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`

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".

Keywords: approximation algorithms, priority algorithms, greedy algorithms, vertex cover, independent set, vertex coloring.

[*] 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). Parts of this work were carried out while these authors were visiting the University of Toronto and the University of California, Irvine.

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 [26], 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 general framework for studying greedy algorithms is still developing. 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 [10] 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 captures algorithms that process the input set in steps, where we assume that the input comes in the form of a set of input items. The greedy-like aspect is modelled by allowing the algorithm to choose, in a restricted way, the order in which the input items are processed. Informally, the restriction is that the order (or prioritizing) must be made by specifying the property the next input item should have, e.g., the item of largest size or the edge of smallest weight, etc. In the next section, we give precise definitions for the format of priority algorithms.

The priority model has two forms: fixed priority and the more general adaptive priority model. For both models, input items are treated one at a time, and each time, some irrevocable decision is made concerning the item. For fixed priority algorithms, a total order on all possible input items is specified in the beginning and input items are then treated one at a time according to that ordering. For adaptive algorithms, the ordering can depend on the items already considered, i.e., the algorithm can decide on a new ordering every time before processing the next item. It is crucial that the ordering is not determined by the actual input set, but rather it must apply to the set all of possible input items.

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. The version of facility location studied in [3] was for the disjoint model where the set of facilities and the set of clients/cities are disjoint sets. In contrast, in the complete model for facility location, there is just a set of cities and every city can be a facility. 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. In contrast, for the weighted vertex cover and maximum independent set problems, Davis and Impagliazzo use a vertex adjacency formulation, where input items are vertices, represented by their names and 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. We devote Section 6 to these discussions, including the issue of possibly reserving the term "greedy" for only a subset of all priority algorithms.

In order to establish lower bounds for priority algorithms, it is important to be precise about the behavior and power of an adversary. Contributions in this direction have been made by Davis and Impagliazzo as well as Angelopoulos. It is also important to define the precise form of the input to make it clear what can be deduced by a priority algorithm from seeing parts of the complete input. We give our definitions in the next section, and we further discuss differences from earlier work and relations between various concepts such as input representation, greediness, and memorylessness in Section 6.

In Sections 3, 4, and 5, we study the graph theoretic problems of vertex cover, independent set, and vertex coloring. 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. In particular, in Section 4, we establish a separation between the results that can be obtained using a vertex adjacency formulation versus an edge adjacency formulation.

2 Priority Algorithms for Graph Problems

Kruskal's and Prim's algorithms for Minimum Spanning Tree are standard examples of greedy algorithms, and both can be viewed as priority algorithms. We will use them as examples to present the two models, fixed and adaptive priority algorithms.

As part of the definition of an algorithmic problem where the input can be viewed as a set of input items, we let Γ denote the set (or universe) of all possible input items.

For some algorithmic problems, all finite subsets of Γ form valid input instances, whereas for other problems, this is not the case. For example, if a graph is given by its edges, any subset of the edges is a valid input instance. However, if a graph is given by vertices, where along with each vertex a list of its neighbors is given, then a subset of vertices not containing the vertex v, but containing a vertex u with v listed as one of its neighbors, would not be a valid input instance. We let Ψ denote the collection of all valid input set instances, i.e., the elements of Ψ are subsets of Γ.

Priority algorithms define (one or more) orderings on Γ which determine the item to be processed next in each iteration of the algorithm. The smallest (or first) item according to the ordering among those remaining in the input instance is processed next. This item has highest priority, hence the name "priority algorithm".

Fixed Priority Algorithms

Figure 1 shows the template for a fixed priority algorithm.

An algorithm is called a fixed priority algorithm if it can be formulated using the template. The notation $\min_{\leq_F} G_{i+1}$ denotes the minimum element in G_{i+1} with respect to the total ordering $\leq_F$. Note that the algorithm does not know the sets G_i, so it bases its irrevocable decisions on the general format for an input item (as captured by Γ) and the items, S, seen so far.

As an example, for Minimum Spanning Tree, the items are edges with their weights. If the weights are integers, the edges are represented by specifying the vertices they are incident to, and the vertices are given as integers, then one can let $\Gamma = \mathbb{Z} \times \mathbb{Z} \times \mathbb{Z}$, where the first component is the weight of the edge and the other two components are its two endpoints. For Kruskal's algorithm, the lexicographical ordering on the items in Γ gives a total ordering, $\leq_F$, which places edge $e_1 = (w_1, u_1, v_1)$ before $e_2 = (w_2, u_2, v_2)$ if $w_1 < w_2$ and breaks ties in some specific way (as we have

```
$\Gamma$ is the set of all possible input items
$G_0 \in \Psi$ is the input instance
decide on a total ordering $\leq_F$ of $\Gamma$
$S$ := $\emptyset$                        { the set of items already seen }
i := 0                                    { $|S|$ }
**while** $G_i \setminus S \neq \emptyset$ **do**
    $G_{i+1}$ := $G_i \setminus S$
    *item* := $\min_{\leq_F} G_{i+1}$
    make an irrevocable decision concerning *item*
    $S$ := $S \cup \{item\}$
    i := i + 1
**end**
```

Figure 1: The template for a fixed priority algorithm

defined it, lexicographically on the pairs (u_i, v_i)). The irrevocable decision made in Kruskal's algorithm is to include the edge in the spanning tree, or to reject it (if it would create a cycle). Thus, Kruskal's algorithm can clearly be expressed using the template.

