Hierarchies and Complexity Results for Priority Algorithms

by

Periklis A. Papakonstantinou

A thesis submitted in conformity with the requirements
for the degree of Master of Science
Graduate Department of Computer Science
University of Toronto

Abstract

Hierarchies and Complexity Results for Priority Algorithms

Periklis A. Papakonstantinou
Master of Science
Graduate Department of Computer Science
University of Toronto
2004

Priority Algorithms is a model of computation that generalizes on-line computation, attempting to formulate the notion of greedy algorithm. We study questions concerning Priority Algorithms for variants of Job Scheduling. In the first part of the thesis we separate the class of adaptive from the class of greedy and adaptive priority algorithms, which was an open question [4]. We also compare the power of restricted classes of priority algorithms defined for the Job Scheduling and we define a memory hierarchy and show that it is robust. The second part studies questions, where given a finite set of jobs, we want to decide whether a given priority algorithm is optimal, or whether there exists, an optimal priority algorithm. For different settings of these questions we derive containment and hardness results for several complexity classes. Finally, we study normal forms and properties of optimal priority algorithms.

Remark

This is an extended version of the MSc thesis submitted to the Department of Computer Science, University of Toronto (January, 2004).
There are also some results (related but) not included in the thesis. Every document related to this thesis is available at the following URL:
http://www.cs.toronto.edu/∼papakons/MSc/

To my beloved Maria

Acknowledgements

My research supervisor Charles Rackoff for teaching me and for the countless discussions.

My second reader, Allan Borodin, for his useful suggestions and remarks. Stephen Cook for his remarks on the complexity of languages related to this thesis, and my office-mate Mark Braverman for the valuable discussions and for being the proof-killer for some of my most ambitious theorems.

My office-mates Aurélie Bataille and Daniel Ivan for their friendship here in the frozen Toronto. My parents and my friends Panayiota Bay, Jenny Bay and Yiorgos Peristeris for supporting me all these years.

Contents

Chapter 1

Definitions and Related work

This chapter discusses some of the previous work done in fields related to this thesis. This work goes in two orthogonal directions of study and we provide previous work, preliminary knowledge and definitions for both of them. Section 1.1 gives some motivation for the main topic of this thesis. Sections 1.2 and 1.3 provide the definitions for the Job Scheduling problem and for the model of Priority Algorithms. In section 1.4 we give an outline of our contribution. Section 1.5 discusses the previous work conducted in this field and in section 1.6 we give some definitions from complexity theory.

1.1 Motivation

Unless we have a proof that $P \neq NP$ it is not possible to state unconditional inapproximability results regarding the existence of *general* approximation algorithms for many common combinatorial properties. By *general* we mean that the model of computation is a polynomial time bounded Turing Machine. During the last twenty years many new proof techniques and powerful mathematical machinery have been developed in this direction. Recent advances in the Probabilistically Checkable Proof (PCP) systems combined with algebraic and combinatorial tools gave many promising results, stating lower bounds on the approximation algorithms. These are all conditional statements and base their validity on assumptions like $NP \neq P$.

In [4] (also in [5]) a formulation of the greedy and "greedy-like" computation model is presented, and with respect to this we would be able to prove unconditional statements for lower bounds on the approximation ratio of Priority Algorithms. This approach comes under the title *"Priority Algorithms"* and it is in essence a generalization of on-line

computation.

1.2 Job and Interval Scheduling

Scheduling theory is a rich field of study which dates back even before the establishment of the foundations of complexity theory. Scheduling theory studies the Job Scheduling Problem, and many variants appeared in the literature [9, 19, 12]. In all scheduling models we are given a set of jobs $S = \{J_1, \ldots, J_n\}$, where the problem is to schedule them on one or more processors.

Definition 1.2.1.

- A *job* J is a vector $(id, r, d, p, w) \in \mathbb{Z}^{\geq 0}$, 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 $p \leq d - r$. The job descriptor is used to distinguish among jobs of the same r, d, p and w. We may omit id given that it is denoted by the job subscript (for example J_i) or it is clearly implied.

- An *interval* is a restricted form of job where $p = r - d$. That is, an interval J is well defined by $J = (id, r, d, w)$.

