Priority Algorithms for Graph Optimization Problems *

Allan Borodin
University of Toronto
bor@cs.toronto.edu

Joan Boyar † and Kim S. Larsen *
University of Southern Denmark, Odense
{joan,kslarsen}@imada.sdu.dk

Nazanin Mirmohammadi
University of Toronto
nazanin@cs.toronto.edu

December 12, 2007

Abstract

We continue the study of priority or "greedy-like" algorithms as initiated in [10] and as extended to graph theoretic problems in [12]. Graph theoretic problems pose some modeling problems that did not exist in the original applications of [10] and [3]. Following [12], we further clarify these concepts. In the graph theoretic setting there are several natural input formulations for a given problem and we show that priority algorithm bounds in general depend on the input formulation. We study a variety of graph problems in the context of arbitrary and restricted priority models corresponding to known "greedy algorithms".

* A preliminary version of this paper appeared in the *Second Workshop on Approximation and Online Algorithms*, Lecture Notes in Computer Science, vol. 3351, pages 126–139, Springer-Verlag, 2005.

† Partially supported by the Danish Natural Science Research Council (SNF) and the IST Programme of the EU under contract number IST-1999-14186 (ALCOM-FT).

1 Introduction

The concept of a greedy algorithm was explicitly articulated in a paper by Edmonds [14] following a symposium on mathematical programming in 1967, although one suspects that there are earlier references to this concept. Since that time, the greedy algorithm concept has taken on a broad intuitive meaning and a broader set of applications beyond combinatorial approximation. The importance of greedy algorithms is well motivated by Davis and Impagliazzo [12] and constitutes an important part of many texts concerning algorithm design and analysis. New greedy algorithms keep emerging, as, for instance, in [25], which considers mechanisms for combinatorial auctions, requiring solutions to difficult optimization problems. Given the importance of greediness as an algorithm design "paradigm", it is somewhat surprising that a rigorous framework, as general as priority algorithms, for studying greedy algorithms is just emerging. Of course, the very diversity of algorithms purported to be greedy makes it perhaps impossible to find one definition that will satisfy everyone. The goal of the priority algorithm model is to provide a framework which is sufficiently general so as to capture "most" (or at least a large fraction) of the algorithms we consider to be greedy or greedy-like while still allowing good intuition and rigorous analysis, e.g., being able to produce results on the limitations of the model and ultimately suggesting new algorithms.

The priority model has two forms, fixed priority and the more general adaptive priority model. The general form of fixed and adaptive priority algorithms is presented in Figures 1 and 2. To make this precise, for each specific problem we need to define the nature and representation (the type) of the input items and the nature of the allowable (irrevocable) decisions. Surprisingly, the issue as to what orderings are allowed has a rather simple and yet very inclusive formalization. Namely, the algorithm can use any total ordering on some sufficiently large set of items from which the actual set of input items will come. (For adaptive algorithms, the ordering can depend on the items already considered.) The priority framework was first formulated in Borodin, Nielsen and Rackoff [10] and applied to (worst case approximation algorithms for) some classical scheduling problems such as Graham's makespan problem and various interval scheduling problems. In a subsequent paper, Angelopoulos and Borodin [3] applied the framework to the set cover and uncapacitated facility location problems. These problems were formulated so that the data items were "isolated" in the sense that one data item did not refer to another data item and hence any set of valid data items constituted a valid input instance. For example, in the makespan problem on identical machines with no precedence constraints, a data item is represented by a processing time and the items are unrelated. The version of facility location studied in [3] was for the

Determine a total ordering of all possible input items
(without knowing the actual input set S of items)
While S is not empty
    $next$:= index of input item I in S that comes first in the ordering
    Make an irrevocable decision concerning I_{next} and remove I_{next} from S
End While

Figure 1: The form of a fixed priority algorithm

While S is not empty
    Determine a total ordering of all possible remaining input items
    (without knowing the input items in S not yet considered)
    $next$:= index of input item I in S that comes first in the ordering
    Make an irrevocable decision concerning I_{next} and remove I_{next} from S
End While

Figure 2: The form of an adaptive priority algorithm

"disjoint model" where the set of facilities and the set of clients/cities are disjoint sets and a facility is represented by its opening cost and a vector of distances to each of the cities. In contrast, in the "complete model" for facility location, there is just a set of cities and every city can be a facility. Here a city is represented by its opening cost and a vector of distances to every other city. In the complete model for facility location, an input item (a city) directly refers to other input items. This is similar to the standard situation for graph theoretic problems when vertices are, say, represented by adjacency lists. Angelopoulos [2] studies the facility location problem in the complete model.

The work of Davis and Impagliazzo [12] extends the priority formulation to graph theoretic problems. They consider a number of basic graph theory problems (single source shortest path, weighted vertex cover, minimum spanning tree, Steiner trees, maximum independent set) with respect to one of two different input formulations depending on the problem and known "greedy algorithms". For the shortest path, minimum spanning tree and Steiner tree problems, the formulation used is the "edge model", where input items are edges represented by their weights, the names of the endpoints, and in the case of the Steiner tree problem by the types (required or Steiner) of the edge endpoints. Note that in this edge representation, input items are isolated and all of the definitions in [10] can be applied. In particular, the definition of a greedy decision is well defined. In contrast, for the weighted vertex cover and maximum independent set problems, Davis and Impagliazzo use

a vertex adjacency list representation, where input items are vertices, represented by the names of the vertices to which they are adjacent, and in some problems also the weight of the vertex. This representation presents some challenges for defining priority algorithms and greedy decisions. These definitional issues have helped to clarify the nature and usefulness of "memoryless priority algorithms".

Davis and Impagliazzo provide a formal model of an algorithm-adversary game which gives a precise definition of the adversarial model used to derive inapproximation results. Angelopoulos [2] refers to this as the DI adversary and he introduces a stronger (but still reasonable in terms of many existing algorithms) adversary which intuitively ensures that "input item id's do not carry information". The Angelopoulos adversary is defined for both fixed and adaptive priority. For our purposes, we need only describe this adversary in terms of the fixed priority model. In the context of complete weighted graphs, whenever two vertices have the same multiset of edge weights, they must be given equal priority and the adversary is then entitled to break this tied priority however it wishes. (For adaptive algorithms, intuitively one has to say whether or not the history thus far can distinguish two unseen vertices.) In the context of unweighted graph problems and fixed priority algorithms as considered in this paper, the Angelopoulos adversary does not let an algorithm's priority distinguish between vertices that have the same degree. We call this the *degree based model*. In the weighted complete graph case, Angelopoulos proves lower bounds for the complete facility location problem (for both fixed and adaptive priority algorithms) and the dominating set problem (for the more general adaptive priority algorithms). It is not clear if Angelopoulos' adaptive priority results can be obtained in the DI adversary model, but even if they can, this simple restriction on priority algorithms certainly makes it easier to derive lower bound proofs. We present a number of inapproximation results in the degree based fixed priority model by using graph constructions involving regular graphs. We believe that all these results (using the same constructions) also hold for the DI model but will require much more delicate arguments.

In this paper, we continue the study of priority algorithms for graph problems using several combinations of priority algorithm models and input formulations. Motivated by current algorithms, we consider two basic input formulations: the *vertex adjacency formulation* as in Davis and Impagliazzo, and an *edge adjacency formulation*, where input items are vertices now represented by a list of adjacent edge names (rather than a list of adjacent vertex names) and possible vertex weights where appropriate. It should be clear that any priority algorithm in the edge adjacency formulation can be simulated in the vertex adjacency formulation (making exactly the same set of decisions). In contrast to fixed priority algorithms, most existing adaptive priority algorithms can function in the edge adjacency formulation;

the authors are unable to recall one which does not. However, we prove (using an example in [12] for showing that memorylessness is restrictive) that the edge adjacency formulation can be restrictive even for adaptive algorithms. For fixed priority algorithms, there is a natural problem which seems to differentiate the vertex and edge adjacency formulations. Namely, as we will see, the simple 2-approximation algorithm for the vertex cover problem that is achieved by the end points of a maximal matching algorithm can be realized as a fixed priority algorithm in the vertex adjacency formulation. However, while the same algorithm can be realized as an adaptive algorithm in the edge adjacency formulation, we do not believe a fixed priority algorithm can achieve any O(1) approximation for vertex cover in the edge adjacency formulation. (See section 3.) We also introduce an "acceptances-first" model and clarify the relation of memoryless algorithms to this "acceptances-first" model rather than to greediness.

With respect to fixed priority algorithms, we study the more restrictive degree based model, where the priority given to an input item (vertex) is a function of *only* the degree of the vertex. Here we can assume that an adversary can choose how to break ties amongst vertices having the same degree. While it may seem that a fixed priority algorithm cannot utilize any information about adjacent node/edge names when assigning priorities, it is not hard to see that such an algorithm can ensure that vertices are considered in adjacent pairs. However, intuitively this seems like the only additional power such a fixed priority algorithm has when compared to a degree based fixed priority algorithm. We have not been able to formalize this intuition for our constructions, but we conjecture that our inapproximation bounds concerning degree based fixed priority algorithms hold for arbitrary fixed priority algorithms in the edge adjacency formulation.[1] We should also note that for regular graphs, it may seem that a degree based fixed priority algorithm is essentially just an online algorithm as the adversary has complete control over the order in which the algorithm considers the inputs. However, in the usual online model for graph problems, when an input vertex v is provided to the algorithm, the algorithm only gets to know those vertices adjacent to v which have previously been input whereas in the degree based fixed priority model, the algorithm gets to know the entire list of adjacent vertices (or edges). We note that in the online model, the edge and vertex adjacent formulations are equivalent. It follows that all of our degree based lower bounds apply to online algorithms.

Before we conclude, we also provide some further comments on the definition of greediness and give examples related to the graph problems discussed in this paper.

[1] The construction in Theorem 8 for the independent set problem allows the algorithm to assign priorities so that vertices are enumerated in adjacent pairs.

