AUTHOR QUERY FORM

ELSEVIER	**Journal:** Theoretical Computer Science **Article Number:** 7797	**Please e-mail or fax your responses and any corrections to:** **E-mail: corrections.esnl@elsevier.river-valley.com** **Fax: +44 1392 285879**

Dear Author,

Any queries or remarks that have arisen during the processing of your manuscript are listed below and highlighted by flags in the proof. Please check your proof carefully and mark all corrections at the appropriate place in the proof (e.g., by using on-screen annotation in the PDF file) or compile them in a separate list.

For correction or revision of any artwork, please consult http://www.elsevier.com/artworkinstructions.

Articles in Special Issues: Please ensure that the words 'this issue' are added (in the list and text) to any references to other articles in this Special Issue.

Uncited references: References that occur in the reference list but not in the text – please position each reference in the text or delete it from the list.	
Missing references: References listed below were noted in the text but are missing from the reference list – please make the list complete or remove the references from the text.	
Location in article	**Query / remark** **Please insert your reply or correction at the corresponding line in the proof**
Q1	Please check the corresponding address.
Q2	Please check the position of end of proof symbol given in all cases.

Electronic file usage

Sometimes we are unable to process the electronic file of your article and/or artwork. If this is the case, we have proceeded by:

☐ Scanning (parts of) your article ☐ Rekeying (parts of) your article ☐ Scanning the artwork

Thank you for your assistance.

Theoretical Computer Science xx (xxxx) xxx–xxx

Contents lists available at ScienceDirect

Theoretical Computer Science

journal homepage: www.elsevier.com/locate/tcs

Randomized priority algorithms[☆]

Spyros Angelopoulos[a,*], Allan Borodin[b]

[a] Max-Planck-Institut für Informatik, Campus E1 4, Saarbrücken 66123, Germany
[b] Department of Computer Science, University of Toronto, Toronto, Ontario, Canada M5S 3G4

ARTICLE INFO

Article history:
Received 16 June 2009
Received in revised form 23 January 2010
Accepted 2 March 2010
Communicated by A. Fiat

Keywords:
Greedy algorithms
Randomized approximation algorithms
Facility location
Makespan scheduling

ABSTRACT

Borodin, Nielsen and Rackoff [13] introduced the class of priority algorithms as a framework for modeling deterministic greedy-like algorithms. In this paper we address the effect of randomization in greedy-like algorithms. More specifically, we consider approximation ratios within the context of randomized priority algorithms. As case studies, we prove inapproximation results for two well-studied optimization problems, namely facility location and makespan scheduling.

© 2010 Published by Elsevier B.V.

1. Introduction

Greedy algorithms are a popular approach in combinatorial optimization and approximation algorithms. This is mainly due to their conceptual simplicity as well as their amenability to analysis. In fact, one reasonably expects a greedy algorithm to be one of the first approaches an algorithm designer employs when facing a specific optimization problem. It would therefore be desirable to know when such an approach is not likely to yield an efficient approximation. However, while it is relatively easy to identify a greedy algorithm based on intuition and personal experience, a precise definition of such a class of algorithms is needed so as to prove limitations on its power. Even more importantly, a rigorous framework for greedy algorithms can hopefully provide insight on how to develop better, more efficient algorithms. Q1

The term *greedy algorithm* seems to have first appeared in print in seminal papers by Edmonds [20,21]. The greedy algorithm concept, however, was surely understood in earlier papers. There is indeed a significant literature concerning problems where a "natural greedy algorithm" performs well. In addition to the work of Edmonds, some early important papers in this regard are (for example) due to Rado [46], Hoffman [32], Graham [28,29], Fisher et al. [23] and Korte and Lovász [36,37]. Despite the long standing popularity and importance of greedy algorithms as an algorithmic paradigm, it was only relatively recently that a formal framework for a general study of greedy algorithms began to emerge. In particular, Borodin, Nielsen and Rackoff [13] introduced the class of *priority algorithms* as a framework for abstracting the main properties of deterministic greedy-like algorithms. Within the proposed framework, they showed how to derive lower bounds on the approximability of various scheduling problems by such a class of algorithms.

In hindsight, the priority framework is very similar to the ideas proposed by Dechter and Dechter [19] who are concerned with optimal greedy algorithms for non-linear objective functions in the context of greedoids. Following the introduction of the priority model, there have been a number of further papers (see Section 3) studying the limitations of *deterministic*

[☆] A preliminary version of this paper appeared in the First Workshop on Approximation and Online Algorithms (WAOA), 2003.
[*] Corresponding address: Max Planck Institute for Computer Science, Campus E1 4, Saarbruecken 66123, Germany. Tel.: +49 681 9325 119; fax: +49 681 9325 199.
E-mail addresses: sangelop@mpi-inf.mpg.de (S. Angelopoulos), bor@cs.toronto.edu (A. Borodin).

0304-3975/$ – see front matter © 2010 Published by Elsevier B.V.
doi:10.1016/j.tcs.2010.03.014

priority algorithms. In this paper we extend the priority algorithm paradigm so as to provide a formal definition of *randomized* greedy-like algorithms. This framework allows us to use the *Von Neumann/Yao principle* in the context of priority algorithms, and thus prove lower bounds for algorithms that behave in a greedy-like fashion but also allow randomization. As case studies, we apply the technique to two well-studied NP-hard problems, namely, metric facility location and makespan minimization on identical machines.

Deterministic greedy-like algorithms are modeled as priority algorithms, which are, informally, algorithms with the following two properties: (i) the algorithm considers each input item in some "allowable order"; and (ii) as each input item is considered, the algorithm makes an "irrevocable decision" concerning the input item. Here, the decision on how to represent an input (i.e., what the input items are) and the precise meaning of an irrevocable decision are pertinent to specific problems. A precise formulation of priority algorithms is given in Section 2.

Depending on whether the ordering of input items can change throughout the execution of the algorithm, we define two distinct classes of priority algorithms. *Fixed* (order) priority algorithms commit to an ordering prior to considering any input item, and the ordering cannot change in any subsequent iteration. On the other hand, *adaptive* (order) priority algorithms can specify a new ordering after each input item is considered and processed.[1] As in [13] we use the term "priority algorithm" to refer to an algorithm in the more general class of adaptive priority algorithms which clearly encompasses fixed priority algorithms.

Following [13], a further distinction can be made among (fixed or adaptive) priority algorithms. We call a priority algorithm *greedy* if every irrevocable decision optimizes the given objective function, as if the current input item was the last to be considered by the algorithm. That is, every decision must optimize the given objective function. In view of this definition not every greedy-like (that is, priority) algorithm is greedy.

There are two natural ways to introduce randomization in a priority algorithm. First, the algorithm may choose a *random ordering* of the input items. Second, the algorithm can make *random decisions* for each input item. We emphasize again that priority algorithms need not be greedy priority algorithms and even for greedy priority algorithms there can be many decisions that are greedy in the sense of optimizing the objective function. In the case where randomization can be applied to both orderings and decisions we refer to such algorithms as *fully randomized algorithms*.

Note that every randomized *online algorithm* is a special case of a fixed priority, random decisions algorithm, where the ordering is precisely the order in which the input items arrive (i.e., as determined by the adversary). It is well known that randomization can be helpful in improving the competitiveness of online algorithms. As an illustrative example, we mention the randomized algorithm of Bartal et al. [7] for makespan scheduling on two machines which achieves the best possible randomized competitive ratio, namely $\frac{4}{3}$; in contrast there is a $\frac{3}{2}$ lower bound on the competitive ratio of every deterministic online algorithm. Likewise, there is evidence that randomization may be useful in improving the approximation ratio of priority algorithms. For instance, Borodin et al. [13] presented a "classify and randomly select" priority algorithm for the problem of interval scheduling, and showed that it is an $O(\log \Delta)$-approximation (here Δ is the ratio of the maximum profit per unit size over the minimum profit per unit size amongst all jobs in the input). In contrast, they showed that, for interval scheduling on a single machine, no deterministic priority algorithm can achieve better than a Δ-approximation.

Contributions of this paper. Our main goal is to be able to derive lower bounds on the approximability of optimization problems by randomized priority algorithms. We apply the framework to the following well-known problems:

1. The *metric facility location problem.* (a) For the model in which the input is represented as a set of facilities, we show that no fully randomized priority algorithm is better than a $\frac{4}{3}$-approximation. For randomized ordering, fixed priority greedy algorithms, we prove a lower bound of $\frac{3}{2}$ on the approximation ratio. (b) For the model in which the input is represented as a set of cities, we show that no fully randomized priority algorithm is better than a 3-approximation.

2. The *makespan scheduling problem* on identical machines. We show that no fully randomized priority algorithm achieves an approximation ratio better than $\frac{12}{11}$. For fixed priority algorithms with (only) randomized decisions, and for the case of at least three machines, we show an approximation lower bound of $\frac{10}{9}$.

We emphasize that as in the case of deterministic priority algorithms (and similar to competitive analysis in the online world), inapproximation bounds are derived by exploiting the syntactic structure of the algorithms, and are orthogonal to any complexity considerations. In other words, we allow the algorithm unbounded time complexity. The inapproximation results suggest that it is the nature and the inherent limitations of this algorithmic paradigm, rather than resource constraints (such as the available space and running time) that cause greedy-like algorithms to be suboptimal.

As one may expect, the criterion of what precisely constitutes a greedy or greedy-like algorithm can be very subjective. We recognize that there exist algorithms which can intuitively be characterized as greedy algorithms, but which do not fit

[1] The algorithm can thus use information about input items already considered to determine a new ordering. A further distinction can be made based on whether the priority algorithm is "memoryless" or not (see [4,17]). For priority algorithms that either accept or reject input items (as for example in the case of the facility input problem studied in Section 4.1), an *acceptances-first* priority algorithm satisfies the property that once an input item is rejected, all remaining input items will be rejected. For such problems, it is observed in [4] that the memoryless and acceptances-first concepts are equivalent for adaptive priority algorithms. The randomized inapproximation results in this paper do not assume any restriction on the algorithm's knowledge of the past.

into the priority algorithm framework. For example, in the context of packing problems, a revocable priority algorithm is allowed to reject previously accepted items so as to maintain feasibility. (See [6,22,33] for revocable priority algorithms for weighted interval scheduling problems.) There are also algorithms which output the best of the partial solutions constructed during the execution of a greedy (i.e., priority) algorithm. (See the "greedy" algorithm for densest subgraph [38,14].) These algorithms can be modeled by extensions of the priority model but are beyond the scope of this paper. Nevertheless, we believe that most greedy algorithms do fall within the priority model. Furthermore, these extended models as described above are not defined for problems such as the makespan problem.

This line of work is motivated by the fact that greedy algorithms are a standard tool in both theory and practice, and thus it is useful to know in which cases the greedy approach will not provide solutions of sufficiently high quality (in terms of approximability). There are two additional reasons we believe it is interesting to focus on the approximation power of the broad class of priority algorithms (as opposed to only considering a few "natural" greedy algorithms for every specific problem). First, as argued in [13], it is expected that the intuition behind the lower bound constructions will provide insight into how to design better (variants of) randomized priority algorithms. Similar insight has proven quite useful in the design of online algorithms. For a simple yet illustrative concrete example, see [51] where the proof that no randomized online algorithm for 2-machine scheduling is better than $\frac{4}{3}$-competitive suggests how to design a randomized algorithm which is $\frac{4}{3}$-competitive (a result due to Bartal et al. [7]).

Second, there exist problems where good approximations are achieved by greedy-like algorithms which, if not "unnatural", certainly are not straightforward. For instance, the natural greedy algorithm of Mahdian et al. [40] for metric facility location does not achieve as good an approximation ratio as the priority algorithm of Jain, Mahdian and Saberi [34], which can be implemented as a priority algorithm but is described as a primal–dual algorithm and does not yet have a "simple combinatorial statement". In fact, several (but not all) primal–dual algorithms can be interpreted as priority algorithms. This means that inapproximation results for priority algorithms have implications that go beyond the strict boundaries of what we identify as "greedy algorithms".