- In the case of *proportional profit* $p = w$. In this case a job is defined as (id, r, d, p) and an interval is defined as (id, r, d).

- Given two intervals $J_1 = (r_1, d_1, w_1)$ and $J_2 = (r_2, d_2, w_2)$, where $d_1 \geq d_2$, we say that they *overlap each other* if $d_2 > r_1$.

Definition 1.2.2.

- Let $S = \{J_1, J_2, \ldots, J_n\}$ be a finite set of jobs. A *feasible schedule* or simply a *schedule* is a set, where each elements is a job from S together with its starting time and the processor where it is scheduled. In a feasible schedule there are no two overlapping jobs, scheduled on the same processor. Notice that a *scheduled job* can be viewed as an interval.

- Let S' be the set of jobs that correspond to a feasible schedule of jobs from S. $profit : S' \rightarrow \mathbb{Z}$ is the cost function denoting the total profit of S', that is $profit(S') = \sum_{J_i \in S'} w_i$.
- For a finite set of jobs S, a schedule is *optimal* if its total profit is maximal over all feasible schedules.

In this thesis we are interested in non-preemptive Job Scheduling where the machine environment consists of identical processors. The optimization *Job Scheduling Problem* is given a finite set of jobs and a number of processors m, find a feasible schedule that maximizes the total profit ($P|r_j|\sum w_j\bar{U}_j$ in [9] notation). If we restrict to the case where we have intervals then we have the *Interval Scheduling Problem*. Naturally, the decision variation of this problem is where we are also given a bound B and we want to decide if there exists a feasible schedule with total profit greater than or equal to B.

1.3 Priority Algorithms for Job Scheduling

In this work we are studying problems related to priority algorithms for the Job Scheduling problem. We study the cases of Job and Interval Scheduling for one and more than one (identical) machine environments. In the Priority Algorithms model the decisions made are not necessarily computable. The main characteristics of the Priority Algorithm model (what makes this computation non-trivial - since the function is an arbitrary one) are: (i) the algorithm does *not* have access to the input set in advance and (ii) the computation proceeds in a round-fashion where in each round an input element is presented to the algorithm which then makes an *irrevocable* decision. The algorithm "interacts" with the input by specifying an ordering for a possibly infinite set and in each round the next, or highest priority, available element is presented. Note that in this sense the algorithm might be an infinite object. The irrevocable decision that the algorithm makes on a job, is realized as a pair of this job with the rejection decision or the job paired with the processor where is scheduled and its starting time. We distinguish between two types of Priority Algorithms. The *Fixed Priority* and the *Adaptive Priority* model. In the Fixed Priority model the algorithm orders the set of jobs S only at the beginning and then proceeds according to that order.

FIXED PRIORITY ALGORITHM

- Determine a total ordering π for the set of *all possible* input elements.
- On a finite set S repeat:
  - The next available input element, according to π, is presented to the algorithm.
  - Make an *irrevocable* decision about scheduling that input element.

  Until decisions are made for all data items on the finite input S.

Adaptive Priority Algorithms or simply *Priority Algorithms*, differ from the above mentioned model to the fact that in each round they can reorder the (remaining) data items.

(ADAPTIVE) PRIORITY ALGORITHM

On a finite input set repeat:

- Determine a total ordering π_i for the set of *all remaining possible* input elements.
- The next available input element, according to π_i, is presented to the algorithm.
- Make an *irrevocable* decision about scheduling that input element.

Until decisions are made for all data items on the finite set S.

Say that A is a priority algorithm (the function that realizes the priority algorithm) on an input S. We denote by $A(S)$ the set of scheduled jobs when the algorithm terminates.

It is obvious that the fixed priority model is a restriction of the general (adaptive) one. We can add two more natural types of restrictions on the priority model. We define the *greedy* property to be a restriction on the above mentioned model, where when a greedy priority algorithm reads a job that can be scheduled it necessarily schedules it.

We also define the *memoryless* restriction to refer to algorithms where every decision and the ordering is based only on the scheduled (not on the rejected jobs). More about the memoryless property can be found in 2.3.