2 Priority Algorithms for Graph Problems

As mentioned in the introduction, we consider two input formulations. In the common vertex adjacency formulation, an input item is a vertex, represented by the tuple $(v, w, v_1, v_2, \ldots, v_d)$, where v is the name of the vertex, w is the weight (if any) of vertex v and $v_1, \ldots, v_d$ is a list of adjacent vertices. In the more restrictive edge adjacency formulation (but still a formulation sufficient to capture many known greedy graph algorithms), an input item is a vertex $(v, w, e_1, e_2, \ldots, e_d)$ where again v is the vertex name, w is the weight (if any) of v and $e_1, e_2, \ldots, e_d$ is a list of incident edges.

In either of the above input formulations, we have the situation that not every set of valid input items constitutes a valid input instance. Clearly, a valid input instance cannot have the same vertex appear as two different items. And in the vertex adjacency formulation, if a vertex v is an input item and v' is in its adjacency list, then v' must also be an input item with v in its adjacency list. Similarly, in the edge adjacency formulation, if an edge e appears in some input item, then e must appear in exactly one other input item. Although the priority algorithm framework is designed to model greedy algorithms, it is possible to define priority algorithms where the irrevocable decisions do not seem greedy. As noted by Davis and Impagliazzo, the definition of "greedy decision" (as formulated in [10]) is no longer well defined when the algorithm "knows" that the current item is not the last. More specifically, in [10], a greedy priority algorithm is one in which all of the irrevocable decisions are "greedy" in the sense that the algorithm acts as if the current item being considered is the last item in the input. In more colloquial terms, greediness is defined by the motto "live for today". We would like to formulate a general concept of a greedy decision that also makes sense when the input items are not isolated. (We would like such a definition to also make sense for non-graph problems, such as scheduling problems with precedence relations amongst the jobs, where one can have non-isolated input items.) We offer one such definition in Section 6. We note, however, that in the context of priority algorithms the greedy versus non greedy distinction is not that important, and to the extent that it is important it is only because greedy is such a commonly used (albeit mostly undefined) concept. We do argue that the priority algorithm formulation is important as it captures a wide variety of existing algorithms which might be called "greedy-like" extending the concept of greedy and including (for example) all online algorithms.

One can always make an ad hoc definition of a greedy decision in the context of any given problem. For example, for the vertex coloring problem, one might define a greedy decision to be one that never assigns a new color to a vertex if an existing color could be used *now*. But for a given input and history of what has been seen, it

may be known to the algorithm that any valid completion of the input sequence will force an additional color and it might be that in such a case one would also allow a new color to be used before it was needed. This can, of course, all be considered as a relatively minor definitional issue and one is free to choose whatever definition seems to be more natural and captures known "greedy algorithms".

Perhaps a more meaningful distinction is the concept of "memoryless" priority algorithms. Although motivated by the concept of memoryless online algorithms, especially in the context of the k-server problem, the concept of memorylessness takes on a somewhat different meaning as applied in [10] and [3]. Namely these papers apply the concept to problems where the irrevocable decision is an accept/reject decision (or at least that acceptance/rejection is part of the irrevocable decision). In this context memoryless priority algorithms are defined as priority algorithms in which the irrevocable decision for the current item (and the choice of next item in the case of adaptive algorithms) depends only on the set of previously accepted items. That is, in the words of [12], a rejected item is treated as a NO-OP. In the accept/reject context, memoryless adaptive algorithms are "essentially" equivalent to *acceptances-first* adaptive algorithms which do not accept any items after the first rejected item. As observed[2] in [10] and [3], we have the following:

Theorem 1 Let $\mathbb{A}$ be a memoryless priority algorithm for a problem with accept/reject decisions. Then there exists an "acceptances-first" adaptive priority algorithm $\mathbb{A}'$ that "simulates" $\mathbb{A}$ in the sense that it accepts the same set of items and makes the same irrevocable decisions.

We observe that many graph theoretic algorithms called greedy may or may not satisfy some generic general definition of greedy. But many of these algorithms (e.g. the maximal matching algorithm for vertex cover when realized as an adaptive priority algorithm) are indeed acceptances-first algorithms and thus memoryless. In terms of a converse for the above theorem, it is obvious that an acceptances-first algorithm can be simulated by a "1-bit algorithm", where this one bit is used to remember whether or not a rejection has already occurred.

To prove negative results, showing that no priority algorithm in some model can achieve an approximation ratio better than ρ for a given problem P, we use an adversary. The adversary initially chooses a set S of valid input items. It interacts with an algorithm $\mathbb{A}$, maintaining the invariant that the items remaining in S,

[2] In [10], this fact is stated in terms of memoryless algorithms being simulated by greedy algorithms, but the essence of that observations really concerns the acceptance-first restriction and not greediness.

together with the items already selected by $\mathbb{A}$, contain at least one valid input instance. At each step, $\mathbb{A}$ selects its highest priority item i remaining in S and makes an irrevocable decision regarding item i; the adversary then removes item i from S and may also remove more items from S at this point, as long as the invariant is maintained.

In most cases the initial set S contains multiple copies of each vertex and possibly additional vertices than occur in the final input graph. After the algorithm chooses a vertex, the adversary removes the other copies of that vertex from S, since its adjacency list is now determined. An adaptive priority algorithm in the edge-adjacency formulation knows the names of the edges adjacent to the vertices already chosen, so it can give vertices with the same edges in their lists either high or low priority. The adversary may still have more than one copy of the neighbors at this time, though. In the vertex adjacency formulation, an adaptive priority algorithm has even more power; it can give the neighbors high or low priority and it can also give the neighbors of the neighbors high or low priority, since it knows the names of the neighbors. Although the adversary may still retain multiple copies of the neighbors, it (intuitively) cannot make arbitrary decisions as to whether or not a vertex chosen by the algorithm is or is not at distance at most two from any chosen vertices.

For some scheduling results in [10], the adversary assumes that the algorithm does not know (or use information concerning) the final number of jobs to be processed. The same holds here for graph problems; in some cases the adversary creates final input graphs that have different sizes for different algorithms. In practice, most priority algorithms do not seem to use the total number of vertices or edges in the graph in assigning priorities or in making the irrevocable decisions, so the results based on adversaries of this type are widely applicable. Unless otherwise stated, the results below assume the algorithm does not know the total number of vertices n or edges m in the graph.

3 Vertex Cover

Minimum Vertex Cover is the problem of finding a smallest subset C of vertices such that all edges are incident to some vertex in C.

This unweighted vertex cover problem is one of the most celebrated open problems in the area of worst case approximation algorithms. As previously mentioned, the simple maximal matching algorithm (taking both adjacent vertices in any maximal matching) provides a 2-approximation. This is essentially the best known polynomial time approximation bound in the sense that there are no known poly-

nomial time $2-\epsilon$ approximation algorithms (for a fixed $\epsilon > 0$), although various algorithms are known which for certain classes of graphs guarantee an approximation better than 2, but converging to 2 as some parameter grows. This maximal matching algorithm provides illustrative examples of priority algorithms. We show below how to implement it both as a fixed priority algorithm in the vertex adjacency formulation and as an acceptances-first adaptive priority algorithm in the edge adjacency formulation. Both implementations are memoryless.

Surprisingly, Johnson [21] showed that the greedy algorithm which chooses the vertex with highest degree in the remaining graph is only an H_n-approximation, and that this bound is tight in that there are arbitrarily large graphs on which the algorithm produces a vertex cover whose size is H_n times the size of the optimal cover. Thus, this adaptive priority algorithm is inferior to the maximal matching algorithm. However, the "list processing algorithm", which is a fixed priority algorithm that simply takes the vertices in non-increasing order of their degree in the original graph, accepting a vertex if any of its edges is still uncovered, is even worse. Avis and Imamura [5] show that any list processing algorithm which orders the vertices based on degrees only (i.e. degree based fixed priority) has an approximation ratio of at least $\Omega(\sqrt{n})$. We show below that their result also applies to all acceptances-first fixed priority algorithms in the edge adjacency formulation. Note that list processing is not the same as acceptances-first, since a list processing algorithm for vertex cover will reject any vertex if all of its edges are already covered, and such a vertex might appear in the ordering before other vertices which will be accepted.

Davis and Impagliazzo [12] show that for the weighted case, no priority algorithm (in the vertex adjacency formulation) can achieve a $(2-\varepsilon)$-approximation ratio, for any $\varepsilon > 0$. Although the weighted vertex cover problem can essentially be reduced (in polynomial time) to the unweighted case (by making multiple copies of vertices), this reduction does not preserve the property of being a priority algorithm and hence the study of the unweighted and weighted vertex cover problems may be substantially different problems in the context of priority algorithms. It turns out that there are several priority algorithms for the weighted case that also achieve a 2-approximation algorithm (or slightly better). One such algorithm is the "layered algorithm" as given in [29]. This algorithm chooses all maximum (current) degree vertices and removes them simultaneously. Another simple to state (and also called greedy) algorithm is given by Clarkson [11]. This algorithm achieves the approximation bound $\frac{\Delta}{\Delta-2}(2-\frac{2n}{\Delta \cdot OPT})$, where Δ is the maximum degree in the graph and n is the number of vertices[3]. Both the layered algorithm and Clarkson's algorithm

[3] The stated bound is not defined for $\Delta \leq 2$. The more general bound that applies to all Δ is that

can be expressed as acceptances-first adaptive algorithms in the edge adjacency formulation. Below, we prove a $\frac{4}{3}$ lower bound on the approximation achievable by any priority algorithm. This matches the upper bound by Clarkson for the case $n = 7$, $\Delta = 3$, and $OPT = 3$.

In addition to priority algorithms, linear programming relaxation techniques have proven useful in designing approximation algorithms for vertex cover. Arora et al. [4] have shown an integrality gap of $2 - o(1)$ for three different families of linear relaxations for vertex cover, implying that many linear programming based algorithms cannot obtain an approximation ratio better than 2.