2. Preliminaries

2.1. Deterministic vs. randomized priority algorithms and the minimax principle

We can define a randomized priority algorithm as a probability distribution over the set of all deterministic priority algorithms *of the same class*. To do so, we first must provide a more precise description of deterministic priority algorithms. (A formal definition can be found in Davis and Impagliazzo [17].) Consider optimization problems Π that are defined as follows. We associate with Π a set $\mathcal{S}$ of all possible *input or data items*. For example, in the problem of makespan scheduling, $\mathcal{S}$ is the (infinite) set of all possible jobs, with each job s_i characterized by its size (or load) p_i and its unique id i. For every input item $s \in \mathcal{S}$, denote by $D(s)$ the set of *allowable decisions* associated with s; e.g., in makespan scheduling with m machines, $D(s)$ can be described by the set $\{1, \ldots, m\}$, where decision $k \leq m$ is interpreted as "schedule job s on machine k". A problem Π is then defined by a family of objective functions $g = \{g^n\}$ with $g^n(s_1, \ldots, s_n, d_1, \ldots, d_n) \in \mathbb{R}$, where each d_i is such that $d_i \in D(s_i)$. On input $I = \{s_1, \ldots, s_n\} \subset \mathcal{S}$, the goal (for problem Π) is to assign a decision d_i to each input s_i so as to maximize (or minimize) g^n. Returning to the makespan problem on m identical machines, the goal is to define a machine $k = \sigma(i)$ for each job s_i so as to minimize the maximum total load on any machine; i.e., minimize the quantity $\max_{k:1\leq k\leq m} \sum_{i:k=\sigma(i)} p_i$ where p_i is the size of job i. Note that it is possible to have instances with multiple "identical" input items (e.g., several jobs of the same weight), hence the definition requires that each input item has a unique id. A priority algorithm can use the id's to distinguish between seemingly identical input items. The id's also allow us to formally include online algorithms as (fixed order) priority algorithms in that the algorithm will use the ordering specified by increasing id's.

As stated in the introduction, adaptive order (deterministic) algorithms will consider each input item $s \in I$ in some order and upon considering item s will make an irrevocable decision $d \in D(s)$. To make this more precise let us first consider fixed order priority algorithms and then later indicate how to modify the definition for adaptive orderings. A fixed order deterministic priority algorithm for Π is specified by two functions: the *ordering function* π and the *decision function* δ. The ordering function produces an allowable ordering on the given set I of input items. One clearly has to restrict the possible orderings; otherwise, using as much time as needed, the algorithm can consider the input set I and choose an ordering that gives rise to an optimal solution. The priority model avoids such "cheating" by assuming that the algorithm must choose an ordering on the set $\mathcal{S}$ of *all possible input items*. The input items that are in the given input set I then inherit their ordering from the ordering on $\mathcal{S}$. Using the language of social choice theory, the allowable orderings satisfy Arrow's IIA axiom [5] (Independence of Irrelevant Attributes) in that the removal of an item C from the input does not change the relative ordering between any two input items A and B. For simplicity, the reader can think of the ordering being defined by a real-valued function f on $\mathcal{S}$ and then ordering items by non-decreasing (or non-increasing) f-values. (If there are many input items having the same f-value, we can think of the algorithm breaking such ties in any way, such as by using the item *id*.) This IIA requirement is precisely what enables us to use an adversarial argument to create "difficult" input sets of finite size by selectively removing items from a large (possibly infinite) set of possible items. We claim that almost all greedy algorithms[2] utilize such IIA orderings.

[2] One example of a non-IIA ordering is the randomized greedy algorithm for the stochastic knapsack problem in [18].

The decision function δ_i is a function which, given the ith input item $s \in \mathcal{S}$ being considered and the *history* of the algorithm's execution thus far, maps s to a decision in $D(s)$. Here, the term history refers to all input items the algorithm has considered in the past and the algorithm's corresponding decisions for those items. That is, the history H_i at the beginning of the ith iteration contains all ordered pairs of the form (s_j, d_j) for $j < i$, where s_j is the item considered at iteration j, and d_j the decision associated with that item. Returning to the makespan problem, the history would contain the order and size of the first $i - 1$ input items and on what machine each of these items was scheduled. We will also distinguish between *greedy* and non-greedy priority algorithms by whether or not the decision function is "greedy" in the sense that every decision $\delta_i()$ produces a choice which optimizes the objective function given the previous decisions. For example, in the makespan problem, a greedy decision will only schedule a job on a machine which will minimize the overall makespan.[3] For facility location (say in the facility input model that we will define), a greedy decision must open the facility being considered if it decreases the total cost (i.e., opening costs thus far plus connections costs of all cities to the nearest open facility) and it will not open a facility if that decision would increase the total cost. A greedy algorithm is free to either open or not open a facility when the total cost is not changed by opening the facility.

Algorithm 1 below presents the format of an *adaptive (order) priority algorithm*.

```
1 $i := 1, H_1 = \emptyset$
2 while $I \neq \emptyset$ do
3     Choose an ordering function $\pi$ on $\mathcal{S}$
      /* This induces an ordering $s_{\pi(i)}, s_{\pi(i+1)}, \ldots$ on the remaining input items in $I$ */
4     Set $d_i = \delta_i(H_i, s_{\pi(i)})$
5     Remove $s_{\pi(i)}$ from $I$ and remove $s_{\pi(i)}$ and all items that precede $s_{\pi(i)}$ in $\pi$ from $\mathcal{S}$
      /* the algorithm now knows that certain input items do not occur in $I$ */
6     $H_{i+1} := H_i \cup \{(s_{\pi(i)}, d_i)\}$
7     $i := i + 1$
8 end
```

Algorithm 1: Statement of an adaptive priority algorithm.

A *fixed (order) priority algorithm* is obtained by simply exchanging steps (2) and (3) in the format (Algorithm 1) of an adaptive algorithm. That is, the algorithm initially chooses an ordering π on $\mathcal{S}$ which induces a fixed ordering $s_{\pi(1)}, s_{\pi(2)}, \ldots$ on all the input items in I.

A priority algorithm A for a cost-minimization problem is a c-*approximation* algorithm if

$$A(\sigma) \leq c \cdot OPT(\sigma), \tag{1}$$

for every input σ, where $A(\sigma)$ and $OPT(\sigma)$ denote the cost of algorithm A and the optimal cost on input σ, respectively. The approximation ratio of A is defined as the infimum of the set of values c for which A is a c-approximation. (Here we are using the definition for a strict (i.e. non-asymptotic) approximation ration. In Section 7, we consider the issue of asymptotic approximation ratios.)

A randomized priority algorithm is a priority algorithm in which either the ordering function(s) or the decision functions, or both, are randomized; we refer to such algorithms as randomized on the orderings, randomized on the decisions, or fully randomized, respectively. For a randomized priority algorithm, the definition of a c-approximation is as in (1), with $A(\sigma)$ replaced by its expected value over the random choices of A, denoted by $\mathbf{E}[A(\sigma)]$. We shall see when considering the makespan problem in Section 5 that we achieve improved inapproximation bounds for the more restrictive *randomized on the decisions* in contrast to *fully randomized* priority algorithms.

One can derive lower bounds for deterministic priority algorithms by evaluating the performance of every such algorithm A on an appropriately constructed nemesis input for A. But how can we show bounds on the approximability of a problem by a randomized priority algorithm? We will assume that an adversary against a randomized priority algorithm has knowledge of the algorithm, but does not have any access to the random choices the algorithm makes in the course of the algorithm. Borrowing the terminology of competitive analysis, we refer to such an adversary as an *oblivious adversary*. Under this assumption, one can resort to the *minimax principle*, also known as the *Von Neumann/Yao principle* [54] in order to prove lower bounds on the approximation ratio of randomized priority algorithms against such an adversary. This is similar to the application of the same principle in the analysis of online algorithms (as introduced by Borodin, Linial and Saks [12]). More specifically, suppose that there exists a constant $c \geq 1$ such that we can present a distribution $\mathcal{D}$ over inputs σ such that

$$\mathbf{E}_{\mathcal{D}}[A(\sigma)] \geq c \cdot \mathbf{E}_{\mathcal{D}}[OPT(\sigma)], \tag{2}$$

for every deterministic priority algorithm A in the class $\mathcal{C}$. Then the cost-minimization problem cannot be approximated by any randomized priority algorithm in the class $\mathcal{C}$ within a ratio better than c.

[3] In makespan scheduling, we would usually think of a greedy decision to be scheduling a job on a machine which currently has the least load but by our definition this is not necessary if a job can be scheduled without increasing the current makespan.

Naturally, the crux in the application of the Von Neumann/Yao principle lies in the selection of the appropriate input distribution. As in competitive analysis of online algorithms, the lower bound constructions for deterministic algorithms are suggestive of the appropriate distribution. A special case arises when one considers fixed priority algorithms with randomization on the decisions *only*, in the sense that one can have better insight on how to identify a good distribution. In this case, the distribution can be determined by a game between a deterministic fixed priority algorithm and an adversary, as follows: the adversary presents a set of potential input items S, and having knowledge only of the ordering that is decided by the algorithm, removes input items from S according to a specific probability distribution, so as to generate a distribution over the actual inputs. An example of such a game applied in the problem of makespan scheduling will be given in Section 5.2. Note that an adaptive priority algorithm with randomized decisions will introduce randomization on the selection of the orderings as well since the next input to be considered depends on the (randomized) decisions made for previously considered input items. Hence, a similar conceptual game is not directly applicable for adaptive priority algorithms.

2.2. Problem definitions and input representations

The input to the *makespan scheduling* problem on m identical machines consists of a set of jobs, each having a certain *size*. The *load* of a machine is the sum of the sizes of jobs scheduled on it. The objective is to minimize the maximum load among the m machines.

In the *(uncapacitated, unweighted) facility location* problem, there are various natural problem formulations and input representations. Facility location is defined in terms of a set $\mathcal{F}$ of facilities and a set $\mathcal{C}$ of cities. In the *complete (graph) model*, $\mathcal{F} = \mathcal{C}$; that is, every city is also a potential facility. In the *disjoint (or bipartite) model*, $\mathcal{F} \cap \mathcal{C} = \emptyset$. Although from a general algorithmic viewpoint these two formulations are equivalent via appropriate reductions, they are not equivalent in the context of priority algorithms. In this paper, we shall restrict attention to the disjoint model. Each facility $i \in \mathcal{F}$ is associated with an *opening cost* f_i which reflects the cost that must be paid to utilize the facility. Furthermore, for every facility $i \in \mathcal{F}$ and city $j \in \mathcal{C}$, the non-negative *distance* or *connection cost* c_{ij} is the cost that must be paid to connect city j to facility i. The objective is to open a subset of the facilities in $\mathcal{F}$ and connect each city in $\mathcal{C}$ to an open facility so that the total cost incurred, namely the sum of the opening costs and the connection costs, is minimized. We focus on the *metric* version of the problem, in which the connection costs satisfy the triangle inequality.

When considering priority algorithms for the above problems, we need to establish a model for representing the input. For the makespan scheduling problem on identical machines, there is no issue as to what constitutes the input: the input is a set of jobs, with each job being characterized by its id and its size. In contrast, there exist two natural ways to represent inputs for the facility location problem within the disjoint model formulation. In particular, we may assume that the input consists of facilities, where each facility is identified by a unique id, its distance to every city and its opening cost. When considering a facility of highest priority, the algorithm must make an irrevocable decision as to whether or not to open this facility. Note that in this model, the decisions d_i made for each input item (i.e., facility) are either to 'open' (i.e., accept) or to 'not open' (i.e., reject). In this model, we do not specify which cities are connected to each open facility as this is completely determined by the closest distance to open facilities. This model of input representation was assumed in [4], since it abstracts several known priority algorithms (see the adaptive priority algorithms in [40,34] as well as a fixed priority algorithm in [42,35]). We refer to the above representation of input items as the *facility input model*.

