Models of Greedy Algorithms for Graph Problems

Sashka Davis* Russell Impagliazzo†

[Extended Abstract]‡

Abstract

Borodin, Nielsen, and Rackoff ([5]) gave a model of greedy-like algorithms for scheduling problems and [1] extended their work to facility location and set cover problems. We generalize their notion to include other optimization problems, and apply the generalized framework to graph problems. Our goal is to define an abstract model that captures the intrinsic power and limitations of greedy algorithms for various graph optimization problems. We prove bounds on the approximation ratio achievable by such algorithms for basic graph problems such as shortest path, vertex cover, and others. Shortest path is an example of a problem where no algorithm in the FIXED priority model can achieve any approximation ratio (even one dependent on the graph size), but for which the well-known Dijkstra's algorithm shows that an ADAPTIVE priority algorithm can be optimal. We also prove that the approximation ratio for vertex cover achievable by ADAPTIVE priority algorithms is exactly 2. Here, a new lower bound matches the known upper bounds ([8]).

1 Introduction

There is a huge variety of known algorithms for a huge variety of computational problems. However, a surprisingly large fraction of the known efficient algorithms fit into relatively few basic design paradigms: e.g., divide-and-conquer, dynamic programming, greedy algorithms, linear programming relaxation, and hill-climbing. While algorithm designers have long had a good intuitive feel for these paradigms, there is still little research formalizing these paradigms and so little is known about their relative computational power. For example, intuitively, greedy algorithms are faster but less powerful than dynamic programming algorithms; how could we make such a statement formal?

To begin to address such questions, Borodin, Nielson and Rackoff recently introduced a formal framework for the greedy algorithm paradigm ([5]). Their framework, which they called *priority algorithms*, was originally given only for scheduling problems. However, Angelopoulos and Borodin [1] extended the priority algorithm framework to facility location and set cover problems. In this paper, we present a more abstract definition of a priority algorithm that can be applied to a variety of problem domains. In particular, we apply this model to formalize greedy algorithms for graph problems.

Even using the standard techniques only, the space of possible algorithms for a problem is large. This profusion of competing algorithmic approaches has led to a growing interest in formalizing and analyzing algorithmic methods, rather than individual algorithms. For example, much of the interest in recent proof complexity has been in studying the relative strength of different approaches to satisfiability algorithms. Extending traditional integrality gap approaches to proving lower bounds for linear programming relaxation algorithms based on specific relaxations, Arora, et. al. [2], have shown lower bounds on very general forms of relaxations. While not as technically difficult as the above work, our work contributes to a more general understanding of the relative power of algorithmic methods. We believe that the priority models described here can be taken as a starting point in formally defining and analyzing more powerful computational paradigms, such as backtracking and dynamic programming.

1.1 Motivation The greedy algorithm paradigm is one of the most important in algorithm design, because of its simplicity and efficiency. Greedy algorithms are used in at least three ways: they provide exact algorithms for a variety of problems; they are frequently the best approximation algorithms for hard optimization problems; and, due to their simplicity, they are frequently used as heuristics for hard optimization problems even when their approximation ratios are unknown or known to be poor in the worst-case. To cover all the uses of greedy algorithms, from simple exact algorithms

*University of California, San Diego, sdavis@cs.ucsd.edu. Research supported by NSF grant CCR-0098197.

†University of California, San Diego, russell@cs.ucsd.edu. Research supported by NSF grant CCR-0098197. Some work done while at the Institute for Advanced Study, supported by the State of New Jersey.

‡A full version of the paper is available at www.cs.ucsd.edu/users/sdavis

to unanalyzed heuristics, one needs to study a cross-section of problems from the easiest (minimum spanning tree) to the hardest (NP-complete problems with no known approximation algorithms).

While greedy algorithms are simple and intuitive, they are frequently deceptive. It is often possible to generate many plausible greedy algorithms for a problem, and one's first choice is often not the best algorithm.

There are also distinctions that can be made between greedy algorithms. For example, Kruskal's algorithm for Minimum Spanning Tree is in some sense simpler than Prim's algorithm, because it just scans through the edges in sorted order, rather than dynamically growing a tree.

The priority model allows one to formally address all of these uses of greedy algorithms and issues in greedy algorithm design. One can use this model to:

1. Tell when a known but slightly complicated greedy algorithm cannot be simplified. This can be done by defining a sub-model of "simple" priority algorithms, and showing that this subclass is weaker than the general priority model. (See the definitions and the separation of FIXED and ADAPTIVE priority algorithms in Section 3.1).

2. Show that sometimes greedy approximation algorithms need to be counter-intuitive. This can be done by defining a sub-model of "sensible" priority algorithms, and showing that this subclass is weaker than the general priority model. (See the definition of memoryless algorithms in Section 4).

3. Formalize the intuition that greedy algorithms are weaker than some of the other paradigms, by proving lower bounds for priority algorithms for problems with known algorithms of a different paradigm.

4. Prove that the known greedy approximation algorithm for a problem cannot be improved, by showing a matching lower bound for any priority algorithm.

5. Rule out the possibility of proving a reasonable approximation ratio for any greedy algorithm for a hard problem. This is particularly interesting for problems where greedy algorithms are used as heuristics.

1.2 Related Work The priority algorithm model resembles that of on-line algorithms. In both models, decisions affecting the output have to be made irreversibly based on partial information about the input. For this reason, the techniques used to prove bounds for priority algorithms often are borrowed from the extensive literature on on-line algorithms (See [3] for a good overview).