In terms of complexity based inapproximation bounds, Dinur and Safra [13] show that it is NP-hard to have a c-approximation algorithm for the (unweighted) vertex cover problem for $c < 1.36$. Assuming the Unique Games Conjecture [23], Khot and Regev [24] show a very strong result, namely that the vertex cover problem has an approximation ratio of at least $2 - \epsilon$ for any $\epsilon > 0$. We note (as in previous papers concerning priority algorithms) that priority algorithm bounds are incomparable with complexity based bounds as priority algorithms can (in principle) utilize arbitrarily complex (and even non computable) functions in determining the priority of an item and the irrevocable decision being made about an item. Of course, in practice, priority algorithms tend to be very time efficient (as well as conceptually simple) and that is, of course, why they are so popular.

3.1 The maximal matching algorithm as a priority algorithm

First, we show the easier of two implementations of the maximal matching algorithm as a priority algorithm.

Theorem 2 The matching algorithm can be implemented as an acceptances-first adaptive priority algorithm in the edge adjacency formulation.

Proof Consider the following algorithm for an input graph $G = (V, E)$:

$C := \emptyset$;
while there are any vertices adjacent to an unmarked edge **do**
    Give the two vertices having some adjacent unmarked edge e highest priority;
    Let u be the first vertex in the ordering and v be the second;
    (do not reorder between the processing of these two)

$w(C_{MG}) \leq w(C_{OPT}) - \frac{2(n-w(C_{MG}))}{\Delta}$. Here, C_{MG} is the cover obtained by Clarkson's Modified Greedy algorithm and C_{OPT} is the cover obtained by OPT.

$C := C \cup \{u, v\}$; (accept u and v)
Mark all edges incident to u;
Mark all edges incident to v;
end while
Reject all remaining vertices.

Note that knowing the number of vertices and/or edges in advance is not necessary, since choosing a vertex next to an unmarked edge only requires knowing the universe for the set of edge names. We note that the algorithm can also be made memoryless in the sense of Davis and Impagliazzo [12]. While the marking of vertices is using memory beyond the set of accepted items, the algorithm knows the adaptive order it has been using and can recreate the entire computation at every iteration to know which edges would have been marked. The above acceptances-first algorithm clearly functions exactly as the maximal matching algorithm. □

Next, we consider a fixed priority implementation of the maximal matching algorithm, but to do this, we need to use the vertex adjacency formulation. In fact, an arbitrary ordering can be used and hence the algorithm can be viewed as an online algorithm (keeping in mind the distinction with traditional online graph-theoretic algorithms that are mentioned in the introduction). The algorithm maintains a list, C, of vertices already accepted and a list, L, initially empty, of vertices which it intends to accept. A vertex in this list has not been processed yet; it is the second vertex incident to some edge which has been chosen by the matching algorithm. This is possible because the algorithm knows which vertices are adjacent to already processed vertices. When the algorithm receives a vertex u from the ordering, it checks if $u \in L$ and accepts u if it is. If the vertex is not in L, it checks if all of its neighbors are in $C \cup L$ and rejects if they are. Otherwise, it accepts u and chooses a designated neighbor v not in $C \cup L$ and adds v to L. The edge (u, v) has thus been added to the matching.

Theorem 3 The maximal matching algorithm can be implemented as a memoryless fixed priority algorithm in the vertex adjacency formulation.[4]

Proof Consider the following algorithm, which was informally described above, for an input graph $G = (V, E)$:

Order the vertices in any way.

[4] This algorithm is memoryless according to the definition provided by Davis and Impagliazzo [12] but to make it acceptances first, the algorithm becomes adaptive.

$C := \emptyset$; $X := V$; $L := \emptyset$;
while X is not empty **do**
    Let u be the vertex with highest priority in X;
    if $u \in L$ **then**
        $C := C \cup \{u\}$; (accept u, the "2nd vertex" of an edge)
        $X := X \setminus \{u\}$;
        $L := L \setminus \{u\}$;
    else if u's adjacency list contains no vertex $v \notin C \cup L$ **then**
        $X := X \setminus \{u\}$; (reject u)
    else
        $C := C \cup \{u\}$; (accept u, the "1st vertex" of an edge)
        $X := X \setminus \{u\}$;
        Choose a vertex v in u's adjacency list, but not in $C \cup L$;
        $L := L \cup \{v\}$; (plan to accept v later)
end while

This is clearly a fixed priority algorithm in the vertex adjacency formulation, and it functions exactly as the maximal matching algorithm. While the algorithm does not seem memoryless in that it is remembering vertices in L, we argue (as we did for Theorem 2) that the algorithm can reconstruct the current list L by considering just the set of vertices in C. Again note that knowing the number of vertices and/or edges in the graph in advance is not necessary. □

It is instructive to consider why the argument in Theorem 3 does not extend to the edge adjacency formulation. Suppose we try to implement the maximal matching algorithm as an online algorithm (i.e., the algorithm does not determine the ordering of the input vertices) as in Theorem 3. Suppose that u is the vertex of highest priority and say edge e is its incident edge that we are using for the matching. Let v be the other vertex incident to e. Then while we can remember to include v in the vertex cover, we do not know until we see v which other edges (i.e., the edges adjacent to v) can be removed. Alternatively, the algorithm could order vertices so that vertices are seen in adjacent pairs as discussed in the introduction. Suppose the first adjacent pair of vertices is u and v_1, but the ordering of the edges is such that the next adjacent pair is the same u and v_2. If v_2 is adjacent to a vertex v_3, which has not been seen yet, then v_2 must be accepted, but now v_3 causes the same problems as the v in the online variant above.

3.2 Limitations on priority algorithms for vertex cover

First, we show that, although both an acceptances-first adaptive priority algorithm in the edge adjacency formulation and a memoryless fixed priority (in fact, online) algorithm in the vertex adjacency formulation can achieve an approximation ratio of 2, Theorem 4 below shows that this is impossible for an acceptances-first fixed priority algorithm in the edge adjacency formulation. In fact, the best obtainable ratio is $\Omega(\sqrt{n})$. Using the intuition following Theorem 3, we conjecture that Theorem 4 applies to *any* fixed order algorithm using the edge adjacency formulation. The proof below uses the construction in Avis and Imamura's proof [5] of the similar result[5] where they proved an $\Omega(\sqrt{n})$ inapproximation bound for any degree based list processing algorithm for vertex cover. Moreover, Avis and Imamura show that an $O(\sqrt{n})$ approximation is achievable by a list processing algorithm for vertex cover in the edge adjacency formulation.

Theorem 4 No acceptances-first fixed priority algorithm in the edge adjacency formulation for vertex cover can achieve an approximation ratio better than $\frac{k^2}{2k-1}$ on graphs with $n = k^2 + 2k - 1$ vertices, $k \geq 3$.

Proof We use a construction suggested by Avis and Imamura [5]: G is a bipartite graph with vertex sets U and V, where $|V| = 2k-1$ and $|U| = k^2$. V is partitioned into two subsets V_1 and V_2, where $|V_1| = k - 1$ and $|V_2| = k$. V_1 and U form a complete bipartite graph, so every vertex in V_1 is adjacent to every vertex in U. Every vertex in V_2 is adjacent to exactly k vertices of U, so every vertex of U is adjacent to exactly one vertex in V_2. All vertices in U and V_2 have degree k. See Figure 3.

We now decide on the total set of vertex labels and edge labels that we will use and we form a set of input items, where each item consists of one vertex label and a set of edge labels. We will give all possible combinations of vertex and edge labels where the size of the set of edge labels is at most k^2 Thus, there are many more input items than there are vertices in the graph. The fixed priority algorithm now creates an ordering of these input items. Recall that the adversary is at liberty to remove input items at the beginning of each round. Thus, we are interested in the relative ordering of the items we will actually use.

Now, think of the graph described above as a structure without any labels on vertices and edges. We will show how the labels from the input items can be written

[5] Although we use the Avis and Imamura construction, our Theorem 4 is incomparable with the Avis and Imamura result since they require a degree based ordering while we require acceptances-first. The list processing requirement is in essence a greedy requirement which says that we take any vertex as long as it covers an uncovered edge.

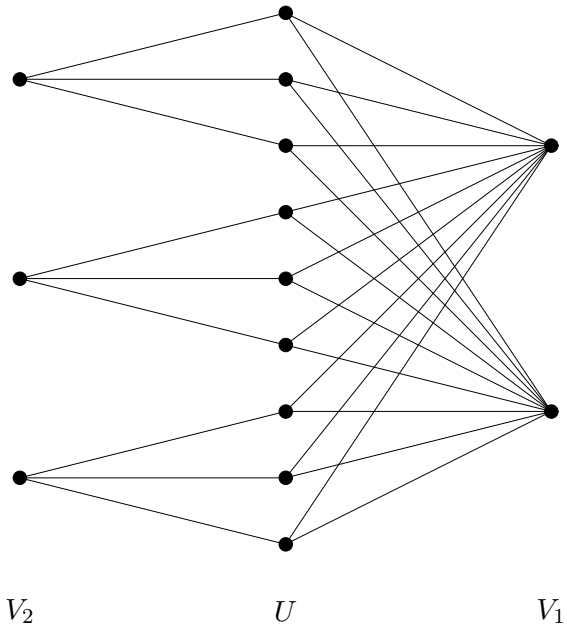

Figure 3: The Avis/Imamura construction for $k = 3$.

onto vertices and edges in the structure such that the fixed priority algorithm processes the vertices in the order giving rise to the lower bound. We start with an arbitrary labelling and gradually arrive at the desired one.

Our goal is to ensure that for all vertices $v \in V_2$, all neighbors of v (which are all in U) are processed and therefore accepted before v is processed. For the fixed priority algorithm, this will give rise to a vertex cover of size at least k^2, since the cover will contain all vertices in U. The optimal cover consisting of all vertices in V is of size $2k - 1$.

What remains is to demonstrate a labelling of the graph such that the described processing order is obtained. We start with an arbitrary labelling and obtain our desired labelling inductively by repeatedly modifying the current one.

