

Priority Algorithms for Graph Optimization Problems

Allan Borodin¹, Joan Boyar^{2***}, and Kim S. Larsen^{2*}

¹ Department of Computer Science, University of Toronto.
`bor@cs.toronto.edu`, fax: +1 416 978 1931.

² Department of Mathematics and Computer Science,
University of Southern Denmark, Odense.
`{joan,kslarsen}@imada.sdu.dk`, fax: +45 6550 2325.

Abstract. We continue the study of priority or “greedy-like” algorithms as initiated in [6] and as extended to graph theoretic problems in [8]. Graph theoretic problems pose some modelling problems that did not exist in the original applications of [6] and [2]. Following [8], 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”.

1 Introduction

The concept of a greedy algorithm was explicitly articulated in a paper by Edmonds [10], 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 [8] and constitutes an important part of many texts concerning algorithm design and analysis. New greedy algorithms keep emerging, as, for instance, in [17], 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 suggesting new algorithms.

* 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).

** Contact author.

```

Determine an allowable ordering of the set of possible input items
(without knowing the actual input set  $S$  of items)
while not empty( $S$ )
   $next :=$  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  $S$ 

```

Fig. 1. The form of a fixed priority algorithm

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 [6] 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 [2] 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 [2] was for the “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.

The work of Davis and Impagliazzo [8] extends the priority formulation to graph theoretic problems. Davis and Impagliazzo consider a number of basic graph theory problems (single source shortest path, vertex cover, minimum spanning tree, Steiner trees, maximum independent set) with respect to one of two different input models depending on the problem and known “greedy algorithms”. For the shortest path, minimum spanning tree and Steiner tree problems, the model 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

```

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

```

Fig. 2. The form of an adaptive priority algorithm

the definitions in [6] can be applied. In particular, the definition of a greedy decision is well defined. In contrast, for the 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”.

Noting that lower bounds for graph theoretic priority algorithms appear to be hard to obtain in (say) the vertex adjacency model, Angelopoulos has recently [1] proposed a reasonable change to the model by restricting what priority algorithms can do, thus increasing the power of the adversary. The basic effect of his change is to force items which are indistinguishable (except for their different identification labels) to receive the same priority. 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’ results can be obtained in the model which we use, but even if they can, this simple restriction on priority algorithms should make it easier to derive lower bound proofs.

In this paper, we continue the study of priority algorithms for graph problems using two models (again motivated by current algorithms), namely the vertex adjacency model as in Davis and Impagliazzo and an “edge adjacency model”, 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 the vertex adjacency model is more general in the sense that any priority algorithm in the edge adjacency model can be simulated in the vertex adjacency model (making exactly the same set of decisions). Most existing priority algorithms can function in the edge adjacency model; the authors were unable to recall one which does not. However, we show (using an example in Davis and Impagliazzo showing that memorylessness is restrictive) that the edge adjacency model can be restrictive. In the appendix, we provide a definition for greedy decisions that can be applied even when input items are not isolated. We also introduce an “acceptances-first” model and clarify the relation of memoryless algorithms to this “acceptances-first” model rather than to greediness. We prove a number of new results within these models.

Many proofs have been omitted, but they are all included in the appendix for the convenience of the program committee.

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 $(v, w, v_1, v_2, \dots, v_d)$, where v is the name of the vertex, w is the weight (if any) of vertex v and v_1, \dots, v_d is a list of adjacent vertices. In the more restrictive edge adjacency formulation (but still a model sufficient to capture most known greedy graph algorithms), an input item is a vertex $(v, w, e_1, e_2, \dots, e_d)$ where again v is the vertex name, w is the weight (if any) of v and e_1, e_2, \dots, e_d is a list of adjacent edges.

In either of the above models, 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 model, 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, 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 [6]) is no longer well defined when the algorithm “knows” that the current item is not the last. More specifically, in [6], 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 Appendix A. 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 widely used (albeit mostly undefined) concept. We do argue that the priority algorithm formulation is important as it captures such 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 [6] and [2]. 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 [8], a rejected item is treated as a NO-OP. In the accept/reject context, memoryless algorithms are equivalent to *acceptances-first* algorithms which do not accept any items after the first rejected item. As observed¹ in [6] and [2], 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 are indeed memoryless (or equivalently, acceptances-first) according to the above definition. (By the definition of memoryless, the converse of the above theorem holds trivially.)

