Available online at www.sciencedirect.com
SCIENCE DIRECT®
Theoretical Computer Science 352 (2006) 181–189
Theoretical Computer Science
www.elsevier.com/locate/tcs

Hierarchies for classes of priority algorithms for Job Scheduling

Periklis A. Papakonstantinou*

Department of Computer Science, University of Toronto, 10 King's College Rd., Toronto, ON, Canada M5S 3G4

Received 18 August 2004; received in revised form 26 October 2005; accepted 31 October 2005

Communicated by G. Ausiello

Abstract

Priority algorithm is a model of computation capturing the notion of greedy and greedy-like algorithm. This paper concerns priority algorithms for Job Scheduling. Three main restrictions are defined for priority algorithms, namely: memoryless, greedy and fixed. It was asked in [A. Borodin, M.N. Nielsen, C. Rackoff, (Incremental) priority algorithms, in: Proc. 13th Annu. Symp. Discrete Algorithms (SODA), January 2002, pp. 752–761 (also in Algorithmica 37(4) (2003) 295–326] whether the general class of priority algorithms is of different power from the restricted class of greedy-priority algorithms (for the Job Scheduling problem). We answer this question affirmatively by showing that a specific priority algorithm cannot be simulated by any greedy-priority algorithm on every input. Furthermore we systematically compare every two classes of priority algorithms for different settings of Job Scheduling. We also define a hierarchy for priority algorithms of bounded memory which lies between the class of memoryless and the class of priority algorithms with memory, and we show that this memory hierarchy is robust.

Keywords: Greedy algorithm; Classes; Algorithmic paradigms

1. Introduction

Greedy algorithms have been extensively applied due to their conceptual simplicity and computational efficiency. Priority algorithm [3] is a model of computation that captures the notion of greedy and greedy-like algorithms by generalizing on-line computation. This approach formulates a much wider class of greedy algorithms than previous ones such as the matroids, greedoids and matroid embeddings, and also gives rise to the study of the approximation power of greedy algorithms. For example, with respect to priority algorithms we are able to prove *unconditional* lower bounds on the approximation ratio of such algorithms. By *unconditional* we mean that the statements do not base their validity on assumptions like $P \neq NP$.

Three restrictions are defined for the (general) model of priority algorithm, namely: fixed, memoryless and greedy. During the study on the approximation power for classes of priority algorithms for Job Scheduling, questions concerning relations among these classes were raised in [3]. In this paper we also study classes of priority algorithms for Job Scheduling. Our techniques are model-specific (i.e. for Job Scheduling) but we believe that similar techniques can be applied for priority models defined for other combinatorial problems. The main technical contribution of this paper

* Tel.: +1 416 978 4299; fax: +1 416 978 1931.
E-mail address: papakons@cs.toronto.edu.

0304-3975/$ - see front matter © 2005 Elsevier B.V. All rights reserved.
doi:10.1016/j.tcs.2005.10.045

concerns a thorough comparison among classes of priority algorithms, and the definition of a memory hierarchy for priority algorithms which is shown to be robust.

Previous work. Borodin et al. [3] initiated the study of priority algorithms. In [3] classes of priority algorithms are defined for variations of the Job Scheduling and the Makespan problems. Priority algorithms and lower bounds on the approximation ratio for classes of priority algorithms are shown. The subset model for the Makespan problem, defined in [3], is considered by Regev [7] who shows a lower bound on the approximation ratio of any fixed-priority algorithm. There are also works formulating problems other than scheduling in the context of prioritized computation. These models can be adjusted in a natural way to incorporate the memoryless, fixed and greedy restrictions on the priority model. Angelopoulos and Borodin [2] define and study priority algorithms for two well-known *NP*-hard optimization problems, Set Cover and (uncapacitated) Facility Location. Motivated by similar works in the field of on-line computation, Angelopoulos [1] considers randomized classes of priority algorithms. Priority algorithms for graph problems are studied by Impagliazzo and Davis [4]. They study the approximation power of priority algorithms for the Shortest Path, Steiner Tree, Weighted Vertex Cover and Independent Set problems; they present an abstract model for priority algorithms; they describe an abstract way of capturing the notion of memoryless algorithm, and they present a general lower bounding technique in terms of a combinatorial game.

Priority algorithms for Job Scheduling. We use the term "priority algorithms" to refer to "priority algorithms for the Job Scheduling problem". In the general (*adaptive*) priority model the computation proceeds in rounds. In each round: (i) a total ordering is chosen over *all possible* (not-yet-considered) jobs and (ii) the next, highest priority, available job is presented to the algorithm which makes an *irrevocable* decision to reject or to schedule this job (specifying the processor and the starting time). The determined orderings and the decisions are arbitrarily complex (maybe non-computable) functions of the so far scheduled and rejected jobs. A natural restriction to this model is the *fixed*-priority model where the ordering is determined only once in the beginning. We define the *greedy* restriction to be such that when a greedy-priority algorithm reads a job that can be scheduled, it necessarily schedules it. Also we define the *memoryless* restriction to refer to algorithms where every decision to schedule/reject a job and the determined ordering are based only on the scheduled (not on the rejected) jobs. We denote as PRIORITY the (unrestricted) class of adaptive priority algorithms. If the class is restricted to the class of fixed, greedy and memoryless priority algorithms, we add the prefix FIXED (or *F* for short), GREEDY (*G*) and MEMORYLESS (*M*), respectively.