A different way of representing the input is to view the cities as the input items. Namely, the input consists of the set of all cities, with each city being identified by its id and its distance to every facility. The opening costs of the facilities are treated as global information. In this context, an irrevocable decision is interpreted as assigning each city to some open facility, possibly opening a new facility, but without affecting the assignment of cities considered in the past. We refer to this model of representing the inputs as the *city input model*.

3. Related work

Since their introduction in [13], priority algorithms have been studied in a variety of problems and settings. Angelopoulos and Borodin [4] showed lower bounds on the approximability of facility location and set cover by priority algorithms. Davis and Impagliazzo [17], Borodin, Boyar and Larsen [9] and Angelopoulos [3] addressed the application of the framework to graph optimization problems. Papakonstantinou [45] provided results on hierarchies for classes of priority algorithms for job scheduling, and Regev [47] gave an $\Omega(\log m/\log\log m)$ fixed priority lower bound for makespan minimization in the subset model (nearly matching the $\log m$ online greedy upper bound). Priority algorithms with revocable decisions were studied by Horn [33] in the context of the job interval selection problem [22] and by Ye and Borodin [55] in the context of the subset-sum problem. All these results pertain to deterministic priority algorithms.

Beyond what might reasonably be called greedy algorithms, there has been a number of algorithmic models which build upon priority algorithms. More specifically, Lesh and Mitzenmacher [39] introduce Bubble Search as a local search method that modifies an initial priority-based solution by randomly perturbing the priority ordering. In order to capture simple forms of dynamic programming and backtracking, Alekhnovich et al. [2] introduce Priority Backtracking, namely a model for algorithms that can branch on decisions so as to form a tree in which every path corresponds to an execution of a priority algorithm. As another example, Borodin, Cashman and Magen [10] introduce *stack algorithms* as a model to capture certain primal–dual/local-ratio algorithms.

With regards to the specific optimization problems considered in this paper, the first constant-factor polynomial time approximation algorithm for (metric) facility location was given by Shmoys, Tardos and Aardal [53]. The past several years have witnessed a steady series of improvements on the approximability of this problem with approaches that employ techniques such as LP-rounding, the primal–dual method, local search, dual fitting, or combinations of the above. (We refer the reader to the survey of Shmoys [52].) Interestingly, the current best approximation ratio (1.52), due to Mahdian, Ye and Zhang [41], can be implemented[4] as an adaptive priority greedy algorithm in the facility input model. With respect to negative results, Guha and Khuller [30] showed that the problem is not approximable by a deterministic (respectively, randomized) polynomial time algorithm within a factor better than 1.463, unless NP $\subseteq$ DTIME($n^{O(\log\log n)}$) (respectively, NP $\subseteq$ RTIME($n^{O(\log\log n)}$)).

We identify other facility location algorithms that can be described as priority algorithms in the facility input model. Mahdian, Markakis, Saberi and Vazirani [40] showed that a natural adaptive priority algorithm gives a 1.861-approximation; the analysis is performed by an elegant application of the dual fitting technique. A more complicated priority algorithm, due to Jain, Mahdian and Saberi [34] yields a 1.61 approximation factor (and is also analyzed by using the dual fitting technique). Mettu and Plaxton [42] gave an algorithm which can be interpreted as a fixed order priority algorithm (where facilities are the input items)[5] that achieves a 3-approximation for the problem.

On the use of randomization, Meyerson [43] provides an algorithm that can be interpreted as a fully randomized priority algorithm with respect to the complete model formulation (where $\mathcal{F} = \mathcal{C}$). The Meyerson algorithm[6] does not lend itself to the disjoint input model in either the facility or city input model. In subsequent work Fotakis [25] presented a deterministic online algorithm in the complete model formulation with competitive ratio $O(\frac{\log n}{\log\log n})$ where n denotes the number of nodes. This algorithm can be interpreted as an online (and hence priority) algorithm in either the facility or city input model. As such, the Fotakis algorithm provides (to our knowledge) the current best polynomial time approximation in the city input model. On the negative side, Fotakis showed that no randomized online algorithm (i.e., using randomization only in determining whether or not to open a new facility) can achieve a competitive ratio better than $\Omega(\frac{\log n}{\log\log n})$, against the oblivious adversary.

The problem of makespan scheduling on identical machines has long been known to be strongly NP-hard [26]. Hochbaum and Shmoys [31] showed that it is possible to derive a PTAS, while Sahni [48] showed that when the number of machines m is fixed, there exists a FPTAS for the problem. For the online version of the problem, Graham [29] showed that the natural "List Scheduling" greedy algorithm (each job is scheduled on the currently least loaded machine) is $(2 - \frac{1}{m})$-competitive. Currently, Fleischer and Wahl [24] have the best competitive ratio 1.9201 (for arbitrary m) achieved by a deterministic online algorithm while the best known lower bound is 1.853, due to Gormley et al. [27]. The algorithm of Fleischer and Wahl is a non-greedy online algorithm; that is, jobs are not necessarily scheduled on the least loaded machine in each iteration and may indeed increase the makespan beyond what would be achieved by greedily scheduling on the least loaded machine.

When randomization is introduced, Albers [1] proposed and analyzed an algorithm that achieves a competitive ratio of 1.916 for general m. Chen, van Vliet and Woeginger [15] and Sgall [50] showed that no randomized online algorithm can be better than $\frac{1}{1-(1-\frac{1}{m})^m}$-competitive which limits to $\frac{e}{e-1} \approx 1.582$ as m increases.

Concerning priority makespan algorithms, Graham [29] showed that a simple algorithm called *Longest Processing Time First* (or LPT, for brevity), which sorts jobs by non-increasing size and assigns the current job to the least loaded machine, is a $\frac{4m-1}{3m}$-approximation. Note that LPT is clearly a fixed order greedy priority algorithm. Seiden, Sgall and Woeginger [49] showed that LPT is optimal for $m = 2$ (i.e., non-greedy decisions cannot help) when jobs are considered by non-increasing size (which they call *semi-online* scheduling); they also presented a $\frac{8}{7}$-approximation semi-online algorithm with random decisions for $m = 2$, and showed it is optimal for the class of semi-online randomized algorithms when $m = 2$. Borodin et al. [13] showed that LPT is optimal with respect to all priority algorithms when $m = 2$ and that, for arbitrary m, no (deterministic) priority algorithm is better than a $\frac{7}{6}$-approximation. They also proved that LPT is optimal for fixed order priority algorithms when $m = 4$.

Tables 1 and 2 summarize the previous work for the two problems, as well as results (in boldface) which appear in this paper. The measures on which we compare the algorithms are as follows: For priority (respectively online) algorithms the measure is the approximation (respectively, competitive) ratio, and lower bounds hold for this class of algorithms, without any complexity-type restrictions (e.g., regardless of whether P $\neq$ NP). For offline algorithms, the measure is again the approximation ratio, but the inapproximation lower bounds hold under certain complexity assumptions (e.g., NP $\subseteq$

[4] Even though the algorithm of [41] works in two phases, and at first sight does not appear to follow the priority framework, it can indeed be seen as a priority algorithm since it is essentially an algorithm that makes irrevocable decisions about the facility it considers. Following the discussion of Section 2.1, the algorithm is greedy in the sense that it will open a facility unless this action increases the overall cost.

[5] The Mettu and Plaxton algorithm is stated in terms of the complete model for facility location in which every vertex in the graph can be used both as a facility and a city. It can, however, also be interpreted as a fixed order priority algorithm with respect to the facility input model. Although their algorithm is called an "online" algorithm, it is not online in the same sense as in competitive analysis, where an adversary dictates the order of input items. Rather, their algorithm is online in the sense that the input cities (and hence potential facilities) are considered one at a time (according to a fixed priority ordering determined by the algorithm) without knowing when there will be no further inputs.

[6] Like the Mettu and Plaxton algorithm, Meyerson's algorithm is formulated in terms of the complete input model and is also called an "online" algorithm but again this is not in the sense of competitive analysis where an adversary determines the input order. Rather, Meyerson's algorithm uniformly randomizes the input order and as such becomes a randomized priority algorithm in our terminology. The algorithm also uses randomization in determining whether the next input city that is considered will be opened as a facility or connected to a city that is already opened.

Table 1
Summary of some approximation bounds concerning metric facility location. Priority results refer to the facility input model. In particular, following [17,13,4] *greedy* refers to the additional assumption that whenever a facility is being considered, it is opened (resp. not opened) if this action decreases (resp. increases) the total opening plus connection cost. The deterministic upper and lower bounds for adaptive priority algorithms apply to the acceptances-first model.

Algorithm	Deterministic bounds		Randomized bounds	
	Upper	Lower	Upper	Lower
Offline	1.52 [41]	1.463 [30]	– [41]	– [30]
Online	$O\left(\frac{\log n}{\log\log n}\right)$ [25]	$\Omega\left(\frac{\log n}{\log\log n}\right)$ [25]	– [25]	– [25]
Fixed priority	3 [42]	Greedy 3 [4] Non-greedy 1.366 [4]	– [42]	Greedy $\frac{3}{2}$
Adaptive priority	1.52 [41]	1.463 [4]	– [41]	$\frac{4}{3}$

Table 2
Summary of some approximation bounds for makespan scheduling. The $\frac{10}{9}$ lower bound for fixed priority algorithms holds for randomized decisions only and $m \geq 3$, whereas the $\frac{12}{11}$ lower bound for adaptive priority algorithms holds for any fully randomized algorithm and $m \geq 2$.

Algorithm	Deterministic bounds		Randomized bounds	
	Upper	Lower	Upper	Lower
Offline	FPTAS [48] PTAS [31]	(strongly) NP-hard [26] (when m is part of input)	–	– [26]
Online	1.9201 [24]	1.853 [27]	1.916 [1]	$\frac{1}{1-(1-\frac{1}{m})^m}$ [15,50]
Fixed priority	$\frac{4m-1}{3m}$ [28]	$\frac{5}{4}$ for $m = 4$ [13]	$\frac{8}{7}$, $m = 2$ [49]	$\frac{10}{9}$
Adaptive priority	$\frac{4m-1}{3m}$ [28]	$\frac{7}{6}$ for all m [13]	– [28]	$\frac{12}{11}$

DTIME[$n^{O(\log\log n)}$]). A dash signifies that no randomized upper bounds better than the corresponding deterministic upper bounds are known. Offline complexity-based randomized inapproximations follow from their deterministic counterparts by strengthening the complexity assumption to concern the randomized polynomial time class (e.g., NP $\subseteq$ RP or NP $\subseteq$ RTIME[$n^{O(\log\log n)}$]).

4. Randomized priority facility location

In this section we show lower bounds on the approximation ratio achieved by randomized priority algorithms for the *metric* facility location problem. All our lower bound constructions use connection costs in $\{1, 3\}$, thus suggesting the following definitions. Let C_f be the set of cities at distance 1 from facility f. We say that f *covers* C_f. Two facilities f and f' are called *complementary* if and only if $C_f \cup C_{f'} = C$ (i.e., f and f' cover all cities in the input), $C_f \cap C_{f'} = \emptyset$, and f and f' have the same opening cost. For convenience, we call f' the *complement* of f (and vice versa), and we use the notation $\overline{f}$ to denote the complement of a given facility f.

It is worth pointing out that for the special case of metrics where distances are either 1 or 3, Guha and Khuller [30] give an improved (non-priority) algorithm which is 1.463-approximation, and thus matches the previously mentioned NP-hardness bound. We again emphasize that inapproximation bounds for priority algorithms are incomparable with inapproximation bounds based on complexity assumptions. In particular, the following theorem (Theorem 1) is incomparable with the stronger inapproximation bound of Guha and Khuller [30].