We denote as PRIORITY the (unrestricted) class of adaptive priority algorithms. If the class is restricted to the class of fixed priority algorithms we add the prefix FIXED or F for short. If the algorithms are restricted to make greedy decisions only then we write as a prefix GREEDY (G) and if the algorithms are memoryless then this is denoted by a prefix MEMORYLESS (M). For example, G-F-PRIORITY denotes the class of fixed priority algorithms for the Job Scheduling problem with memory where the decisions are greedy, and M-PRIORITY denotes the class of memoryless (adaptive) priority algorithms.

1.4 Contribution

In this thesis we are dealing with models of Priority Algorithms for the Job Scheduling problem and its variants. Taking into account all possible combinations of the restrictions, for classes of priority algorithms, we have eight classes for priority algorithms. In chapter 2, we develop 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 *LPF* (Largest Processing Time first) algorithm is optimal w.r.t. its approximation ratio for one processor Interval Scheduling with proportional profits. Since, LPF $\in$ MEMORYLESS-GREEDY-FIXED-PRIORITY (the lowest level of the hierarchy) it is impossible to separate classes of priority algorithms, for this specific problem, based on the approximation ratio. We base our separation results on the weaker notion of not-feasible-simulation. We say that a class of priority algorithms C_2 *dominates* ($\geq$) a class of priority algorithms C_1 if for every algorithm in C_1 there exists an algorithm in C_2 with at least the same profit on every input. For example, PRIORITY dominates over GREEDY-PRIORITY, since GREEDY-PRIORITY is just a restriction of the general class. We say that C_2 *partially dominates* C_1 if for every algorithm in C_1 there exists an algorithm in C_2 such that there exists an input instance where the algorithm in C_2 has greater profit. Partial domination is the weakest notion that implies a "not-feasible-simulation notion".

One of the early stated open questions was whether greediness is a real restriction to the model [4]. We answer this question affirmatively by separating GREEDY-PRIORITY from PRIORITY even for the case of one processor Interval Scheduling with propor-

tional profits. Furthermore, chapter 2 contains a thorough comparison of every pair of classes of priority algorithms for the case of: (i) Job Scheduling, (ii) Interval Scheduling for one processor machine environments and (iii) Interval Scheduling on multiple processors. Hence, 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 lemmata. In the same chapter we define variations of the memoryless property and we also define classes of priority algorithms of bounded memory. We show that the variations correspond to algorithms of different power. Notice that there are two extremes. The first is where a priority algorithm makes decisions based on the sets of the so far scheduled and rejected jobs and the other is where the algorithm makes decisions based only on the set of the so far scheduled jobs (and the algorithm can remember all information about a job). In section 2.3 we see that the stronger case (fact 2.3.1) is the same as if the priority algorithm has a memory where it is allowed to store arbitrary data. Furthermore, in the same section, we define a hierarchy of classes of algorithms that have bounded memory. Thus, we define a hierarchy of classes of algorithms that exist between these two extremes. 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. Based on a variation of the bounded memory model we show that priority algorithms with k memory bits cannot be simulated (in general) by priority algorithms having $k-1$ memory bits.

Chapter 3 discusses the complexity of languages related to priority algorithms. Previous (to priority algorithms) formulations of greedy algorithms through connections to matroids, greedoids and matroid embeddings (e.g. [6, 10, 18, 14]) capture a restricted class of greedy algorithms, which are optimal in the exact sense. On the other hand, priority algorithms formulate a wide class of greedy algorithms which may not though be optimal (in the exact sense). We say that a priority algorithm is optimal on a given set of jobs, if it gains optimal profit on every subset of this set. Simple examples show that sets admitting optimal priority algorithms have associated subset systems which are of very different structure than matroids and greedoids. With respect to the priority algorithms model we would like to know the structure of inputs admitting (exact) optimal solutions. Furthermore, we would like to investigate properties of such optimal priority algorithms. We study two types of questions. The first type is given a finite set of jobs or intervals and the description of a priority algorithm, we want to decide whether the algorithm is optimal for this set. The second type of questions is given a finite set of jobs or intervals,

we want to decide whether there exists an optimal priority algorithm for this set.