In [3] the authors give some indications that "greediness is a real restriction to the model". These indications stem from a randomized analog studied (for the Job Scheduling problem). In [4] separation results based on the approximation ratio are presented for classes of priority algorithms for graph problems. This will not work in the case of Job Scheduling. The *LPT* (Longest Processing Time first) priority algorithm is optimal w.r.t. its approximation ratio for one processor Interval Scheduling with proportional profits [3]. Since, $LPT \in$ MEMORYLESS-GREEDY-FIXED-PRIORITY it is impossible to separate such classes of priority algorithms based on the approximation ratio. Instead we show that there exists a natural setting in which we can show such a separation. We base our separation results on the weaker notion of not-feasible-simulation introduced in Definition 1. Specifically, we show that a given priority algorithm *A* cannot be simulated by any priority algorithm belonging to a given class *C*, showing that no algorithm from *C* can do as well as *A* on every set of jobs.

Contribution. We compare classes of priority algorithms for the (non-preemptive) Job and Interval Scheduling on one and more than one (identical) machine environments. Our study shows that restricting (using one of the three above mentioned restrictions) a given class of priority algorithm usually results in a weaker class. In this sense we have a natural notion of hierarchy among classes of priority algorithms. Taking into account all possible combinations of restrictions, for classes of priority algorithms, we have eight classes. We develop some basic combinatorial tools that we use to construct input instances so as to compare every possible pair of classes of priority algorithms for the Job Scheduling Problem. The separation results are based on the (weak) notion of not-feasible-simulation. In Section 2 we define relations that capture this notion. One open question was whether greediness is a real restriction to the model [3]. We answer this question affirmatively by separating GREEDY-PRIORITY from PRIORITY even in case of one processor Interval Scheduling with proportional profits. Furthermore, we provide a comparison for every pair of classes of priority algorithms in case of: (i) Job Scheduling, (ii) Interval Scheduling for one processor machine environments and (iii) Interval Scheduling on multiple processors. For each case we have 64 comparisons. With respect to cases (ii) and (iii), the same relations hold between every pair of priority algorithm classes. Thus, in total we have 128 lemmas (Section 4) which are corollaries of seven main lemmas (Section 3). One of our main contributions regards memory models for priority algorithms. In Section 5 we define an (infinite) hierarchy of classes of priority algorithms of bounded memory.

We show that this hierarchy is robust; that is, in general, priority algorithms that remember k rejected jobs cannot be simulated by priority algorithms that have memory only for $k-1$ rejected jobs. In the same section we show that another natural bounded memory hierarchy, where the memory of the algorithm consists of bits, is also robust.

2. Preliminaries

A *job* J is $(id, r, d, p, w) \in \mathbb{N}^5$, where *id* is the job descriptor, r, d, p is its release, deadline and processing time and w is its profit (weight). Furthermore, $r < d$ and $p \leqslant d - r$. We omit *id* given that it is denoted by the job subscript (for example J_i). An *interval* is a restricted form of job where $p = d - r$. That is, an interval J is well defined by $J = (id, r, d, w)$. In case of *proportional profit* $p = w$ a job is defined as (id, r, d, p) and an interval is defined as (id, r, d). For two distinct $(i \neq j)$ jobs $J_i = (i, r, d, p, w)$, $J_j = (j, r, d, p, w)$ we may abuse notation by writing $J_i = J_j$ meaning that J_i and J_j have the same release, deadline, processing time and profit. An instance of the non-preemptive Job (Interval) Scheduling on identical machines problem ($P|r_j|\sum w_j \bar{U}_j$ in [5] notation) consists of m machines and n jobs (intervals). The goal is to assign jobs to machines maximizing the sum of profits of the scheduled jobs. Given a finite set of jobs S and a priority algorithm A, we write $A(S)$ to denote the set of jobs scheduled by A when computing on S. Also, w_J denotes the profit of job J and we write $\mathit{profit}(S) = \sum_{J \in S} w_J$.

In this paper the memoryless-priority algorithms make decisions by having access to all the information of a scheduled job (including its identifier) and on which processor it has been scheduled. Minor modifications to the proofs of Section 3 make the lemmas hold for weaker definitions of memoryless models.

Definition 1.