4.1. Facility input model

Theorem 1. *No fully randomized (adaptive) priority algorithm for facility location has approximation ratio better than $\frac{4}{3} - \epsilon$, for arbitrarily small ϵ.*

Proof. Suppose we have $n = 2^k$ cities, each with an id in $\{0, \ldots, 2^k - 1\}$ in binary representation. In addition, we define a set F, which consists of k pairs of complementary facilities, as the set of potential facilities. The facilities are identified by the cities they cover in the following way: The i-th pair of complementary facilities, denoted by $f_i, \overline{f}_i$ with $i \leq k$ is such that facility f_i covers exactly those cities whose i-th bit is 0, while $\overline{f}_i$ covers only cities whose i-th bit is 1. We also define the distance between a facility and a city it does not cover to be equal to 3. Note that this definition gives rise to metric distances. By construction, any j facilities in F no two of which are complementary cover exactly $n\sum_{i=1}^{j} 2^{-i} = n(1-2^{-j})$ cities. Each facility has an opening cost of $\frac{n}{4}$.

We will now define how to generate the actual input, as a distribution over all possible inputs defined on the above set of cities and potential facilities. Only one of the k pairs of complementary facilities in F appears in the actual input, and this pair is chosen uniformly at random, i.e., with probability $\frac{1}{k}$. For each of the remaining $k - 1$ pairs, only one of the two facilities will be in the actual input, and it is likewise chosen uniformly at random; i.e., with probability $\frac{1}{2}$. No other facilities are in the actual input.

Note that for every actual input I drawn from the distribution defined above, there always exists a unique pair of complementary facilities in the input. An algorithm that opens this pair only has cost $2\frac{n}{4} + n = \frac{3n}{2}$, hence $\mathbf{E}[OPT] \leq \frac{3n}{2}$; moreover, the optimal cost is the same for every random I. On the other hand, it is easy to observe the following: (i) an algorithm that opens one facility only has cost at least $2n$, for $n \geq 2$; (ii) an algorithm that opens only two or three facilities, but does not open a pair of complementary facilities, has cost at least $2n$; and (iii) an algorithm that opens at least four facilities incurs cost at least $3n$. Since there is always a pair of complementary facilities in I (but only one such pair), it follows that any algorithm that does not open both facilities in such a pair has approximation ratio at least $\frac{4}{3}$. For the remainder of the proof we refer to the (unique) pair of complementary facilities in I as the *optimal pair*.

It suffices then to argue that any deterministic priority algorithm A for this problem that opens at most 3 facilities on a random input I will fail to open both facilities in the optimal pair, with high probability. This is formalized in the following lemma (Lemma 2). In particular, applying Lemma 2, it follows that with probability at least $1 - 8/k$, any priority algorithm A pays cost at least $2n$ on a random input I. Since the optimal solution on I has cost at most $3n/2$, the approximation ratio of A can be made arbitrarily close to 4/3, for large n.

Lemma 2. *Given a random input I, if A opens at most 3 facilities on I, then the probability that A opens both facilities in the optimal pair in I is at most $8/k$.*

Proof. We will show the result for an algorithm that is at least as powerful as an adaptive priority algorithm. Recall that F denotes the set of all potential facilities, and I an actual input according to the specified probability distribution. In each iteration, the algorithm will *probe* a facility in F. Formally, let P_i denote the set of facilities that the algorithm has probed by the end of iteration i ($P_0 = \emptyset$). In iteration $i+1$, the algorithm will probe a facility from the set $F \setminus P_i$: if the facility is present (that is, if the facility is in the actual input I), the algorithm will consider it in iteration $i+1$ and make an irrevocable decision concerning it. Otherwise, that is if the facility is absent, the algorithm will do nothing; however it acquires some information concerning the actual input. For instance, if facility f is not in I, the algorithm deduces that $\overline{f}$ must be in I.

We call every algorithm in the above class a *probe* algorithm. Every (adaptive) priority algorithm can be implemented as a probe algorithm (although it is not clear if the opposite is necessarily true). The potential added power of a probe algorithm over a priority algorithm is that it is allowed to change the ordering after learning that a potential input item is not in the actual input.

Let A then denote a deterministic probe algorithm. We can further assume that A works in *phases* where each phase consists of two consecutive iterations, in which the algorithm probes two complementary facilities in F. To see this, suppose that in iteration i, A probes facility f. We consider the following cases:

- f is not present in I. Then the algorithm deduces that the pair $(f, \overline{f})$ is not the optimal pair in I. In this case, A will probe $\overline{f}$ in iteration $i+1$ (which must be in I) and reject it.
- f is present in I, and A opens it in iteration i. Then A can probe $\overline{f}$ in iteration $i+1$: if it is the case that $\overline{f} \in I$, A has identified the optimal pair (and thus will open $\overline{f}$). Otherwise, A does not take any action in iteration $i+1$.
- f is present in I and A does not open it in iteration i. Then once again, A can probe $\overline{f}$ in iteration $i+1$: if it is in I, then regardless of the decision of A concerning $\overline{f}$, the algorithm has failed to open both facilities of the optimal pair. If the facility in question is not in I, then the algorithm deduces that f is not in the optimal pair, and will not take further action on this iteration.

We will now show how to bound the probability that A opens a facility that belongs in the optimal pair. Let $p_1, \ldots p_k$ denote the k pairs probed by A in phases $1 \ldots k$. Suppose that in phase i, A decides to open a facility of pair p_i. Following the earlier argument, this means that the first facility of p_i that is considered by A is opened. We define the event that *A succeeds in phase i* if this facility that is opened by A, say f_i, is one of the facilities in the optimal pair. We have:

$$\begin{aligned}&\Pr(A \text{ succeeds in phase } i)\\ &\quad= \Pr(f_i \in \text{optimal pair} \mid \text{pairs } p_1, \ldots p_{i-1} \text{ are not optimal and } f_i \in I)\\ &\qquad \cdot \Pr(\text{pairs } p_1, \ldots p_{i-1} \text{ are not optimal}). \qquad (3)\end{aligned}$$

Eq. (3) is derived from the observation that A is successful in phase i if the following hold: the first facility that A opens during that phase (that is, facility f_i) belongs in the optimal pair, and all previously probed pairs are not optimal. Also note that since A is a probe algorithm, if it decides to open A, it knows that $f_i \in I$, otherwise it would not open it (hence the conditional dependence on the event $f \in I$).

We will show that $\Pr(A \text{ succeeds in phase } i) \leq 2/k$. First, note that

$$\Pr(\text{pairs } p_1, \ldots p_{i-1} \text{ are not optimal}) = \prod_{j=1}^{i-1} \frac{k-j}{k-j+1} = \frac{k-i+1}{k}. \qquad (4)$$

Let E denote the event $(f_i \in I \mid p_1 \ldots p_{i-1} \text{ not optimal})$. We then obtain

$$\begin{aligned}&\Pr(f_i \in \text{optimal pair} \mid p_1, \ldots p_{i-1} \text{ are not optimal})\\ &\quad= \Pr(f_i \in \text{opt. pair} \mid p_1, \ldots p_{i-1} \text{ not optimal and } f_i \in I) \cdot \Pr(E)\\ &\qquad + \Pr(f_i \in \text{opt. pair} \mid p_1, \ldots p_{i-1} \text{ not optimal and } f_i \notin I) \cdot \Pr(\overline{E}). \qquad (5)\end{aligned}$$

We can simplify Eq. (5). First, we have

$$\Pr(f_i \in \text{optimal pair} \mid p_1, \ldots p_{i-1} \text{ are not optimal}) = \frac{1}{k-i+1}. \quad (6)$$

Next, it is easy to see that

$$\Pr(E) = \frac{1}{2} \cdot \frac{k-i}{k-i+1} + \frac{1}{k-i+1} > \frac{1}{2}. \quad (7)$$

Last, note that f_i cannot belong in the optimal pair if $f_i \notin I$. Using this observation, as well as (6) and (7), (5) becomes

$$\frac{1}{k-i+1} > \Pr(f_i \in \text{optimal pair} \mid p_1, \ldots p_{i-1} \text{ are not optimal and } f_i \in I) \cdot \frac{1}{2} + 0$$

from which it follows that

$$\Pr(f_i \in \text{optimal pair} \mid p_1, \ldots p_{i-1} \text{ are not optimal and } f_i \in I) < \frac{2}{k-i+1}. \quad (8)$$

Substituting (4) and (8) into (3) we obtain

$$\Pr(A \text{ succeeds in phase } i) < \frac{2}{k-i+1} \cdot \frac{k-i+1}{k} = \frac{2}{k}.$$

To conclude the proof of the lemma, recall that we can assume that A can open facilities which belong to at most three pairs. Since A is deterministic, on any input I, it is characterized by three phases $i_1, i_2, i_3 \le k$ in which it will open the first facility of that phase. As shown above, the probability the optimal pair will be one of the i_j is at most $2/k$ and hence the probability that it opened the optimal pair is at most $8/k$. □

As noted, the proof of the lemma concludes the proof of the Theorem. □

The following theorem shows that it is possible to obtain a stronger lower bound when considering the class of fixed priority greedy algorithms.

Theorem 3. *There is a lower bound of $\frac{3}{2} - \epsilon$ on the approximation ratio of every randomized (on the orderings) fixed priority greedy facility location algorithm.*

Proof. Suppose we have a set C of n cities with id's $1 \ldots n$. Define the set F of potential facilities as follows: For a fixed integer $i \in [\frac{n}{2}]$ (i.e., $i \in \{1, \ldots, \frac{n}{2}\}$) facility f_i covers cities $1 \ldots \frac{n}{2}$ excluding city i, and also covers city $\frac{n}{2}+i$. The complement of facility $\bar{f}_i$ is defined as the facility that covers cites $\frac{n}{2}+1 \ldots n$ excluding city $\frac{n}{2}+i$, and also covers city i. The set F consists of all f_i's and $\bar{f}_i$'s, for $i \in [\frac{n}{2}]$, and each facility has an opening cost equal to $2-\epsilon$, for arbitrarily small ϵ. Consider a deterministic fixed priority greedy algorithm A, and denote by I_i (respectively $\bar{I}_i$) the potential input that consists of all facilities of the form $f \in F$ (resp. $\bar{f} \in F$) as well as facility $\bar{f}_i$ (resp. f_i), for $i \in [\frac{n}{2}]$. The actual input to A, which we denote by z, is selected uniformly at random from the set $I \cup \bar{I}$, where $I = \{I_i, i \in [\frac{n}{2}]\}$ and $\bar{I} = \{\bar{I}_i, i \in [\frac{n}{2}]\}$, i.e., a specific actual input is chosen with probability equal to $\frac{1}{n}$. Since every possible input set contains (exactly) one pair of complementary facilities, the optimal algorithm (OPT) can cover all n cities by opening the two complementary facilities, hence $\mathbf{E}[OPT] = n + 2(2-\epsilon)$. Denote by A_f the cost of facilities opened by A. Note that

$$\begin{aligned}\mathbf{E}[A_f] &= \Pr(z \in I) \cdot \mathbf{E}[A_f|z \in I] + \Pr(z \in \bar{I}) \cdot \mathbf{E}[A_f|z \in \bar{I}] \\ &= \frac{1}{2}(\mathbf{E}[A_f|z \in I] + \mathbf{E}[A_f|z \in \bar{I}]). \quad (9)\end{aligned}$$

In order to show the desired result, we give expressions for $\mathbf{E}[A_f|z \in I]$ and $\mathbf{E}[A_f|z \in \bar{I}]$; the intuition is that not both expressions can be very small, hence $\mathbf{E}[A_f]$ is large.

Lemma 4. $\mathbf{E}[A_f|z \in I] + \mathbf{E}[A_f|z \in \bar{I}] \ge n(1 - \frac{\epsilon}{2})$.

