Order-Preserving Transformations and Greedy-Like Algorithms

Spyros Angelopoulos[*]

School of Computer Science, University of Waterloo,
Waterloo, Ontario, Canada N2L 3G1
sangelop@cs.uwaterloo.ca

Abstract. Borodin, Nielsen and Rackoff [5] proposed a framework for abstracting the main properties of greedy-like algorithms with emphasis on scheduling problems, and Davis and Impagliazzo [6] extended it so as to make it applicable to graph optimization problems. In this paper we propose a related model which places certain reasonable restrictions on the power of the greedy-like algorithm. Our goal is to define a model in which it is possible to filter out certain overly powerful algorithms, while still capturing a very rich class of greedy-like algorithms. We argue that this approach better motivates the lower-bound proofs and possibly yields better bounds. To illustrate the techniques involved we apply the model to the well-known problems of (complete) facility location and dominating set.

Keywords: Priority algorithms, inapproximability results, facility location, dominating set.

1 Introduction

Greedy algorithms have been a widely popular approach in combinatorial optimization and approximation algorithms. This is mainly due to their conceptual simplicity as well as their amenability to analysis. In fact, one reasonably expects a greedy algorithm to be one of the first approaches an algorithm designer employs when facing a specific optimization problem. It would therefore be desirable to know when such an approach is not likely to yield an efficient approximation. However, while it is relatively easy to identify a greedy algorithm based on intuition and personal experience, a precise definition of such a class of algorithms is needed so as to prove limitations on its power. Even more importantly, as argued in [5] it is expected that a rigorous framework for greedy algorithms can provide insight on how to develop better, more efficient algorithms.

Despite the popularity and importance of greedy algorithms as an algorithmic paradigm, it was only recently that a formal framework for their study emerged. In particular, Borodin, Nielsen and Rackoff introduced in [5] the class of *priority algorithms* as a model for abstracting the main properties of deterministic

---

[*] Research done while at the Department of Computer Science, University of Toronto.

greedy-like algorithms (we will hereafter refer to their model as the *BNR model*). In addition, Borodin, Nielsen and Rackoff showed how the framework can yield lower bounds on the approximation ratio achieved by priority algorithms for a variety of classical scheduling problems. In a follow-up paper, Regev [15] addressed one of the open questions in [5] related to scheduling in the subset model. The priority framework was subsequently applied in the context of the facility location and set cover problems by Angelopoulos and Borodin [2], and it has also been extended to capture greedy-like algorithms that allow randomization [1]. More recently, Davis and Impagliazzo [6] showed how to modify the BNR model in order to show limitations on the power of priority algorithms for *graph optimization problems*. To illustrate the technique, they applied the game to basic graph problems such as shortest path, metric Steiner tree, independent set and vertex cover. We refer to their model as the *DI model*.

In this paper we continue the study of greedy-like algorithms as formulated by priority algorithms. We first provide some key definitions. According to [5] priority algorithms are characterized by the following two properties:

1. The algorithm specifies an ordering of "the input items" and each input item is considered in this order.
2. As each input item is considered, the algorithm must make an "irrevocable decision" concerning the input input.

As one would expect, a precise definition of "input items" and "irrevocable decisions" pertains to the specific problem at hand.

Depending on whether the ordering changes throughout the execution of the algorithm, two classes of priority algorithms can be defined:

- Algorithms in the class FIXED PRIORITY decide the ordering before any input item is considered and this ordering does not change throughout the execution of the algorithm.
- Algorithms in the broader class ADAPTIVE PRIORITY are allowed to specify a new ordering after each input item is processed. The new ordering can thus depend on input items already considered.

In [5], **greedy** algorithms are defined as (fixed or adaptive) priority algorithms, which satisfy an additional property: the irrevocable decision is such that the objective function is *locally optimized*. More specifically, the objective function must be optimized as if the the input currently being considered is the last input. Note that in this context not every greedy-like (that is, priority) algorithm is greedy.

In both the work of Borodin *et al* and Davis and Impagliazzo, in order to show a lower bound on the approximation ratio one evaluates the performance of every priority algorithm for an appropriately constructed nemesis input. The construction of such a nemesis input can be seen as a *game* between *an adversary* and the algorithm. In both games, the adversary presents initially a (large) set of potential input items, and in each round removes certain input items according to the corresponding decisions made by the algorithm.
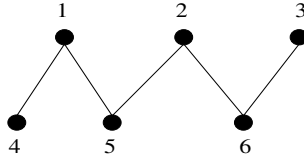
Fig. 1. An input for a graph problem

As noted in [5] it is expected that the study of priority algorithms will provide insights about how to develop better, more efficient greedy-like algorithms. To this end, it is essential that the adversary involved in the lower-bound arguments is "reasonably" powerful, or, from a different scope, that the priority algorithm does not have "unreasonable" power. Otherwise, it is expected that i) the arguments behind the lower bound proofs will be very elaborate (something which becomes even more critical in the context of graph problems, where the input items refer to each other) and it will be hard to get reasonably good lower bounds; and most importantly ii) the lower bounds will not necessarily reflect the limitations of "real" greedy-like algorithms, but rather those of artificial algorithms which use information that is conceptually difficult to generalize.