Assume that the vertex labels of vertices in V_2 in the order decided by the fixed priority algorithm's ordering of input items are $v_1, v_2, \ldots, v_k$ and assume that we currently have a labelling such that for any $v \in \{v_1, v_2, \ldots, v_{i-1}\}$, $i \geq 1$, all neighbors of v come before v in the ordering given by the fixed priority algorithm.

Consider vertex v_i. Assume that the neighbors of v_i are named $u_1, u_2, \ldots, u_k$. Remove all vertex labels from v_i and $u_1, u_2, \ldots, u_k$ as well as all edge labels incident to any of these vertices. Consider all input items which can be formed using one of these vertex labels and k of these edge labels. Of all these items, choose the one which has been giving the lowest priority in the ordering made by the fixed priority algorithm, and assign these labels to v_i and its incident edges. Assign all the other removed labels back to vertices and edges in an arbitrary fashion.

We argue that now for any $v \in \{v_1, v_2, \ldots, v_i\}$, all neighbors of v come before v in the ordering given by the fixed priority algorithm.

Clearly, by the most recent relabelling it holds for v_i. Consider any v_j, $1 \leq j < i$. Refer to the set of vertices consisting of $v_i, u_1, u_2, \ldots, u_k$ and all neighbors of these vertices as the *active set*. Note that only vertices of the graph structure which are in the active set may have changed place in the ordering. Since vertices in U have only one neighbor in V_2, all neighbors of the vertices $u_1, u_2, \ldots, u_k$ other than v_i belong to V_1. Now, v_j clearly does not belong to the active set and any of the neighbors of v_j belong to U and cannot be a neighbor of v_i, so they do not belong to the active set either. Thus, the relative ordering of v_j and its neighbors is unchanged.

This completes the induction step and the proof. □

Note that restrictions on rejection, which both acceptances-first and list processing impose, appear to be necessary using the above construction to establish lower

bounds. Without these restrictions, an algorithm could give highest priority to the high degree vertices, accept them, reject all vertices adjacent to them and accept all other vertices to get the minimum vertex cover.

Removing the acceptances-first restriction from the previous result, we obtain a much weaker result, which we conjecture is not tight:

Theorem 5 No degree based fixed priority algorithm $\mathbb{A}$ in the edge adjacency formulation for vertex cover can achieve an approximation ratio better than 2.

Proof The adversary uses copies of the following construction, $\mathbb{G}$, which is a modification of a construction due to Hochbaum [19]:

Construction $\mathbb{G}$: There are two sets of vertices, U and V. The set U consists of k independent $(k+1)$-cliques, and the set V is an independent set consisting of k^2 vertices, each of which is adjacent to every vertex in every $(k+1)$-clique.

Note that all vertices in $\mathbb{G}$ have degree k^2+k. Thus, $\mathbb{A}$ cannot distinguish between the vertices when assigning priorities. The optimum vertex cover includes every vertex in U and has size k^2+k.

The adversary arranges that the selected vertices are independent during the first of the two phases. We let n' denote the number of vertices processed so far. The first phase continues until either $\mathbb{A}$ has rejected at least $c = \lceil \frac{n'}{k} \rceil$ vertices or $n' = k^2$; whichever happens first.

If the first phase stopped because at least c vertices were rejected, then the adversary creates c copies of the construction $\mathbb{G}$. There are enough cliques so that each of the n' vertices can be placed in distinct cliques in the copies of U, and the rejected vertices can be placed in separate copies of $\mathbb{G}$. This means that in each construction, all vertices in V must be accepted in the second phase. In addition, the algorithm must take at least k vertices in every clique in U. This gives a ratio of $\frac{k^2+k^2}{k^2+k} = \frac{2k}{k+1}$.

If the first phase stopped because $n' = k^2$, the adversary uses a single copy of the construction $\mathbb{G}$. The n' vertices are in V. Note that the number of rejected vertices is at most $\lceil \frac{k^2}{k} \rceil = k$, since otherwise the algorithm would have terminated for that reason. If any of the n' vertices are rejected, then everything in U must be accepted in the second phase, giving a total of $k^2 - k + k^2 + k = 2k^2$. Even if all the vertices in V are accepted, at least k vertices must be accepted from every clique in the second phase. This gives a total of at least $k^2 + k^2$. Thus, in both cases the ratio is at least $\frac{2k}{k+1}$. □

In contrat to vertices in U, the vertices in V have identical adjacency lists. Since this distinction can be detected in the vertex adjacency formulation, the above proof depends on the edge adjacency formulation.

The following lower bound applies to all priority algorithms for the vertex cover problem:

Theorem 6 No adaptive priority algorithm in the vertex adjacency formulation can achieve an approximation ratio better than $4/3$ for the vertex cover problem.

Proof First note that both graphs in Fig. 4 have vertex covers of size 3.

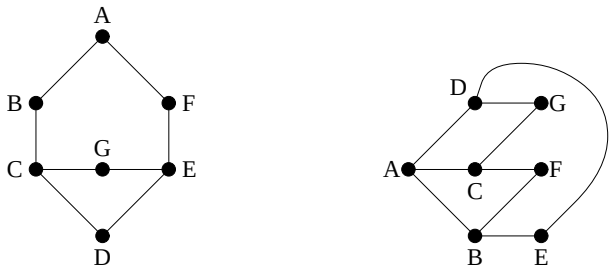

Figure 4: Graph 1 to the left and Graph 2 to the right.

We now force any adaptive priority algorithm $\mathbb{A}$ to choose at least 4 vertices.

In the first step, $\mathbb{A}$ must choose either a degree 2 or a degree 3 vertex, and it can choose to accept or reject. We treat these four cases.

If $\mathbb{A}$ rejects a degree 2 vertex first, we let it be vertex A in Graph 1. If $\mathbb{A}$ accepts a degree 2 vertex first, we let it be vertex B in Graph 1. If $\mathbb{A}$ rejects a degree 3 vertex first, we let it be vertex C in Graph 1. If $\mathbb{A}$ accepts a degree 3 vertex first, we let it be vertex A in Graph 2.

(As an example, Clarkson's algorithm would first accept a vertex of degree 3, so the adversary would give Graph 2. After accepting vertex A, the algorithm must accept at least three more vertices to cover all edges.) □

Note that the numbers of vertices in the two graphs used in the proof of the above theorem are the same, so the theorem holds true in a model where the algorithms know the number of vertices.[6]

[6] If the number of nodes and edges are both known to the algorithm, we can add a cycle of 4 new vertices to Graph 2 and a cycle of 4 new vertices with one diagonal to graph 1 and obtain a bound of $6/5$.

In addition, the results hold for arbitrarily large graphs, since disjoint copies of the constructions can be used.

In more restrictive models, we obtain stronger lower bounds.

Theorem 7 In the vertex adjacency formulation, no acceptances-first adaptive priority algorithm can achieve an approximation ratio better than $3/2$ for the vertex cover problem (even if the number of edges and vertices in the graph is known to the algorithm).

Proof Consider a chain of five vertices. In the acceptances-first model, the first vertex chosen (the one that initially gets highest priority) must be accepted. If the first vertex chosen has degree 1, at least two other vertices must be chosen to cover all the edges. If the first vertex chosen has degree 2, the adversary makes it the center vertex, C, and again at least two others must be chosen. The smallest vertex cover consists of the two vertices adjacent to degree 1 vertices. Thus, one obtains the ratio $3/2$. □

4 Independent Set

Maximum Independent Set is the problem of finding a largest subset, I, of vertices in a graph such that no two vertices in I are adjacent to each other.

The independent set problem and the clique problem, which finds the same set in the complement of the graph, are well studied NP-hard problems, where approximation also appears to be hard. The bounded degree maximal independent set problem is one of the original MAX SNP-Complete problems [27]. Håstad [16] has shown a general lower bound on the approximation ratio for the independent set problem of $n^{1-\epsilon}$, for all ϵ, provided that NP $\neq$ ZPP, where ZPP is the class of languages decidable by a random expected polynomial-time algorithm that makes no errors A general upper bound of $O(n/\log^2 n)$ was presented by Boppana and Halldórsson [8], and an upper bound of $6/5$ for graphs of degree 3 was shown by Berman and Fujito [7]. These algorithms are not priority algorithms.

Davis and Impagliazzo [12] have shown that no adaptive priority algorithm (in the vertex adjacency formulation) can achieve an approximation ratio better than $\frac{3}{2}$ for the maximum independent set problem, and their proof uses graphs with maximum degree 3.

The Davis and Impagliazzo bound is the current best inapproximation bound for adaptive priority algorithms, although there are better results for more restricted

models. In our preliminary conference paper [9], we claimed that no fixed order priority algorithm in the vertex adjacency formulation can achieve an approximation ratio better than $\Omega(n^{1/3})$ where n is the number of nodes. We soon realized that our proof was assuming a degree based fixed priority algorithm. Following the online algorithm results of Halldórsson et al. [17], we provide a corresponding inapproximation bound for degree based fixed order algorithms in the edge adjacency formulation. The following construction is a special case[7] of a construction in the thesis of Mirmohammadi [26].