Proof. Let σ be the ordering of facilities in F as produced by the fixed priority greedy algorithm A. Let $f_{i_1} \ldots f_{i_{\frac{n}{2}}}$ and $\bar{f}_{j_1} \ldots \bar{f}_{j_{\frac{n}{2}}}$ be the order in which facilities of the two types appear in σ, noting that $i_1 \ldots i_{\frac{n}{2}}$ and $j_1 \ldots j_{\frac{n}{2}}$ are permutations of $\{1, \ldots, \frac{n}{2}\}$. For a given $k \in [\frac{n}{2}]$, define by x_k the number of f-facilities that precede $\bar{f}_{j_k}$ and also follow $\bar{f}_{j_{k-1}}$ in σ (for the special case $k=1$, we define x_1 as the number of f-facilities that precede $\bar{f}_{j_1}$). Note that the total number of f-facilities that precede $\bar{f}_{j_k}$ is $\sum_{l=1}^{k} x_l$; denote this sum by p_k.

First, consider the event that z (the actual input to A) is $I_{j_l} \in I$, with $l \in [\frac{n}{2}]$. The greedy criterion dictates that every time the algorithm considers an f-facility that precedes $\bar{f}_{j_l}$, it will open it: this is because the algorithm will pay a total of $3-\epsilon$

for opening the f-facility and connecting a currently "uncovered" city to the f-facility, thereby improving upon the cost of 3 that must be paid if the algorithm does not open the f-facility. Hence, A must open p_l facilities.

Therefore,

$$\mathbf{E}[A_f|z \in I] = \sum_{l=1}^{\frac{n}{2}} \mathbf{Pr}(z = I_{j_l}|z \in I) \cdot \text{cost of opened facilities on input } I_{j_l}$$

$$\geq \sum_{l=1}^{\frac{n}{2}} \frac{2}{n}(2-\epsilon)p_l = \frac{2(2-\epsilon)}{n} \sum_{l=1}^{\frac{n}{2}} \sum_{i=1}^{l} x_i,$$

and by expanding the double summation we obtain

$$\mathbf{E}[A_f|z \in I] \geq \frac{2(2-\epsilon)}{n} \sum_{l=1}^{\frac{n}{2}} \left(\frac{n}{2} - l + 1\right) x_l. \tag{10}$$

Next, consider the event that z is $\bar{I}_{i_l} \in \bar{I}$, with $l \in [\frac{n}{2}]$. Similar to the argument shown earlier concerning the greedy property, every time A considers an $\bar{f}$-facility which precedes f_{i_l} in σ, it must open the facility in question. Observe that if l is such that $p_t < l \leq p_{t+1}$, for some $t \in [\frac{n}{2} - 1]$ (and thus for $p_{t+1} - p_t = x_{t+1}$ possible values of l), there are t $\bar{f}$-facilities that precede f_{i_l} in σ. We can rephrase the above observation by saying that in a randomly chosen input from $\bar{I}$, the probability that A opens t facilities is at least $\frac{x_{t+1}}{\frac{n}{2}} = \frac{2x_{t+1}}{n}$, for $t \in [\frac{n}{2} - 1]$. We further note that if $p_{\frac{n}{2}} < \frac{n}{2}$, then there exist $\frac{n}{2} - p_{\frac{n}{2}}$ f-facilities each of them following all $\frac{n}{2}$ $\bar{f}$-facilities in σ; again, we can restate the previous observation by saying that with probability at least $\max\left\{\frac{\frac{n}{2} - p_{\frac{n}{2}}}{\frac{n}{2}}, 0\right\} = \max\left\{1 - \frac{2}{n}\sum_{i=1}^{\frac{n}{2}} x_i, 0\right\}$, A will open $\frac{n}{2}$ facilities. Summarizing, we can bound $\mathbf{E}[A_f|z \in \bar{I}]$ as follows:

$$\mathbf{E}[A_f|z \in \bar{I}] \geq \sum_{t=1}^{\frac{n}{2}-1} (2-\epsilon)t\frac{2x_{t+1}}{n} + (2-\epsilon)\frac{n}{2} \cdot \max\left\{1 - \frac{2}{n}\sum_{i=1}^{\frac{n}{2}} x_i, 0\right\},$$

thus getting

$$\mathbf{E}[A_f|z \in \bar{I}] \geq \frac{2(2-\epsilon)}{n} \sum_{t=1}^{\frac{n}{2}-1} t x_{t+1} + (2-\epsilon)\frac{n}{2} \cdot \max\left\{1 - \frac{2}{n}\sum_{i=1}^{\frac{n}{2}} x_i, 0\right\}. \quad \square \tag{11}$$

By adding (10) and (11) we obtain

$$\mathbf{E}[A_f|z \in I] + \mathbf{E}[A_f|z \in \bar{I}] \geq \frac{2(2-\epsilon)}{n} \sum_{i=1}^{\frac{n}{2}} \frac{n}{2} x_i + (2-\epsilon)\frac{n}{2} \cdot \max\left\{1 - \frac{2}{n}\sum_{i=1}^{\frac{n}{2}} x_i, 0\right\}. \tag{12}$$

Observe that if $\sum_{i=1}^{\frac{n}{2}} x_i \geq n/2$, the RHS of (12) is at least $(2-\epsilon)\frac{n}{2}$, whereas if $\sum_{i=1}^{\frac{n}{2}} x_i < n/2$, the RHS becomes

$$(2-\epsilon)\sum_{i=1}^{\frac{n}{2}} x_i + (2-\epsilon)\frac{n}{2} \cdot \left(1 - \frac{2}{n}\sum_{i=1}^{\frac{n}{2}} x_i\right) = (2-\epsilon)\frac{n}{2}$$

which concludes the proof. □

The theorem then follows, since from (9) and Lemma 4 we have $\mathbf{E}[A_f] \geq \frac{n}{2}(1 - \frac{\epsilon}{2})$, and thus the expected total cost paid by A is at least $n + \frac{n}{2}(1 - \frac{\epsilon}{2})$, whereas the optimal cost is at most $n + 2(2-\epsilon)$. □ Q2

4.2. City input model

As already noted in Section 3, we do not know of any results that provide an $O(1)$-approximation in the city input model. In particular, for the city input model, the best known upper bound is the online bound due to Fotakis [25]. In this section we establish a constant inapproximation bound (namely 3) for the city input model that contrasts with the 1.52 approximation ratio obtained by the Mahdian, Ye and Zhang [41] algorithm for the facility input model.

Theorem 5. *No fully randomized priority algorithm for facility location in the city input model is better than a $(3-\epsilon)$-approximation.*

Proof. Let the set C of potential input items (cities) consist of n^2 cities. In addition, let there be $\binom{n^2}{n}$ facilities, each covering (i.e., at distance 1) a set of n cities in C, and such that no two facilities cover the same set of cities. The distance from a given facility to every city it does not cover is 3, and the opening cost of every facility is equal to 2. Each of the n^2 cities in C will appear in the actual input with probability $\frac{1}{n}$, thus, the expected number of cities in the actual input is n. Using the Chernoff bound (see, e.g., [44]), we obtain that the probability that the number of cities exceeds $2n$ is at most $2e^{-\frac{n}{3}}$. If there are at most $2n$ cities in the actual input, then two facilities suffice to cover them, otherwise, up to n facilities may be required. Therefore we obtain

$$\mathbf{E}[OPT] \le (1-2e^{-\frac{n}{3}})(n+4) + 2e^{-\frac{n}{3}} \cdot (2n+3n^2). \tag{13}$$

Next, we will bound the expected cost of a deterministic algorithm A. First, note that every time a city c not covered by an opened facility is considered, it is to the algorithm's benefit to open a new facility: the total cost of opening a new facility (say f) that covers c and connecting c to f is 3, which is the connection cost that must be paid if no facility is opened. Therefore, we assume that when A considers an uncovered city it must open a new facility.

We now argue that with high probability, the algorithm must open more than $n - \ln n$ facilities. Suppose the algorithm opens at most $n - \ln n$ facilities implying that at most $n^2 - n \ln n$ cities are covered or equivalently that at least $n \ln n$ cities are uncovered. We let F_A denote the set of facilities opened by A (for a given actual input upon termination), and let $\overline{C}$ denote the set of cities in C which are not covered by F_A. Clearly, every city in $\overline{C}$ must not be in the actual input, otherwise A would consider it and open an additional facility, contradicting the definition of F_A. Since $|\overline{C}| \ge n \ln n$, the probability that all the cities in $\overline{C}$ will not be in the actual input is at most

$$\left(1-\frac{1}{n}\right)^{n \ln n} \le e^{-\ln n} = \frac{1}{n}.$$

That is, with probability at least $(1 - 1/n)$, an uncovered city will be in the actual input contradicting the definition of F_A. It follows that with probability at least $(1 - 1/n)$, the algorithm must open at least $n - \ln n$ facilities and therefore, the expected cost of facilities opened by the algorithm is at least $(1 - 1/n) \cdot 2(n - \ln n)$. On the other hand, using the Chernoff bound, with probability at least $1 - 2e^{-\frac{n^{1/3}}{3}}$, there are at least $n - n^{\frac{2}{3}}$ cities in the actual input, which incur a connection cost of at least $n - n^{\frac{2}{3}}$. Summarizing, the expected cost of the algorithm is

$$\mathbf{E}[A] \ge (1-1/n) \cdot 2(n - \ln n) + (1 - 2e^{-\frac{n^{1/3}}{3}})(n - n^{\frac{2}{3}}). \tag{14}$$

Using (13) and (14), we observe that the ratio $\mathbf{E}[A]/\mathbf{E}[OPT]$ becomes arbitrarily close to 3, as n approaches infinity. □

5. Randomized priority makespan scheduling

In this section we show lower bounds on the approximation ratio of randomized priority algorithms for makespan scheduling on identical machines. The results hold under the assumption that the algorithm does not know the number n of jobs in the input. The same assumption is used in deriving lower bounds for deterministic priority algorithms in [13].[7] While it is clearly easy for an offline algorithm to determine n, we note that known priority algorithms (e.g., the Longest Processing Time or LPT algorithm, as discussed in Section 3) and all online algorithms for the makespan problem operate as if n is unknown. However, currently we cannot tell if a priority makespan algorithm can take advantage by knowing n so as to reduce the approximation ratio for either deterministic or randomized algorithms.

Once we assume that the algorithm cannot know the number of actual inputs (until the input set is exhausted), we are also implicitly accepting some additional assumptions as to the nature of the input items and the execution of the algorithm. Namely, we assume that the input item identifier for each input is an arbitrary integer. If, say instead, the input items were numbered sequentially as $1, 2, \ldots, n$ where n is the number of inputs, then the algorithm can immediately determine n in the first iteration. It then follows (using Lemma 2 of [13]) that for a fixed order priority algorithm, the jobs are ordered according to the size of the job with all jobs having the same size being adjacent in the ordering. That is, the input identifier plays no role in determining the ordering. The same observation applies to each iteration of an adaptive priority algorithm. Hence, we are considering the following algorithmic model: In each iteration, the algorithm chooses an ordering of possible (remaining) job sizes, say the ordering $s_1, s_2, \ldots$. Suppose that there is a job of size s_j and no jobs of size $s_1, \ldots, s_{j-1}$. The algorithm must then make an irrevocable scheduling decision for a job of size s_j and remove that particular job from the input set. The algorithm terminates when no job remains.

5.1. Fully randomized priority makespan scheduling

Theorem 6. *No fully randomized priority makespan scheduling algorithm has approximation ratio better than $\frac{12}{11}$, for all $m \ge 2$.*

Proof. As in the proof of the deterministic $\frac{7}{6}$ bound in [13], the set S of potential jobs consists of $2\lfloor \frac{m}{2} \rfloor$ jobs of size 3, and $3\lceil \frac{m}{2} \rceil$ jobs of size 2. If the actual input is S itself, the optimal makespan is 6, and is achieved by scheduling two jobs of size 3 on each of $\lfloor \frac{m}{2} \rfloor$ machines and three jobs of size 2 on each of the remaining $\lceil \frac{m}{2} \rceil$ machines. We will make use of the following fact that is shown in [13]:

Fact 7. *Any algorithm on input S that schedules two jobs of different sizes on the same machine has makespan at least 7.*

[7] If the number of jobs is known to the algorithm we can still obtain lower bounds, albeit much weaker.

