one Steiner node s_j , fanning out with explicit Steiner edges to some k' of the terminal nodes. This acts as $l - 1$ connecting “edges” but only costs l , 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 accepted “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 nodes into c connected components. Having enough intuition, we are now ready to prove the theorem.

There will be k terminal nodes and $h \geq k$ Steiner nodes for some constant h . These informations are known to an algorithm. All terminal nodes are connected each other and each Steiner node has connection to all terminal nodes. Each explicit Steiner edge has weight 1 whereas each implicit Steiner edge has weight 3. And, each terminal edge will be always explicit and has weight 2. An adversary initially constructs P_0 narrowing down the universe of all possible data items to all possible terminal edges and all possible explicit Steiner edges $\{t_g, s_j\}$. If, later, the adversary deletes $\{t_g, s_j\}$ from P_i , then the deleted explicit edge becomes implicit.

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

We prove the loop invariants are maintained as follows. Initially, no edge has been seen. Therefore, S_1 applies to all Steiner nodes, 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 2 depicts a possible adversarial set P_{i-1} .

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

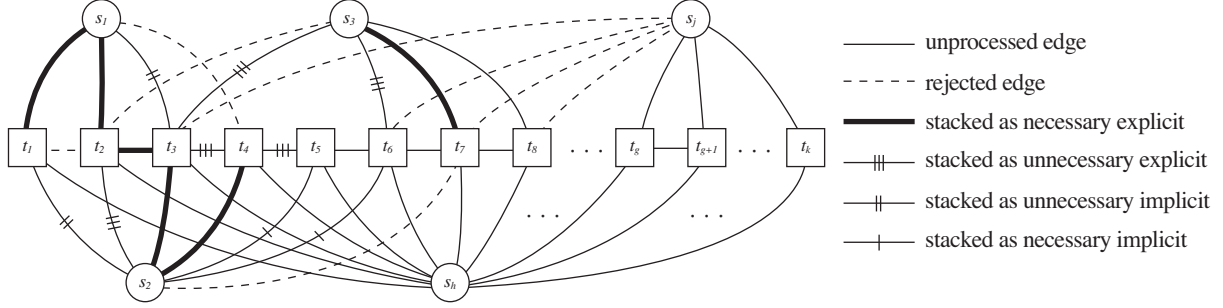


Figure 2: A possible situation at the beginning of the i^{th} iteration.

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_{π_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 an 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 $P_i = P_{i-1}$. The s_j must have been of type S_1 and remains to be type S_1 . Hence b, b', r and c are unchanged and, of course, $r + b + b' \geq k - 1$ holds.

Case 3: The algorithm stacks the second necessary and explicit Steiner edge $a_{\pi_i} = \{t_g, s_j\}$ incident to Steiner node 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_{π_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. Hence, 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 and c does not change, but 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 inform 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 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 nodes. 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 1, the adversary knows that the terminal nodes 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 nodes of type S_2 connect the k terminal nodes into one component and hence, we know that the number of such Steiner nodes cannot exceed this number. Because there are k Steiner nodes, the loop invariant ensures us that there is at least one Steiner node remaining of type S_1 . All the edges incident to such a node 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 node of type S_1 will also be rejected. All other edges incident to this Steiner node 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 nodes together and hence are all needed for the solution. Each terminal edge has weight 2. There are two Steiner edges incident to each Steiner node 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 5: Not allowing the implicit edges to be accepted was referred above as an issue. When the model has a stack, this does not seem to be a matter. However, without a stack and without this ability, we can prove an arbitrarily large lower bound. The main changes to the proof of Theorem 7 is that all the terminal edges are implicit and hence can't be accepted. 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 nodes will be increased to be much larger than k . To find the k -star it must consider each of these Steiner nodes, 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 nodes. Then the adversary allows the algorithm to accept the k -star incident to the remaining Steiner node. 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. ■

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 1 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 7. Suppose that the algorithm does not know either the number of terminal or Steiner nodes 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 7 one of the Steiner nodes is the root of a star with edges of weight 1 to each of the terminal nodes, then the algorithm will know which Steiner node 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 use 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 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 node 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 node. One has to be a little careful the one does not disconnect the graph. But that aside, we conjecture that there exists a nearly optimal solution not containing any of these rejected edges. The algorithm uses its unbounded computational power to find such a near optimal solution. ■

3.5 Clean Up Phase on the MST

In this section, we prove a separation between priority algorithm and the stack algorithm through a restricted version of the MST problem. We also improve the result in another version of the MST problem where we allow each data item of the priority algorithm to have more local information. We call this large data item model. At last, we show that the popup phase is not necessary if a priority algorithm is allowed to keep two solution sets.

The first restricted version is same as the MST problem except that there is no edge between any two terminal nodes, that there is no edge between any two Steiner nodes, and that all Steiner edges, where each edge has one Steiner node and one terminal node as endpoints, have same weights. Let us name this version MST-I and assume k is the number of the terminal nodes and h is the number of the Steiner nodes.

Theorem 8 *No priority algorithm in the edge model can achieve a constant approximation ratio for the MST-I problem.*

Proof of Theorem 8: Every time the alg accepts an edge, the adversary deletes all other unseen edges adjacent to this steiner node. That is, if the algorithm rejected d edges adjacent to steiner node s and then accepted one adjacent to it, then this node s will have degree $d + 1$ and the one accepted is useless. The last Steiner node considered has an edge to each terminal node. Hence, $\mathcal{R} = \text{Alg}/\text{Opt} = [(h - 1) + k]/[k]$.