- For two classes of priority algorithms C_1 and C_2, C_1 *is partially dominated by* C_2, $D(C_1, C_2)$, iff there exists a priority algorithm $A \in C_2$, such that for every algorithm $A' \in C_1$ there exists a finite set of input jobs S such that $\mathit{profit}(A(S)) > \mathit{profit}(A'(S))$.
  The D-relation describes the case where a priority algorithm cannot be simulated by priority algorithms belonging to another class, even when this simulation is not computable.
- If for every $A \in C_1$ there exists an $A' \in C_2$, such that for every input instance S, $\mathit{profit}(A(S)) \leqslant \mathit{profit}(A'(S))$ then we write $C_1 \leqslant C_2$. Note that $C_1 \leqslant C_2$ means that C_1 can be simulated by C_2, that is $\neg D(C_2, C_1)$ holds.
- If $C_1 \leqslant C_2$ and $C_2 \leqslant C_1$ then we write $C_1 \approx C_2$. If $D(C_1, C_2)$ and $C_1 \leqslant C_2$ then we write $C_1 < C_2$. If it is true that $D(C_1, C_2)$ and $D(C_2, C_1)$ then we write $C_1 \perp C_2$, that is C_1 and C_2 are essentially incomparable.

It easily follows that relation D is not transitive, whereas relations $\leqslant$ and $<$ are. Also, the relation $<$ defines a partial order since it is also anti-symmetric; relations $\perp$ and $\approx$ are symmetric. The following proposition is a direct consequence of the definitions.

Proposition 1. (i) *Let* C_1, C_2, C_1', C_2' *be classes of priority algorithms. If* $C_1' \leqslant C_1$ *and* $C_2 \leqslant C_2'$ *and* $D(C_1, C_2)$ *then* $D(C_1', C_2')$. (ii) *If* C *and* C' *are classes of priority algorithms then there exists a unique* R *among* $\{<, >, \approx, \perp\}$, *such that* $C\ R\ C'$ (*where* $>$ *has the obvious definition*).

Lemma 2. M-G-PRIORITY $\approx$ M-PRIORITY.

This result (from [3]) means that in case of a memoryless-priority algorithm, greediness is not a real restriction. This lemma, as well as those that directly follow from it, are the only ones where greediness is not a restriction. Lemma 2 is the only previous result concerning comparison among classes of priority algorithms for Job Scheduling. It also holds for every variation of the Job Scheduling problem considered in this work, since its proof is based on a (general) simulation argument.

Gadgets used throughout the constructions of the proofs. We systematically construct instances to separate classes of priority algorithms. In our proofs we use three types of "gadgets": (i) indicator jobs, (ii) switches and (iii) blockers. An *indicator job* is a job (or interval) that works as a "promise for the following jobs in the input set, giving advantage to an adaptive over all fixed-priority algorithms. A *switching gadget* is (intuitively) used to implement memory bits (giving advantage to algorithms with memory), for a set of jobs (or intervals) where their rejection or non-presence

provides information about the rest of the jobs in the input instance. A *blocker* is a job which helps a greedy algorithm to overcome part of its greediness when the blocker is scheduled in a place which prevents other jobs/intervals from being scheduled.

3. Partially dominated classes of priority algorithms

All the D-relation proofs presented in this section are constructive. Specifically, in order to prove $D(C_1, C_2)$ we give an algorithm in C_2, and a specific finite set of inputs S, such that every algorithm in C_1 performs worse than the algorithm in C_2, for at least one subset of S.

Remark 1. All the separation lemmas presented in this section also hold in case of machine environments with $m > 1$ identical processors. The modification is easy. Add $m - 1$ jobs each of which overlaps with every other job in the input set considered in the initial proof. The set of lemmas presented in this section is minimal. That is, all the D-relations proved here are in the strongest possible form, in the sense that removing restrictions from the first argument or restricting the second argument makes the relation not hold. Observe that the given simulations hold for the arbitrary profit case whereas for the proofs of D-relations it suffices to consider the proportional profit case, thus making our results stronger. It is interesting to note that in all separation results the presented algorithms do not gain optimal profit on every subset of the presented set of jobs. In [6] we study priority algorithms that gain optimal profit on every subset of given sets and we also study related complexity questions. Moreover, note that our examples are engineered in such a way that by scaling the job profits or by adding multiple non-overlapping copies of the input set examples we can construct examples of sets of jobs where the two separated priority classes have arbitrarily large gaps in the gained profit.

3.1. Partially dominated classes of priority algorithms for one processor Interval Scheduling

We consider one machine environments where the jobs are intervals of proportional profit. We provide the full proof of Lemma 3 and we sketch the rest since the arguments are of very similar flavor.

Lemma 3. D(FIXED-PRIORITY, MEMORYLESS-GREEDY-PRIORITY)

Proof. Let $S = \{J_1, J_2, J_3, J_4\}$, where $J_1 = (0, 1)$, $J_2 = (1, 3)$, $J_3 = (3, 5)$, $J_4 = (2, 5)$. Here J_1 corresponds to an *indicator job*. Let $A \in$ M-G-PRIORITY be the following algorithm, with initial ordering $J_1 > J_2 > J_3 > J_4$:

- If J_1 is in the input then schedule it and determine an ordering starting with J_4 and proceed greedily.
- Else, schedule greedily (that is J_2, J_3 and then J_4).