To illustrate the argument above, consider the graph shown in Figure 1, and suppose it is used as input to a FIXED PRIORITY algorithm for a certain unweighted graph optimization problem. In the DI model it is possible that the algorithm specifies the ordering 1,5,4,2,3,6 from highest to lowest priority. Note that in this ordering a vertex of degree one (namely vertex 3) receives both higher and lower priority than vertices of degree two (namely vertices 6 and 2, respectively). In other words, the algorithm has somehow the power to differentiate between two input items of the same degree such as vertices 6 and 2. However, it is very counterintuitive to think of a FIXED PRIORITY algorithm which can make such an unnatural distinction of seemingly identical input items.

In this paper we put forward a formal model which intends to capture the above observation, namely that the priority algorithm normally should give equal priority to input items which "look alike" (which is not necessarily the case in the DI and BNR models). We propose an adversary which applies to priority algorithms that are not necessarily as powerful as the algorithms assumed by the DI model, while still being able to capture a very wide class of natural greedy-like algorithms for graph problems. In a nutshell, we do not allow algorithms to acquire useful information from the id's of the input items (vertices). Interestingly, the proposed adversary does not remove input items, but instead can apply a more wide range of transformations over the potential input items. In particular, we allow transformations that do not affect the ordering in which the algorithm considers the input items, in the presence of an adversary. We believe this model reflects the fact that local information does not necessarily yield knowledge of the global structure of an instance, which is only self-evident in greedy-like algorithms.

To demonstrate our techniques, we apply our model to the *complete metric facility location* and *dominating set* problems. The former is a variant of the

classic metric facility location problem in which every node is both a facility and a city (unlike the *disjoint* variant in which facilities and cities form disjoint sets); the latter can be seen as a variant of the set cover problem, where now each vertex can both cover and "be covered by" other vertices. We focus on these two graph optimization problems for two reasons: i) their corresponding variants, namely disjoint facility location and set cover have already been studied in [2] from the point of view of approximability by priority algorithms using the BNR game; ii) the lower bounds of [2] do not carry over to the problems we consider, since the input items are no longer "isolated", but refer to each other.

How do our results compare to the work of Davis and Impagliazzo? It is not clear to us whether the bounds as stated in Theorems 2 and 3 can be reproduced using the DI adversary. Similarly, we do not know whether Theorem 5 can be shown within the DI framework, but we believe that if so it would require a more elaborate proof. Theorems 1 and 4 can definitely be reproduced by the DI adversary. We nevertheless include them since not only they illustrate our transformations, but they are also interesting on their own (Theorem 1 uses an instance where the facilities have non-uniform opening costs).

We emphasize that as in deterministic priority algorithms (and similar to competitive analysis of online algorithms) the lower bounds are derived by exploiting the syntactic structure of the algorithms, and are orthogonal to any complexity considerations. In other words, we allow the algorithm unbounded time complexity.

Very recently and independently to this work, Borodin, Boyar and Larsen [4] addressed further the topic of priority algorithms for graph optimization problems. Their focus is primarily on the effect of memory on priority algorithms as defined in [5] and [6]; in particular they considered a model in which memoryless algorithms do not accept an input item once some other input item was rejected in a previous iteration (which they call the "acceptances-first" model) and presented lower bounds for problems such as vertex cover, independent set and vertex coloring. In addition, they showed that the "vertex-adjacency" model of representing input items (also assumed in our work) is more general than the "edge-adjacency" model. Finally, they proposed a formal definition of "greediness" in the context of graph problems; however, it is not clear whether their definition can lead to lower bounds for the class of "greedy" priority algorithms. The contributions of our paper are orthogonal to the work of Borodin, Boyar and Larsen even though the two papers address a similar topic.