However, where an on-line algorithm sees the parts of its input in an adversarial order or one imposed by some real-world constraint such as availability time, a priority algorithm can specify the order in which inputs are examined. [5] considered two variants: *FIXED priority* algorithms where this order is independent of the instance and constant throughout the algorithm, and *ADAPTIVE priority* where the algorithm can change the order of future parts based on the part of the instance that it has seen. They also defined some subclasses of "intuitive" priority algorithms: *GREEDY* priority algorithms which are restricted to make each decision in a locally optimal way, and *MEMORYLESS* adaptive priority algorithms, which must base decisions only on the set of previously accepted data items.

[5] proved many non-trivial upper and lower bound results for a variety of scheduling problems (interval scheduling with unit, proportional, and arbitrary profits; job scheduling, and minimum makespan). They showed a separation between the class of ADAPTIVE priority algorithms and FIXED priority algorithms, by proving lower bound of 3 on the performance of FIXED priority algorithms on the interval scheduling on identical machines with proportional profit and observed an ADAPTIVE priority algorithm with approximation ratio 2 for the same problem. For the interval scheduling problem with proportional profit [5] proved that the Longest Processing Time heuristics is optimal within the class of FIXED priority algorithms. They also proved a separation between the class of deterministic and randomized priority algorithms. The problem [5] considered is interval scheduling with arbitrary profits. They showed a lower bound of Δ (the ratio of the maximum to the minimum unit profit among all intervals) on the performance of ADAPTIVE priority algorithms, for multiple and single machine configurations. However, if a FIXED priority not necessarily greedy algorithm is given access to randomness then it can achieve an approximation ratio of $O(\log \Delta)$.

Angelopoulos and Borodin proved that no ADAPTIVE priority algorithm can achieve an approximation ratio better than $\ln n – \ln \ln n + \Theta(1)$. This bound is tight because the greedy set cover heuristic, classified as an ADAPTIVE priority algorithm, achieves the bound. [1] also considered the unrestricted facility location problem, and proved a tight bound of $\Omega(\log n)$ on the performance of any ADAPTIVE priority algorithm, which is matched by the known greedy heuristic for the problem. For the uniform metric facility location problem, they were able to show a tight bound of 3 on the approximation ratio achieved by fixed priority algorithms.

Boyar and Larsen ([4]) proved a lower bound of $\frac{4}{3}$ for the Vertex Cover problem in the general priority model, where a data item encodes a vertex name along with the names of its neighbors. They also considered the Independent Set and Vertex Coloring problems in more restrictive priority models where a data item encodes only the degree of the vertex, excluding the names of the vertices adjacent to it, and proved lower bounds matching known upper bound results in those models.

1.3 Our Results In this paper, we extend their model to combinatorial optimization problems of other domains. In particular, we look at models for graph problems, and evaluate the performance of priority algorithms for some classical graph problems. Our main contributions are:

Abstract priority model We present an abstract definition of priority algorithm that applies to a variety of problem domains. Previous work defined a new model for each domain, e.g, scheduling algorithms ([5]) or facility location problems ([1]). We define two instantiations of our abstract model for graph problems, the node model and the edge model.

Characterization of models We show how to characterize the power of three of the models (FIXED, ADAPTIVE, and MEMORYLESS) in terms of combinatorial games between a Solver and an Adversary. This characterization holds for the abstract definitions of priority algorithms, and so is problem-independent.

Shortest Paths We show a strong separation between FIXED and ADAPTIVE priority algorithms. We consider the problem of finding the shortest path in graph G from node s to t. Dijkstra's algorithm is seen to be an ADAPTIVE priority algorithm in this model, for the case when edge weights are positive. In the case of positive edge weights, we show that no FIXED priority algorithm can achieve any approximation ratio, even a ratio that is a function of the graph size. Secondly, for graphs with negative edge weights (but no negative cycles), we show that no ADAPTIVE priority algorithm can achieve any approximation ratio.

Steiner Trees We consider the Steiner Tree problem for metric spaces. Here, a standard FIXED priority algorithm achieves an approximation ratio of 2. We show that no ADAPTIVE priority algorithm can achieve an approximation ratio better than 1.18, even for the special case when every positive distance is between 1 and 2. We give an improved algorithm for this special case, in the ADAPTIVE priority model.

Weighted Vertex Cover We considered the Weighted Vertex Cover problem in the node model. For the weighted vertex cover problem, the standard 2-approximation algorithm fits into the ADAPTIVE priority model. Moreover, we show that no ADAPTIVE priority algorithm can achieve a better approximation ratio. Thus, the known algorithm is optimal within the class of priority algorithms.

Independent Set We consider the independent set problem for graphs of small degree, in the node model. For degree-3 graphs, the standard greedy algorithm achieves an approximation ratio of $\frac{5}{3}$, [7]. This algorithm fits in the ADAPTIVE priority model. We show that no ADAPTIVE priority algorithm can achieve an approximation ratio better than $\frac{3}{2}$.

Memoryless Algorithms We define a formal model of memoryless algorithms and prove a separation between the class of ADAPTIVE priority algorithms (with memory) and memoryless adaptive priority algorithms.

1.4 What is a greedy algorithm? The term "greedy algorithm" has been applied to a wide variety of optimization algorithms, from Dijkstra's shortest path algorithm to Huffman's coding algorithm. The pseudo-code for the algorithms in question can appear quite dissimilar. There are few high-level features common to most greedy algorithms, unlike say divide-and-conquer algorithms that almost always have a certain recursive structure, or dynamic programming algorithms, which almost always fill in a matrix of solutions to sub-problems. What do these algorithms have in common, that they should all be placed in the same category?