Let $A' \in$ F-PRIORITY. If in the ordering of A', J_2 or J_3 precedes (has higher priority than) J_4 then on input $S' = \{J_1, J_2, J_4\}$ (or on $\{J_1, J_3, J_4\}$) $profit(A'(S')) \leqslant 3 < 4 = profit(A(S'))$. Else if J_4 precedes J_2 and J_3 then on input $S' = \{J_2, J_3, J_4\}$ $profit(A'(S')) \leqslant 3 < 4 = profit(A(S))$. □

Lemma 4. D(GREEDY-PRIORITY, FIXED-PRIORITY)

Proof sketch. Let $S = \{J_1, J_2, J_3, J_4\}$, $J_2 = (0, 2)$, $J_3 = (2, 4)$, $J_4 = (1, 4)$ and $J_1 = (2 - \varepsilon, 2 + \varepsilon)$, for a fixed $\varepsilon < 1$ (J_1 is an *indicator job*). For every FIXED-PRIORITY there exists a subset of S where the following algorithm $A \in$ FIXED-PRIORITY, with ordering $J_1 > \mathbf{J_4} > J_2 > J_3$ gains more profit: (i) If J_1 is read from the input then reject it, and schedule greedily (schedule as many intervals as possible). (ii) Else if J_1 is not the first interval read (that is, J_1 is not rejected when one of J_4, J_2, J_3 is read) then reject J_4 (if present) and schedule the rest greedily. □

Lemma 5. D(GREEDY-FIXED-PRIORITY, MEMORYLESS-FIXED-PRIORITY)

Proof sketch. Let $S = \{J_1, J_2, J_3, J_4\}$, where $J_1 = (0, 1)$, $J_2 = (1, 3)$, $J_3 = (3, 5)$, $J_4 = (2, 5)$. Intuitively, J_1 is an *indicator job*. The lemma is proved by considering the following algorithm $A \in$ M-F-PRIORITY, with fixed ordering

$J_1 > J_4 > J_2 > J_3$: (i) If J_1 is present then schedule it, reject J_4 (if present) and schedule the rest greedily. (ii) Else schedule greedily. □

Lemma 6. D(MEMORYLESS-GREEDY-PRIORITY, GREEDY-PRIORITY)

Proof sketch. Let $S = \{J_1, J_2, J_3, J_4, J_5\}$ such that $J_1 = J_2 = (0, 1)$ (J_1, J_2 is a switching gadget) and $J_3 = (1, 3)$, $J_4 = (3, 5)$, $J_5 = (2, 5)$. The lemma is proved by considering $A \in$ G-PRIORITY, with initial ordering $J_1 > J_2 > J_3 > J_4 > J_5$. A has the following description: (i) If J_1 and J_2 are both present then determine the ordering $J_5 > J_3 > J_4$. (ii) Else schedule greedily keeping the initially determined ordering. □

3.2. Lemmas that are different in case of one processor Interval Scheduling, multiple processors Interval Scheduling and Job Scheduling problem

The following lemma corresponds to the case of $m = 2$ processors, Interval Scheduling with proportional profits problem. We provide full-proofs for Lemmas 7 and 8 since they are getting a bit more involved than the proofs of the previous subsection.

Lemma 7. D(MEMORYLESS-PRIORITY, GREEDY-FIXED-PRIORITY), *for proportional profit Interval Scheduling on two-machine environments*.

Proof. Say that we have two processors P_1 and P_2. Let $S = \{J_1, J_2, J_3, J_4, J_5, J_6, J_7, J_8\}$, where: (i) $J_1 = J_2 = J_3 = (0, 1)$ are the switches, (ii) $J_4 = J_6 = (1, 3)$, $J_5 = J_7 = (3, 5)$ and $J_8 = (2, 5)$ (the intervals J_4 and J_5 are used as blockers). Recall that the blockers help us to overcome the fixed and the greedy nature of the algorithm since we use them to reject J_8 (if needed). Let $A \in$ G-F-PRIORITY with ordering $J_1 > J_2 > J_3 > \mathbf{J_4} > \mathbf{J_5} > J_8 > J_6 > J_7$. A is as follows: (i) Schedule greedily from $\{J_1, J_2, J_3\}$. (ii) If all J_1, J_2 and J_3 are present then schedule J_4 at P_1 and J_5 at P_2. (iii) Else schedule both J_4 and J_5 at P_1. (iv) Schedule the rest greedily.

Let $A' \in$ MEMORYLESS-PRIORITY. If A' has at least the same profit as A on every input, then on input S, A' must schedule all of J_4, J_5, J_6, J_7. Assume that $U \in \{J_4, J_5, J_6, J_7\}$ is the one read last through this run of A'. Since A' is deterministic, when running on $S' = S\backslash\{U\}$ A' still rejects J_8 (by scheduling all of $\{J_4, J_5, J_6, J_7\}\backslash\{U\}$). Let $U' \in \{J_1, J_2, J_3\}$ be the interval read last when A' runs on S' and thus U' is the one not scheduled. Since A' is memoryless, on input $S'' = S'\backslash\{U'\}$ A also rejects U and therefore $profit(A'(S'')) = 7 < 8 = profit(A(S''))$. □

The same lemma holds in case of the Job Scheduling problem for one processor.