2   The Model

2.1   Preliminaries

Input representation. An instance of a graph optimization problem Π can be described as an (undirected) graph $G = (V, E)$, with vertex and edge weights. A reasonable representation of an input item for Π is a pair $\langle d_v, w_v \rangle$ where

$v \in V$. Here, w_v is the *weight vector*[1] of v, and d_v is the *distance vector* of v. The weight vector represents the weight assignments for vertex v (e.g., the cost that is payed if v is included in the solution), while the distance vector is the vector of distances (edge weights) from v to every other vertex in V. In addition, each input item has a unique id. In what follows we use the notation $\langle v \rangle$ to denote the input item that corresponds to vertex v, although in some cases we will not make the distinction when it is clear from the context. Note that every time the priority algorithm considers an input item $\langle v \rangle$, all the information for $\langle v \rangle$ becomes available to the algorithm (including the id's of the neighbouring vertices in its distance vector).

Priority functions. As in [6] we will assume that the priority algorithm uses a *priority function* to assign real-valued priorities to the input items. Thus, in each iteration, the input item with the highest priority is the one that the priority function maps to the highest value among all remaining input items. To make the definition more precise, we must distinguish between the two classes of priority algorithms. Let $\langle V \rangle$ denote the infinite set of input items of the form described earlier. For FIXED PRIORITY algorithms, the priority function is $P : \langle V \rangle \to \mathbb{R}$, that is, the priority of an input item is determined solely by the input item itself. In contrast, in iteration k, an ADAPTIVE PRIORITY algorithm will take into consideration the *history* of the algorithm's execution, namely the set $H_{k-1} = \langle v_1 \rangle, \langle v_2 \rangle, \ldots, \langle v_{k-1} \rangle$ of input items considered in iterations 1 through $k - 1$. Hence for ADAPTIVE PRIORITY algorithms we can describe the priority function at iteration k as $P_k : \langle V \rangle \times H_{k-1} \to \mathbb{R}$.

We will assume that in the event two input items have the same highest priority, the algorithm cannot distinguish them. Equivalently, we will assume an adversary that dictates which input item should be considered next, in the event of a tie.

We also need to impose some natural, realistic restrictions on the capacity of the algorithm to assign priorities and differentiate between input items. Let us first introduce some notation. For a distance vector d_v denote by $M(d_v)$ the distance multiset of v, namely the multiset of all distances from v to every other vertex in G. Also, for a given set $S \subseteq V$, denote by $d_v(S)$ the vector of distances $d(v, u)$, for all $u \in S$.

Consider first algorithms in the class FIXED PRIORITY. Let $\langle v \rangle$, $\langle u \rangle$ be two input items. The priority function P must obey the following rule:

FP Rule: $P(\langle v \rangle) = P(\langle u \rangle)$ if $M(d_v) = M(d_u)$ and $w_v = w_u$. $(*)$

The interpretation of the above rule is that *id's do not carry information*. In this view the distance vectors degenerate to distance multisets, and thus two

---

[1] In general, the weight vector can represent more than one weights, according to the specific problem at hand. E.g., for the weighted complete facility location problem, each weight vector will store the weight of the facility as well as its opening cost.

input items with the same distance multisets and the same weight vectors will be indistinguishable to the algorithm.

A similar assumption will be made for ADAPTIVE PRIORITY algorithms; in this case, however, we must take into account the history. More specifically, let $\langle v \rangle$, $\langle u \rangle$ be two input items not in H_{k-1}. Also, let $\pi : V \setminus H_{k-1} \cup \{u\} \cup \{v\} \rightarrow V \setminus H_{k-1} \cup \{u\} \cup \{v\}$ be a permutation of vertices in $V \setminus H_{k-1} \cup \{u\} \cup \{v\}$. Then P_k must observe the following rule:

AP Rule: $P_k(\langle v \rangle, H_{k-1}) = P_k(\langle u \rangle, H_{k-1})$ if

- $d_v(H_{k-1}) = d_u(H_{k-1})$ and $w_u = w_v$,
- There exists a permutation π such that for every $x \in V \setminus H_{k-1} \cup \{u\} \cup \{v\}$, $d(u, x) = d(v, \pi(x))$, $d_x(H_{k-1}) = d_{\pi(x)}(H_{k-1})$ and $w(x) = w(\pi(x))$.   (**)

The first constraint is related to the fact that the algorithm knows the distance vectors of all vertices in H_{k-1}. It implies that the history by itself is not sufficient to distinguish u from v. The second constraint is related to vertices not yet considered, and likewise signifies that such vertices cannot help in distinguishing u from v, assuming that id's do not carry information.

We conclude this section by mentioning that every time the priority algorithm considers an input item $\langle v \rangle$ it has to make an irrevocable decision concerning the input item. In several graph problems, the decision is whether to include the input item in the partial solution (we then say that the algorithm *accepts* the input item).

2.2   Order-Preserving Transformations

Similar to [5] and [6], in order to establish a lower bound on the approximability of a graph problem by a deterministic priority algorithm, we will use the concept of a game between the algorithm and an adversary. The game evolves in rounds, with each round corresponding to an iteration of the algorithm as described below (we focus on the more general case of ADAPTIVE PRIORITY algorithms).