A standard undergraduate textbook by Neapolitan and Naimipour ([10]) describes the greedy approach as follows: a greedy algorithm "grabs data items in sequence, each time taking the one that is deemed 'best' according to some criterion, without regard for the choices it has made before or will make in the future." This seems to us a fairly clear and concise informal working definition, except for the words "without regard for the choices it has made before" which we think does not in fact apply to most of the "canonical" greedy algorithms. (For example, a coloring algorithm that assigns each node the first color not used by its neighbors seems to be a typical greedy algorithm, but certainly bases its current choice on previously made decisions.)

This is the sense of a greedy algorithm the priority model is meant to capture. More precisely, a priority algorithm:

1. Views the instance as a set of "data items".
2. Views the output as a set of "choices" (decisions) to be made, one per "data item".
3. Defines a "criterion" for "best choices", which orders data items. (Making this formal leads to two models, FIXED vs. ADAPTIVE priority algorithms.)
4. In the order defined by this criterion, makes and commits itself to the choices for the data items.
5. Never reverses a choice once made (i.e., decisions are irrevocable).
6. In making the choice for the current data item, only considers the current and previous data items, not later data items.

All but the third point are also true of on-line algorithms. The main difference is that on-line algorithms have the order of choices imposed on them, whereas priority algorithms can define this order in a helpful way. Many of the lower bound techniques here, in [5], and in [1] are borrowed from the extensive on-line algorithm literature.

Are the characteristics listed above the defining features of "greedy algorithms"? Many of the known algorithms can be classified as priority algorithms (ADAPTIVE or FIXED). For example, Prim's and Kruskal's algorithms for the Minimum Cost Spanning Tree problem are classified as ADAPTIVE and FIXED priority algorithms, respectively. Dijkstra's single source shortest path algorithm also can be seen to fit the ADAPTIVE priority model. The known greedy approximation [8] for the Weighted Vertex Cover problem (WVC) can be classified as an ADAPTIVE priority algorithm. The greedy approximation for the independent set problem [7] also fits our model. In [1] it was shown that the best known greedy approximation algorithm for the set cover problem also fits the framework of priority algorithms, and similarly the greedy algorithms for the facility location in arbitrary and metric spaces have priority models. However, the term "greedy algorithm" is used in at least one other sense which the priority model is not meant to capture. A hill-climbing algorithm that uses the "steepest ascent" rule, looking for the local change that leads to the largest improvement in the solution, is frequently called a "greedy hill-climbing" or simply "greedy" algorithm. The Dijkstra heuristic for Ford-Fulkerson, which finds the largest capacity augmenting path during each iteration, is "greedy" in this sense. As far as we can tell, there is no real connection between this sense of greedy algorithm and the one defined above. We make no claims that any of our results apply to steepest ascent algorithms, or any other classes of algorithms that are metaphorically "greedy" but do not fit the above description[1].

1.5 Priority Models To make the above definition precise, we need to specify a few components: What is a decision? What choices are available for each decision? What is the "data item" corresponding to a decision? What is a criterion for ordering decisions? The exact answers to these questions will be problem-specific, and there may be multiple ways to answer them even for the same problem. In this section, we give a format for specifying the answers to these questions, leaving parameters to be specified later to model different problems.

The general type of problem we are discussing is a combinatorial optimization problem. Such a problem is given by an instance format, a solution format, a constraint (a relation between instances and solutions), and an objective function (of instance and solution, giving a real number value). The problem is, given the instance, among the solutions meeting the constraint, find the one that maximizes (or minimizes) the objective function.

We want to view an instance as a set of ***data items***, where a solution makes one decision per data item. Let Γ denotes the type of a data item; thus an instance is a set of items of type Γ, $I \subseteq \Gamma$ [2]. The solution format will assign each $\gamma \in I$ a ***decision*** σ from a set of ***options*** Σ, so a *solution* is a set of the form $\{(\gamma_i, \sigma_i) | \gamma_i \in I\}$.

For example, for k-colorings of graphs on up to n nodes, we need to assign colors to nodes. So $\Sigma = \{1, \ldots, k\}$, and Γ should correspond to the *information available about a node* when the algorithm has to color it. This is not uniquely defined, but a natural choice, and the one we will consider here, is to let the algorithm see the name and adjacency list for v when considering what to color v. Then the data item corresponding to a node is the name of the node and the adjacency

[1]While there are a few uses of the phrase "greedy algorithm" that do not seem to fit the priority model, this seems more a matter of the inherent ambiguity of natural language than a weakness in the model. A useful scientific taxonomy will not always classify things according to common usage; e.g., a shellfish is not a fish. There will also always be borderline objects that are hard to classify, e.g., is a marsupial a mammal? We should not be overly concerned if a few intuitively greedy algorithms go beyond the restrictions of the priority model; however, this will be motivation to try to extend the model in future work.

[2]We are not necessarily assuming that every subset of data items constitutes a valid instance. For scheduling problems any sequence of jobs is a valid instance. Instances of graph problems have more structure, which prevents some sets of data items from being valid graphs. We also frequently want to restrict to instances with some global structure, e.g., metric spaces or directed graphs with no negative cycles.

list of a node, i.e., Γ would be the set of pairs, $(NodeName, AdjList)$, where a $NodeName$ is an integer from $\{1, \ldots, n\}$; $AdjList$ is a list of $NodeNames$. We then view G as being given in adjacency list format: G is presented as the set of nodes v, each with its adjacency list $AdjList(v)$ [3].

More generally, a **node model** is the case when the instance is a (directed or undirected) graph G, possibly with labels or weights on the nodes, and data items correspond to nodes. Here, Γ is the set of pairs or triples consisting of possible node name, node weight, or label (if appropriate), and a list of neighbors. Σ varies from problem to problem; often, a solution is a subset of the nodes, corresponding to $\Sigma = \{accept, reject\}$.