When expressing concrete algorithms, we emphasize readability, and, thus, deviate slightly from the template. We introduce additional variables and control structures in order to compute the intended orderings and irrevocable decisions where this is most convenient. In all cases, however, it should be clear that the algorithms could be written so as to strictly follow the template.

One can view any online algorithm as a fixed priority algorithm where an adversary (rather than the algorithm) determines the ordering. The added power of a fixed priority algorithm is that it imposes a total ordering on Γ (albeit independent of the actual input set) and, because of this ordering, as items are being revealed, it also learns that certain items cannot be in the actual input set.

Adaptive Priority Algorithms

In Prim's algorithm, the next edge chosen is the lowest weight edge incident to the portion of the spanning tree which has already been constructed, but not creating a cycle. One can view this as choosing the first item in a total ordering which changes for each new item, as explained below.

Figure 2 shows the template for an adaptive priority algorithm.

```
$\Gamma$ is the set of all possible input items
$G_0 \in \Psi$ is the input instance
$S$ := $\emptyset$                 { the set of items already seen (processed) }
i := 0                  { $|S|$ }
**while** $G_i \setminus S \neq \emptyset$ **do**
    decide on a total ordering $\leq_{i+1}$ of $\Gamma$
    $G_{i+1}$ := $G_i \setminus S$
    *item* := $\min_{\leq_{i+1}} G_{i+1}$
    make an irrevocable decision concerning *item*
    $S$ := $S \cup \{$*item*$\}$
    i := i + 1
**end**
```

Figure 2: The template for an adaptive priority algorithm

An algorithm is called an adaptive priority algorithm if it can be formulated using the template. Note also here that the algorithm has no knowledge of the sets G_i, but bases its choices (the orderings and the irrevocable decisions) on the general format for an input item (as captured by Γ) and the items, S, seen so far.

Returning to Prim's algorithm, let N be the set of edges which have one vertex in the set $S \subseteq \Gamma$ of vertices already processed (in the current spanning tree) and one vertex outside that set (in $\Gamma \setminus S$), and let L be all other edges. Place the edges in N before all those in L in the total ordering. Within the two sets, N and L, the edges are ordered lexicographically, as in Kruskal's algorithm, so that edges with smaller weight come first. If the set N is non-empty, the first edge in this ordering is added to the tree and to S. Otherwise, the edge is rejected, but still added to S. Since Prim's algorithm can be expressed this way, it is an adaptive priority algorithm.

The extra power, in comparison with a fixed priority algorithm, is the ability to change the priorities of items in each iteration. This prioritizing is based on a total ordering defined on all possible input items, and is defined using only information about items already processed, i.e., those in S. From these items, the algorithm, ALG, can make some deductions concerning what items from Γ could be in the remaining part of the input sequence. Certainly, the items in S cannot be given again. Additionally, no item less than the last item chosen, according to the ordering $\leq_i$, can be in the input, and possibly other input items originally in Γ cannot lead to a valid input instance. However, the placement of these items in the total ordering cannot affect the computation since they will not be part of the input instance (since $G_0 \in \Psi$), i.e., they will not be elements of G_{i+1}.

Further Restrictions

We consider a variety of restricted forms for priority algorithms. The first of these is relevant for problems where the irrevocable decision concerning an input item is simply an accept/reject decision, e.g., should an edge be included in the minimum spanning tree or not. For such problems, we refer to a priority algorithm as *acceptances-first*, if it can choose orderings such that as soon as it has rejected an item, it never accepts an item again (and of course still solves the algorithmic problem in question).

The following two restrictions have to do with how much information algorithms use and store. The first of these applies to the class of accepts/reject problems. Assume that we partition the set S of items seen up to a given point in time into the sets A and R of accepted and rejected items, respectively, i.e., $A \cup R = S$ and $A \cap R = \emptyset$. Then we refer to an algorithm as *memoryless* if all its decisions (defining orderings and making irrevocable decisions concerning input items) are based only on A, i.e., all rejected items are ignored.

Finally, for graph problems where input items are vertices, we refer to a priority algorithm as *degree-based* if only the degrees of vertices are used when defining an ordering (as opposed to, for example, information regarding neighbors that may have been seen already or the names of the vertices or edges). Thus, vertices of the same degree cannot be distinguished when defining the ordering.

Input Representations

In the remainder of this paper, we assume that the input items are vertices in a graph. We use two different input representations for these vertices, depending on whether connections to neighbors are expressed through edges or vertices. Vertices and edges have names or labels (or are numbered in some way) to distinguish them from each other.

In the *vertex adjacency formulation*, the neighbors of a vertex v are given by a list of the vertices that are neighbors to v. In the *edge adjacency formulation*, neighbors are given as a list of edges.

The latter gives less information. For instance, consider a vertex v which has both u and w as neighbors. In the vertex adjacency formulation, if we see u and w (and they are not neighbors), then whether or not we have already seen v, we can conclude that u and w are at a distance two apart, since v appears as a neighbor of both. In the edge adjacency formulation, we would just get the names of the two different edges connecting u to v and v to w, and would not be able to infer that distance information.