Lemma 8. D(MEMORYLESS-PRIORITY, GREEDY-FIXED-PRIORITY), *for proportional profit Job Scheduling on one-machine environments*.

Proof. The representation of the input is given in Fig. 1. This is the only proof of this section where we do not use Interval Scheduling. In the figure three types of objects are depicted: (a) $\{J_1, J_2, J_3, J_4, J_5\}$ are proportional profit intervals. (b) Job J_6 is a proportional profit job and its release and deadline times are depicted at the ends of the dashed line. (c) $\{h_1, h_2\}$ are the two *holes*. The holes are not jobs (or intervals). They are "time intervals" lying between the intervals of (a). Notice that the job J_6 has processing time equal to the length of each hole. Regarding the processing

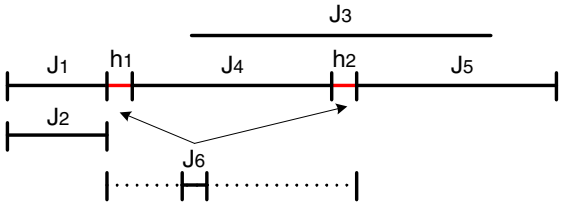

Fig. 1. Graphical representation of the input for the proof of the Lemma 8.

time: J_1, J_2 have processing time 1, J_4, J_5 have processing time 2, J_3 has processing time 3, and J_6 strictly less than 1, say 0.9. Consequently the length of each hole is 0.9. Thus, J_1 and J_2 are used as *switches* and J_6 as a *blocker*. Say that $A \in$ G-F-PRIORITY has job ordering $J_1 > J_2 > J_6 > J_3 > J_4 > J_5$, and works as follows: (i) Schedule greedily from $\{J_1, J_2\}$. (ii) If J_1 and J_2 are both present: schedule J_6 in h_2 (that is preventing J_3 from being scheduled) and schedule the rest greedily. (iii) If J_1 or J_2 is not present: schedule J_6 in h_1 and the rest greedily.

Let $A' \in$ M-PRIORITY. If A' has at least the same profit as A on every input, then on input S A' schedules J_4, J_5. Let $U \in \{J_4, J_5\}$ be the one scheduled last by A'. Thus on input $S' = S\backslash\{U\}$ A' rejects J_3 since it still schedules one among $\{J_4, J_5\}$. Let $U' \in \{J_1, J_2\}$ be the rejected interval (among $\{J_1, J_2\}$) when A' was running on S'. Since A' is memoryless, on input $S'' = S'\backslash\{U'\}$, $profit(A'(S'')) = 3.9 < 4.9 = profit(A(S''))$. □

Things are different in case of proportional profit Interval Scheduling on one processor. An immediate corollary of the following lemma and the derived corollary D(G-F-PRIORITY, M-G-PRIORITY) is that M-G-F-PRIORITY $\approx$ G-F-PRIORITY $<$ M-G-PRIORITY $\approx$ M-PRIORITY. This corollary does not hold in case of Job Scheduling or Interval Scheduling on more than 1 processor.

Lemma 9. MEMORYLESS-GREEDY-FIXED-PRIORITY $\approx$ GREEDY-FIXED-PRIORITY, *in case of Interval Scheduling and* $m = 1$. *This directly implies that in case of Interval Scheduling and* $m = 1$, GREEDY-FIXED-PRIORITY $<$ MEMORYLESS-PRIORITY.

Proof. This proof is a trivial simulation. Since $A \in$ G-F-PRIORITY, A cannot make any decisions (because we have only one machine and intervals) based on the already rejected and scheduled intervals. Hence, $A \in$ M-G-F-PRIORITY. □

4. Comparing classes of priority algorithms

We compare every two classes of priority algorithms for the Job Scheduling problem. Adding a restriction to a class of priority algorithms most of the time yields a weaker class. In this sense the corresponding hierarchy appears to be almost robust. We have three cases: (i) Job Scheduling on any number of processors (recall Remark 1), (ii) Interval Scheduling on $m \geqslant 2$ processors and (iii) Interval Scheduling on one processor ($m = 1$). Since, all the "D-relations" that hold for Job Scheduling ($m \geqslant 1$) hold also for Interval Scheduling on $m \geqslant 2$, then the same relations between classes of priority algorithms hold in these two cases. Table 1 concerns the comparison of classes for (i) and (ii) and Table 2 concerns (iii). Each table entry contains the relation and the lemmas used to derive this relation. If the table entry (r, c) contains the relation R, the class which corresponds to the rth row is C and the class which corresponds to the cth column is C', then we read "$C\ R\ C'$ is true". Only the upper triangular part of the table is filled, since all other relations trivially follow. PR is an abbreviation for PRIORITY. Also, notice that the column M-PR could be missing as well, because M-PR $\approx$ M-G-PR.