Alternatively, in an **edge model**, the data items requiring a decision are the edges of a graph. In an edge model, Γ is the set of (up to) 5-tuples with two node names, node labels or weights (as appropriate to the problem), and an edge label or weight (as appropriate to the problem). In an edge model, the graph is represented as the set of all of its edges. Again, the options Σ are determined by the problem, with $\Sigma = \{accept, reject\}$ when a solution is a subgraph.

As another example, [5] consider scheduling problems, where jobs are to be scheduled on p identical machines. Here, we have to decide whether to schedule a job, and if so, on which machine and at what starting time. So $\Sigma = \{(m_i, t) | 1 \leq i \leq p, t \in R\} \cup \{\textit{Not scheduled}\}$. They allow the algorithm to see all information about a job when scheduling it, a data item is represented by (a_i, d_i, t_i, w_i), where a_i is the arrival time of job i, d_i its deadline, t_i its processing time, and w_i its weight. Thus, $\Gamma = \{(a, d, t, w) | a < d, t \geq d - a, w \geq 0\}$.

In fact, we can put pretty much any search or optimization problem in the above framework. Let $D(I)$ determine ***decision points*** for an instance I so that solutions can be described as arrays indexed by $D(I)$, $S \in \Sigma^{D(I)}$. Assume we give the algorithm access to the information $LocalInfo(d, I)$ when making decision $d \in D(I)$. Then we can set $\Gamma = \{(d, LocalInfo(d, I)) \mid d \in D(I),\ I \text{ is a valid instance}\}$. Since the union of all $LocalInfo$ is all we are given to solve the problem, we can view I as $\{(d, LocalInfo(d, I)) \mid d \in D(I)\}$.

As in [5], we distinguish between algorithms that order their data points at the start, and those that reorder them at each iteration. A FIXED priority algorithm orders the decisions at the start, and proceeds according to that order. The format for a FIXED priority algorithm is as follows:

```
FIXED PRIORITY ALGORITHM
Input: instance $I \subseteq \Gamma$, $I = \{\gamma_1, \ldots, \gamma_d\}$.
Output: solution $S = \{(\gamma_i, \sigma_i) | i = 1, \ldots, d\}$.
- Determine a criterion for ordering the decisions, based
  on the data items[4]: $\pi : \Gamma \to \mathbb{R}^+ \cup \{\infty\}$
- Order $I$ according to $\pi(\gamma_i)$, from smallest to largest

Repeat
  • Go through the data items $\gamma_i$ in order
  • In step $t$, observe the $t$'th data item according to
    $\pi$, let that be $\gamma_{i_t}$. Make an irrevocable decision
    $\sigma_{i_t} \in \Sigma$, based only on currently observed data items
    (i.e., the $t$ smallest under the priority function $\pi$).
    Update the partial solution: $S \leftarrow S \cup \{(\gamma_{i_t}, \sigma_{i_t})\}$
  • Go on to the next data item

Until (decisions are made for all data items)
Output $S = \{(\gamma_i, \sigma_i) | 1 \leq i \leq d\}$.
```

ADAPTIVE priority algorithms, on the other hand, have the power to reorder the remaining decision points during the execution, and clearly can simulate the simpler FIXED priority algorithms.

```
ADAPTIVE PRIORITY ALGORITHM
Input: instance $I \subseteq \Gamma$, $I = \{\gamma_1, \ldots, \gamma_d\}$.
Output: solution vector $S = \{(\gamma_i, \sigma_i) | 1 \leq i \leq d\}$
- Initialize the set of unseen data points $U$ to $I$, an
  empty partial instance $PI$, an empty partial solution
  $S$, and a counter $t$ to 1.

Repeat
  • Based only on the previously observed data items
    $PI$, determine an ordering function
    $\pi_t : \Gamma \to \mathbb{R}^+ \cup \{\infty\}$
  • Order $\gamma \in U$ according to $\pi_t(\gamma)$
  • Observe the first unseen data item $\gamma_t \in U$, and add
    it to the partial instance, $PI \leftarrow PI \cup \{\gamma_t\}$.
  • Based only on $PI$, make an irrevocable decision $\sigma_t$
    and add $(\gamma_t, \sigma_t)$ to the partial solution $S$
  • Remove the processed data point $\gamma_t$ from $U$, incre-
    ment $t$

Until (decisions are made for all data items, $U = \emptyset$)
Output $S$
```

The current decision made depends in an arbitrary way on the data points seen so far. The algorithm also has an implicit knowledge about the unseen data

[3] As mentioned before, not all sets of data items will code graphs. To actually code an undirected graph, a set of data items has to have distinct node names, and have the property that, if $x \in AdjList(y)$ then also $y \in AdjList(x)$.

[4] We could instead use a more general notion, where π is a total ordering of Γ. Because we use only finite sets of instances in our lower bounds, all of our lower bounds also hold for this more general class. Our upper bounds use orderings based on real-valued priority functions as given here.

points: no unseen point has a smaller value of the priority function π_t than γ_t.

[5] also define two other restricted models: GREEDY and MEMORYLESS priority algorithms. They define a greedy priority algorithm as follows: "*A greedy algorithm makes its irrevocable decision so that the objective function is optimized as if the input currently being considered is the final input.*" This concept of GREEDY algorithms does not seem to be well-defined for arbitrary priority models, in particular graph models, where not every set of data items constitutes a valid instance. An interesting problem for future research is to define an analogous notion of GREEDY priority algorithm for graph problems.

We formalize the notion of MEMORYLESS algorithms in Section 4 and show a separation between the class of memoryless algorithms and adaptive priority algorithms.