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 , together with the items already selected by \mathbb{A} , contain at least one valid input instance. At each step, the adversary removes the item i remaining in S , to which \mathbb{A} has given the highest priority. It 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 in the final graph, and possibly even more different vertices than in the final 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 model 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 model, 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 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 [6], the adversaries assume 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

¹ In [6], this fact is stated in terms of memoryless algorithms being simulated by greedy algorithms, but the essence of that observation really concerns the acceptance-first restriction and not greediness.

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 or edges in the graph.

3 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 [18]. Håstad [12] has shown a general lower bound on the approximation ratio for the independent set problem of $n^{1-\epsilon}$, for all ϵ , provided that $\text{NP} \neq \text{ZPP}$, 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 [5], and an upper bound for graphs of degree 3 of $6/5$ was shown by Berman and Fujito [4]. These algorithms are not priority algorithms.

Davis and Impagliazzo [8] have shown that no adaptive priority algorithm (in the vertex adjacency model) can achieve an approximation ratio better than $\frac{3}{2}$ for the maximum independent set problem², and their proof used graphs with maximum degree 3. We consider algorithms in more restrictive models. We again note that many known greedy-like graph algorithms are acceptances-first priority algorithms.

In the proofs of Theorems 2, 3, and 8, the adversary uses a modification of a construction due to Hochbaum [14].

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, initially, \mathbb{A} cannot distinguish between the vertices when assigning priorities. The optimum independent set includes every vertex in V and has size k^2 . If n is the total number of vertices in \mathbb{G} , $k \in \Theta(\sqrt{n})$.

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

The proof of this result depends on the first vertex being accepted. One can obtain a similar result, removing the acceptances-first assumption, if the algorithm \mathbb{A} is a fixed priority algorithm in the edge adjacency model.

² We have defined the approximation ratio so that all ratios are at least one.

Theorem 3. *No fixed priority algorithm \mathbb{A} in the edge adjacency model for independent set (or clique) can achieve an approximation ratio better than $\Omega(n^{1/3})$, where n is the number of vertices.*

Proof. The adversary uses possibly several copies of the construction, \mathbb{G} . Since \mathbb{A} is a fixed priority algorithm and all vertices have the same degree, \mathbb{A} cannot distinguish the vertices when assigning priorities.

The adversary arranges that the selected vertices are independent during the first phase. We let n' denote the number vertices processed so far. The first phase continues until either \mathbb{A} has accepted 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 accepted, 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 U . The accepted vertices are placed such that at most one is in each construction. This means that in each construction, \mathbb{G} , all vertices in V must be rejected. In addition, the algorithm can accept at most 1 vertex in every clique in U . This gives a ratio of at least $\frac{c \cdot k^2}{c \cdot k} = k = \Omega(n^{1/3})$.

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 accepted vertices is strictly smaller than $\lceil \frac{k^2}{k} \rceil = k$, since otherwise the algorithm would have terminated for that reason. If any of the n' vertices are accepted, then no vertices from U can be; otherwise at most one vertex from each clique can be accepted. Thus, the best ratio is when all n' vertices from phase one are rejected: $\frac{k^2}{k} = k = \Omega(\sqrt{n})$. \square

Combining the acceptances-first requirement with the fixed priority requirement, gives a model which is so weak that it appears to be uninteresting. 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.

Our next result is based on the example used in Davis and Impagliazzo to show that memoryless priority algorithms are less powerful than those which use memory. Namely, we consider $\text{WIS}(k)$, 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 model there is an adaptive priority algorithm whose approximation ratio approaches one as k goes to infinity. We now show a lower bound of $\frac{3}{2}$ for the approximation ratio for this same problem in the edge adjacency model, thus showing that the edge adjacency model can be restrictive when compared to the vertex adjacency model.

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

Proof. We will 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 $\frac{3}{2}$. \square

The following result shows that a $\frac{3}{2}$ approximation ratio for $\text{WIS}(k)$ can be achieved in the edge adjacency model³.

Theorem 5. *For the $\text{WIS}(k)$ problem, there is an adaptive priority algorithm in the edge adjacency model with approximation ratio $\frac{3}{2}$ for $k \geq 2$.*

Proof. The algorithm proceeds as follows:

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. If there were 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.