Construction $\mathbb{P}$: There are three sets of vertices, A, B, and W (refer to Figure 5). The sets A and B each consist of k vertices, $a_1, a_2, ..., a_k$ and $b_1, b_2, ..., b_k$, and the

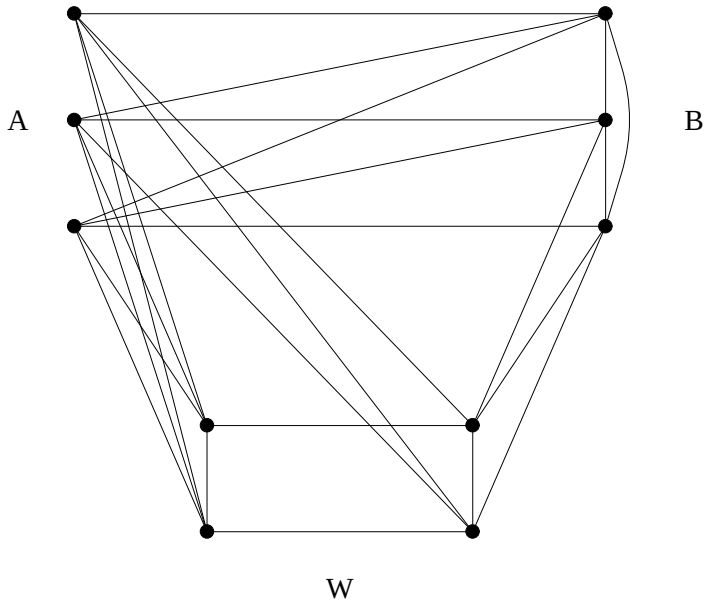

Figure 5: Construction $\mathbb{P}$ for $k = 3$.

set W consists of $2k - 2$ vertices. All vertices have degree $2k - 1$. The edges, E,

[7] The Halldórsson et al (respectively, Mirmohammadi) construction is designed to prove a result for online (respectively, degree based fixed priority pBT algorithms [1]) that allow (polynomially) many solutions to be simultaneously constructed. Moroever, strong inapproximation results are still obtained in [26] when the model is extended to allow revocable acceptances as previously studied in (for example) [20, 1, 30]. However, the construction in [26] does not provide an inapproximation bound for the vertex adjacency formulation.

are as follows: For $1 \leq i \leq k$, $(a_i, b_i) \in E$ and $(b_i, a_j), (b_i, b_j) \in E$ for $i < j \leq k$, so the vertices in A and B come in pairs which are matched, and each vertex in B is adjacent to all later vertices in both A and B. In addition, the vertices in A and B are adjacent to enough vertices in W so that every vertex in A and B has degree $2k-1$. Then, partitioning the vertices in W into two sets of size $k-1$, making each of these sets a clique, and adding the edges of a perfect matching between these two sets will also cause every vertex in W to have degree $2k-1$.

Now we consider any degree based fixed priority algorithm and show an $\Omega(n)$ lower bound. This result is clearly asymptotically optimal.

Theorem 8 No degree based fixed priority algorithm $\mathbb{A}$ in the edge adjacency formulation for independent set (or clique) can achieve an approximation ratio better than $\frac{n+2}{12}$, where n is the number of vertices.

Proof The adversary uses the construction, $\mathbb{P}$. We note that the optimum independent set in $\mathbb{P}$ includes every vertex in A and has size k. If n is the total number of vertices in $\mathbb{P}$, then $k = \frac{n+2}{4}$.

Since $\mathbb{A}$ is a degree based fixed priority algorithm and all vertices have the same degree, $\mathbb{A}$ cannot distinguish between the vertices when assigning priorities.

The adversary arranges that the selected vertices are given (in adjacent pairs) in the order

$$a_1, b_1, a_2, b_2, ..., a_i, b_i, ...$$

as long as $\mathbb{A}$ rejects the vertices. When $\mathbb{A}$ accepts its first vertex (assuming that less than $2k$ vertices have already been rejected), the adversary makes it b_j, where j is the index of the first vertex in B not yet processed. Note that this is possible for the adversary, even if the last vertex processed was also a B-vertex, because a_j and b_j are both adjacent to all previous B-vertices and no previous A-vertices. Since the edge adjacency representation is used, the edges to unprocessed vertices are simply labels which are distinct from any edges previously seen. Thus, the adversary can successfully complete the construction, regardless whether a_j or b_j is processed at this point. Since b_j is adjacent to all later A- and B-vertices, $\mathbb{A}$ must reject all of them, except b_j. The vertices in W form two cliques, so at most two of them can be accepted. Thus, $\mathbb{A}$ accepts at most three vertices, compared to the optimal k, giving a ratio of $\frac{k}{3}$. (If $\mathbb{A}$ rejects $2k$ vertices initially, only the vertices in W are unprocessed and thus at most two vertices are accepted, giving an even worse ratio.)

Since a clique is a complement of an independent set, the same result holds for the clique problem, by complementing the construction. □

Note that if the algorithm, $\mathbb{A}$, in the above proof accepts the first vertex, the adversary will arrange that no other vertices can be included in the independent set. Hence, the following is obtained.

Theorem 9 No acceptances-first adaptive algorithm $\mathbb{A}$ in the vertex adjacency formulation for independent set (or clique) can achieve an approximation ratio better than $\frac{n+2}{12}$, where n is the number of vertices (even if the number of vertices and edges in the graph is known to the algorithm).

Combining the acceptances-first requirement with the fixed priority requirement, gives a model which is so weak that it appears to be uninteresting for this problem. Consider, for example, a complete bipartite graph with n vertices in each part. All vertices look the same to the algorithm as it assigns priorities, so the adversary can decide that the two vertices with highest priority are adjacent. If the algorithm is acceptances-first, since it must reject the second vertex, it cannot accept more than one vertex in all.

We use a special case of the maximal independent set problem to prove that the edge adjacency formulation is weaker than the vertex adjacency formulation for adaptive priority algorithms. Our result is based on the example used in Davis and Impagliazzo [12] to show that memoryless priority algorithms are less powerful than those which use memory. Namely, we consider WIS(k), the weighted maximum independent set problem when restricted to cycles whose vertex weights are either 1 or k. In their proof separating the power of memoryless algorithms from those which use memory, Davis and Impagliazzo show that in the vertex adjacency formulation there is an adaptive priority algorithm whose approximation ratio approaches one as k goes to infinity. In contrast, for the WIS(k) problem in the vertex adjacency formulation, Davis and Impagliazzo [12] show a 2-approximation lower bound for memoryless algorithms. We now show a lower bound of $\frac{3}{2}$ for the approximation ratio for the WIS(k) problem in the edge adjacency formulation, thus showing that the edge adjacency formulation can be restrictive when compared to the vertex adjacency formulation.

Theorem 10 For the WIS(k) problem with $k \geq 4$, no adaptive priority algorithm in the edge adjacency formulation can obtain an approximation ratio better than $\frac{3}{2}$.

Proof We represent the cycles by lists of weights. Two neighbors in the list are also neighbors in the cycle. In addition, the first and last element in the list are also neighbors in the cycle.

We use w^+ to denote a vertex accepted by the priority algorithm and w^- to denote a vertex rejected by the priority algorithm. To demonstrate a best possible result which the priority algorithm can obtain given the accept/reject actions it has already made, we use w^c to mark vertices which could be included in addition to the already accepted vertices. Finally, we indicate an optimal vertex cover by marking vertices in one such cover by $\underline{w}$. Neither the vertices marked w^c nor $\underline{w}$ can in general be chosen uniquely, but their total weight will be unique.

The argument is structured according to the choices made by the priority algorithm, beginning with whether the first vertex has weight 1 or k and whether the priority algorithm accepts or rejects that vertex. In all but one case, the adversary can immediately guarantee a specific approximation ratio, but in one case, the next vertex chosen by the algorithm must also be used by the adversary:

First accept weight k vertex: $(k^+, \underline{k}, 1^c, \underline{k})$ gives $\frac{2k}{k+1}$.

First reject weight k vertex: $(\underline{k}^-, 1^c, 1)$ gives $\frac{k}{1}$.

First accept weight 1 *vertex*: $(1^+, \underline{k}, 1^c, \underline{k})$ gives $\frac{2k}{2}$.

First reject weight 1 *vertex*: We now ensure that no vertex of weight k will appear as a neighbor of the rejected vertex. All the remaining cases are subcases of the current case.

Next accept non-neighbor weight k vertex: $(\underline{1}^-, 1^c, \underline{k}, k^+, \underline{k}, 1^c)$ gives $\frac{2k+1}{k+2}$.

Next accept non-neighbor weight 1 *vertex*: $(1^-, 1^c, \underline{k}, 1^+, \underline{1})$ gives $\frac{k+1}{2}$.

Next accept neighbor weight 1 *vertex*: $(\underline{1}^-, 1^+, \underline{k}, 1^c)$ gives $\frac{k+1}{2}$.

Next reject non-neighbor weight k vertex: $(\underline{1}^-, 1^c, \underline{k}^-, 1^c)$ gives $\frac{k+1}{2}$.

Next reject non-neighbor weight 1 *vertex*: $(\underline{1}^-, 1^c, \underline{1}, 1^-, \underline{1}, 1^c)$ gives $\frac{3}{2}$.

Next reject neighbor weight 1 *vertex*: $(\underline{1}^-, 1^-, \underline{1}, 1^c)$ gives $\frac{2}{1}$.

Choosing $k \geq 4$ ensures the stated approximation ratio lower bound of $\frac{3}{2}$. □

The following result shows that in the edge adjacency formulation, a $\frac{3}{2}$-approximation ratio for WIS(k) can be achieved.

Theorem 11 For the WIS(k) problem, there is an adaptive priority algorithm in the edge adjacency formulation with approximation ratio $\frac{3}{2}$ for $k \geq 2$.

Proof The algorithm proceeds as follows:

If there are no vertices of weight 1, accept one of weight k. Then follow it around the cycle, accepting every other vertex until finding a vertex adjacent to two already processed vertices. That last vertex must be rejected.

If there is at least one vertex of weight 1, do the following:

I. Give highest priority to vertices with weight 1 which are not adjacent to anything processed yet, as long as this is possible. Reject them all.

II. Repeat the next two steps as long as possible:

1. If there is a vertex with both neighbors already processed, accept it. (The neighbors have been rejected.)

2. If there is a vertex with weight k adjacent to exactly one vertex which was already processed, accept it. Then, reject its other neighbor.

III. If there are any vertices remaining, there must be a vertex of weight 1 adjacent to only one already processed vertex. Reject this vertex of weight 1 and accept its unprocessed neighbor. Follow this around the cycle, accepting every other vertex until reaching a vertex which has already been processed. Repeat this step until all processed chains have been joined.

Note that this algorithm maintains the invariant that for any maximal chain of vertices already processed, the endpoints have been rejected.

Case 1: All vertices have weight k. The algorithm finds a maximum weight independent set.