2 A General Lower Bound Technique

In this section, we give a characterization of the best approximation ratio achievable by a (deterministic) ADAPTIVE priority algorithm, in terms of a combinatorial game. The techniques used in this section are borrowed from competitive analysis of on-line algorithms.

Let Π be a maximization problem, with objective function μ, and Σ, and Γ be a priority model for Π. Let T be a finite collection of instances of Π. The ADAPTIVE priority game for T and ratio $\rho \geq 1$ between two players Solver and Adversary is as follows:

1. Initialize an empty partial instance PI and partial solution PS. The Adversary picks any subset $\Gamma_1 \subseteq \Gamma$.
2. Repeat until $\Gamma_t = \emptyset$.
   begin; (Round t)
   (a) The Solver picks $\gamma_t \in \Gamma_t$, and $\sigma_t \in \Sigma$.
   (b) γ_t is added to PI, and deleted from Γ_t. (γ_t, σ_t) is added to PS.
   (c) The Adversary replaces Γ_t with a subset $\Gamma_{t+1} \subseteq \Gamma_t$.

   end; (Round t)
3. In the endgame, the Adversary presents a solution S for PI.
4. The Solver wins if $PI \notin T$, S is not a valid solution, or if PS is a valid solution and $\rho \geq \frac{\mu(PS)}{\mu(S)}$ for maximization problem, or $\rho \geq \frac{\mu(S)}{\mu(PS)}$ for minimization problem.

LEMMA 2.1. *There is a winning strategy for the Solver in the above game if and only if there is an ADAPTIVE priority algorithm that achieves an approximation ratio of ρ on every instance of Π in T.*

COROLLARY 2.1. *If there is a strategy for the Adversary, in the game defined above, that guarantees a payoff of at most ρ, then there is no ADAPTIVE priority algorithm that achieves an approximation ratio better than ρ.*

We can similarly characterize the FIXED priority model by replacing steps 1 and 2(a) as follows. The rest of the game is the same.

1' Initialize an empty partial instance PI and a partial solution PS. The Solver picks a total ordering $<$ on Γ. The Adversary picks any subset $\Gamma_1 \subseteq \Gamma$.

2(a)' Let $\gamma_t \in \Gamma_t$ be the $<$-first element of Γ_t. The Solver picks $\sigma_t \in \Sigma$.

LEMMA 2.2. *There is a winning strategy for the Solver in the above game if and only if there is a FIXED priority algorithm that achieves an approximation ratio ρ on every instance of Π in T.*

3 Results for Graph Problems

In this section we present our results for FIXED and ADAPTIVE priority algorithms. The proofs of lower bounds here and in Section 4 resemble derivation of integrality gaps for given LP formulation, in that easy instances are used to establish bounds on the approximation ratio (See [14] for examples). Our results are used to evaluate priority algorithms as a model for greedy heuristics, rather than to establish hardness of approximation results for the particular problem.

3.1 Shortest Paths FIXED priority algorithms are simpler and ADAPTIVE priority algorithms can simulate them. We want to show that the two priority models ADAPTIVE and FIXED, are not equivalent in power. We define the following graph optimization problem:

DEFINITION 3.1. **(ShortPath Problem)** *Given a directed graph $G=(V,G)$ and two nodes $s \in V$ and $t \in V$, find a directed tree of edges, rooted at s. The objective function is to minimize the combined weight of the edges on the path from s to t.*

We consider the ShortPath problem in the edge model. The data items are the edges in the graph, represented as a triple (u, v, w), where the edge goes from u to v and has weight w. The set of options is $\Sigma = \{accept, reject\}$. A valid instance of the problem is a graph, represented as a set of edges, in which there is at least one path from node s to t. An alternative definition of the ShortPath problem would insist on the edges selected to form a path, rather than a tree. However, most standard algorithms construct a single source shortest paths tree, rather than a single path. In fact, not only is constructing a tree a more general version of the problem it is not difficult to show that no priority algorithm can guarantee a path.

The well-known Dijkstra algorithm, which belongs to the class of ADAPTIVE priority algorithms, solves this problem exactly.

THEOREM 3.1. *No FIXED priority algorithm can solve the ShortPath problem with any approximation ratio ρ.*

Proof sketch: We show an Adversary strategy for the FIXED priority game for any ρ. Let $k \geq 2\rho$. Let T be the set of directed graphs on four vertices s, t, a, b with edge weights either k or 1, so that t is reachable from s. The Adversary selects the set Γ_1, as shown on Figure 1. For example, u stands for the edge from s to a, with weight k. Note that the parallel edges in the figure are just possible data items; the instance graph will itself be simple. The next move is by the Solver. She must assign distinct priorities to all edges, prior to making any decisions, and this order cannot change. Thus one of the edges y and z must appear first in the order. Since the set of data items is symmetrical, we assume, without loss of generality, that y appears before z in this order. The Adversary then removes edges v and w, restricting the remaining set of data items to $\Gamma_2 = \{x, y, z, u\}$. The

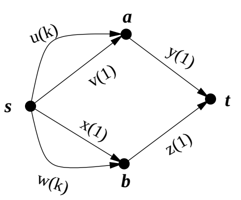

Figure 1: Adversary selects $\Gamma_1 = \{x, y, z, u, v, w\}$.