The adversary presents, in the beginning of the first round of the game, a graph $G_1 = (V_1, E_1)$. Denote by $v_1^1, v_2^1, \ldots, v_n^1$, the input items for G_1, with the superscript "1" identifying the round of the game, and subscripts denoting the *labels* of vertices [2]. The algorithm considers the input item of highest priority, say v_1^1, and makes an irrevocable decision concerning it. More generally, suppose that by the end of the k-th round the algorithm has considered the input items with labels $1, \ldots, k$. At the end of the k-th round, the adversary applies a *transformation* $\phi_k : \langle V \rangle \rightarrow \langle V \rangle$. This transformation maps an input item $\langle v_i^k \rangle = \langle d_{v_i^k}, w_{v_i^k} \rangle$

---

[2] We distinguish between "labels" and "id's" intentionly, since the adversary is allowed to permute id's, as will become clear later. We need some invariant piece of information to refer to the input items in the order they are considered in the game, and the labels serve precisely this purpose.

to an input item $\langle v_i^{k+1} \rangle = \langle d'_{v_i^{k+1}}, w'_{v_i^{k+1}} \rangle$; in other words, the distance and weight vectors of the item may change. We emphasize that the transformation applies only to input items with labels $k+1, \ldots, n$, i.e., the adversary cannot modify items considered in previous rounds. This gives rise to a new graph G_{k+1}, which can be uniquely described by the input items $v_1^{k+1}, v_2^{k+1}, \ldots, v_n^{k+1}$, for which $v_i^{k+1} \equiv v_i^k$, for all $i \leq k$. The game proceeds until the last round, namely round n. At that point the algorithm has considered all input items in the graph.

As one might expect, only limited types of transformations can be helpful for our purposes. We call ϕ_k an *order-preserving* transformation, if and only if for every $j \leq k$, and $i \geq k+1$,

$$P_j(v_i^{k+1}, H_{j-1}) \leq P_j(v_j^j, H_{j-1}) \tag{1}$$

where H_{j-1} is the history at the end of round $j-1$ of the game, namely $H_{j-1} = \{v_l^l \mid l \leq j-1\}$.

Informally, the definition suggests that as v_i^k is "replaced" by v_i^{k+1}, the ordering of input items (in terms of their labels) considered by the algorithm up to round k will not be affected.

The following lemma[3] formalizes the use of the adversary/algorithm game as a tool for bounding the approximation ratio:

Lemma 1. *Suppose that the graph defined by the input items $v_1^n, \ldots v_n^n$ (as determined by the game between the adversary and the algorithm) is given as input to the algorithm. Then on this specific input, and in iteration i, the algorithm will consider input item v_i^n. In other words, the algorithm will consider input items in the order of their labels.*

The following transformations are implied in our model, but we emphasize them since they are of particular importance. First, if σ is the ordering of a set of input items, as produced by a FIXED PRIORITY algorithm, then the adversary can swap the positions in σ of two input items which have the same priority. Second, suppose that G_1 and G_2 are two **isomorphic** graphs, in the sense that there exists a permutation of the id's of vertices in G_1 which produces G_2. Suppose that on input G_1 the algorithm assigns label i to the vertex it considers in the i-th iteration. Then on input G_2, the algorithm will consider in the i-th iteration the vertex with label i, and will make the same decision as the decision made by the algorithm on input G_1 and in the i-th iteration.

2.3   What Is a "Greedy" Algorithm for a Graph Problem?

In the context of graph problems, a definition for greediness that treats the current input item as if it were the last one becomes problematic. First, note that throughout its execution the algorithm acquires local information which reveals

---

[3] In this preliminary version we omit certain proofs due to space restrictions. Full proofs will be provided in the journal version of the paper.

only part of the input graph. Hence, it is difficult to think of a specific input item as being the last one when the partial information suggests that there definitely exist input items that should follow. Second, and more importantly, it is not easy to identify a unique "locally optimal" decision at each iteration, precisely because the partial information does not represent a valid graph instance (some distances and weights have not yet been revealed to the algorithm). For the above reasons we will employ only intuitive and in a sense ad-hoc definitions of **greedy** (as opposed to greedy-like) algorithms which are specific to the problems we consider. The definitions which we propose are still broad enough to capture known algorithms, and provide some flavour of what a "locally optimal" decision is meant to be. We insist on providing some meaningful definitions not only because of the historical interest in this concept, but mainly because the concept itself is widely used in practice. As one would expect, it is possible to show much better lower bounds for greedy priority algorithms (for instance the bounds of Theorem 3 and Theorem 5 are tight for the corresponding classes of algorithms). In such cases, the bounds suggest some directions towards the design of better algorithms: namely, that in order to improve the approximation ratio it is essential that the algorithm is not greedy.