Case 2: All vertices have weight 1. Then, at least $\frac{1}{3}$ of them are accepted. At most $\frac{1}{2}$ are in a maximum weight independent set, so the ratio is at least $\frac{3}{2}$.

Case 3: There are some vertices of weight 1 and some of weight k. For any maximal chain of weight-k vertices, one of the endpoints is accepted and then every other vertex is accepted. For any maximal chain S of weight-1 vertices, both endpoints are adjacent to vertices of weight k, though this may be the same weight-k vertex. Thus, for each such chain, there is a distinct vertex of weight k which is accepted. The smallest possible number of acceptances in such a chain of length s occurs when the next to last vertex on either end of the chain was selected in Step I and rejected, and every third vertex between these two was also selected in Step I and rejected. Then, at least $\frac{1}{3}(s-3)$ vertices in the chain must be accepted in Step III. (If there are some vertices chosen in Step I which have only one vertex between them, instead of two, they will be accepted in Step II.1, increasing the fraction accepted.) Consider the vertex of weight k assigned to this chain. Suppose there were t vertices in its chain C of weight-k vertices. There are two subcases based on whether t is even or odd.

Subcase t even: Then $\frac{t}{2}$ of these weight-k vertices were accepted, and $\frac{t}{2}$ of these vertices are in any maximum weight independent set. Since t is even, the algorithm

cannot accept both endpoints of C. Next to the endpoint it does not accept, it will accept a vertex of weight 1, which has not been accounted for in the $\frac{1}{3}(s'-3)$ vertices accepted in any maximal chain of weight-1 vertices which has length s'.

Subcase t odd: In this case, a maximum weight independent set contains both endpoints of C, and the algorithm also accepts both endpoints, so $\frac{t+1}{2}$ vertices in C are accepted and are in a maximum weight independent set.

Let E be the set of even-length maximal chains of weight-k vertices, let O be the set of odd-length maximal chains of weight-k vertices, and let I be the set of maximal chains of weight-1 vertices, Let $l(C)$ denote the number of vertices in a chain C. For each chain in E, there is one endpoint of a maximal chain of weight-1 vertices which cannot be in a maximum weight independent set. Similarly, for each chain in O, there are two endpoints of maximal chains of weight-1 vertices which cannot be in a maximum independent set. Thus, amortized over all chains, $C \in I$, of weight-1 vertices, a maximum weight independent set contains at most $\sum_{C\in I}(\frac{l(C)}{2}) - \frac{1}{2}|O|$ weight-1 vertices.

Thus, the ratio r of the weight of the independent set accepted by this algorithm to the weight of a maximum independent set is at most

$$\begin{aligned} r &\leq \frac{\sum_{C\in E}(\frac{k\cdot l(C)}{2})+\sum_{C\in O}(\frac{k\cdot(l(C)+1)}{2})+\sum_{C\in I}(\frac{l(C)}{2})-\frac{1}{2}|O|}{\sum_{C\in E}(\frac{k\cdot l(C)}{2}+1)+\sum_{C\in O}(\frac{k\cdot(l(C)+1)}{2})+\sum_{C\in I}(\frac{(l(C)-3)}{3})} \\ &= \frac{\sum_{C\in E}(\frac{k\cdot l(C)}{2})+\sum_{C\in O}(\frac{k\cdot(l(C)+1)}{2})+\sum_{C\in I}(\frac{l(C)}{2})-\frac{1}{2}|O|}{\sum_{C\in E}(\frac{k\cdot l(C)}{2})+\sum_{C\in O}(\frac{k\cdot(l(C)+1)}{2})+\sum_{C\in I}(\frac{(l(C))}{3})-|O|} \\ &\leq \frac{6k|O|+3\sum_{C\in I}(l(C))-3|O|}{6k|O|+2\sum_{C\in I}(l(C))-6|O|}. \end{aligned}$$

For $k \geq 2$, this is at most $\frac{3}{2}$. □

5 Vertex Coloring

Minimum Vertex Coloring is the problem of coloring the vertices in a graph using the minimum number of different colors in such a way that no two adjacent vertices have the same color. The problem is also known as Graph Coloring and as Chromatic Number.

Hardness results are known for minimum vertex coloring under various complexity theoretical assumptions: minimum vertex coloring is NP-hard to approximate within $\Omega(n^{1-\epsilon})$, for all ϵ, provided that NP $\neq$ ZPP [15]. It is NP-hard to approximate within $n^{\frac{1}{5}}$ provided that NP $\neq$ coRP and within $n^{\frac{1}{7}}$ provided that P $\neq$ NP [6].

From [22], it is known that it is NP-hard to 4-color a 3-chromatic graph and NP-hard to color a k-chromatic graph with at most $k + 2\lceil k/3 \rceil - 1$ colors.

On the positive side, a general upper bound of $O(n \log \log^2 n / log^3 n)$ is shown by Halldórsson [18]. In [28], an upper bound of $\lambda(G) + 1$ is established, where λ is any function of graphs $G = (V, E)$ such that

$$(G' \subset G \;\Rightarrow\; \lambda(G') \leq \lambda(G)) \;\;\wedge\;\; \lambda(G) \geq \min_{v \in V} \mathit{deg}(v).$$

Let $d(G)$ be the maximum over all vertex-induced subgraphs of the minimum degree in that subgraph. The result in [28] constructively establishes that any graph is $d(G) + 1$ colorable, so a corollary of the theorem below is that the algorithm from [28] is not a priority algorithm. This theorem is proven using an adversary which is defined using a lengthy case analysis.

Theorem 12 No priority algorithm in the edge adjacency formulation can 3-color all graphs G with $d(G) = 2$.

Proof The adversary begins with edge lists such that many graphs could be found by removing different subsets of the edge lists. Each of the final graphs the adversary might produce in the following contains one degree 2 vertex and the remainder of the vertices have degree 3. Each graph has $d(G) = 2$ and thus can be colored with three colors, but an adaptive priority algorithm $\mathbb{A}$ will be forced to use at least four colors. In order to satisfy the degree requirements, extra vertices and edges will need to be added to what is described in each case. This can often be done by creating several copies of the same subgraph and attaching them where the degree is too low.

Note that attaching the degree 2 vertex in the subgraph of Fig. 6 to some vertex

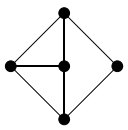

Figure 6: Attachment graph for incrementing vertex degrees while only adding degree 3 vertices.

u in some partially specified graph will increase the degree of u by one while all the added vertices will have degree 3. Thus, any partially specified graph (where

degrees are not already too large) can be completed to a graph of the type we are interested in (one degree 2 vertex and the rest degree 3 vertices).

In many of the cases below, we use completions of variants of the graph $K = (V, E)$, where $V = \{A, B, C, D, E, F, G, H\}$ and $E = \{\{A, B\}, \{A, E\}, \{A, H\}, \{B, C\}, \{B, G\},$
$\{C, D\}, \{C, F\}, \{D, E\}, \{D, F\}, \{E, F\}\}$; see Fig. 7.

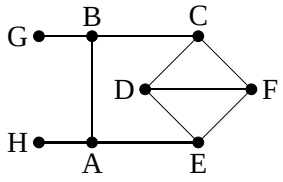

Figure 7: Graph K.

In some cases, the vertices G and H will be replaced by a single vertex adjacent to both vertices A and B. This merged vertex will be adjacent to an extra vertex of degree 2, to make its degree 3 also. The entire graph is then repeated on the other side of this new degree 2 vertex, so it is symmetric about this vertex. In other cases, the graph will be completed such that G (or H) will be the degree 2 vertex and H (or G) will be a degree 3 vertex.

Note that, in the graph above, since removing G and H (or the vertex replacing them) from K leaves a vertex induced subgraph with minimum degree 2, $d(K) \geq 2$. It can easily be seen that no vertex induced subgraph has higher degree.

If vertices C and E get assigned different colors, then C, D, E, and F must together have at least four different colors, and we are done. Giving vertices A and C the same colors will force C and E to get different colors, accomplishing the same. The goal in most of the following cases is to force one of these conditions.

In the following, the notation $c(X)$ will be used for the color the priority algorithm $\mathbb{A}$ gives vertex X.

Case A: The degree 2 vertex is never chosen; the adversary never shows it adjacent to anything until $\mathbb{A}$ has been forced to use four colors and the entire graph is revealed. In all of Case A, we use the graph variant shown in Fig. 8. The first vertex chosen, W, has degree 3.

Case A.1: The next vertex chosen, X, is adjacent to W. The adversary restricts the input so that there exists another degree 3 vertex, Z, adjacent to both of them, plus one vertex adjacent to X, and another adjacent to W. No vertex, other than Z, W,

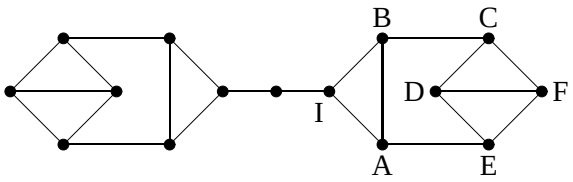

Figure 8: The Case A Graph.

and X will be adjacent to two of W, X, and Z; see Fig. 9. The next vertex chosen

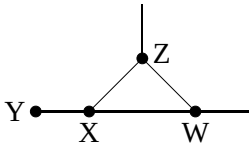

Figure 9: $\mathbb{A}$'s initial view of the graph in Case A.

may be Z. Whenever Z is chosen, it is given the third color. In the following, we ignore the actual timing of when it is chosen.

Case A.1.1: The next vertex chosen Y is adjacent to one of W, X and Z. Without loss of generality, assume Y is adjacent to X.

Case A.1.1.1: If $c(Y) = c(W)$, let $A = W$, $B = X$, and $C = Y$, $Z = I$, and we are done since we must have $c(A) = c(C)$.

Case A.1.1.2: If $c(Y) \neq c(W)$, let $A = Z$, $W = I$, $B = X$, and $C = Y$, and we are done since $c(A) = c(C)$.