Table 1
Comparison between classes of priority algorithms in the cases of (i) general Job Scheduling ($m \geqslant 1$) and (ii) Interval Scheduling where $m \geqslant 2$

	F-PR	G-PR	G-F-PR	M-PR	M-F-PR	M-G-PR	M-G-F-PR
PR	> 3	> 4	> 3	> 6	> 6	> 6	> 6
F-PR		$\perp$ 3,4	> 4	$\perp$ 2,3,4	$> 2,4$	$\perp$ 3,4	$> 2,4$
G-PR			> 3	> 6	$> 2,3$	$> 6,2$	$> 6,2$
G-F-PR				$\perp$ 3,7or 8	$\perp$ 5,7 or 8	$\perp$ 3,7or 8	> 7 or 8
M-PR					> 3	≈ 2	> 3
M-F-PR						$< 2,3$	> 5
M-G-PR							> 3

Table 2
Comparison between classes of priority algorithms for one processor, proportional profit, Interval Scheduling

	F-PR	G-PR	G-F-PR	M-PR	M-F-PR	M-G-PR	M-G-F-PR
PR	> 3	> 4	> 3	> 6	> 3	> 6	> 3
F-PR		$\perp$ 3,4	> 4	$\perp$ 2,3,4	$> 2{,}4$	$\perp$ 3,4	$> 2{,}4$
G-PR			> 3	$> 6{,}2$	$> 2{,}3$	> 6	> 6
G-F-PR				$< 3{,}9$	$< 5{,}9$	$< 3{,}9$	≈ 9
M-PR					> 3	≈ 2	> 3
M-F-PR						$< 2{,}3$	> 5
M-G-PR							> 3

5. Memory and priority algorithms

Observe that so far we have considered two extremes regarding priority algorithms for the Job Scheduling problem; algorithms with memory and memoryless algorithms. Note that for an algorithm with memory it suffices to only have access to the sets of scheduled and rejected jobs (in the sense that given an algorithm which maintains its memory contexts throughout its execution, we can iteratively reconstruct them by just checking the description of the algorithm and the set of scheduled and rejected jobs).

We now give two definitions for bounded memory which imply natural memory hierarchies that we show to be robust in most settings.

Bounded memory model with irrevocable store and jobs.

Definition 2. We write for a priority algorithm A, that $A \in M(k) - C$ where C is a class of priority algorithms with memory, to mean that A makes its (irrevocable) decisions by remembering any set of rejected jobs of cardinality less than or equal to $k \in Z^{\geqslant 0}$. Each decision of A to store a job in its memory is also irrevocable; that is, when A stores a rejected job this job cannot be removed/updated later on.

Remark 2. Looking at the proof of Lemma 4 we see that Lemma 2 cannot be extended even when we have memory for only one rejected job. Actually, the same proof (as the one for Lemma 4) shows that D(M(1)-Greedy-Priority, M(1)-Fixed-Priority).

Notice that in Definition 2 the algorithm is allowed to decide whether it will store a rejected job in its memory or not. The following theorems are proved (even) for the weaker case where the algorithm (on the right-hand side of the relation D) stores, without choice, the first up to k, so far rejected jobs.

Theorem 10. *We consider two different settings.* (i) *Interval Scheduling where the machine environment consists of one processor and C stands for any one among* {Fixed-Priority, Greedy-Priority, Priority}. (ii) *Job Scheduling and we have a machine environment of $m \geqslant 1$ identical processors, or Interval Scheduling and we have a machine environment of $m \geqslant 2$ identical processors, and C stands for one among* {Greedy-Fixed-Priority, Fixed-Priority, Greedy-Priority, Priority}. *Given* (i) *or* (ii) *we have that $M(k+1) - C > M(k) - C$.*

A corollary of Lemma 9 is M(k)-G-F-Priority $\approx$ G-F-Priority for Interval Scheduling on one machine environment and every $k \in \mathbb{Z}^{\geqslant 0}$. Theorem 10 is a corollary of the three following lemmas (also recall Remark 1).

Lemma 11. *For the two machine environment Interval Scheduling,*
D(M(k)-Priority, M($k+1$)-G-F-Priority) *even for proportional profit intervals.*

Proof sketch. We do not give the full proof of this lemma since it is very similar to the proof of Lemma 7. Say that the two processors are P_1 and P_2. Let $S = \{J_{i_1}, \ldots, J_{i_{k+3}}\} \cup \{J_1, J_2, J_3, J_4, J_5\}$ be a set of intervals where (i) $J_{i_1} = J_{i_2} = \cdots = J_{i_{k+3}} = (0, 1)$ are the switches, (ii) $J_1 = J_3 = (1, 3)$, $J_2 = J_4 = (3, 5)$ and $J_5 = (2, 5)$, (iii) the jobs J_1 and J_2 are used as blockers. Let $A \in$ M($k+1$)G-F-Priority with initial ordering $J_{i_1} > J_{i_2} > \cdots > J_{i_{k+3}} >$

$J_1 > J_2 > \mathbf{J_5} > J_3 > J_4$. A is as follows:

- Schedule greedily from $\{J_{i_1}, \ldots, J_{i_{k+3}}\}$.
- If all $J_{i_1}, \ldots, J_{i_{k+3}}$ are present then schedule J_1 at P_1 and J_2 at P_2.
- Else schedule both J_1 and J_2 at P_1.
- Schedule the rest greedily.