We will bound the expected makespan of any deterministic algorithm A for a probability distribution over the actual inputs that is defined as follows: with probability p, to be determined later, the actual input to A is S, while with probability $1-p$ the actual input is a set $S' \subset S$, which consists of all jobs of size 3 in S (namely, all $2\lfloor m/2 \rfloor$ of them) as well as $2\lfloor m/2 \rfloor$ jobs of size 2, if m is even, or $2\lfloor m/2 \rfloor + 2$ such jobs if m is odd, respectively (note that $2\lfloor m/2 \rfloor + 2 < 3\lceil m/2 \rceil$ when m is odd).

We will make use of the following important observation. Suppose that, by the end of iteration i, the algorithm has considered x_2 jobs of size 2 and x_3 jobs of size 3, such that both x_2 and x_3 are strictly smaller than the corresponding number of jobs in S'. Then at the beginning of iteration $i+1$ the algorithm cannot know which of the two potential inputs chosen at random by the adversary (S or S'), is the actual one. This is due to the fact that at this point, for both potential inputs, jobs of both sizes would have to be considered in future iterations. In what follows we will describe this situation by saying that A *cannot distinguish S from S'* at the beginning of iteration $i+1$. Note that by "beginning of iteration $i+1$" we mean the point in which the algorithm has considered the input item of iteration $i+1$ but has not yet made a decision concerning it.

We will also need the following lemma.

Lemma 8. *Let x_2 and x_3 be the number of jobs of size 2 and 3, respectively, that A has considered by the end of iteration i, and suppose that at the beginning of iteration $i+1$ A cannot distinguish S from S'. If $x_2 > \lceil m/2 \rceil$ or $x_3 > \lfloor m/2 \rfloor$ then at the end of iteration i, either A has scheduled two jobs on the same machine, or in the event S is the actual input, A's makespan is at least 7.*

Proof. Suppose that no two jobs were scheduled on the same machine by the end of iteration i. In the event where S is the actual input, then if A on input S schedules two jobs of different sizes on the same machine, from Fact 7 it follows that its makespan will be at least 7. Otherwise, i.e., if no two jobs in S of different sizes were scheduled on the same machine, let l_2, l_3 denote the number of machines on which jobs in S of size 2 and 3 were scheduled, respectively. If $x_2 > \lceil m/2 \rceil$ then $l_3 < \lfloor m/2 \rfloor$, and thus at least 3 jobs of size 3 are scheduled on the same machine. Similarly, if $x_3 > \lfloor m/2 \rfloor$ then $l_2 < \lceil m/2 \rceil$, and thus at least 4 jobs of size 2 are scheduled on the same machine. In either case the makespan is at least 7. □

Let $\mathbf{E}[A]$ denote the expected makespan of A. Our goal is to show that either $\mathbf{E}[A] \geq 7p + 5(1-p)$, or $\mathbf{E}[A] \geq 6$; then choosing $p = 1/2$ will yield the desired lower bound, as will become clear later. Denote by T the multiset of the m jobs that A considered (and irrevocably scheduled) by the end of iteration m. To simplify the proof, we will consider two cases: the case where m is even, and the case where m is odd.[8]

m is even: Note that S' consists of m jobs of size 2 and m jobs of size 3. The optimal algorithm on S' schedules one job of size 2 and one job of size 3 on each machine, hence $\mathbf{E}[OPT] = 6p + 5(1-p)$.

• *Case* 1: The number of both size 2 and size 3 jobs in T is in the range $[1, m-1]$. Then at the beginning of iteration $m+1$, A cannot distinguish S from S'.

Subcase 1a: A scheduled all m jobs in T on distinct machines. Denote by s the job considered at iteration $m+1$. Either s will be scheduled on a machine where a job of the same size as s has been scheduled, or not. In the former case, A's makespan will be at least 6 whether S or S' is the actual input, hence $\mathbf{E}[A] \geq 6p + 6(1-p) = 6$. In the latter case, Fact 7 guarantees that $\mathbf{E}[A] \geq 7p + 5(1-p)$.

Subcase 1b: A scheduled two jobs in T on the same machine. If the two jobs are of different sizes, then on input S, A's makespan would be at least 7. Otherwise, on input S', A's makespan would be at least 6. The corresponding bounds are then $\mathbf{E}[A] \geq 7p + 5(1-p)$, and $\mathbf{E}[A] \geq 6$, respectively.

• *Case* 2: T is not as in Case 1, i.e., T consists of m jobs, all of the same size $r \in \{2, 3\}$. At the beginning of iteration m/2+1, A cannot distinguish S from S'. If A scheduled all jobs of size r considered through the first $m/2+1$ iterations on separate machines, then in the event S is the actual input, A's makespan will be at least 7 (from Lemma 8), hence $\mathbf{E}[A] \geq 7p + 5(1-p)$. Otherwise two jobs of the same size r were scheduled on the same machine, which implies that on the event S' is the actual input, A's makespan will be at least 6, hence $\mathbf{E}[A] \geq 6$.

m is odd. Note that S' consists of $m-1$ jobs of size 3 and $m+1$ jobs of size 2. The optimal algorithm on S' has makespan 5, since it will schedule pairs of jobs of sizes 2 and 3 on each of the first $m-1$ machines, and two jobs of size 2 on the last machine. Hence $\mathbf{E}[OPT] \leq 6p + 5(1-p)$. This case is more technically involved due to the asymmetry of the optimal solution.

We will need the following straightforward lemma

Lemma 9. *Suppose that at some point in the execution of A, its schedule includes two machines each with two jobs of size 2. Then A's makespan is at least 6 (on input S or S').*

Proof. If a third job is scheduled on one of the machines assigned two jobs of size 2, then the lemma holds. Otherwise, the remaining $m-2$ machines must accommodate a load of at least $3(m-1) + 2(m+1) - 8 = 5m - 9 > 5(m-2)$, hence one of the $m-2$ machines will have load at least 6. □

We now consider cases on the multiset of jobs T, as defined earlier in the proof. Denote by s_1, s_2 the jobs considered at iterations $m+1$ and $m+2$, respectively.

• *Case* 1: The number of size jobs of size 2 in T is in the range $[2, \ldots m-1]$, and hence the number of size 3 jobs in T is in the range $[1, \ldots, m-2]$. Note that at the beginning of iteration $m+1$, A cannot distinguish S from S'; moreover, if s_1 is a

[8] The reader may want to skip the case where m is odd, on first reading, since this case is somewhat technical. The case where m is even reflects the essence of the proof.

2-job, then again at the beginning of iteration $m+2$, A cannot distinguish the two potential inputs S and S'.
Subcase 1a: A scheduled all jobs in T on separate machines. Consider now job s_1. If s_1 is of size 3, then either it is scheduled on a machine already assigned a job of size 3 (in which case $\mathbf{E}[A] \geq 6$), or otherwise from Fact 7, $\mathbf{E}[A] \geq 7p+5(1-p)$. Suppose then s_1 is of size 2. If s_1 is scheduled on a machine where a 3-job has been scheduled, from Fact 7, $\mathbf{E}[A] \geq 7p+5(1-p)$. Otherwise, consider job s_2.

- Subcase 1aa: Suppose s_2 is a 2-job. If s_2 is scheduled on a machine where a 3-job was scheduled, then $\mathbf{E}[A] \geq 7p+5(1-p)$. Otherwise, since there are at least 4 jobs of size 2 among the first $m+2$ jobs considered, it follows that either one machine has load at least 6, or there exist two machines of load 4. In the latter case, Lemma 9 yields $\mathbf{E}[A] \geq 6$.
- Subcase 1ab: Suppose s_2 is a 3-job. If it is scheduled on a machine where a 3-job was scheduled, then $\mathbf{E}[A] \geq 6$. Otherwise, from Fact 7, $\mathbf{E}[A] \geq 7p+5(1-p)$.

Subcase 1b: A scheduled at least two jobs of T on the same machine. We can assume, without loss of generality, that a single machine was assigned exactly two jobs of size 2, one machine is empty, and every other machine was assigned only one job in T (since in any other case, Fact 7 and Lemma 9 imply that $\mathbf{E}[A] \geq 7p+5(1-p)$, or $\mathbf{E}[A] \geq 6$, using arguments as earlier). Let m_1 denote the machine assigned two 2-jobs, and m_2 denote the empty machine. Consider job s_1 of size $r \in \{2,3\}$. If s_1 is scheduled on some non-empty machine, then $\mathbf{E}[A] \geq 7p+5(1-p)$, or $\mathbf{E}[A] \geq 6$. Suppose then that s_1 is assigned to m_2. If s_1 is of size 2, then at the beginning of iteration $m+2$, A cannot distinguish S from S'. At the beginning of this iteration, all machines are occupied, hence no matter which machine s_2 is assigned to, $\mathbf{E}[A] \geq 7p+5(1-p)$, or $\mathbf{E}[A] \geq 6$ (due to Fact 7 and Lemma 9, respectively). Suppose then that s_1 is a 3-job. We need to consider the following subcases:

- *Subcase* 1ba: T consists of at most $m-3$ jobs of size 3. Then at the beginning of iteration $m+2$, A cannot distinguish S from S'. Using similar arguments as in subcase (1b) we conclude that $\mathbf{E}[A] \geq 7p+5(1-p)$, or $\mathbf{E}[A] \geq 6$. More specifically, if s_2 is a 2-job, then any assignment for s_2 will give rise to a schedule with either at least two machines having two jobs of size 2, or a machine with jobs of both sizes, or a machine with 3 jobs of size 2. On the other hand, if s_2 is a 3-job, then any assignment of s_3 will give rise to either a schedule with a machine with jobs of both sizes, or a schedule with a machine with two jobs of size 3.
- *Subcase* 1bb: T consists of $m-2$ jobs of size 3. Suppose that A scheduled all the $m-2$ jobs of size 3 on separate machines (otherwise $\mathbf{E}[A] \geq 6$) and $m > 3$, then from Lemma 8, $\mathbf{E}[A] \geq 7p+5(1-p)$. Otherwise, i.e., if $m=3$, the layout of machines before s_1 is scheduled is as follows: m_1 is assigned two jobs of size 2, m_2 a job of size 3, and m_3 is empty. If s_1 is scheduled on machine m_3, then Fact 7 gives then that $\mathbf{E}[A] \geq 7p+5(1-p)$. Otherwise, it is clear that $\mathbf{E}[A] \geq 6$.

Case 2: Let x_2, x_3 denote the number of size 2 and 3 jobs in T respectively. This case corresponds to the situation in which $(x_2, x_3) \in \{(0,m), (1,m-1), (m,0)\}$. In all these cases, "long" sequences of jobs of the same size are considered; using arguments similar to the ones above, we will derive again that either $\mathbf{E}[A] \geq 7p+5(1-p)$, or $\mathbf{E}[A] \geq 6$. The technical details are as follows.

Suppose first that $x_3 \leq m-1$. If by the end of iteration m all x_3 jobs were scheduled on different machines, then since $m-1 > \lfloor \frac{m}{2} \rfloor$ for all $m \geq 3$, from Fact 7 we have that $\mathbf{E}[A] \geq 7p+5(1-p)$. Otherwise, trivially $\mathbf{E}[A] \geq 6$.

Remains then to consider the case $x_2 = m$. If $m > 3$, and the algorithm schedules the m 2-jobs on different machines, then from Fact 7 we have that $\mathbf{E}[A] \geq 7p+5(1-p)$. Otherwise, either at least 2 machines were assigned two jobs of size 2 (hence $\mathbf{E}[A] \geq 6$, from Lemma 9) or a machine was assigned at least 3 jobs of size 2 (hence again $\mathbf{E}[A] \geq 6$). The only case that remains is the case $m=3$. Recall that s_1 and s_2 denote the jobs considered at iterations $m+1=4$ and $m+2=5$, respectively.