We first provide a definition of what we consider to be a greedy priority algorithm for complete facility location. Let v_i be the node considered by the algorithm at iteration i. We capture the greedy behaviour of the algorithm (which one would informally describe by the motto "live for today"), by requiring that it **always opens v_i if this results in lowering the cost of the "current solution"**. Of course, we must clearly define the intuitive term "current solution". Note that the algorithm has only limited information (i.e., what can be deduced by the triangle inequality) about the distance $d(u, v)$ of every two nodes u, v which have not been considered yet (prior to iteration i), as well as the facility cost of such nodes. The current solution is then the optimal solution with the constraint that the algorithm cannot open a yet unconsidered node (which implies that every unconsidered node has to be connected to a node which was opened by iteration i), and cannot revoke the decision about nodes which were considered before the current iteration (i.e., cannot open a facility it did not open in a previous iteration, and cannot close a facility that it opened in a previous iteration).

For dominating set, we propose the following definition of a greedy priority algorithm. Let v be the vertex considered in the current iteration. Then v will be accepted **if it is adjacent to at least two vertices which have not been considered yet and which are not dominated by vertices accepted thus far**. We emphasize that in the case where the above condition does not hold the algorithm may or may not open v; in other words no restrictions are placed on the algorithm in situations other than the one we described earlier. This is important since we do not want a definition that forces the algorithm to open a large number of vertices, because the lower bounds then become artificial. For instance, consider the situation where v is adjacent to only one yet unconsidered and undominated vertex u, and furthermore v is adjacent to no other undominated

vertices. Then it would make sense for the algorithm to reject v, wait until u is considered in a subsequent iteration, and then accept u; this would not increase the total cost of the algorithm.

3   Applications: Complete Facility Location and Dominating Set

Problem definitions. In the *(uncapacitated, unweighted) facility location* problem, the input consists of a set $\mathcal{F}$ of facilities and a set $\mathcal{C}$ of cities. The set $\mathcal{N} = \mathcal{F} \cup \mathcal{C}$ corresponds to the set of *nodes* in the graph, i.e., a node can be either a facility or a city, or both. Each facility $i \in \mathcal{F}$ is associated with an *opening cost* f_i which reflects the cost that must be paid to utilize the facility. Furthermore, for every facility $i \in \mathcal{F}$ and city $j \in \mathcal{C}$, the non-negative *distance* or *connection cost* c_{ij} is the cost that must be paid to connect city j to facility i. In the version of the problem that is known as the *complete facility location* problem, we have $\mathcal{F} = \mathcal{C} = \mathcal{N}$, i.e., every node is both a facility and a city. The objective is to open the facilities at a subset of the nodes and connect every other node to some open facility so that the total cost incurred, namely the sum of the cost of open facilities and the total connection cost is minimized[4]. We shall focus exclusively on the *metric* version of the problem, in which the connection costs satisfy the triangle inequality.

Observe that from the point of view of algorithms (upper bounds), the disjoint version (namely the version in which $\mathcal{F} \cap \mathcal{C} = \emptyset$) subsumes the complete version. This is because we can always "split" a node in the complete version of the problem to a corresponding facility and a corresponding city at zero distance from each other. However, we emphasize that the lower bounds for priority algorithms for the disjoint version (see [2]) do not carry over to the complete version.

In the *dominating set* problem, the input is an undirected, unweighted graph $G = (V, E)$. We seek a set $V' \subseteq V$ of smallest cardinality such that every vertex $u \in V \setminus V'$ is adjacent to at least one vertex in V'.

3.1   Complete (Metric) Facility Location

The first constant-factor polynomial-time approximation algorithm for (metric) facility location was given by Shmoys, Tardos and Aardal [16]. Interestingly, the best-known approximation ratio (1.52) is due to Mahdian, Ye and Zhang [13], and is achieved by a priority algorithm. Other algorithms that follow the priority framework include the ADAPTIVE PRIORITY greedy algorithm of Mahdian, Markakis, Saberi and Vazirani [12], which is a 1.861-approximation algorithms, and the ADAPTIVE PRIORITY algorithm of Jain, Mahdian and Saberi [9]. On the other hand, Mettu and Plaxton [14] showed that an algorithm which belongs in the class FIXED PRIORITY greedy yields a 3-approximation for the

---

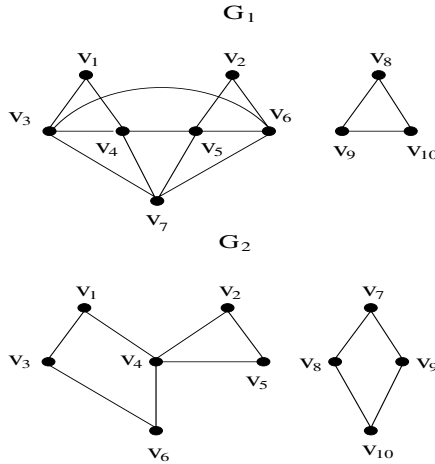[4] We say that a node is *opened*, when the facility on the said node is opened.