Adversary's strategy is to wait until the Solver considers edge y before deleting any other items, and applies the following strategy after he observes Solver's decision on y:

1. If the Solver decides $\sigma_y = reject$, then the Adversary removes z from the remaining set of data items. The Adversary does not subsequently remove any other data items. Thus, the instance will be $I = \{u, x, y\}$. The Adversary outputs a solution $S = \{y, v\}$, while the Solver's solution $PS \subseteq \{u, x\}$ cannot contain any path from s to t.
2. If the Solver decides $\sigma_y = accepted$. Then the Adversary never deletes any data items, making $I = \{u, x, y, z\}$. In the end game, the Adversary presents solution $S = \{x, z\}$, with cost 2. If the Solver picks edge z, then the Solver failed to satisfy the solution constraints, since no sub-graph with both y and z can be a directed tree. Otherwise, $PS \subseteq \{u, x, y\}$ and thus the cost of the Solver is at least $k+1$. The approximation ratio is: $\frac{k+1}{2} > \rho$, so the Solver loses. $\square$

We conclude that the two classes of algorithms FIXED and ADAPTIVE priority are not equivalent in power. Dijkstra's algorithm can solve the above problem exactly and belongs to the class of ADAPTIVE priority algorithms.

Dijkstra's algorithms, however, does not work on graphs with negative weight edges. Is dynamic programming necessary for this problem? Perhaps there exists an ADAPTIVE priority algorithm which can solve the Single Source Shortest Paths problem on graphs with negative weight edges, but no negative weight cycles?

THEOREM 3.2. *No ADAPTIVE priority algorithm can solve the ShortPath problem for graphs with weight function $w_e : (E) \to \mathbb{R}$, allowing negative weights, but no negative weight cycles.*

This result shows a separation between priority algorithms and dynamic programming algorithms for shortest path problems; a similar separation was shown by [5] for interval scheduling on a single machine with arbitrary profits.

3.2 The Weighted Vertex Cover Problem

Next we examine the performance of ADAPTIVE priority algorithms on the Weighted Vertex Cover problem (WVC). WVC is **NP**-hard problem and no polynomial time algorithm can solve it exactly, or even with approximation ratio $1 + \epsilon$, for some $\epsilon > 0$ unless $P = NP$, [14]. The well known 2-approximation algorithm [8], fits our ADAPTIVE priority model, and we show that no ADAPTIVE priority algorithm can achieve an approximation ratio better than 2.

We consider the Vertex Cover problem in the node model. The data items are nodes, with their name, weight, and adjacency list. The set of options is $\Sigma = \{accept, reject\}$, meaning the vertex is added to the vertex cover or thrown away.

THEOREM 3.3. *No ADAPTIVE priority algorithm can achieve an approximation ratio better than 2 for the Weighted Vertex Cover problem.*

Proof For any $\rho < 2$, we show a winning strategy for the Adversary in the game defined in Section 2. For a suitably large n, the Adversary picks a complete bipartite graph $K_{n,n}$ and sets T to be the set of instances with this underlying graph, where node weights are either 1 or n^2. Since the underlying graph is fixed, Γ contains two data items for each node, varying only in the node weight.

Each time the Solver selects a data item corresponding to a node v, the Adversary deletes the other such item (avoiding inconsistency). The Adversary otherwise does not delete any items until one of the following three events occurs:

- **Event 1:** Solver accepts a node v with weight n^2.
- **Event 2:** Solver rejects a node v (of any weight).
- **Event 3:** Solver accepts $n-1$ nodes of weight 1 from either side of the bipartite graph.

Eventually, one of these three events must occur. If **Event 1** occurs first, then the Adversary fixes the weights of all nodes on the opposite side to 1, by deleting all data items giving weight n^2. This is possible, since previously the Solver has only accepted nodes of weight 1, so no data item giving a node weight 1 has been deleted. Similarly, for each node on the same side with two possible weights, Adversary deletes the data item of weight 1. This fixes the instance. Eventually, the Solver will consider all remaining data items, and output a solution PS with $v \in PS$, so PS has cost at least n^2. The Adversary outputs a solution S consisting of all nodes on the other side, which has cost n, winning if $\rho < n^2/n = n$.

If **Event 2** occurs, the Adversary fixes the weights of all unseen nodes on the opposite side of v to n^2 and the weights of remaining nodes on the same side to 1 (by deleting the data items of other weights.) Since neither Events 1 or 3 have previously occurred, it is possible for all nodes on the same side to have value 1, and there are at least 2 unseen nodes on the opposite side. Eventually the Solver outputs a vertex cover PS with $v \notin PS$. Hence, all nodes on the opposite side must be in PS, for a total cost of at least $2n^2$. The Adversary outputs all nodes on the same side of v, for a cost of n^2+n-1. The Adversary wins if $\rho < \frac{2n^2}{n^2+n-1} = 2 - o(1)$.

If **Event 3** occurs first, the Solver has committed to all but one vertex on one side, say A, of the bipartite graph. Then the Adversary fixes the weight of the last unseen vertex in A to n^2 (by deleting the data item giving it value 1) and unseen nodes on the other side are set to weight 1.

The Solver outputs a set either containing all of A and hence having weight at least n^2, or containing all but one node of A and containing all of the other side B, giving a total weight of $2n-1$. The Adversary takes side B, winning if $\rho < \frac{2n-1}{n} = 2 - o(1)$.

Thus, for any $\rho < 2$, the above is a winning strategy for the Adversary for a suitably large value of n. □

The class of instances $K_{n,n}$ can be solved easily. However, what Theorem 3.3 shows is that a large class of greedy algorithms cannot approximate the WVC problem with approximation ratio better than 2. This bound is tight, because the known greedy heuristic achieves an approximation ratio 2, and thus is optimal in the class of adaptive priority algorithms.

It was important to our bound that nodes had weights. Boyar and Larsen ([4]) consider the unweighted version and prove a lower bound of $\frac{4}{3}$ for any priority algorithm.