Note that opposed to the situation for some online formulations, in our setting, a vertex comes with a complete neighbor list (in terms of either edges or vertices). In contrast, in some formulations of online problems, only connections to neighbors already seen are given.

Adversarial Arguments

Most often, the harder part of establishing the limits of what can be obtained using priority algorithms has to do with establishing lower bounds. When establishing lower bounds, one often uses the concept of an adversary that designs the input to make it hardest possible for an algorithm trying to solve the problem at hand.

For adaptive priority algorithms in particular, one has to be fairly careful when devising these types of arguments, since the algorithms can choose a new ordering for each iteration of the loop and the adversary has an obligation to end up with a valid input instance.

In Figure 3, we give an alternative formulation of an adaptive priority algorithm which clarifies the relationship between the algorithm and the adversary. This formulation is equivalent (just more formal) to the one already given. We will call an algorithm ALG an adaptive priority algorithm if it is possible to define functions δ and σ such that no matter which choices are made by the adversary, ADV, the result of running the template in Figure 3 is always the same as running the algorithm ALG on the input defined by ADV.

The template highlights the restrictions on the "game" most often written up in proofs as a case analysis based on the choices made by an algorithm in each iteration. The primary purpose of this template is to act as a precise definition of the power of priority algorithms (by defining exactly which adversary such algorithms are working against). Concrete algorithms will be formulated following the earlier templates which come much closer to a "programmer's view" on priority algorithms.

We now explain this template to clarify all the relevant concepts.

H_i represents the history after i iterations. It is an ordered sequence of pairs consisting of an input item together with the irrevocable decision that was made concerning that item. The decision function δ makes the irrevocable decision based on the history and the input item at hand. The ordering function specifies a total ordering of the elements of Γ. This ordering can be based on the history.

The restrictions on the adversary just formalize that it must present a valid input

```
$\Gamma$ is the set of all possible input items
$\Psi$ is the collection of all valid input sets

Specify:
    • a decision function $\delta(H, \textit{item})$
    • an ordering function $\sigma(H, \Gamma)$

$H_0$ := $\langle\rangle$          { the history of input items and irrevocable decisions }
$i$ := $0$                { number of input items given }

while ADV does not choose to terminate
    $\leq_{i+1}$ := $\sigma(H_i, \Gamma)$                      { ordering on input items}
    ADV gives next item, $\textit{item}_{i+1}$                 { next input item }
    $d_{i+1}$ := $\delta(H_i, \textit{item}_{i+1})$                 { an irrevocable decision }
    $H_{i+1}$ := $H_i$ ++ $\langle(\textit{item}_{i+1}, d_{i+1})\rangle$           { updating the history }
    $i$ := $i + 1$
endwhile
```

There are the following restrictions on ADV's choice to terminate and on ADV's choice of input items to give. Let $\textit{item}_1, \textit{item}_2, \ldots, \textit{item}_n$ be the sequence of items given before ADV terminates the loop. Then ADV must ensure that

$$\{\textit{item}_1, \textit{item}_2, \ldots, \textit{item}_n\} \in \Psi, \text{ and}$$

$$\forall i \in \{1, \ldots, n\}\ \forall j \in \{1, \ldots, i-1\}:\ \textit{item}_j <_j \textit{item}_i$$

Figure 3: A formal template for an adaptive priority algorithm.

set, and that it must be consistent with the total orderings defined by the priority algorithm, i.e., the adversary is not allowed to present an item which according to an earlier ordering is smaller than the item given at that time. Or phrased positively: when presenting the priority algorithm with the next item, an item given in an earlier round must be smaller according to the ordering used in that earlier round.

An alternative, but equivalent, way of viewing this is as follows: Let $\Gamma_0 = \Gamma$. The adversary choosing $\textit{item}_{i+1}$ is equivalent to its restricting the items which could still be part of the remaining input to the subset $\Gamma_{i+1} = \Gamma_i \setminus \{x \in \Gamma_i \mid x \leq_{i+1} \textit{item}_{i+1}\}$. In this view, ADV defines sets $\Gamma_1 \supset \Gamma_2 \supset \Gamma_3 \supset \cdots$ before each new input item. Before the choice of and processing of the ith item, Γ_i is a set of items, all of which can lead to a well-defined input instance, i.e., an input instance in Ψ. The input to the algorithm, $\{\textit{item}_1, \textit{item}_2, \ldots, \textit{item}_n\}$, is the sequence

$\langle \min_{\leq_1} \Gamma_1, \min_{\leq_2} \Gamma_2, \ldots, \min_{\leq_n} \Gamma_n \rangle$.