Fig. 2. The graphs G_1 and G_2 for the proof of Theorem 1. The edges shown indicate edges of distance 1, while all other distances are equal to 2

problem. It should be mentioned that their algorithm is in fact a re-statement of a primal-dual algorithm due to Jain and Vazirani [10].

On the negative side, Guha and Khuller [8] have shown that the *disjoint* facility location problem is not approximable within a factor better than 1.463, unless $NP \subseteq DTIME(n^{O(\log \log n)})$. They also showed, that under the same complexity assumption *complete* facility location is not approximable within a factor better than 1.278. Negative results for facility location priority algorithms in the disjoint model were given by Angelopoulos and Borodin in [2]. In particular, they showed lower bounds of 4/3 and 1.463 for general ADAPTIVE PRIORITY algorithms and *memoryless* ADAPTIVE PRIORITY algorithms respectively, as well as a lower bound of 3-ϵ for FIXED PRIORITY *greedy* algorithms.

In the lower-bound constructions shown in this Section we say that a node v *covers* a set $U \subset \mathcal{N}$ of nodes if and only if the distance from v to every node in U is equal to 1. The vertices of the graphs in our constructions will correspond to the nodes of the facility location instance.

Theorem 1. *No* ADAPTIVE PRIORITY *(not necessarily greedy) algorithm is better than an α-approximation, where α is a constant slightly greater than 36/35.*

Proof. We will use instances that consist of 10 nodes. Every node will cover either 2 or 4 facilities, and its facility cost will be denoted by f_2 and f_4, for the two cases, respectively. Here, f_2 and f_4 are suitably chosen constants (in particular, the 36/35 bound is obtained for $(f_2, f_4) = (3, 4.5)$).

Initially the adversary presents graph G_1, shown in Figure 2. We consider the following cases:

Case 1: The algorithm gives highest priority to an f_2 node, which the adversary specifies to be v_1, and the algorithm does not open it. In this case the input to the algorithm is G_1 itself (no transformation takes place).

Case 2: The algorithm gives highest priority to an f_2 node, (again, assume it to be v_1) which it opens. The adversary will perform an order-preserving transformation to derive graph G_2 also shown in Figure 2.

The remaining two cases, namely when the algorithm considers first a f_4 node, and either opens it or does not open it, are symmetric to Cases (1) and (2). In particular, in the case where the algorithm opens the f_4 node, the input to the algorithm will be graph G_1, while in the event it does not open it, the input is graph G_2. To complete the proof, it suffices to optimize with respect to f_2 and f_4.   □

Using ideas similar to Theorem 1, we can show the following:

Theorem 2. *There exists $\beta > \alpha$, where α is the lower bound of Theorem 1 such that no* FIXED PRIORITY *(but not necessarily greedy) algorithm achieves an approximation ratio better than β. In particular, β is slightly greater than 34/33.*

The lower bound we show in the following theorem matches the upper bound of Jain and Vazirani [10] and Mettu and Plaxton [14] (which it is easy to show that they belong in the class FIXED PRIORITY greedy).

Theorem 3. *No* FIXED PRIORITY greedy *algorithm has an approximation ratio better than $3 - \epsilon$ for arbitrarily small ϵ.*

Proof. The adversary presents to the algorithm a graph $G = (V, E)$, with $V = \{v_1, v_2, \ldots v_k, u_1, u_2, \ldots u_l\}$ and $l = c + d(k-1)$. Here, c and d are large constants, such that $c >> d^2$ (and whose importance will become evident later). Note that the total number of nodes in the graph is $n = k + c + d(k-1) = k(d+1) + c - d$; that is, n is a linear function of k. Each v_i, with $i \in [k]$ is at distance 1 from nodes $u_{d(i-1)+1} \ldots, u_{c+d(i-1)}$, and has a facility cost equal to $2d - \epsilon'$, where ϵ' is infinitesimally small. The distance between any two nodes v_i, v_j as well as the distance between any two nodes u_i, u_j is equal to 2. Every other distance is equal to 3. The facility cost of every u_i node is infinite (arbitrarily large). Figure 3 illustrates G for $k = 5$, $d = 2$, $c = 4$, with edges denoting distances equal to 1.