3.3 The Metric Steiner Tree Problem We examine the performance of the ADAPTIVE priority algorithms on the Metric Steiner Tree problem. The instance of the problem is a graph $G = (V, E)$, with the vertex set partitioned into two disjoint subsets, required and Steiner. Each edge $e \in E$ has a positive weight $w(e)$. The problem is to find a minimum cost tree, spanning the required vertices which may contain any number of Steiner nodes. We are interested in the metric version, where the edge weights obey the triangle inequality. The standard 2-approximation algorithm for the Steiner tree problem discovered independently by [11] and [12] belongs to the class of FIXED priority greedy algorithms. In the restricted case when the edges of the graph have weights either 1 or 2, known as the *Steiner*$(1,2)$ problem, Bern and Plassmann [6] proved that *the average distance heuristic* ([9], [13]), is a $\frac{4}{3}$-approximation. The *average distance heuristic*, however, does not seem to fit our priority model.

Our lower bounds are for an intermediate class of Steiner problems, where edge weights are in the interval $[1,2]$. This very local restriction implies the metric property, which helps the adversary argument. To show that we cannot get a tight bound of 2 using this restriction, we give a new priority algorithm for this restricted class.

THEOREM 3.4. *No ADAPTIVE priority algorithm in the edge model can achieve an approximation ratio better than $\frac{13}{11}$ for the Metric Steiner Tree problem, even when edge weights are restricted to the interval $[1,2]$. There is a 1.875-approximation adaptive priority algorithm for the $[1,2]$ Metric Steiner Tree problem.*

3.4 Maximum Independent Set Problem We study the performance of ADAPTIVE priority algorithms for the MIS problem and prove a lower bound of $\frac{2}{3}$, for the MIS problem on graphs with maximum degree 3.

THEOREM 3.5. *No ADAPTIVE priority algorithm in the node model can achieve an approximation ratio better than $\frac{3}{2}$ for the MIS problem, even for graphs of degree at most 3.*

4 Memoryless Priority Algorithms

Memoryless priority algorithms are a subclass of adaptive priority algorithms. Although the model is general, it can only be applied for problems where the decision options are $\Sigma = \{accept, reject\}$. Problems with such priority model include scheduling problems, some graph optimization problems (vertex cover, Steiner trees, maximum clique, etc.), and also facility location and set cover problems. Memoryless priority algorithms have a restriction on what part of the instance they can remember. We would like to think of the decision of the algorithm to reject a data item as a *'no-op'* instruction. The state of the algorithm and the remaining data items and their priorities do not change, but the current data item is "forgotten" by the algorithm and removed from the remaining sequence of data items. The algorithm stores in its memory (state) only data items that were accepted. Each decision made by the algorithm is based on the information presented by the current data item and the state. The formal framework of a memoryless adaptive priority algorithm is:

```
MEMORYLESS PRIORITY ALGORITHM
Input: instance $I \subseteq \Gamma$, $I = \{\gamma_1, \ldots, \gamma_d\}$
Output: solution $S = \{(\gamma_i, \sigma_i) | \ \sigma_i = accept\}$
- Initialize: a set of unseen data items $U \leftarrow I$, a partial
  solution $S \leftarrow \emptyset$, and a counter $t \leftarrow 1$
- Determine an ordering function: $\pi_1 : \Gamma \rightarrow \mathbb{R}^+ \cup \{\infty\}$
- Order $\gamma \in U$ according to $\pi_1(\gamma)$
Repeat
    • Observe the first unseen data item $\gamma_t \in U$
    • Make an irrevocable decision $\sigma_t \in \{accept, reject\}$
    • If ($\sigma_t = accept$) then
        – update the partial solution: $S \leftarrow S \cup \{\gamma_t\}$
        – determine an ordering function:
          $\pi_{t+1} : \Gamma \rightarrow \mathbb{R}^+ \cup \{\infty\}$
        – order $\gamma \in U - \{\gamma_t\}$ according to $\pi_{t+1}$
    • If ($\sigma_t = reject$) then
        – Forget $\gamma_t$, i.e., delete it from current state.
    • Remove the processed data item $\gamma_t$ from $U$; $t \leftarrow t+1$
Until ($U = \emptyset$)
Output $S$
```

The differences between adaptive priority algorithms and memoryless algorithms are:

- **Reordering the inputs** Priority algorithms with memory can reorder the remaining data items in the instance after each decision, while memoryless algorithms can reorder the remaining input after they *accept* a data item.
- **State** Memoryless algorithms *forget* data items that were rejected, while memory algorithms keep in their state all data items and the decisions made.
- **Decision making process** In making decisions, memory algorithms consider all processed data items and the decisions made, while memoryless algorithms can only use the information about data items that were accepted.

On one side, memoryless algorithms are intuitive. Consider Prim's and Dijkstra's algorithms. Both algorithms are adaptive priority algorithms and grow a tree by adding an edge at each iteration. In the case of Prim's algorithm, when all nodes of the graph are connected by the currently grown spanning tree the algorithm rejects all remaining edges of the graph. In this sense Prim's algorithm is memoryless, since the priority function and the decisions made depend only on the edges added to the current spanning tree. Note that once this algorithm rejects an edge it never accepts another edge (in this sense any algorithm with this structure can trivially be regarded as a memoryless algorithm). Similarly, Dijkstra, and the known greedy heuristics for the facility location, set cover, and vertex cover problems can be classified as memoryless.