Since the algorithm A can store all possibly rejected intervals and any algorithm with memory bounded to k rejected jobs cannot, this proof follows in a very similar way as the proof of Lemma 7. □

Lemma 12. *For the one machine environment Job Scheduling*,
$D(\mathrm{M}(k)\text{-PRIORITY}, \mathrm{M}(k+1)\text{-G-F-PRIORITY})$.

Proof sketch. This proof is also very similar to the proof of Lemma 8. Instead of having two intervals of the same release and deadline times, we now have $k+2$ intervals of the same form. The rest of the proof is similar to the one of Lemma 8 with all the modifications made in the obvious way. □

Actually, we can take Lemmas 7 and 8 as corollaries of the above two.

Lemma 13. *For the one machine environment Interval Scheduling*,

(i) $D(\mathrm{M}(k)\text{-PRIORITY}, \mathrm{M}(k+1)\text{-F-PRIORITY})$,
(ii) $D(\mathrm{M}(k)\text{-PRIORITY}, \mathrm{M}(k+1)\text{-G-PRIORITY})$.

Proof. (i) Let $S = \{J_{i_1}, J_{i_2}, \ldots, J_{i_{k+2}}\} \cup \{J_1, J_2, J_3\}$, where $J_{i_1} = J_{i_2} = \cdots = J_{i_{k+2}} = (0,1)$, $J_1 = (1,3)$, $J_2 = (3,5)$, $J_3 = (2,5)$. Let $A \in \mathrm{M}(k+1)\text{-F-PRIORITY}$ with ordering $J_{i_1} > J_{i_2} > \cdots > J_{i_{k+2}} > \mathbf{J_3} > J_1 > J_2$ and the following description:

- Schedule greedily from $\{J_{i_1}, J_{i_2}, \ldots, J_{i_{k+2}}\}$.
- If all $J_{i_1}, J_{i_2}, \ldots, J_{i_{k+2}}$ are present then reject J_3 (if present) and schedule the rest greedily.
- Else schedule greedily.

Let $A' \in \mathrm{M}(k)\text{-PRIORITY}$. If A' has at least the same profit as A then on input S schedules $J \in \{J_{i_1}, \ldots, J_{i_{k+2}}\}$, J_1 and J_2 (thus rejecting J_3). Say that $U \in \{J_1, J_2\}$ is the one scheduled last (among J_1, J_2). On the set $S' = S \backslash \{U\}$, A' schedules the remaining one from $\{J_1, J_2\}$. Thus, J_3 is rejected. In the run of A' on S', there exists at least one interval $J' \in \{J_{i_1}, J_{i_2}, \ldots, J_{i_{k+2}}\}$ rejected and not stored in memory. Since A' has memory for only k rejected jobs it schedules the same jobs on S' and on $S'' = S' \backslash \{J'\}$. Thus, $\mathit{profit}(A'(S'')) = 3 < 4 = \mathit{profit}(A(S''))$.

We show $D(\mathrm{M}(k)\text{-PRIORITY}, \mathrm{M}(k+1)\text{-G-PRIORITY})$ using a similar argument. The algorithm A is modified so as to reorder jobs (giving J_3 low Priority) instead of rejecting J_3. □

Bounded memory with updates and bits.

We consider a modification of the bounded memory model for priority algorithms for the Job Scheduling problem.

Definition 3. Let C be a class of priority algorithms with memory. We write $\mathrm{M}_b(k) - C$ to denote the class of priority algorithms C where an algorithm from this class has k memory bits. In every round an algorithm from $\mathrm{M}_b(k) - C$ can update any number of its memory bits. We assume that initially all memory bits are zero.

The same theorem, as in the irrevocable storage memory management model, holds also for this memory model.

Theorem 14. *We consider two different settings*. (i) *Interval Scheduling*, *where the machine environment consists of one processor and C stands for any one among* {FIXED-PRIORITY, GREEDY-PRIORITY, PRIORITY}. (ii) *Job Scheduling where we have a machine environment of* $m \geqslant 1$ *identical processors*, *or Interval Scheduling and we have a machine environment of* $m \geqslant 2$ *identical processors and C stands for one among* {FIXED-PRIORITY, GREEDY-PRIORITY, GREEDY-FIXED-PRIORITY, PRIORITY}. *Given* (i) *or* (ii) *we have that* $M_b(k+1) - C > M_b(k) - C$.

We show this theorem by modifying the job sets used in the proofs of Lemmas 11–13. We present the proof only in the case of the fixed-priority D-relation. The others follow in the obvious way (by modifying appropriately the proofs of Lemmas 11–13).