III. 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 vertex with weight k adjacent to exactly one vertex which was already processed, accept it. Then, reject its other neighbor.

³ In contrast, for the WIS problem in the vertex adjacency model, Davis and Impagliazzo show a 2-approximation lower bound for memoryless algorithms.

IV. 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. The remainder of the proof is a case analysis and is given in the appendix. \square

4 Vertex Cover

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

The unweighted vertex cover problem is one of the most celebrated open problems in the area of worst case approximation algorithms. The naive 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 polynomial time $2 - \epsilon$ approximation algorithms (for a fixed $\epsilon > 0$), although various algorithms are known that guarantee an approximation better than 2 but converging to 2 as some parameter grows. This maximal matching algorithm is easily seen to be an acceptances-first adaptive priority algorithm in the edge adjacency model. Surprisingly, Johnson [15] showed that the greedy algorithm which chooses the vertex with highest degree in the remaining graph is only a 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. Although the weighted vertex cover problem can be essentially 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 Johnson's "greedy algorithm" (the layered algorithm as given in Vazirani's excellent text on approximation algorithms [20]). Essentially for the vertex cover problem this algorithm chooses all maximum degree vertices and removes them simultaneously. Another simple to state (and also called greedy) algorithm is given by Clarkson [7]. This algorithm achieves the approximation bound $\frac{\Delta}{\Delta-2}(2 - \frac{2n}{\Delta \cdot OPT(I)})$ where Δ is the maximum degree in the graph and n is the number of vertices⁴. Both the layered algorithm and Clarksons's algorithm can be expressed as acceptances-first adaptive algorithms in the edge adjacency model. In terms of complexity lower bounds, Dinur and Safra [9] provide a sophisticated proof that it is NP-hard to have a c -approximation algorithm for the (unweighted) vertex cover problem for $c < 1.36$.

⁴ The stated bound is not defined for $\Delta \leq 2$. The more general bound that applies to all Δ is that $w(C_{MG}) \leq w(C_{OPT}) - \frac{2(n-w(C_{MG}))}{\Delta}$.

Davis and Impagliazzo show that for the weighted case, priority algorithms (in the vertex adjacency model) cannot essentially do better than a 2-approximation. Priority lower bounds for the unweighted case seem more difficult.

The following $\frac{4}{3}$ lower bound matches the upper bound by Clarkson for the case $n = 7$, $\Delta = 3$, and $OPT(I) = 3$. In this case, Clarkson's algorithm on our graph 2 in the proof of the theorem below would give a vertex cover with four vertices. The results hold for arbitrarily large graphs, since disjoint copies of the constructions can be used.

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

Proof. First note that both graphs in Fig. 3 have vertex covers of size 3. We will now



Fig. 3. Graph 1 to the left and graph 2 to the right.

force any adaptive priority algorithm 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. \square

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.⁵ Notice that with graph 2 in the proof, as long as the algorithm accepts the first vertex it processes, it will accept at least four vertices. Thus, only the one graph is necessary, when the algorithm is acceptances-first, so the algorithm can be given the number of vertices and the number of edges.

Theorem 7. *In the vertex adjacency model, 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).*

Theorem 8. *No fixed priority algorithm \mathbb{A} in the edge adjacency model for vertex cover can achieve an approximation ratio better than 2.*

The 2-approximation algorithms for vertex cover are adaptive rather than fixed priority, so the above result may not be tight. (We do not know of a fixed priority algorithm which is an $O(1)$ -approximation algorithm for vertex cover.)

⁵ If the number of edges should also be the same, we can add a cycle of 4 vertices to graph 2 and a cycle of 4 vertices with one diagonal to graph 1 and obtain a bound of $6/5$.

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 k -Colorability 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 $\text{NP} \neq \text{ZPP}$ [11]. It is NP-hard to approximate within $n^{\frac{1}{5}}$ provided that $\text{NP} \neq \text{coRP}$ and within $n^{\frac{1}{7}}$ provided that $\text{P} \neq \text{NP}$ [3].

From [16], it is known that it is NP-hard to 4-color a 3-chromatic graph, NP-hard to color a k -chromatic graph with at most $k + 2\lceil k/3 \rceil - 1$ colors, and NP-hard to approximate within n^ϵ for some fixed ϵ as the chromatic number of graphs tend towards infinity.