Suppose, without loss of generality, that at the end of iteration 3, A scheduled two jobs of size 2 on machine 1 and one job of size 2 on machine 2. Note that at the beginning of iteration 4 A cannot distinguish between S and S'. If s_1 is a 2-job and A schedules it on either machine 2 or machine 3, then $\mathbf{E}[A] \geq 6$ and $\mathbf{E}[A] \geq 7p+5(1-p)$, respectively (and trivially $\mathbf{E}[A] \geq 6$ if it is scheduled on machine 1).

Suppose then that s_1 is a 3-job. A should schedule s_1 on machine 3, otherwise $\mathbf{E}[A] \geq 7p+5(1-p)$ or $\mathbf{E}[A] \geq 7$. Consider now job s_2, and notice that at the beginning of iteration 5 A cannot distinguish S from S'. If s_2 is a 2-job, and A schedules it on either machine 1 or machine 2, then $\mathbf{E}[A] \geq 6$, whereas if it is scheduled on machine 3, $\mathbf{E}[A] \geq 7p+5(1-p)$. If s_2 is a 3-job, if it is scheduled on machines 1 or 3, then $\mathbf{E}[A] \geq 6$, whereas if is scheduled on machine 2, $\mathbf{E}[A] \geq 7p+5(1-p)$.

We have thus exhausted all possible cases, and for every case we have that $\mathbf{E}[A] \geq 7p+5(1-p)$ or $\mathbf{E}[A] \geq 6$.

It follows that $\max_p \frac{\min(7p+5(1-p),6)}{6p+5(1-p)}$, is a lower bound on the approximation ratio of A. The expression is maximized at $p=\frac{1}{2}$, which gives a lower bound of $\frac{12}{11}$. $\square$

5.2. Fixed priority makespan scheduling with randomized decisions

Theorem 10. *No fixed priority randomized decisions algorithm for makespan scheduling with $m \geq 3$ has approximation ratio better than $\frac{10}{9}$.*

Proof. The set of potential jobs S is as defined in the proof of Theorem 6, i.e., $3\lceil \frac{m}{2} \rceil$ jobs of size 2, and $2\lfloor \frac{m}{2} \rfloor$ jobs of size 3. Fix a deterministic priority algorithm A, and on input S, let σ denote the sequence of the first m jobs as chosen by A. In order to determine an appropriate input distribution we will consider cases for σ as follows.

• *Case* 1: The number of jobs of size 2 in σ equals $\lceil \frac{m}{2} \rceil$ and the number of jobs of size 3 in σ equals $\lfloor \frac{m}{2} \rfloor$. In this case, with probability p_1 (to be determined later), the input to A consists only of the set of the m jobs in σ, while with probability $1-p_1$ the input contains all jobs in σ and one additional job J of size 3; denote this set by S'.
Subcase 1a: A schedules all jobs in σ on different machines. In this case, it is easy to see that $\mathbf{E}[A] = 3p_1 + 5(1-p_1)$.
Subcase 1b: A schedules two jobs in σ on the same machine. In this case, $\mathbf{E}[A] \geq 4$.

The optimal algorithm on input σ will schedule all m jobs on different machines, while on input S' it will place two jobs of size 2 on the same machine and all 3-jobs on different machines, hence $\mathbf{E}[OPT] = 3p_1+4(1-p_1)$, and the approximation ratio of A in case 1 is at least $\max_{p_1} \frac{\min(3p_1+5(1-p_1),4)}{3p_1+4(1-p_1)}$, which equals $\frac{8}{7}$, for $p_1 = \frac{1}{2}$.

• *Case* 2: The number of jobs of size 2 in σ is greater than $\lceil \frac{m}{2} \rceil$ *or* the number of jobs of size 3 in σ is greater than $\lfloor \frac{m}{2} \rfloor$. In this case, with probability p_2 (to be determined later) the actual input to A is the set S, while with probability $1-p_2$, the actual input consists only of jobs in σ.
Subcase 2a: A schedules all m jobs in σ on different machines. It is easy to see that on input S, A will schedule two jobs of different size on the same machine, or a machine will be assigned either 3 jobs of size 3 or 4 jobs of size 2. In either case, a minimum makespan of 7 is incurred (from Fact 7). Hence $\mathbf{E}[A] \geq 7p_2 + s(1-p_2)$, where s is the maximum size of jobs in σ.
Subcase 2b: A schedules two jobs in σ on the same machine. Using arguments as above it follows that $\mathbf{E}[A] \geq 6p_2+4(1-p_2)$.

On the other hand the optimal algorithm has makespan 6 on input S, and makespan equal to s on input σ. Thus, $\mathbf{E}[OPT] = 6p_2 + s(1-p_2)$, and the approximation ratio of A in the second case is at least

$$\max_{p_2} \frac{\min(7p_2 + s(1-p_2), 6p_2 + 4(1-p_2))}{6p_2 + s(1-p_2)},$$

with the constraint $s \in \{2, 3\}$. The expression is maximized for $p_2 = \frac{1}{2}$, in which case it is equal to $\frac{10}{9}$.

In either case, the approximation ratio of A is at least $\frac{10}{9}$. □

6. Oblivious vs. adaptive adversaries

Thus far we have assumed that the adversarial input is generated by an oblivious adversary. More precisely, an oblivious adversary has access only to the statement of the priority algorithm and not the random choices made throughout its execution. But what if the adversary is not oblivious, but instead it can observe the outcome of the algorithm's random choices in determining the ordering and/or random decisions? We call such an adversary *adaptive*[9], again using terminology from the analysis of online algorithms. Since an adaptive adversary has knowledge of the algorithm's random choices, it can adaptively remove input items from the set of potential input items based on the outcome of the algorithms choices. Consequently, the actual input to the algorithm, and hence the value of the optimal solution will be random variables. A priority algorithm is a c-approximation against an adaptive adversary if $\mathbf{E}[A(\sigma)] \leq c \cdot \mathbf{E}[OPT(\sigma)]$.

The following theorem shows that in the context of priority algorithms, the adaptive adversary is too powerful to be of any interest (reminiscent of the online world, where there is no advantage to using randomization against *adaptive-offline* adversaries [8]).

Theorem 11. *Suppose that a problem is not c-approximable by a deterministic priority algorithm. Then the problem is not c-approximable by any randomized priority algorithm against adaptive adversaries.*

Proof. We will assume the general case of an adaptive priority algorithm, and randomization on both ordering and decisions. Assume also, without loss of generality, that the problem in question is a cost-minimization problem. Since the problem is not c-approximable by a deterministic priority algorithm, it follows from [17] that the lower bound of c can be derived by an explicit game between a deterministic algorithm, denoted by A_d and an adversary (denoted by ADV_d). More precisely, the game can be described as a tree T, in which each node v corresponds to a subset of the (finite) set of potential input items S, denoted by S_v. In particular, the root r corresponds to a subset $S_0 \subseteq \mathcal{S}$ initially chosen by the adversary before execution of the algorithm, every leaf l in T corresponds to a singleton $s_l \in S$, and every internal node corresponds to a subset of S of cardinality at least 2. The tree can be described as follows:

For every internal node v, and every $s \in S_v$ and $d_s \in D(s)$, there is an edge labeled $\langle s, d_s \rangle$ which corresponds to the case that the next item considered by A_d is s, and the corresponding decision is d_s. Let v_k be an internal node in the tree, i.e., $|S_{v_k}| \geq 2$, at depth k, and let $v_0, v_1, v_2, \ldots, v_{k-1}$ be the nodes on the path from the root $r = v_0$ to v_k. Suppose that each edge on the path is labeled by $\langle s_j, d_{s_j} \rangle$ $(j < k)$, and denote by $seq(v_k)$ the sequence of input items $s_1, \ldots, s_{k-1}$. Then, for every pair s, d_s where $s \in S_{v_k}$ and $d_s \in D(s)$, v_k has a child v' and a set $S_{v'}$ of potential items corresponding to the subset of $S \setminus \{s_1, \ldots s_{k-1}, s\}$ that is maintained (i.e., not removed) by ADV_d, in the case the algorithm considers the input items $s_1, s_2, \ldots s_{k-1}, s$ (in this order) with corresponding decisions $d_{s_1}, \ldots d_{s_{k-1}}, d_s$. Note that the construction is such that every leaf l in T corresponds to a singleton $s_l \in S$. We let $seq(v_k)$ denote the sequence of input items $s_1, \ldots, s_{k-1}, s$. The fact that A_d cannot be a c-approximation priority algorithm means that for every leaf l, $A(seq(l)) > c \cdot OPT(seq(l))$.

[9] Not to be confused with adaptive priority algorithms.

Consider now a randomized priority algorithm A_r for the problem against the adaptive adversary denoted by ADV_r. Let X_i, Y_i be the random variables that indicate the i-th input item considered by A_r, and the decision concerning X_i, respectively. Recall that at every iteration the adversary has knowledge of both random variables. We describe how ADV_r works. The adversary presents S as the set of potential input items; we emphasize that S is identical to the set of potential input items used by the deterministic adversary ADV_d, described above. Suppose that, in the first round of the game between A_r and ADV_r, we have that $(X_1, Y_1) = (s, d_s)$; then the adversary removes all input items except the (unique) set that corresponds to the child of the root in the tree T which is labeled by $\langle s, d_s \rangle$. In general, suppose that, for a fixed i, $(X_j, Y_j) = (s_j, d_{s_j})$ for all $j \leq i$. Then, in the i-th round of the game, ADV_r will have removed all input items from S except for the (uniquely defined) set that corresponds to the vertex v in T which can be reached from the root r by following a path of vertices labeled by $\langle s_j, d_{s_j} \rangle$ in T.

The game between A_r and ADV_r terminates when the set of remaining inputs is a singleton. The random variable σ which denotes the input to A_r, as produced by A_r and ADV_r is identical to the sequence $seq(l)$ to which some leaf l in T corresponds. This implies that $A_r(\sigma) > c \cdot OPT(\sigma)$ and hence $\mathbf{E}[A_r[(\sigma)] > c \cdot \mathbf{E}[OPT(\sigma)]$. $\square$

7. Lower bounds on asymptotic approximability

The definition of a c-approximation algorithm as provided by (1), refers to *strict* approximability. Alternatively, we say that A is an *asymptotic* c-approximation algorithm if $A(\sigma) \leq c \cdot OPT(\sigma) + o(OPT(\sigma))$. This is a weaker definition of approximability, since it ignores the (sometimes pathological) performance of the algorithm on small input sets. We would like to be able to extend this definition to priority algorithms. This task is straightforward for deterministic algorithms, but it becomes more challenging when dealing with randomization. A similar complication arises in competitive analysis of online algorithms (see, e.g., [16]).

In order to apply the Von Neumann/Yao principle with respect to asymptotic approximability, we need to supplement condition (2) in Section 2.1 with additional constraints, similar to the argument of Chrobak et al. [16] in the context of online computation. In particular, we must show that for arbitrarily large l the distribution $\mathcal{D}$ we present must be such that $\mathbf{E}_{\mathcal{D}}[OPT(\sigma)] \geq l$, and that for all inputs σ (over which $\mathcal{D}$ is defined), we have that $\mathbf{E}_{\mathcal{D}}[OPT(\sigma)] \geq f(OPT(\sigma))$, where f is a function with $f(n) \in \Omega(n)$. We can then apply the argument of Chrobak et al. to show that no randomized priority algorithm can be an asymptotic c-approximation.

Some of our proofs can be modified so as to guarantee the above constraints. For the makespan scheduling problem, it suffices to scale the sizes of the jobs accordingly; for instance, we can replace the jobs of sizes 2 and 3 by jobs of sizes $2l$ and $3l$, respectively. For the problem of facility location in the facility input model, and in particular Theorems 1 and 3, the constraints are also satisfied since the expected cost of the optimal solution as well as the optimal solution on a random input are linear functions of n. However, for facility location in the city input model, the construction for the lower bound in Theorem 5 is such that there exist inputs σ for which $OPT(\sigma)$ can be as large as quadratic on n, whereas $\mathbf{E}_{\mathcal{D}}[OPT(\sigma)]$ is $O(n)$. Thus the result of Theorem 5 applies only to strict (non-asymptotic) approximability.