This chapter contains two main results that show, for languages corresponding to the above mentioned questions, two problems to be in P. The first corresponds to the case of whether there exists an optimal greedy-fixed algorithm for the proportional profit Interval Scheduling problem on one processor. This problem has a natural (and elegant) representation through the ordering of the intervals of the input set. We combinatorially characterize such optimal algorithms and we use this characterization so as to show that the problem is in NC^2.

Perhaps the most interesting result concerns the existence of optimal priority algorithms for given sets of intervals (when scheduling in one machine environments). We derive combinatorial properties for sets of intervals that admit optimal priority algorithms. Perhaps surprisingly, we show that the problem is in NC^2 even for arbitrary (general) priority algorithms. It is also interesting to note that if an optimal priority algorithm exists (for a given set) then there also exists an optimal priority algorithm in MEMORYLESS-GREEDY-FIXED-PRIORITY.

The complexity of languages related to the existence of optimal priority algorithms (for a given finite set of jobs) is partially resolved. Obviously, the general problem is in $ESPACE$. We study the case where the sets contain jobs (instead of containing only intervals) and we derive normal forms and properties for optimal priority algorithms.

A more complete treatment of the subject would require the study of relevant languages when the algorithm cannot be simply described through a total ordering on the set of jobs. Regarding the optimality of a priority algorithm, we observe that when the algorithm can take decisions of where to schedule a job or interval then the problem is not well-defined unless we agree on specific computational restrictions on the algorithm. In this case we do not provide some results, since when the algorithm is implemented in an "efficient" (resource bounded) model of computation the problem has (i) trivial answers and (ii) it does not have an elegant definition.

In chapter 3 we also provide other related complexity results. For example, we show that deciding whether a greedy-fixed priority algorithm is optimal on a set of intervals, each of which of the same profit, is in uniform AC^0.

We also provide an NL-completeness result. Say that we are given a set of intervals, where each interval has profit a polynomial to the number of intervals in the set. The problem of determining whether there exists a schedule of at least a specific profit is NL-complete. This seems interesting since we have shown (as an intermediate step in

one of our proofs) that the same problem for arbitrary profits is in NC^2, whereas there does not seem to exist an obvious way that shows the problem to be in NL.

1.5 Related work

In this section we briefly review some complexity results for variants of Job Scheduling and we also discuss the previous work in the field of priority algorithms.

The scheduling theory is dated even before the existence of the foundations of complexity theory [8]. We review some complexity results for variations of the Job Scheduling problem. When the problem is restricted to inputs of intervals, each of which has the same profit, then the problem can be easily shown to be in P [3]. The well-known algorithms $SFTF$ (Shortest Finishing Time First) and $LSTF$ (Largest Starting Time First) are optimal on such sets. It also follows as a corollary that their priority algorithm analogs are also optimal. We note that these algorithms are in the most restricted class of priority algorithms, that is Memoryless-Greedy-Fixed-PRIORITY. For the case of intervals with arbitrary profits, using flows or Dynamic Programming, it can be shown that the problem is in P [3]. Even for the one-processor case, Interval Scheduling with proportional profits, it is shown [4] that no priority algorithm can achieve an approximation ratio better than 3. This intuitively means that no "greedy-like" algorithm can solve this problem in the exact sense, despite the fact that the problem is in P. If the given set contains jobs then determining the optimal schedule is NP-hard, even for one processor machine environments. The corresponding decision version is NP-complete in the strong sense. This problem is solvable within pseudopolynomial time if the numbers in the release and deadline times are bounded by a constant, but it remains NP-complete even when the deadline and the release times can only take two values. It can also be solved in polynomial time even if the release and the deadline times are arbitrary and there are precedence constraints, so long as all jobs are of equal length.

The rest of this section is devoted to the previous work in the field of priority algorithms.