Typical structure of an adversarial argument

In graph problems where the graphs are given by vertices, an adversarial argument will typically involve one or more graph constructions, where the algorithm should receive some vertices before others. In the set (or universe), Γ, of all possible input items, each item would contain a vertex label, plus a set of edge (or vertex) labels for the edges incident to (vertices adjacent to) that vertex. For each vertex degree, d, possible in the construction, there will typically be an item with each possible vertex label associated with each possible subset of d edge (or vertex) labels. Based on the algorithm's ordering of the items in Γ, the adversary can assign labels to the vertices and edges in its constructions. Thus, the adversary can, for example, decide that the first vertex in the ordering is any of the vertices in the construction with the same degree (and weight if the vertices have weights). Later choices as to which vertex is the next chosen will be restricted by which of the already processed vertices it is adjacent to and by its degree (and weight). In the vertex adjacency formulation, there is an additional restriction based on neighbors that this vertex has in common with vertices already processed.

Input Size

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. 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 polynomial 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 (that is 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 (or slightly better). One such algorithm is the "layered algorithm" as given in [30]. 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[1]. Both the layered algorithm and Clarkson's algorithm 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 [24], Khot and Regev [25] 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

The matching algorithm for vertex cover proceeds by continually choosing some edge not yet covered and adding both of the edge's endpoints to the current cover, C. First, we show the easier of two implementations of the maximal matching algorithm as a priority algorithm.

Theorem 1 The matching algorithm for vertex cover can be implemented as an acceptances-first (memoryless) adaptive priority algorithm in the edge adjacency formulation.

Proof Suppose the input set, V, is a subset of the set, Γ, of possible vertices. In order to see that the matching algorithm can be written as an acceptances-first

[1] The stated bound is not defined for $\Delta \leq 2$. The more general bound that applies to all Δ is that $w(C_{MG}) \leq w(C_{OPT}) - \frac{2(n-w(C_{MG}))}{\Delta}$. Here, C_{MG} is the cover obtained by Clarkson's Modified Greedy algorithm and C_{OPT} is the cover obtained by OPT.

adaptive priority algorithm, we use an ordering satisfying the following property: $P($*Marked*$, \leq)$ is satisfied by an ordering $\leq$ if and only if all vertices incident to some edge not in *Marked* are smaller (i.e., have higher priority) than any vertex only incident to edges in *Marked*. The algorithm is listed in Figure 4.

```
Marked := $\emptyset$                    { the set of edges already covered }
$C$ := $\emptyset$                         { the cover }
i := 0
ChooseNewEdge := true
while $V \neq \emptyset$
    if ChooseNewEdge
        choose a total ordering $\leq_{i+1}$ of $\Gamma$ satisfying $P(Marked, \leq_{i+1})$
        $u$ := $\min_{\leq_{i+1}} V$
        $V$ := $V \setminus \{u\}$
        if $\exists$ $e$ incident to $u$ such that $e \notin Marked$
            $C$ := $C \cup \{u\}$
            edge := $e$
            ChooseNewEdge := false
    else
        choose an ordering $\leq_{i+1}$ of $V$ with vertices incident to edge first
        $v$ := $\min_{\leq_{i+1}} V$
        $V$ := $V \setminus \{v\}$
        $C$ := $C \cup \{v\}$
        Marked := Marked $\cup$ { all edges incident to $u$ or $v$ }
        ChooseNewEdge := true
    i := i + 1
endwhile
```

Figure 4: The matching algorithm for Vertex Cover as an acceptances-first adaptive priority algorithm.

Note that knowing the number of vertices and/or edges in advance is not necessary.

The marking of edges does not need additional memory other than that for the set C of accepted items, since the marked edges are those incident to vertices in C. Thus, it is memoryless. The algorithm is acceptances-first because when the first vertex is rejected, there are no more uncovered edges. $\square$

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. 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. This gives us the following:

Theorem 2 The maximal matching algorithm can be implemented as a (memoryless) fixed priority algorithm in the vertex adjacency formulation.[2]

Proof Consider the algorithm in Figure 5, which was informally described above, for an input graph $G = (V, E)$. 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, 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. $\square$

It is instructive to consider why the argument behind Theorem 2 does not extend to the edge adjacency formulation. Suppose we try to implement the maximal matching algorithm as an online algorithm (an algorithm which does not determine the ordering of the input vertices) as in Theorem 2. Suppose that u is the vertex of highest priority (first in the ordering) 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 (adjacent to v) can be removed. Thus, all of v's neighbors could be accepted before seeing v, and each one could be "matched" to v. Beyond online algorithms, a priority algorithm could order vertices so that vertices are seen in adjacent pairs, by placing vertices adjacent to a particular edge e_1 first in the total ordering, then those (still unseen) vertices adjacent to e_2, and so on. Suppose the first adjacent pair of vertices is u and v_1, so u and v_1 are put in the cover C. If the ordering of the edges is such that the next adjacent pair is the same u and v_2

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

```
Choose any ordering $\leq_F$
$C := \emptyset$
$X := V$
$L := \emptyset$
while $X \neq \emptyset$
    $u := \min_{\leq_F} X$
    if $u \in L$
        $C := C \cup \{u\}$        { 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 $v \notin C \cup L$
        $X := X \setminus \{u\}$        { 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 }
endwhile
```

Figure 5: The matching algorithm for Vertex Cover as a fixed priority algorithm.

and if v_2 is adjacent to a vertex v_3, which has not been seen yet, but is “matched” to v_2 by the maximal matching algorithm, then v_2 must be accepted and we must later accept v_3. However, now v_3 causes the same problems as the v in the online variant above. That is, we do not know what other vertices share edges with v_3.