Case A.1.2: The next vertex U chosen is not adjacent to W, X, or Z. Without loss of generality, assume $c(U) = c(Z)$, the third color. Let $A = W$, $B = X$, $D = U$, $Z = I$. Then either C and/or E are given a fourth color or $c(C) = c(A)$, and we are done.

Case A.2: The next vertex X is not adjacent to W.

Case A.2.1: If $c(W) = c(X)$, let $A = W$ and $C = X$, and we are done.

Case A.2.2: If $c(W) \neq c(X)$, let $C = W$ and $E = X$, and we are done.

Case B: The degree 2 vertex is chosen at some point. The adversary ensures that the connected component $\mathbb{A}$ sees containing the degree 2 vertex never becomes adjacent to other vertices $\mathbb{A}$ has processed until $\mathbb{A}$ can be forced to use a fourth color and the entire graph is revealed. The following describes how the adversary

handles vertices $\mathbb{A}$ chooses after the degree 2 vertex is chosen, in the connected component processed by $\mathbb{A}$ and containing the degree 2 vertex. The adversary may build graphs on either side of the degree 2 vertex; they are only connected at the degree 2 vertex, and we will only consider one direction; the other can be treated similarly. If vertices are chosen which are not connected by a path to the degree 2 vertex, they are treated as in Case A. (Note that at most four degree 3 vertices not in the same connected component as the degree 2 vertex are sufficient for the adversary to end in Case A.) Thus, we assume that a degree 3 vertex Z, adjacent to the degree 2 vertex and a degree 3 vertex X, adjacent to Z, have been chosen and assigned different colors. The adversary will not present any vertices adjacent to both X and Z.

In these cases, we often use the Graph K from Fig. 7 where either G or H will be the degree 2 vertex, depending on which vertex is interpreted to be Z.

Case B.1: A vertex Y adjacent to Z is chosen next; see Fig. 10.

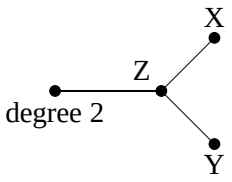

Figure 10: Initial part of graph in Case B.1.

Case B.1.1: If $c(Y) = c(X)$, let $A = X$, $B = Z$, and $C = Y$ in the graph K. Then $c(A) = c(C)$, and we are done.

Case B.1.2: If $c(Y) \neq c(X)$, the adversary's graph will contain two vertices, U and V, both adjacent to X and Y and each other. One of U and V must be given a fourth color.

Case B.2: A vertex Y adjacent to X is chosen next; see Fig. 11.

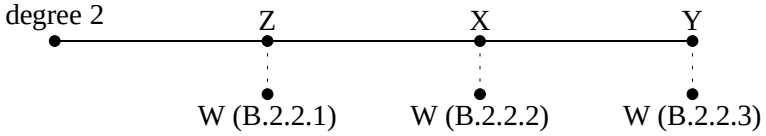

Figure 11: Initial part of graph in Case B.2. The position of W depends on subcases.

Case B.2.1: If $c(Y) = c(Z)$, let $A = Z$, $B = X$, and $C = Y$ in the graph K. Then $c(A) = c(C)$, and we are done.

Case B.2.2: Assume $c(Y) \neq c(Z)$.

Case B.2.2.1: Assume a vertex W adjacent to Z is chosen next.

Case B.2.2.1.1: If $c(W) = c(X)$, let $A = X$, $B = Z$, $C = W$, and $E = Y$ in the graph K. Then $c(C) \neq c(E)$, and we are done.

Case B.2.2.1.2: If $c(W) = c(Y)$, let $A = W$, $B = Z$, and $C = Y$. The adversary will replace the edge $\{B, C\}$ in the graph K the edges $\{B, X\}$ and $\{X, C\}$. Then $c(A) = c(C)$, and we are done, since the vertices D, E, and F will have the same adjacencies as in K. X will be a degree 3 vertex by adding a construction shown in Fig. 6.

Case B.2.2.2: Assume a vertex W adjacent to X is chosen.

Case B.2.2.2.1: If $c(W) = c(Z)$, let $A = Z$, $B = X$, and $C = W$ in the graph K. Then $c(A) = c(C)$, and we are done.

Case B.2.2.2.2: If $c(W) = c(Y)$, let $A = W$, $B = X$, and $C = Y$. Then $c(A) = c(C)$, and we are done.

Case B.2.2.3: Assume a vertex W adjacent to Y is chosen.

Case B.2.2.3.1: If $c(W) = c(Z)$, let $A = Z$, $B = X$, and $C = W$. The adversary will replace the edge $\{B, C\}$ by the edges $\{B, Y\}$ and $\{Y, C\}$. Then $c(A) = c(C)$, and we are done. The vertex Y is made into a degree 3 vertex by adding the subgraph of Fig. 6.

Case B.2.2.3.2: If $c(W) = c(X)$, let $A = X$, $B = Y$, and $C = W$. Then $c(A) = c(C)$, and we are done.

Thus, in every case, the adversary is able to force $\mathbb{A}$ to use at least four colors. □

In more restrictive models, we obtain stronger lower bounds. The following two results apply to models which include the simplest and most natural greedy algorithm; namely, order the vertices in any way and then color vertices using the lowest possible numbered color.

Theorem 13 Any degree based fixed priority algorithm in the edge adjacency formulation must use at least $d + 1$ colors on a bipartite graph of maximum degree d.

Proof The adversary will create many independent portions of a bipartite graph, each with the same number of vertices and the same colors in each part. These portions will grow in size and it may be necessary to join two portions, making

the correct decision as to which partition of the one portion is placed with which partition of the other. At the end all vertices will have degree d, so in assigning priorities, the degree based fixed priority algorithm will continually choose vertices of degree d. Its only choice is which color to give after it is told which already colored vertices the chosen vertex is adjacent to.

Initially, the adversary will arrange that all vertices chosen are independent. The number chosen at this stage will be large enough so that there are either $d+1$ colors given or enough vertices given the same color to make the remainder of the proof possible. It will be clear that some large number will be sufficient. This stage 1 ends when there are enough vertices given the same color, which we call color 1.

In stage i, we have a large number of independent bipartite graphs, where both sides contain vertices with colors $1, 2, ..., i-1$, but no other colors. The vertices chosen are made adjacent to one vertex of each color $1, 2, ..., i-1$, all from one partition of one of the graphs. If there are a large enough number of graphs which get the same additional color on both sides, this color is called color i and the adversary proceeds to stage $i+1$. Otherwise, there will eventually be enough graphs given the same two additional colors, which will be called i and $i+1$. Graphs of this type can be joined in pairs. For each pair, the adversary joins them so that both partitions in the resulting bipartite graph have both colors i and $i+1$. Then, the adversary proceeds to stage $i+2$.

The adversary stops this process as soon as $d+1$ colors have been used, and more vertices are included to create a bipartite graph where all vertices have degree d. Note that if fewer than $d+1$ colors are used before stage $d+1$, a $(d+1)$st color will be used then, since the vertices in that stage will be adjacent to each of the colors $1, 2, ..., d$. If there is no stage $d+1$, because the adversary went from stage d to $d+2$, the $(d+1)$st color was used in stage d. □

In the next result, we consider adaptive priority algorithms which use different information in its two main phases. When assigning priorities to vertices, it only considers the number of uncolored neighbors a vertex has and the vector $(n_1, \ldots, n_k)$ of the k colors used so far where n_i is the number of nodes that have already been colored with color i. In this phase, the algorithm may not use information about how many of a vertex's neighbors have already been colored or what colors these neighbors have been given. For the irrevocable decision of coloring a vertex, the color given will simply be a function of the set of colors already given to the neighbors. This could, for example, be the lowest possible numbered color.

Theorem 14 Any adaptive priority algorithm in the edge adjacency formulation, which gives priorities based only on the current degree of the vertex and the already

processed subgraph, must use $d+1$ colors on a d-colorable graph of maximum degree d, when the color given is a function of the set of currently adjacent colors (no state information).

Proof The adversary uses the following graph. It creates two K_d cliques A and B. Two selected vertices, a in A and b in B are then connected by an additional edge. The remaining vertices in A and B may or may not be connected via a single edge to additional copies of K_d cliques. This will depend on the degree of vertices chosen by the adaptive priority algorithm $\mathbb{A}$. At any point in time during the execution of $\mathbb{A}$, $\mathbb{A}$ will have a choice of two consecutive degrees within A, and two (possibly different) consecutive degrees within B. Whenever it chooses the higher degree, the chosen vertex will be connected to one of the additional K_d cliques. At most $2d-2$ of the additional K_d cliques may be necessary. The adversary must present this many originally. When a vertex with the lower of the two possible degrees from A or B is chosen, one of the additional K_d cliques is removed from the possibilities the adversary gives $\mathbb{A}$, so that vertices not present in the graph are never chosen.

The adversary will ensure that a is the last of the vertices in A which is colored, and b is the last in B. Thus, the last one of them colored will be adjacent to d different colors and get the $(d+1)$st color. When there is a choice between choosing vertices in A or B or in the additional cliques, vertices from A or B are chosen. For the additional K_d cliques, if the connecting vertex is chosen after the adjacent vertex in A or B, then there is no problem; the additional K_d, G, cannot have any influence on A or B. If the connecting vertex is chosen before the adjacent vertex in A or B, there will be fewer colors used in G than in A or B, whichever it is adjacent to. So the connecting vertex will be assigned a color which is already among the neighbors of the connecting vertex in A or B; again there will be no influence on how A or B are colored. As soon as the connecting vertex has been chosen, it is connected to some vertex in A or B which has not been colored yet, further restricting the number of possible vertices of the lower degree.

Note that any graph constructed in this manner is easily d colorable, since the cliques can be connected via vertices of different colors. □

6 Definition of Greedy Priority Algorithms Revisited

As observed in Davis and Impagliazzo [12] and as mentioned in the introduction, the definition of a "greedy" priority algorithm becomes problematic when input items are not isolated. We again note that the greedy distinction amongst priority

algorithms may not be a critical issue but, at the very least, the historical interest in this concept makes it seem necessary to provide a meaningful definition.