8. Conclusion

Our objective, in this work, is to extend the priority algorithm framework so as to capture greedy-like algorithms with access to randomization. We show that in this framework it is possible to obtain non-trivial lower bounds on the approximability of optimization problems by randomized priority algorithms. As case studies we considered two fundamental problems, namely facility location and makespan scheduling. We believe that this framework is applicable to a wider class of problems; in particular, we believe that it can be applied in the model of Davis and Impagliazzo [17], so as to address the power of randomized greedy-like algorithms for graph-related problems.

The obvious open question is whether our lower bounds can be improved. A much more compelling question is whether the intuition behind the lower bound constructions can lead to the design of better randomized priority algorithms. For instance, would the results of Section 5 suggest better upper bounds for randomized priority makespan scheduling? This would be a true testament to the importance of priority algorithms as an algorithmic paradigm. More specifically, we do not know whether the knowledge of the size of the input is critical in the context of makespan scheduling. Our results follow [13], and assume that the algorithm does not know the number of jobs in the input. However, we have not been able to remove this assumption, which points towards the possibility that a better algorithm could use this information to its benefit, in a clever manner. Nevertheless, we can still obtain some lower bounds when this information is available to the algorithm, albeit much weaker than the ones of Section 5.

Questions which apply to the competitive analysis of online algorithms can also be asked about the class of algorithms we study. For instance, how critical is memory in randomized priority algorithms? When are greedy decisions restrictive? When does randomization help? (See [11] for a discussion of the situation in the online world.) Wherever greedy algorithms are applied, it is relevant to understand the power and limitations of (randomized) greedy algorithms. The priority model (with irrevocable decisions) as studied in this paper provides a general framework for understanding the limitations of such algorithms. It is not the only such model but it does capture many of the known greedy algorithms and is amenable to analysis. More general models based on a priority ordering of the input items can be found (for example) in [2,10,55]. So far, only deterministic versions of these models have been studied.

Acknowledgements

The authors would like to thank Jeff Edmonds, Travis Gagie, Avner Magen, Tasos Viglas and an anonymous referee for their comments and suggestions.

References

[1] S. Albers, On randomized online scheduling, in: Proceedings of the 34th Annual ACM Symposium on Theory of Computation, STOC, 2002, pp. 134–143.
[2] M. Alekhnovich, A. Borodin, J. Buresh-Oppenheim, R. Impagliazzo, A. Magen, T. Pitassi, Toward a model for backtracking and dynamic programming, in: Proceedings of the 20th Annual IEEE Conference on Computational Complexity, CCC, 2005, pp. 308–322.
[3] S. Angelopoulos, Order preserving transformations and greedy-like algorithms, in: Proceedings of the Second Workshop on Approximation and Online Algorithms, WAOA, 2004, pp. 27–40.
[4] S. Angelopoulos, A. Borodin, The power of priority algorithms for facility location and set cover, Algorithmica 40 (4) (2004) 271–291.
[5] K. Arrow, Social Choice and Individual Values, Wiley, Chichester, 1951.
[6] A. Bar-Noy, S. Guha, J. Naor, B. Schieber, Approximating throughput in real-time scheduling, SIAM Journal of Computing 31 (2) (2001) 331–352.
[7] Y. Bartal, A. Fiat, H. Karloff, R. Vohra, New algorithms for an ancient scheduling problem, Journal of Computer and System Sciences 51 (3) (1995) 359–366.
[8] S. Ben-David, A. Borodin, R. Karp, G. Tardos, A. Wigderson, On the power of randomization in online algorithms, Algorithmica 11 (1) (1994) 2–14.
[9] A. Borodin, J. Boyar, K.S. Larsen, Priority algorithms for graph optimization problems, in: Proceedings of the 2nd Workshop on Approximation and Online Algorithms, WAOA, 2004, pp. 126–139.
[10] A. Borodin, D. Cashman, A. Magen, How well can primal-dual and local-ratio algorithms perform, in: Proceedings of the 32nd International Colloquium on Automata, Languages and Programming, ICALP, 2005, pp. 943–955.
[11] A. Borodin, R. El-Yaniv, Online Computation and Competitive Analysis, Cambridge University Press, 1998.
[12] A. Borodin, N. Linial, M.E. Saks, An optimal algorithm for metrical task systems, Journal of the ACM 39 (4) (1992) 745–763.
[13] A. Borodin, M.N. Nielsen, C. Rackoff, (Incremental) priority algorithms, Algorithmica 37 (4) (2003) 295–326.
[14] M. Charikar, Greedy approximation algorithms for finding dense components in a graph, in: Proceedings of the Third International Workshop on Approximation Algorithms for Combinatorial Optimization, APPROX, 2000, pp. 84–95.
[15] B. Chen, A. van Vliet, G. J. Woeginger, A lower bound for randomized on-line scheduling algorithms, Information Processing Letters 51 (1994) 219–222.
[16] M. Chrobak, L. Larmore, N. Reingold, J. Westbrook, A better lower bound on the competitive ratio of the randomized 2-server problem, Information Processing Letters 63 (1997) 79–83.
[17] S. Davis, R. Impagliazzo, Models of greedy algorithms for graph problems, in: Proceedings of the 15th Symposium on Discrete Algorithms, SODA, 2004, pp. 381–390.
[18] B.C. Dean, M.X. Goemans, J. Vondrák, Approximating the stochastic knapsack problem: the benefit of adaptivity, in: Proceedings of the 44th Annual Symposium on Foundations of Computer Science, FOCS, 2004, pp. 208–217.
[19] A. Dechter, R. Dechter, On the greedy solution of ordering problems, ORSA Journal on Computing 1 (3) (1989) 181–189.
[20] J. Edmonds, Submodular functions, matroids, and certain polyhedra, in: R. Guy (Ed.), Proceedings of the Calgary International Conference on Combinatorial Structures and Their Applications, Gordon and Breach, 1970, pp. 69–87.
[21] J. Edmonds, Matroids and the greedy algorithm, Mathematical Programming 1 (1971) 127–136.
[22] T. Erlebach, F.C.R. Spieksma, Interval selection: applications, algorithms, and lower bounds, Journal of Algorithms 46 (1) (2003) 27–53.
[23] M.L. Fisher, G.L. Nemhauser, L.A. Wosley, An analysis of approximations for maximizing submodular set functions — ii, Mathematical Programming Studies 8 (1978) 73–87.
[24] R. Fleischer, M. Wahl, Online scheduling revisited, in: Proceedings of the 8th Annual European Symposium on Algorithms, ESA, 2000, pp. 202–210.
[25] D. Fotakis, On the competitive ratio for online facility location, Algorithmica 50 (1) (2008) 1–57.
[26] M.R. Garey, D.S. Johnson, Computers and Intractability: A Guide to the Theory of NP-Completeness, 2nd edition, Freeman, New York, NY, 1983.
[27] T. Gormley, N. Reingold, E. Torng, J. Westbrook, Generating adversaries for request-answer games, in: Proceedings of the 11th ACM-SIAM Symposium on Discrete Algorithms, SODA, 2000, pp. 564–565.
[28] R.L. Graham, Bounds for certain multiprocessing anomalies, Bell System Technical Journal 45 (1966) 1563–1581.
[29] R.L. Graham, Bounds on multiprocessing timing anomalies, SIAM Journal of Applied Mathematics 17 (2) (1969) 416–429.
[30] S. Guha, S. Khuller, Greedy strikes back: Improved facility location algorithms, Journal of Algorithms 31 (1) (1999) 228–248.
[31] D.S. Hochbaum, D.B. Shmoys, Using dual approximation algorithms for scheduling problems: theoretical and practical results, Journal of the ACM 34 (1) (1987) 144–162.
[32] A. Hoffman, On simple linear programming problems, in: Proceedings of 7th Symposium in Pure Mathematics, American Math Society, 1963, pp. 317–327.
[33] S.L. Horn, One-pass algorithms with revocable acceptances for job interval selection, M.Sc. Thesis, University of Toronto, 2004.
[34] K. Jain, M. Mahdian, A. Saberi, A new greedy approach for facility location problems, in: Proceedings of the 34th Annual ACM Symposium on Theory of Computation, STOC, 2002, pp. 731–740.
[35] K. Jain, V.V. Vazirani, Approximation algorithms for metric facility location and k-median problems using the primal-dual schema and lagrangian relaxation, Journal of the ACM 48 (2) (2001) 274–296.
[36] B. Korte, L. Lovász, Mathematical structures underlying greedy algorithms, Lecture Notes in Computer Science: Fundamentals of Computation Theory 177 (1981) 205–209.
[37] B. Korte, L. Lovász, Greedoids and linear objective functions, SIAM Journal Algebraic and Discrete Methods 5 (1984) 229–238.
[38] G. Kortsarz, D. Peleg, Generating sparse 2-spanners, Journal of Algorithms 17 (2) (1994) 222–236.
[39] N. Lesh, M. Mitzenmacher, A simple heuristic for improving priority-based greedy algorithms, Information Processing Letters 97 (2006) 161–169.
[40] M. Mahdian, E. Markakis, A. Saberi, V. V. Vazirani, A greedy facility location algorithm analyzed using dual fitting, in: Proceedings of the 4th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems, APPROX, 2001, pp. 127–137.
[41] M. Mahdian, J. Ye, J. Zhang, A 1.52-approximation algorithm for the uncapacitated facility location problem, in: Proceedings of the 5th International Workshop on Approximation Algorithms for Combinatorial Optimization Problems, APPROX, 2002, pp. 229–242.
[42] R.R. Mettu, C.G. Plaxton, The online median problem, SIAM Journal of Computing 32 (3) (2003) 816–832.
[43] A. Meyerson, Online facility location, in: Proceedings of the 42nd Annual IEEE Symposium on Foundations of Computer Science, FOCS, 2001, pp. 426–431.
[44] R. Motwani, P. Raghavan, Randomized Algorithms, Cambridge University Press, 1995.
[45] P. Papakonstantinou, Hierarchies for classes of priority algorithms for job scheduling, Theoretical Computer Science 352 (2006) 181–189.
[46] R. Rado, A note on independence functions, Proceedings of the London Mathematical Society 7 (1957) 300–320.
[47] O. Regev, Priority algorithms for makespan minimization in the subset model, Information Processing Letters 84 (3) (2002) 153–157.
[48] S.K. Sahni, Algorithms for scheduling independent tasks, Journal of the ACM 23 (1) (1976) 116–127.
[49] S. Seiden, J. Sgall, G.J. Woeginger, Semi-online scheduling with decreasing job sizes, Operations Research Letters 27 (5) (2000) 215–221.
[50] J. Sgall, A lower bound for randomized on-line multiprocessor scheduling, Information Processing Letters 63 (1997) 51–55.

[51] J. Sgall, On-line scheduling—a survey, in: A. Fiat, G. Woeginger (Eds.), On-line Algorithms: The State of the Art, in: Lecture Notes in Computer Science, Springer Verlag, 1998, pp. 196–231.
[52] D.B. Shmoys, Approximation algorithms for facility location problems, in: Proceedings of the Third International Workshop on Approximation Algorithms for Combinatorial Optimization, APPROX, 2000, pp. 27–33.
[53] D.B. Shmoys, E. Tardos, K. Aardal, Approximation algorithms for facility location problems (extended abstract), in: Proceedings of the 29th Annual ACM Symposium on Theory of Computing, STOC, 1997, pp. 265–274.
[54] A.C.C. Yao, Probabilistic computations: toward a unified measure of complexity, in: Proceedings of the Eighteenth Annual IEEE Symposium on Foundations of Computer Science, FOCS, 1977, pp. 222–227.
[55] Y. Ye, A. Borodin, Priority algorithms for the subset-sum problem, in: Proceedings of the 13th Annual Computing and Combinatorics Conference, COCOON, in: Lecture Notes in Computer Science, vol. 4598, 2007, pp. 504–514.