3.2 Limitations on priority algorithms for vertex cover

Although both an acceptances-first adaptive priority algorithm in the edge adjacency formulation and a fixed priority (in fact, online) algorithm in the vertex adjacency formulation can achieve a 2-approximation ratio by implementing a maximal matching algorithm, Theorem 3 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 2, we conjecture that Theorem 3 applies to *any* fixed priority algorithm using the edge adjacency formulation. The proof below uses the construction in Avis and Imamura’s proof [5] of the similar result[3] where they proved an $\Omega(\sqrt{n})$ inapprox-

[3] Although we use the Avis and Imamura construction, our Theorem 3 is incomparable with the Avis and Imamura result since they require a degree-based ordering while we require acceptances-

imation 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 3 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 6 which illustrates the construction for $k = 3$.

We consider any acceptances-first fixed priority algorithm that computes a vertex cover for the graph G. Each input item then consists of one vertex label and a set of edge labels representing the edges incident to the vertex. The set, Γ, of possible input items consists of all possible combinations of vertex and edge labels where the size of the set of edge labels (the size of the edge adjacency list) is either k or k^2. There are, of course, many more possible input items than there are vertices in the graph and only certain subsets of n vertices will constitute a valid input set. The fixed priority algorithm must create a total ordering $\leq_F$ of all possible input items. To derive an inapproximability result, we consider an adversary which is at liberty to select the set of input items (i.e., to set the actual labels for vertices and edges) that will comprise the actual input set.

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 (by the acceptances-first assumption) 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 how the adversary creates a labelling of the graph such that the described processing order is obtained. Let v_k with adjacency list $\{e_k^1, \ldots, e_k^k\}$ be the last (w.r.t. $\leq_F$) input item in Γ of a vertex having degree k. Then label one of the vertices in V_2 by v^k and its adjacent edges by the $\{e_k^i\}$. Now let v_{k-1} with adjacency list $\{e_{k-1}^1, \ldots, e_{k-1}^k\}$ be the last (w.r.t. $\leq_F$) input item in

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.

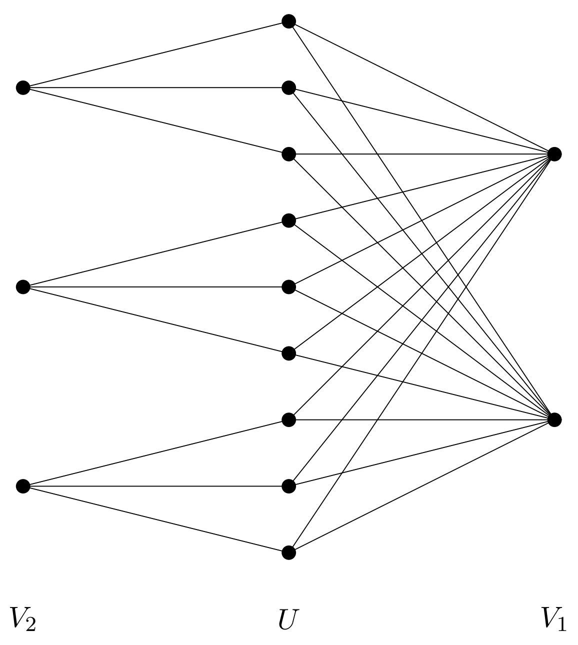

Figure 6: The Avis/Imamura construction for $k = 3$.

Γ such that $v_{k-1} \neq v_k$ and $e_k^i \neq e_{k-1}^j$ for all $1 \leq i, j \leq k$. Label another vertex in V_2 by v_{k-1} and its adjacent edges by the $\{e_{k-1}^i\}$. We continue in this way to inductively label all k vertices in V_2. We use the remaining vertex and edge labels to consistently label the vertices in U and V_1 and the edges in $U \times V_1$ so as to create a valid input instance. Such a labelling clearly has the desired property that each vertex in V_2 comes after its neighbors in the $\leq_F$ ordering. $\square$

Note that restrictions on rejection, which both acceptances-first and list processing impose, are necessary in using the above construction to establish the stated lower bound. 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, especially given the conjecture that $\Omega(\sqrt{n})$ is the best approximation ratio for fixed priority algorithms in the edge adjacency model.

Theorem 4 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 would stop 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 would stop 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 contrast 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 assumption that the algorithm only considers degrees in ordering prevents the algorithm from ensuring that two adjacent vertices are chosen first.

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

Theorem 5 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 Figure 7 have vertex covers of size 3.

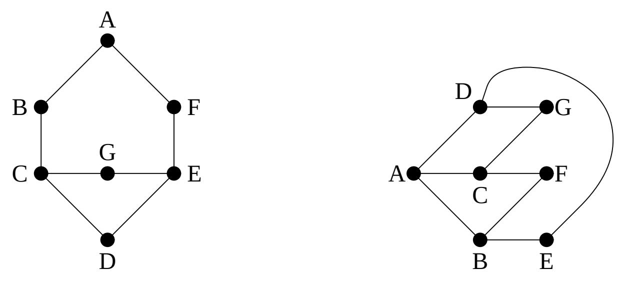

Figure 7: Graph 1 to the left and Graph 2 to the right.