On the positive side, a general upper bound of $O(n \log \log^2 n / \log^3 n)$ is shown by Halldórsson [13]. In [19], an upper bound of $\lambda(G) + 1$ is established, where λ is any function of graphs $G = (V, E)$ such that

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

Let $d(G)$ be the maximum over all vertex-induced subgraphs of the minimum degree in that subgraph. The result in [19] constructively establishes that any graph is $d(G) + 1$ colorable, so a corollary of the theorem below is that the algorithm from [19] is not a priority algorithm.

Theorem 9. *No priority algorithm in the edge adjacency model can 3-color all graphs G with $d(G) = 2$.*

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 10. *Any fixed priority algorithm in the edge adjacency model must use at least $d + 1$ colors on a bipartite graph of maximum degree 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, \dots, 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 11. *Any adaptive priority algorithm in the edge adjacency model, 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).*

References

1. S. Angelopoulos. Ordering-preserving transformations and greedy-like algorithms. Manuscript, 2004.
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*, volume 2462 of *Lecture Notes in Computer Science*, pages 26–39. Springer-Verlag, 2002.
3. Mihir Bellare, Oded Goldreich, and Madhu Sudan. Free bits, PCPs and non-approximability—towards tight results. *SIAM Journal on Computing*, 27:804–915, 1998.
4. Piotr Berman and Toshihiro Fujito. On approximation properties of the independent set problem for degree 3 graphs. In *Fourth International Workshop on Algorithms and Data Structures*, volume 955 of *Lecture Notes in Computer Science*, pages 449–460. Springer-Verlag, 1995.
5. Ravi Boppana and Magnús M. Halldórsson. Approximating maximum independent sets by excluding subgraphs. *Bit*, 32:180–196, 1992.
6. A. Borodin, M. Nielsen, and C. Rackoff. (Incremental) priority algorithms. *Algorithmica*, 37:295–326, 2003.
7. Kenneth L. Clarkson. A modification of the greedy algorithm for vertex cover. *Information Processing Letters*, 16:23–25, 1983.
8. S. Davis and R. Impagliazzo. Models of greedy algorithms for graph problems. In *Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms*, 2004.
9. 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.
10. Jack Edmonds. Matroids and the greedy algorithm. *Mathematical Programming*, 1:127–136, 1971.
11. Urid Feige and Joe Kilian. Zero knowledge and the chromatic number. *Journal of Computer and System Sciences*, 57:187–199, 1998.
12. Johan Håstad. Clique is hard to approximate within $n^{1-\varepsilon}$. *Acta Mathematica*, 182:105–142, 1999.
13. Magnús M. Halldórsson. A still better performance guarantee for approximate graph coloring. *Information Processing Letters*, 45:19–23, 1993.
14. D. Hochbaum. Efficient bounds for the stable set, vertex cover and set packing problems. *Discrete Applied Mathematics*, 6:243–254, 1983.
15. David S. Johnson. Approximation algorithms for combinatorial problems. *Journal of Computer and System Sciences*, 9(3):256–278, 1974.
16. Sanjeev Khanna, Nathan Linial, and Shmuel Safra. On the hardness of approximating the chromatic number. *Combinatorica*, 20(3):393–415, 2000.
17. D. Lehmann, L. O’Callaghan, and Y. Shoham. Truth revelation in approximately efficient combinatorial auctions. *Journal of the ACM*, 49(5):1–26, 2002.
18. C. Papadimitriou and M. Yannakakis. Optimization, approximation and complexity classes. *Journal of Computer and System Sciences*, 43:425–440, 1991.
19. G. Szekeres and Herbert S. Wilf. An inequality for the chromatic number of graphs. *Journal of Combinatorial Theory*, 4:1–3, 1968.
20. Vijay V. Vazirani. *Approximation Algorithms*. Springer-Verlag, 2001.

A Defining Greedy Priority Algorithms

As observed in Davis and Impagliazzo [8] 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 $\delta(I)$ 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. 4. 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 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.

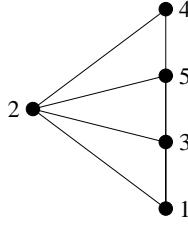


Fig. 4. Choosing the minimal color is not greedy.