On the other hand, memoryless algorithms can be considered counterintuitive, in a sense that the algorithm could explore the structure of the instance by giving lower priority to "unwanted" data items and rejecting them, and thus could achieve better performance. For example, consider the weighted independent set problem on cycles. If an algorithm rejects the smallest weight node, then the algorithm has learned 1) the value of the smallest weight; 2) no other vertex of the instance has a smaller weight; 3) the name of the vertex with the smallest weight; 4) the names of the neighbors of the node with the smallest weight. Perhaps exploring this knowledge could give the algorithm more power? This idea is used to show the following result.

THEOREM 4.1. *No memoryless adaptive priority algorithm can achieve an approximation ratio better than* 2 *for the weighted independent set problem on cycles with weights* 1 *and* k *(WIS-2 problem). There is a* $(1+\frac{2}{k-1})$*-approximation adaptive priority algorithm for the WIS-2 problem.*

5 Open Questions and Future Directions

Priority algorithms are a formal framework for greedy algorithms, and many greedy algorithms fit this model. We were able to show lower bounds for large class of algorithms for various graph optimization problems, which shows the weakness of the technique.

However, several questions remain unanswered in our current model for priority algorithms. Can we obtain an improved lower bound for the general Metric Steiner tree problem, with unrestricted edge weights? Our current upper and lower bounds do not match, and we would like to know whether we can close the gap. What lower bounds can we obtain for priority algorithms for the Maximum Independent Set problems for graphs of arbitrary degrees and for the Weighted Independent Set problem and other graph optimization problems?

Memoryless priority algorithms were proved less powerful for graph optimization problems, yet most of the known approximation algorithms are classified as memoryless algorithms. Can we design improved approximation schemes using memory?

The priority model seems a flexible and useful tool for understanding the limitations of the simple greedy algorithms. However, some "intuitively greedy" algorithms fall outside our model. Thus in future work we might want to consider extensions of the model, so that the extended model captures a larger class of algorithms. One such natural extension will be to consider a model where the algorithm is given access to "global information", such as the length of the instance. For graph problems this will be the number of edges or vertices in the graph. What lower bounds can we prove for priority algorithms in this extended model? [5] and [4] showed that some of their lower bounds for scheduling problems do hold for this extended model and they considered this as evidence of robustness of the model.

Another extension of priority algorithms for graph problems is to redefine the notion of *local information* associated with a data item. For example, suppose the type of data item encodes not just the names of the neighbors, but also neighbors of the neighbors of a node as well, assuming the problem is viewed in the node model. Would that additional information, given to the algorithm during the decision-making process, increase the power of the priority algorithms? A different direction would be to introduce randomization in the model. Our current lower bounds hold only for deterministic algorithms.

Greedy algorithms are simple and efficient. However, dynamic programming algorithms and backtracking algorithms are more powerful[5]. We would like to define similar general frameworks that would capture the defining characteristics of those powerful classes of algorithms. Can we design similar formal models and frameworks for proving lower bounds? Can we establish both negative results on their performance but also identify the strength of the technique and the problems on which it performs well? If we build formal models for the known efficient algorithm design paradigms (greedy, dynamic programming, hill-climbing, etc.) then negative results will show the limits of the known (natural) approaches to optimization.

[5]We proved that the Single Source Shortest Path problem on graphs with negative weight edges cannot be solved by any priority algorithm.

Acknowledgments:

The authors are thankful to Allan Borodin, Joan Boyar, Jeff Edmonds, Valentin Kabanets, Kim Larsen, Vadim Lyubashevsky, and Morten Nielsen.

References

[1] S. Angelopoulos and A. Borodin, *On the Power of Priority Algorithms for Facility Location and Set Cover*, 5th International Workshop on Approximation Algorithms for Combinatorial Optimization, September, 2002.

[2] S. Arora and B. Bollobás and L. Lovász, *Proving Integrality Gaps Without Knowing the Linear Program*, 43rd Symposium on Foundations of Computer Science, 2002, pp. 313-322.

[3] A. Borodin and R. El-Yaniv, *Online Computation and Competitive Analysis*, Cambridge University Press, New York, 1998.

[4] J. Boyar and K. S. Larsen *Preliminary Results on Priority Algorithms for Graph Problems*, Manuscript, 2003.

[5] A. Borodin and M. N. Nielsen and C. Rackoff, *Incremental) Priority Algorithms*, Thirteen Annual ACM-SIAM Symposium on Discrete Algorithms, 2002.

[6] M. Bern and P. Plassmann, *The Steiner Problem with Edge Lengths 1 and 2*, Information Processing Letters, (32) 1989, pp. 171–176.

[7] M. Halldorsson and K. Yoshihara, *Greedy Approximations of Independent Sets in Low Degree Graphs*, Sixth International Symposium on Algorithms and Computation, 1995.

[8] D. Johnson, *Algorithms For Combinatorial Problems*, J of Comp. and System Sci., (9) 1974, pp. 256–278.

[9] B. Maxman and M. Imase, *Worst case performance of Rayward-Smith's Steiner tree heuristic*, Information Processing Letters, (29) 1988, pp. 283–287.

[10] R. Neapolitan and K. Naimipour, *Foundations of Algorithms*,

[11] L. Kou and G. Markowsky and L. Berman, *A Fast Algorithm for Steiner Trees*, Acta Informatica, (15) 1981, pp. 141–145. Jones & Bartlett Publishing, 1997.

[12] J. Plesnìk, *A Bound for Steiner Tree problem in graphs*, Math. Slovaca, (31) 1981, pp. 155–163.

[13] V. J. Rayward-Smith, *The Computation of nearly minimal Steiner Trees in graphs*, Internat J. Math. Educ. Sci. Tech., (14) 1983, pp. 15–23.

[14] V. Vazirani, *Approximation Algorithms*, Springer-Verlag, Berlin, 2001.