We propose a very liberal definition for what can constitute a greedy algorithm. Namely, a greedy (priority) algorithm is one which always makes an irrevocable "greedy" decision whenever such a decision is available. This, of course, has pushed the definitional problem to that of defining a "greedy decision" which we now proceed to do.

Consider a priority algorithm that has processed some number of input items. As stated in the introduction, we interpret the underlying philosophy of "greediness" to be that of "live for today". When input items are isolated, this leads to a very natural concept for being greedy, namely the irrevocable decision must be made to be consistent with optimizing the objective function assuming the current input item being processed will be the last input item. But for non-isolated inputs, it may be the case that any valid input instance will require further input items, e.g., items are vertices represented by their vertex adjacency lists and there are vertices known to exist, but not yet processed. Let P be the set of items already processed plus the item currently being considered. We say that a set S of input items is a *minimal completion set* if $P \cup S$ constitutes a valid input instance and for no set $S' \subset S$ is $P \cup S'$ a valid input instance. In the case of isolated input items, only the empty set is a minimal completion set. A greedy decision for an item I satisfies the property that for *every* minimal completion S, there is a set of decisions for the items in S such that no other set of decisions for I and the items in S would result in a better value for the given objective function. Note that we are not concerned with whether or not the set of minimal completions is finite (or even countable) or whether or not it is (efficiently) computable to determine whether or not a decision is greedy. Clearly for any unweighted graph problem, the set of minimal completions is finite and it is computable (but maybe not efficiently) to determine if a decision is greedy.

Note that any acceptances-first algorithm for the independent set problem is greedy by this definition, since no known, but unprocessed, vertices are independent from what has already been processed. Any priority algorithm for the vertex cover problem which accepts a vertex if not all of its incident edges are already covered is also greedy by this definition. However, not every algorithm for vertex coloring which chooses the lowest numbered color possible at every step is greedy by this definition. To see this, consider the graph $G = (V, E)$, where $V = \{1, 2, 3, 4, 5\}$, and

$$E = \{\{1, 2\}, \{1, 3\}, \{2, 3\}, \{2, 4\}, \{2, 5\}, \{3, 5\}, \{4, 5\}\};$$

see Fig. 12. If the vertices are chosen in the order $\langle 1, 2, 3, 4, 5\rangle$, then vertex 4 will get color 1 if the lowest numbered color possible is given, and thus the last vertex

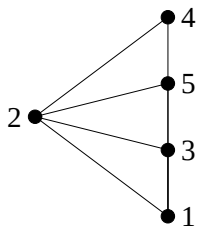

Figure 12: Choosing the minimal color is not greedy.

will get color 4. However, giving vertex 4 color 3 at this point will allow the last vertex to be colored with color 1, and only three colors will be used in all.

7 Conclusions and Open Problems

We have considered priority algorithms in the vertex adjacency and edge adjacency formulations, and it was shown that the edge adjacency formulation can be more restrictive than the vertex adjacency formulation for adaptive priority algorithms. Most known priority algorithms, however, can be implemented using the edge adjacency formulation. Thus, it would be interesting to find natural problems for which the input formulations are provably different with respect to the best approximation ratio attainable, and if different how much better one can do using the vertex adjacency formulation.

With respect to lower bounds for fixed priority algorithms, we considered the degree based model. It is unclear if arbitrary fixed priority algorithms are more powerful than the degree based model in either the vertex or edge adjacency formulations. We conjecture that the three results we have for degree based fixed orderings also hold for arbitrary fixed priority algorithms in the edge adjacency formulation.

For problems where an adaptive priority algorithm makes only accept/reject decisions for each vertex, acceptances-first algorithms are equivalent to memoryless algorithms. The acceptances-first model was introduced and applied to the Maximum Independent Set and Vertex Cover problems.

Many of the lower bound results do not meet the upper bounds provided by known algorithms. It would be interesting to close some of these gaps. For example, in the result for the unweighted vertex cover, our adaptive priority $4/3$ lower bound meets Clarkson's result in the case when the maximum degree is three. But what if the maximum degree is larger than three? Can one prove a better lower bound? It has long been an open problem whether or not the optimal (polynomial time)

approximation ratio for vertex cover is $2 - o(1)$.

References

[1] Michael Alekhnovich, Allan Borodin, Joshua Buresh-Oppenheim, Russell Impagliazzo, Avner Magen, and Toniann Pitassi. Towards a model for backtracking and dynamic programming. In *Proceedings of the Twentieth Annual IEEE Conference on Computational Complexity*, pages 308–322, 2005. Journal version submitted to Computational Complexity Journal.

[2] Spyros Angelopoulos. Ordering-preserving transformations and greedy-like algorithms. In *Proceedings of the Second Workshop on Approximation and Online Algorithms*, volume 3351 of *Lecture Notes in Computer Science*, pages 197–210. Springer-Verlag, 2005.

[3] Spyros Angelopoulos and Allan 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*, volume 2462 of *Lecture Notes in Computer Science*, pages 26–39. Springer-Verlag, 2002.

[4] Sanjeev Arora, Béla Bollobás, and László Lovász. Proving integrality gaps without knowing the linear program. In *Proceedings of the 43th Annual IEEE Conference on Foundations of Computer Science*, pages 313–322, 2002.

[5] David Avis and Tomokazu Imamura. A list heuristic for vertex cover. *Operations Research Letters*, 35(2):201–204, 2007.

[6] Mihir Bellare, Oded Goldreich, and Madhu Sudan. Free bits, PCPs and non-approximability—towards tight results. *SIAM Journal on Computing*, 27:804–915, 1998.

[7] Piotr Berman and Toshihiro Fujito. On approximation properties of the independent set problem for degree 3 graphs. In *Proceedings of the Fourth International Workshop on Algorithms and Data Structures*, volume 955 of *Lecture Notes in Computer Science*, pages 449–460. Springer-Verlag, 1995.

[8] Ravi Boppana and Magnús M. Halldórsson. Approximating maximum independent sets by excluding subgraphs. *Bit*, 32:180–196, 1992.

[9] Allan Borodin, Joan Boyar, and Kim S. Larsen. Priority algorithms for graph optimization problems. In *Proceedings of the Second Workshop on Approximation and Online Algorithms*, volume 3351 of *Lecture Notes in Computer Science*, pages 126–139. Springer-Verlag, 2005.

[10] Allan Borodin, Morten N. Nielsen, and Charles Rackoff. (Incremental) priority algorithms. *Algorithmica*, 37:295–326, 2003.

[11] Kenneth L. Clarkson. A modification of the greedy algorithm for vertex cover. *Information Processing Letters*, 16:23–25, 1983.

[12] Sashka Davis and Russell Impagliazzo. Models of greedy algorithms for graph problems. In *Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 381–390, 2004.

[13] Irit Dinur and Shmuel Safra. The importance of being biased. In *Proceedings of the 34th Symposium on Theory of Computing*, pages 33–42. ACM Press, 2002.

[14] Jack Edmonds. Matroids and the greedy algorithm. *Mathematical Programming*, 1:127–136, 1971.

[15] Urid Feige and Joe Kilian. Zero knowledge and the chromatic number. *Journal of Computer and System Sciences*, 57:187–199, 1998.

[16] Johan Håstad. Clique is hard to approximate within $n^{1-\varepsilon}$. *Acta Mathematica*, 182:105–142, 1999.

[17] Magnús Halldórsson, Kazuo Iwama, Shuichi Miyazaki, and Shiro Taketomi. Online independent sets. In *Proceedings of the Sixth Annual International Computing and Combinatorics Conference*, volume 1858 of *Lecture Notes in Computer Science*, pages 202–209, 2000.

[18] Magnús M. Halldórsson. A still better performance guarantee for approximate graph coloring. *Information Processing Letters*, 45:19–23, 1993.

[19] Dorit S. Hochbaum. Efficient bounds for the stable set, vertex cover and set packing problems. *Discrete Applied Mathematics*, 6:243–254, 1983.

[20] Stephanie Lorraine Horn. One-pass algorithms with revocable acceptances for job interval selection. MS Thesis, University of Toronto, 2004.

[21] David S. Johnson. Approximation algorithms for combinatorial problems. *Journal of Computer and System Sciences*, 9(3):256–278, 1974.

[22] Sanjeev Khanna, Nathan Linial, and Shmuel Safra. On the hardness of approximating the chromatic number. *Combinatorica*, 20(3):393–415, 2000.

[23] Subhash Khot. On the power of unique 2-prover 1-round games. In *Proceedings of the 34th Symposium on Theory of Computing*, pages 767–775, 2002.

[24] Subhash Khot and Oded Regev. Vertex cover might be hard to approximate within $2-\epsilon$. In *Proceedings of the 18th IEEE Conference on Computational Complexity*, pages 379–387, 2003.

[25] Daniel J. Lehmann, Liadan O'Callaghan, and Yoav Shoham. Truth revelation in approximately efficient combinatorial auctions. *Journal of the ACM*, 49(5):1–26, 2002.

[26] Nazanin Mirmohammadi. Inapproximation lower bounds for the maximum independent set problem in pBT model—and other observations. MS Thesis, University of Toronto, 2007.

[27] Christos H. Papadimitriou and Mihalis Yannakakis. Optimization, approximation and complexity classes. *Journal of Computer and System Sciences*, 43:425–440, 1991.

[28] G. Szekeres and Herbert S. Wilf. An inequality for the chromatic number of graphs. *Journal of Combinatorial Theory*, 4:1–3, 1968.

[29] Vijay V. Vazirani. *Approximation Algorithms*. Springer-Verlag, 2001.

[30] Yuli Ye and Allan Borodin. Priority algorithms for the subset-sum problem. In *Proceedings of the Thirteenth Annual International Computing and Combinatorics Conference*, pages 504–514, 2007. Journal version accepted for publication in Journal of Combinatorial Optimization.