Lemma 2. $cost(OPT) \leq (2d^2/c + d + 3)k + O(1)$ *and* $cost(ALG) \geq k(3d - \epsilon')$ *for arbitrarily small ϵ'.*

The lower bound on the approximation ratio follows directly from Lemma 2. As k grows to infinity, and for large constants c, d, with $c >> d^2$, it is easy to verify that $cost(ALG)/cost(OPT) \geq 3 - \epsilon$, for arbitrarily small ϵ.   □
For completeness we mention that a simple argument can be used to show the following theorem.

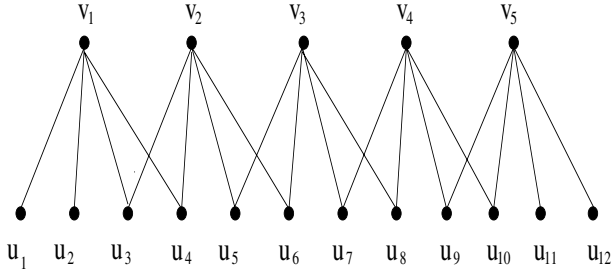**Theorem 4.** *No* ADAPTIVE PRIORITY greedy *algorithm is better than a 10/9-approximation.*

Fig. 3. Input for the proof of Theorem 3, for the case $k = 5$, $c = 4$, $d = 2$. The edges shown correspond to edges of distance 1 only

3.2   Dominating Set

Dominating set is known to be equivalent to Set Cover under L-reductions [3]. This result implies that dominating set is not approximable within a factor better than $(1 - \epsilon) \ln n$ unless $NP \subseteq DTIME(n^{O(\log \log n)})$, by a result due to Feige [7]. Here, n is the number of vertices in the graph. The well known greedy algorithm for set cover [11] can in fact be applied to yield a $\Theta(\log(n))$-approximation priority algorithm for dominating set, which is also greedy (as defined in Section 2.3, since the algorithm will always accept vertices as long as undominated vertices remain). Note that in [2] it was shown that no priority algorithm for set cover is better than $\ln n - o(\ln n)$-approximation, however the result does not carry over to dominating set.

Theorem 5. *Every* ADAPTIVE PRIORITY greedy *algorithm has approximation ratio* $\Omega(\log n)$ *where n is the number of vertices in the graph.*

Proof. The adversary presents the graph G, defined as follows. There is a set V of $k = 2^m$ triangles (3-cliques) of vertices v_l^1, v_l^2, v_l^3, with $l \in \{0, \ldots, k-1\}$, for some integer m. We call triangle $\{v_l^1, v_l^2, v_l^3\}$ *triangle l*, and we say that a vertex *is adjacent to triangle l* if and only if it is adjacent to all three vertices of triangle l. In addition, there is a set U of $m + 1 = \log k + 1$ vertices $u_1, \ldots, u_{m+1}$ with the following property: vertex u_j, with $j \in [m]$ is adjacent to all triangles i for which the binary representation of i has the j-th most significant bit equal to 0. Vertex v_{m+1}, on the other hand, is adjacent to all triangles whose least significant bit is equal to 1. We call vertices $u_i, u_j \in U$ *complementary* if and only if every triangle is adjacent to one of u_i, u_j. Note that u_m and u_{m+1} are the only complementary vertices in G. Figure 4 illustrates G for the case $m = 3$ (for the sake of clarity we substituted the triangles by filled-in nodes). We remind the reader that the u_i's and v_j's play the role of the id's. Note that the total number of vertices in G is $\Theta(k)$. Then OPT has a cost of at most 3, since it suffices to accept vertices u_m, u_{m+1}, v_0^1 in order to dominate every vertex in G.

Consider the class $\mathcal{A}$ of priority algorithms with the following statement. On input G, every algorithm $A \in \mathcal{A}$ works in rounds, with a round consisting of several iterations. In particular, in the beginning of round j, A considers and
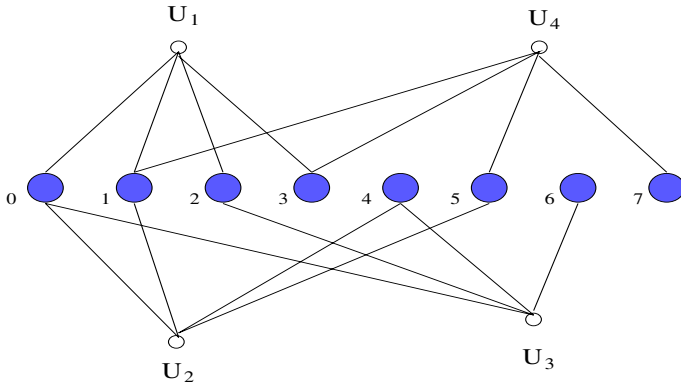
Fig. 4. The graph G for the proof of Theorem 5. Each filled-in node j corresponds to a triangle, and an edge between a vertex u_i and a triangle j indicates that all three vertices v_j^1, v_j^2, v_j^3 that comprise triangle j are adjacent to u_i