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, the adversary lets it be vertex A in Graph 1. If $\mathbb{A}$ accepts a degree 2 vertex first, the adversary lets it be vertex B in Graph 1. If $\mathbb{A}$ rejects a degree 3 vertex first, the adversary lets it be vertex C in Graph 1. If $\mathbb{A}$ accepts a degree 3 vertex first, the adversary lets 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.) $\square$

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.[4]

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.

[4] If the number of vertices 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 $6/5$.

Theorem 6 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 smallest vertex in the first total ordering) 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 $3/2$. $\square$

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 [28]. 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 vertices. 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[5] of a construction in the thesis of Mirmohammadi [27].

Construction $\mathbb{P}$: There are three sets of vertices, A, B, and W (refer to Figure 8). 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

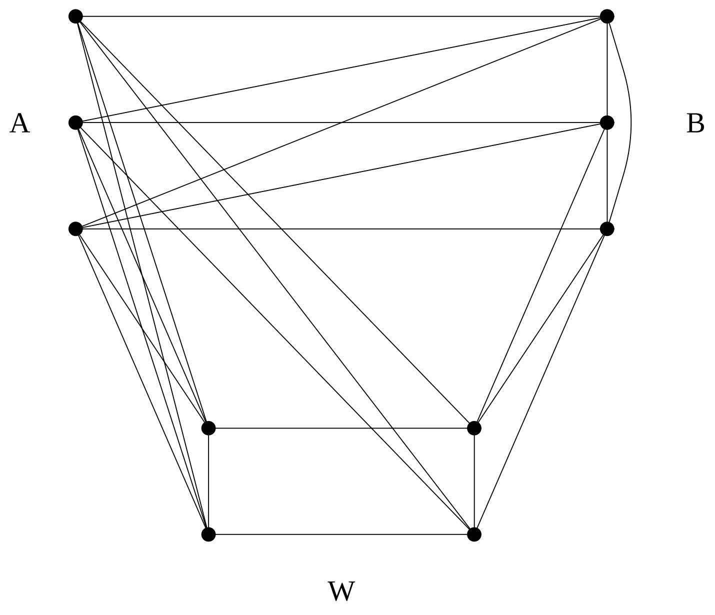

Figure 8: Construction $\mathbb{P}$ for $k = 3$.

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

[5] 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. Moreover, strong inapproximation results are still obtained in [27] when the model is extended to allow revocable acceptances as previously studied in (for example) [20, 1, 31]. However, the construction in [27] does not provide an inapproximation bound for the vertex adjacency formulation.

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 7 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 formulation 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. $\square$

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 8 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 that come first in the ordering 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 9 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 10 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:

The algorithm initially orders vertices so that all vertices of weight 1 precede vertices of weight k. If there are no vertices of weight 1, accept the first (in the ordering) vertex of weight k. Then follow it around the cycle (by adaptively changing the ordering to give priority to a neighbor not already seen), 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. Place vertices with weight 1 which are not adjacent to anything processed yet first in the ordering, 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 $\frac{3}{2}$. $\square$

Thus, with regards to adaptive priority algorithms for the WIS(k) problem, $\frac{3}{2}$ is the exact approximation ratio which can be obtained in the *edge* adjacent formulation. In combination with the result from Davis and Impagliazzo [12], described above, stating that in the *vertex* adjacency formulation there is an adaptive priority algorithm whose approximation ratio approaches one as k goes to infinity, we obtain the following:

Corollary 11 For adaptive priority algorithms, there is a strict separation between the approximation ratios that can be obtained in the vertex adjacency formulation and the edge adjacency formulation, respectively.

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 theoretic assumptions: minimum vertex coloring is NP-hard to approximate within $n^{\frac{1}{7}}$ [6]. Provided that NP $\neq$ ZPP, Khot [23] shows that it is NP-hard to approximate within $\Omega\left(\frac{n}{2^{(\log n)^{1-\gamma}}}\right)$, for some $\gamma > 0$. This improves the earlier result of $\Omega(n^{1-\epsilon})$, for all ϵ, under the same condition [15].

Khot also shows that for sufficiently large k, it is NP-hard to color a k-chromatic graph with $k^{\frac{1}{25}\log k}$ colors, asymptotically improving the earlier result that it is NP-hard to color a k-chromatic graph with at most $k+2\lceil k/3\rceil - 1$ colors [22]. In [22], it is also shown that it is NP-hard to 4-color a 3-chromatic graph.

On the positive side, a general upper bound of $O(n \log\log^2 n/log^3 n)$ is shown by Halldórsson [18]. In [29], an upper bound of $\lambda(G)+1$ is established, where λ is any function of graphs $G=(V,E)$ such that

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

Let $d(G)$ be the maximum over all vertex-induced subgraphs of the minimum degree in that subgraph. The result in [29] constructively establishes that any graph is $d(G)+1$ colorable, so a corollary of the theorem below is that the algorithm from [29] 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 Figure 9 to some vertex

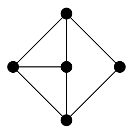

Figure 9: Attachment graph for incrementing vertex degrees while only adding degree 3 vertices.

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 Figure 10.