Lemma 15. $D(\mathrm{M}_b(k)\text{-PRIORITY}, \mathrm{M}_b(k+1)\text{-FIXED-PRIORITY})$

Proof. We prove even for proportional profit Interval Scheduling on one machine environment. Fix $k \in \mathbb{Z}^{\geqslant 0}$. Let $S_1 = \{J_0, J_1, \ldots, J_{2^{k+1}-1}\}$, $S_2 = \{J', J'', J\}$ and $S = S_1 \cup S_2$, where $J_0 = (0, 2)$, $J_1 = J_2 = \cdots = J_{2^{k+1}-1} = (0, 1)$, $J' = (2, 4)$, $J'' = (4, 6)$, $J = (3, 6)$. We will only consider inputs where J_0 is present. Consider the following algorithm $A \in \mathrm{M}_b(k+1)$-FIXED-PRIORITY with ordering $J_0 > J_1 > J_2 > \cdots > J_{2^{k+1}-1} > \mathbf{J} > J' > J''$:

- Accept J_0 (and reject other intervals from S_1). Use the $k+1$ memory bits to store the value of a counter. When rejecting an interval from S_1, if the decimal value of the counter is less than $2^{k+1}-1$ then increase the counter by 1; otherwise leave the counter at $2^{k+1}-1$.
- After having read everything from S_1, if the counter is $2^{k+1}-1$ then reject J (if present) and schedule the rest greedily. Else, schedule greedily.

Fix an arbitrary algorithm $A' \in$ PRIORITY. Note that if A' has at least the same profit as A then it accepts J_0 (and it rejects every other interval from S_1). It is easy to see that if A' gains at least the same profit as A then J_0 must be read before every other job from S_1. Suppose that the first interval A' reads is J_i, $i > 0$. Clearly, A' has to accept this interval. Therefore, on input $\{J_i, J_0\}$ A' gains less profit than A. Now, we will show that if A' has at least the same profit as A on every input then: if the input contains every interval from S_1 then A' must read every interval from $S_1 \backslash \{J_0\}$ before reading an interval from S_2. Fix an input S' where $S_1 \subseteq S'$. If A' reads a proper subset $S'_1 \subsetneq (S_1 \backslash \{J_0\})$ before reading an interval from S_2 then we have the following. Consider the first interval A' reads from S_2. Clearly, A' has to accept this interval. If the first interval A' reads from S_2 is J' (or J'') then on input $S'_1 \cup \{J, J'\}$ (or $S'_1 \cup \{J, J''\}$) A' gains less profit than A. Else, the first interval it reads from S_2 is J and on input $S_1 \cup S_2$ we have that A' gains less profit than A. Therefore, if the input contains S_1 and A' has at least the same profit as A on every input then A' has read at least every interval from $S_1 \backslash \{J_0\}$ before reading an interval from S_2.

Furthermore, let us now assume $A' \in \mathrm{M}_b(k)$-PRIORITY. We define the memory state of A' after a round to be the content of its k memory bits. Consider a run of A' on S. Since, A' has k memory bits and the intervals in $S_1 \backslash \{J_0\}$ are $2^{k+1}-1$, by the pigeonhole principle a memory state has to repeat when A' is reading intervals from $S_1 \backslash \{J_0\}$. Say that between the two occurrences of the same memory state A' has read the intervals from $\hat{S} \subseteq S_1$. If A' has at least the same profit as A then it schedules J_1 and J_2 from S_2. Say that $U \in \{J_1, J_2\}$ is the one read last among $\{J_1, J_2\}$. Therefore, on input $S' = (S_1 \backslash \hat{S}) \cup \{U, J\}$ we have that $\mathit{profit}(A'(S')) = 4 < 5 = \mathit{profit}(A(S'))$. □

Acknowledgements

I am thankful to Charles Rackoff for supervising this research and for the valuable suggestions on drafts of this paper. I also want to thank Allan Borodin for the useful discussions and suggestions.

References

[1] S. Angelopoulos, Randomized priority algorithms, in: Proc. First Workshop on Approximation and Online Algorithms (WAOA), 2003.
[2] S. Angelopoulos, A. Borodin, On the power of priority algorithms for the facility location and set cover, in: Proc. Fifth Workshop on Approximation Algorithms for Combinatorial Optimization (APPROX), 2002, pp. 26–39 (to appear in a special issue of Algorithmica dedicated to papers from APPROX 2002).
[3] A. Borodin, M.N. Nielsen, C. Rackoff, (Incremental) priority algorithms, in: Proc. 13th Annu. Symp. Discrete Algorithms (SODA), January 2002, pp. 752–761 (also in Algorithmica 37(4) (2003) 295–326).
[4] S. Davis, R. Impagliazzo, Models of greedy algorithms for graph problems, in: Proc. 15th Symp. Discrete Algorithms (SODA), 2004.
[5] R.L. Graham, E.L. Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, Optimization and approximation in deterministic sequencing and scheduling: a survey, Ann. Discrete Math. 5 (1979) 287–326.
[6] P.A. Papakonstantinou, Hierarchies and complexity results for priority algorithms, Master Thesis, Computer Science Department, University of Toronto, January 2004.
[7] O. Regev, Priority algorithms for makespan minimization in the subset model, Inform. Process. Lett. 84 (3) (2003) 153–157.