accepts a vertex in U. In subsequent iterations during round j, A considers (and possibly accepts) certain vertices in V, all of which are dominated by the set of vertices in U which were accepted by A in rounds $1, \ldots, j$. Round j ends (and round $j+1$ begins) when A considers and accepts a new vertex in U.

We shall focus on $\mathcal{A}$ since it is easier to lower-bound the cost of algorithms in this class. First, we show that the cost of every greedy priority algorithm is lower-bounded (within a factor of $1/2$) by the cost of some algorithm in $\mathcal{A}$.

Lemma 3. *Let A' be a greedy priority algorithm. Then there is an algorithm $A \in \mathcal{A}$ such that on input G $cost(A') \geq (1/2) \cdot cost(A)$.*

It now suffices to prove the following lemma. Interestingly, we will show that permuting the id's of the vertices is sufficient for the adversary to force a logarithmic bound on the approximation ratio.

Lemma 4. *Let A be an algorithm in class $\mathcal{A}$. For every $j \leq m$ there exists a graph G_j isomorphic to G such that on input G_j the adversary can force A to consider and accept j vertices in U no two of which are complementary in the first j rounds of A.*

The theorem follows from Lemma 3 and Lemma 4, and the observation that on input G (or any graph isomorphic to G) no algorithm in $\mathcal{A}$ is correct unless it accepts a pair of complementary vertices. ☐

References

1. S. Angelopoulos. Randomized priority algorithms. In *Proceedings of the 1st International Workshop on Approximation and Online Algorithms*, pages 27–40, 2003.

2. S. Angelopoulos and A. Borodin. On the power of priority algorithms for facility location and set cover. In *Proceedings of the 5th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*, pages 26–39, 2002.

3. R. Bar-Yehuda and S. Moran. On approximation problems related to the independent set and vertex cover problems. *Disc. Appl. Math*, 9:1–10, 1984.

4. A. Borodin, J. Boyar, and K. Larsen. Priority algorithms for graph optimization problems. These proceedings.

5. A. Borodin, M. Nielsen, and C. Rackoff. (Incremental) priority algorithms. *Algorithmica*, 37:295–326, 2003.

6. S. Davis and R. Impagliazzo. Models of greedy algorithms for graph problems. In *Proceedings of the 15th ACM-SIAM Symposium on Discrete Algorithms*, 2004.

7. U. Feige. A threshold of ln n for approximating set cover. *Journal of the ACM*, 45(4):634–652, 1998.

8. S. Guha and S. Khuller. Greedy strikes back: Improved facility location algorithms. In *Proceedings of the 9th ACM-SIAM Symposium on Discrete Algorithms*, pages 649–657, 1998.

9. K. Jain, M. Mahdian, and A. Saberi. A new greedy approach for facility location problems. In *Proceedings of the 34th Annual ACM Symposium on Theory of Computation*, pages 731–740, 2002.

10. K. Jain and V.V. Vazirani. Approximation algorithms for metric facility location and k-median problems using the primal-dual schema and lagrangian relaxation. *Journal of the ACM*, 48(2):274–296, 2001.

11. D.S. Johnson. Approximation algorithms for combinatorial problems. *JCSS*, 9(3):256–278, 1974.

12. M. Mahdian, E. Markakis, A. Saberi, and V. V. Vazirani. A greedy facility location algorithm analyzed using dual fitting. In *Proceedings of the 4th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*, pages 127–137, 2001.

13. M. Mahdian, J. Ye, and J. Zhang. Improved approximation algorithms for metric facility location problems. In *Proceedings of the 5th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems (APPROX)*, pages 229–242, 2002.

14. R. R. Mettu and C. G. Plaxton. The online median problem. In *Proceedings of the 41st Annual IEEE Symposium on Foundations of Computer Science*, pages 339–348, 2000.

15. O. Regev. Priority algorithms for makespan minimization in the subset model. *IPL*, 84(3):153–157, 2002.

16. D.B. Shmoys, E. Tardos, and K. Aardal. Approximation algorithms for facility location problems (extended abstract). In *Proceedings of the 29th Annual ACM Symposium on Theory of Computing*, pages 265–274, 1997.