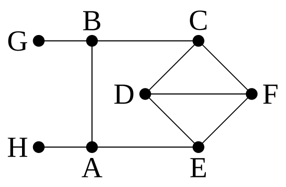

Figure 10: Graph K.

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 algorithm never gives an ordering where a degree 2 vertex comes first in the ordering before ending in a position where it is forced to use four colors); 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 Figure 11. The first vertex

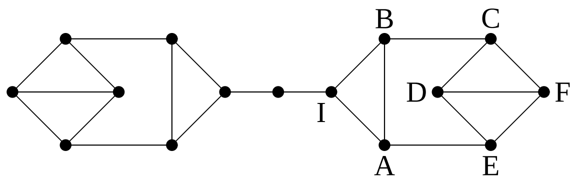

Figure 11: The Case A Graph.

chosen, W, has degree 3.

Case A.1: The next vertex chosen, X, is adjacent to W. The adversary ensures 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, and X will be adjacent to two of W, X, and Z; see Figure 12. The next vertex chosen may be

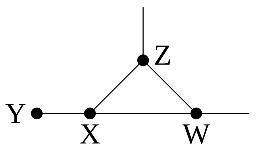

Figure 12: $\mathbb{A}$'s initial view of the graph in Case A.

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 Figure 10 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 Figure 13.

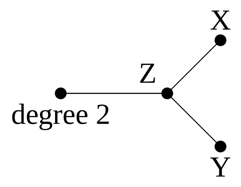

Figure 13: Initial part of graph in Case B.1.

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 Figure 14.

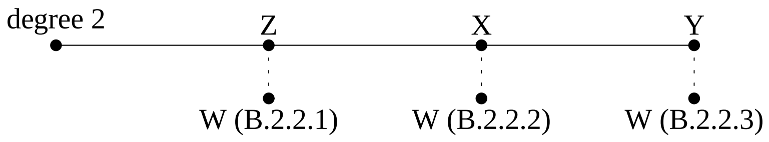

Figure 14: Initial part of graph in Case B.2. The position of W depends on subcases.

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 Figure 9.

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 Figure 9.

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. $\square$

In more restrictive models, we obtain stronger lower bounds. The following result applies, for example, to the simplest and most natural fixed priority algorithm: order the vertices by non-increasing (or non-decreasing) degree and then color vertices using the lowest possible numbered color. We note that this natural greedy algorithm colors any graph having maximum degree d using at most $d + 1$ colors.

Theorem 13 Any degree-based fixed priority algorithm in the edge adjacency formulation must use at least $d + 1$ colors on a (2-colorable) bipartite graph of maximum degree d.

Proof Informally, the adversary will create a d-regular bipartite graph containing many disjoint "portions" (induced bipartite subgraphs), each of which will have the same number of vertices and the same colors in each side of the partition (there could be additional colors outside these portions). These portions will grow in size and it may be necessary to join two portions, making the correct decision as to which side of the one portion is placed with which side of the other. At the end all vertices will have degree d, so the degree-based fixed priority algorithm has no real power in assigning priorities; as with on-line algorithms, the adversary has complete control over the input sequence. The algorithm's only choice is which color to assign after seeing which of its adjacent vertices are already colored.

Consider any degree-based fixed priority algorithm in the edge adjacency formulation. 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 are given the same color to make the remainder of the proof possible. Throughout our construction, whenever the algorithm first uses $d + 1$ colors, the remainder of the construction will add additional vertices (as explained later) so as to form a d-regular bipartite graph. This requires at all stages of the construction, that the partial graph that has so far been specified can be so completed. We will not specify an upper bound on the number of vertices used in this construction, but it will be clear that some large finite number will be sufficient. Stage 1 ends when there are enough vertices given the same color, which we call color 1. In stage 2, vertices adjacent to exactly one vertex having color 1 are given. That is, the adversary is creating a matching between color 1 and color 2 vertices. This continues similarly until either $d + 1$ colors are given in all or enough of these new vertices are given the same color, which we call color 2.

Now all the portions being considered consist of two adjacent vertices, one with color 1 and one with color 2. It is essential to note that the adversary does not have to commit to which side of the partition it will eventually place these adjacent nodes. In stage 3, these portions are combined in pairs. Suppose the adjacent vertices (v_1^1, v_2^1) are combined with (v_1^2, v_2^2), where v_j^i has color j. Two new vertices u_1 and u_2 are given, with u_1 adjacent to both v_1^1 and v_2^2 and with u_2 adjacent to both v_2^1 and v_1^2. The result is a bipartite graph (a 6-cycle), with u_1 and u_2 in different parts of the partition. Since each u_i is adjacent to one vertex of color 1 and one of color 2, the algorithm must give each u_i a new color; the two could either get the same color or different colors. Continue combining pairs of portions until either $d + 1$ colors are used or there are enough (and evenly many) combined portions with the same color, 3, for each u_i, or the same pair of colors, 3 and 4, for these two vertices. In the former case, each 6-cycle has each of the three colors on both sides of the partition. In the latter case, since the adversary has not committed to which side of the partition the v_j^i lie in, it is also free to later choose which side of the partition each u_i lies in. Pairs of these 6-cycles will be combined so that for each pair, there is at least one vertex of each of the first four colors in each part. These pairs are the induced subgraphs for the next stage. In this case, the next stage is stage 5, but if there were enough cases where both u_1 and u_2 got the same color, the next stage is stage 4.