Theorem 9 *A trivial algorithm, Online Alg, with a second pass achieves the competitive ratio 2 for the MST-I problem.*

Proof of Theorem 9: The algorithm has the following strategy. (Stage 1:) Consider edges in any (online) order. “Stack” the next edge as long as it does not create a cycle. (Stage 2:) If you have only stacked one edge adjacent to a Steiner node s , then delete that edge. At the end, the algorithm has accepted at most $2(k - 1)$ edges whereas the optimal solution includes at least k edges. The worst case is when every steiner node (except one that has k cross edges) has degree 2 (or zero). Note that the algoirthm in Theorem 9 can be understood within the Stack model. ■

We now improve the above result in two ways considering another version of the MST problem. We prove that a priority algorithm in the large data item model is not helpful whereas the two phase algorithm (stack algorithm) gets the optimal solution; and we show that the popup phase is replaceable by allowing the priority algorithm to have two solution sets. A problem with this approach is that we consider a promise problem.

Theorem 10 *There is a version of the Steiner tree problem (or st -connectivity) that*

- A. a priority algorithm cannot achieve a bounded competitive ratio with data item size $q = o(\sqrt{n})$.*
- B. a stack algorithm can optimally solve it with data item size $q = 1$.*
- C. a priority algorithm keeping two solution sets can optimally solve it with data item size $q = 1$.*

Problem Def.: We consider the Steiner tree problem with the following restrictions on the input instance. There are only two terminal nodes s and t . The rest of the nodes are considered to be Steiner nodes. There are $\frac{h}{8q} - 1$ disjoint paths (cycles) of length $4q$ from s back to s , another $\frac{h}{8q} - 1$ from t back to t , and one path of length $4q$ between these terminal nodes. All edges have weight 1. The output is a subset of the edges that must contain the edges in the unique path from s to t . The cost of a solution is the number of edges taken. This problem can be seen as st -Connectivity problem. Let us call this version MST-II.

Proof of Theorem 10: We first prove B. (Stage 1:) “Stack” ALL edges. (Stage 2) Pop stack as stated in papers. After the second state, only the unique path from s to t will be kept. This proves the second.

We next prove A. For each of these disjoint paths P , consider the first edge e in it for which the algorithm receives the data item G_e . When it received $G_{e'}$ for edges e' from other paths P' , it may have seen q edges coming into P from each of the two ends. G_e may reveille another q edges somewhere in the middle of P . Hence, the algorithm may know whether e is on a path out of s or out of t . However, because the paths are of length $4q$, it has not seen the entire path. It follows that the algorithm has no way of knowing whether this edge connects s and t or not. If the algorithm rejects e then we fix that it is on the only path between s and t . By rejecting e , the algorithm loses any hope of connecting s and t . On the other hand, if the algorithm accepts e , then the adversary reveals that e is on a cycle from s back to s or from t back to t as is consistent with the information that the algorithm has. To be nice, the adversary can go on to reveal all the edges within this cycle. After this has been done either for $\frac{h}{8q} - 1$ cycles from s back to s or for $\frac{h}{8q} - 1$ cycles from t back to t , then the adversary reveals that e is on the path from s to t . The algorithm must accept at least $\frac{h}{8q}$ edges while the optimal must accept only $4q$ edges. This gives a competitive ratio of at least $Alg/Opt = \lceil \frac{h}{8q} \rceil / [4q] = \frac{h}{32q^2}$ which is unbounded as long as $q = o(\sqrt{h})$. This proves the first.

Finally, we proves C. The algorithm does depth first search from s looking for t . Note that it is key that the algorithm is adaptive. As the algorithm travels down the current path P out of s , it keeps two solutions. The first solution rejects every edges seen so far. The second solution rejects every edges seen before it started P , but accepts ever edges seen so far in P . When algorithm

reaches the end of the current path P , if it reaches t , then the second solution becomes the optimal solution simply by rejecting all unseen edges. If instead, the algorithm cycles to s , then the second solution is abandoned and the first solution is extended into two solutions as the algorithm starts to search the next path P' out of s . ■

4 Summary and Future Directions

For the MST problem, we started by improving the $\frac{5}{4}$ priority lower bound of Davis and Impagliazzo [5] to $\frac{4}{3}$ for the MST problem when the graph is small and the edge weights are known. See Section 3.3. 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. For the MST-II problem, let us consider not to just contain paths out of s and t but binary trees with height and width w . One may prove how much tree width is needed for the problem.

References

- [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] 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)*.
- [3] A. BORODIN, M. NIELSEN, AND C. RACKOFF, (2002) (Incremental) Priority algorithms, *Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms*.
- [4] 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.
- [5] S. DAVIS AND R. IMPAGLIAZZO (2004) Models of greedy algorithms for graph problems, *Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms*.
- [6] JEFF EDMONDS (2008). *How to Think About Algorithms*. Cambridge University Press.
- [7] H. KWON, (2008) Improved results on models of greedy and primal-dual algorithms, MSc Thesis, York University.
- [8] V. VAZIRANI (2001). *Approximation Algorithms*. Springer.
- [9] C. GRÖPL, S. HOUGARDY, T. NIERHOFF, AND H. PRÖMEL, Approximation Algorithms for the Steiner Tree Problems in Graphs, *Steiner Trees in Industries*, editors: D. Z. Du and X. Cheng, Kluwer.