B Omitted Proofs

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

Proof. The adversary uses the construction, \mathbb{G} , described in Section 3. Since all vertices have the same degree, the algorithms cannot distinguish between them and the algorithm might first select some $u \in U$. Since \mathbb{A} is acceptances-first, u is accepted (or else no vertices will be accepted). Since all vertices in V are adjacent to u , they must all be rejected. Since \mathbb{A} can accept at most one vertex in each clique, it accepts at most k vertices, giving a ratio of at least $\frac{k^2}{k} = k \in \Omega(\sqrt{n})$. \square

Theorem 5: Proof that the algorithm given in the body of the paper achieves an approximation ratio of $\frac{3}{2}$ for $k \geq 2$:

Proof. 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 IV. (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 III.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, $S \in I$, of weight-1 vertices, a maximum weight independent set contains at most $\sum_{S \in I} (\frac{l(S)}{2}) - \frac{1}{2}|O|$ 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

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

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

Theorem 7. *In the vertex adjacency model, 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 W 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 one, at least two other vertices must be chosen to cover all the edges. If the first vertex chosen has degree two, 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 one vertices. Thus, one obtains the ratio $3/2$. □

Theorem 8. *No fixed priority algorithm \mathbb{A} in the edge adjacency model for vertex cover can achieve an approximation ratio better than 2.*

Proof. The adversary uses copies of construction \mathbb{G} from section 3. Recall that all vertices have degree $k^2 + k$, so \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 phase. 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 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, 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. This gives a total of at least $k^2 + k^2$. Thus, in both cases the ratio is at least $\frac{2k}{k+1}$. \square

Theorem 9. *No priority algorithm in the edge adjacency model can 3-color a graph G with $d(G) = 2$, where $d(G)$ is the maximum over all vertex-induced subgraphs of the minimum degree in that subgraph.*

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 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.

In many of the cases below, we use 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. 5.

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 3, to make its degree 3 also. This will be denoted below by $\text{merge}(G, H)$. This extra degree three vertex can, for example, be made adjacent to two vertices which are adjacent to each other and to an additional vertex of degree 2.

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 F 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

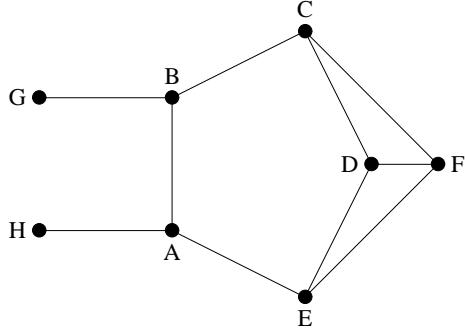


Fig. 5. Graph K .

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 gives vertex X .

Case A: The degree 2 vertex is never chosen; the adversary never shows it adjacent to anything until A has been forced to use four colors and the entire graph is revealed. 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 will be adjacent to two of W , X , and Z . The next vertex chosen 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 , but not more than one. 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 $C = Y$, $Z = \text{merge}(G, H)$ in the graph K , and we are done since $c(A) = c(C)$.

Case A.1.1.2: If $c(Y) \neq c(W)$, let $A = Z$, $B = \text{merge}(G, H)$, $C = Y$, and $D = X$, 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 $A = W$, $B = X$, $C = U$, $D = \text{merge}(G, H)$. Then $c(C) \neq c(E)$, 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 containing it which \mathbb{A} has processed. 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. 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 .

Case B.1: A vertex Y adjacent to Z is chosen next.

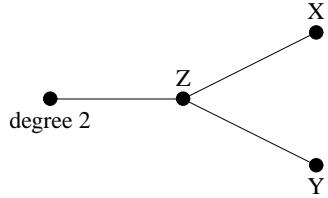


Fig. 6. Initial part of graph in Case B.1.

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.

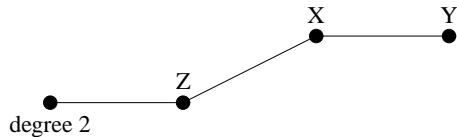


Fig. 7. Initial part of graph in Case B.2.

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 .

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.

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. \square

Theorem 10. *Any fixed priority algorithm in the edge adjacency model 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 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 $1, 2, \dots, i - 1$, but no other colors. The vertices chosen are made adjacent to one vertex of each color $1, 2, \dots, i - 1$, 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 $1, 2, \dots, d$. 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 . \square

Theorem 11. *Any adaptive priority algorithm in the edge adjacency model, 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 \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.

□