In stage $i \geq 4$, the partial graph constructed thus far contains a large number of disjoint equi-partitioned bipartite subgraphs G_i with maximum degree at most $i - 2$, where both sides contain vertices with colors $1, 2, ..., i - 1$ (the purpose of separately treating stages 1 through 3 is to establish this property of having all of the first $i-1$ colors on both sides). There are additional vertices that were also seen but not used in these G_i subgraphs. We need only claim that all such additional vertices have degree less than $i - 1$ with equal number of vertices on each side of any induced partition so that these vertices can also be extended to be part of the final bipartite graph. The next vertices chosen are made adjacent to one vertex of each color $1, 2, ..., i - 1$, all from one partition of one of the induced subgraphs. As in stage 3, this is done for both parts of the partition. If there are a large enough number of subgraphs 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 subgraphs given the same two additional colors, which will be called i and $i + 1$. Graphs of this type can be combined in pairs, as in the case where colors 3 and 4 were given, arranging that for each pair one vertex of color i and one of color $i+1$ is on each side of the partition. Then, the adversary proceeds to stage $i + 2$.

The adversary stops this process as soon as $d + 1$ colors have been used. At that point, some vertices may have degree less than d in the induced subgraph created so far. However, every vertex which is presented to the algorithm must have degree d. We now explain how additional vertices and edges can be added to the construction so that every vertex gets degree d. This must be done in a manner consistent with the algorithm's observations during the different stages, i.e., we cannot introduce an edge between two vertices that have already been treated (colored) and have been observed to be independent. This can be done because each part of the partition in the induced subgraph has the same number of vertices, say n, in any induced partition, and the sum of their degrees is the same, say s. One can add $2(dn - s)$ edges, each one to a new vertex, to get the degree of each of the original $2n$ vertices up to d. This gives $dn - s$ new vertices in each part. Add s new vertices to each part and edges to make a matching between them. Now there are dn new vertices on each side, each of degree 1. For any d, it is easy to create a $(d - 1)$-regular bipartite graph on d vertices. Add the edges for this to n disjoint subsets of the new vertices.

Note that if fewer than $d + 1$ colors are used before stage $d + 1$, a $(d + 1)$st color will be used on the first vertex in that stage, 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. □

6 Priority Concepts and Relationships

In this section, we discuss the issue of defining a natural concept of greediness. We also discuss memorylessness, adversaries, restricted models, and input representations, as well as relationships between these concepts.

In either of the input formulations used in this paper, 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 this section.

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.

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, 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. However, for non-isolated inputs, it may be the case that any valid input instance will require further input items, e.g., if items are vertices represented by their vertex adjacency lists and there are vertices known to exist, but not yet processed.

Let S be the set of items already processed plus the item currently being considered. We say that a set T of input items is a *minimal completion set* if $S \cup T$ constitutes a valid input instance and $S \cup T'$ is not a valid input instance for any set $T' \subset T$. 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 T, there is a set of decisions for the items in T such that no other set of decisions for I and the items in T 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.

Any priority algorithm for the vertex cover problem which accepts a vertex if not all of its incident edges are already covered is 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 Figure 15. If the vertices are chosen in the order $\langle 1, 2, 3, 4, 5\rangle$, then vertex 4

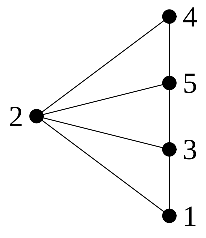

Figure 15: Choosing the minimal color is not greedy.

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.

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*. However, 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], [3], and here. 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 ("no operation"). 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[6] in [10] and [3], we have the following:

Theorem 14 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. However, many of these algorithms (for instance 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.

With regards to the formulation of the power of the adversary, 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 and we use the same model. 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

[6] 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.

not let an algorithm's priority distinguish between vertices that have the same degree. This corresponds to our 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 have presented 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.

With regards to input representation, we note 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 established in Theorem 9 (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. The matching algorithm for vertex cover can be realized as a fixed priority algorithm in the vertex adjacency formulation (Theorem 2). However, while the same algorithm can be realized as an adaptive algorithm in the edge adjacency formulation (Theorem 1), we do not believe a fixed priority algorithm can achieve any $O(1)$ approximation for vertex cover in the edge adjacency formulation.

With respect to fixed priority algorithms, we have studied 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. While it may seem that a fixed priority algorithm cannot utilize any information about adjacent vertex/edge names when assigning priorities, it is not hard to see that such an algorithm can ensure that vertices are considered in adjacent pairs, for example by placing all the vertices incident to a certain edge first in the ordering, followed by those incident to another edge, etc. 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. 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.

7 Concluding Remarks 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. However, 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. *Algorithmica*, 54:269–317, 2009.

[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. Improved inapproximability results for maxclique, chromatic number and approximate graph coloring. In *Proceedings of the 42nd Annual IEEE Symposium on Foundations of Computer Science*, pages 600–609, 2001.

[24] 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.

[25] 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.

[26] 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.

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

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

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

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

[31] 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.