Borodin, Nielsen and Rackoff [4] initiated the study on the field of Priority Algorithms. This work provides the basic definitions of the models of computation for the Priority Algorithms for the Job Scheduling, the Makespan problems and some of their variants. In this work there is an extensive intuitive discussion on the goals of this study, its applicability and arguments on what classes of greedy and greedy-like algorithms this

model captures. There is a discussion of the connection of Priority Algorithms with on-line computation and "greedy" approximation algorithms that have appeared in the field of approximating NP-hard optimization problems. It appears that priority algorithms cannot even solve, in the exact sense, Job Scheduling optimization problems that are known to be in FP; where FP stands for the function analog of P. For example it is known that even for arbitrary profits the Interval Scheduling on multiple machines can be solved optimally in polynomial time using flows [3]. The focus on this work is on Job Scheduling and Makespan problems. Results from on-line computation and scheduling theory are ported in the context of priority algorithms. The authors present lower bounds on the approximation ratio for classes of priority algorithms and some of them are shown to be tight. They also ask questions of what happens if the algorithm knows in advance the length of the input or other "global" parameters for the input set. In the same work there is also a first attempt to define randomized classes of Priority Algorithms. In chapter 2 of this thesis we make an extensive use of a powerful simulation lemma from [4] which proves that Memoryless-PRIORITY $\approx$ Memoryless-Greedy-PRIORITY. In general, we consider this as a seminal paper in the field of complexity theory, where unconditional lower bounds are derived through the formalization of specific algorithmic paradigms.

In [4] the Makespan problem is considered on multiple homogeneous machine environments. In the same work there is also consideration of the subset model for the same problem, where every job can be scheduled on specified subsets of machines. Continuing the work of [4], Regev in [20] showed a $\Omega\left(\frac{\log m}{\log \log m}\right)$ lower bound on the approximation ratio of any Fixed-Priority Algorithm, where m is the number of machines.

Angelopoulos and Borodin [7] define and study applications on the model of Priority Algorithms for two well-known NP-hard optimization problems, Set Cover and the (uncapacitated) Facility Location. Since, (in the intuitive sense) greedy approximation algorithms have been proposed for uncapacitated facility location and set cover, it is natural to study whether unconditional lower bounds can be derived for these problems, given that the computational model captures the notion of greediness. Furthermore, the authors argue that many primal-dual algorithms, which have an alternative combinatorial statement, can be seen as Priority Algorithms. The authors also study closely the relations of priority algorithms with proposed approximation algorithms for these problems and port some of them in the priority algorithms area. It is shown that no priority algorithm for the uniform metric facility location problem can achieve an approximation

ratio better than $\frac{4}{3}$, where for the general (not necessarily metric) case a lower bound of $\Omega(\log n)$ is shown, where n is the number of facilities. Porting a well-known approximation algorithm, a tight bound $\ln n - \ln \ln n + \Theta(1)$ on the approximation ratio is proved for the adaptive-greedy class (under the specialized model definition) of algorithms. The same work extends its study for the case of memoryless and fixed priority algorithms.

A more thorough treatment of randomized classes for priority algorithms is presented by Angelopoulos in [1]. This work is motivated by similar studies on the field of on-line computation. The author distinguishes between the ways that someone can introduce randomization in a priority algorithm. One is where the algorithm chooses a random ordering of the data items and the other is where the algorithm makes random decisions on each data item. When randomization is applied both in the orderings and the decisions we have the case of a fully randomized algorithm. This paper shows lower bounds for the metric facility location and the makespan scheduling problems, under this randomized setting. Specifically, for the metric facility location problem, it shows that no fully-randomized priority algorithm is better than a $\frac{4}{3}$ approximation and for the case of the randomized ordering it is shown that a 1.5 lower bound holds for the greedy-fixed priority algorithms. This work provides different lower bounds for alternative representations of the input. For the makespan scheduling problem on identical machines it is shown that no priority algorithm achieves an approximation ratio better than $\frac{12}{11}$ and when the algorithm is fixed-priority with randomized decisions (for environments with at least three machines) the lower bound on the approximation ratio is $\frac{10}{9}$.

An extensive study of various priority algorithms for graph problems is in [7], by Impagliazzo and Davis. This work studies the approximation power of priority algorithms for the Shortest Path, Steiner Tree, Weighted Vertex Cover and Independent Set problems. The authors describe ("define") an abstract model for priority algorithms and they provide a definition for every problem under question. They also describe an abstract way of capturing the notion of memoryless algorithms and they present a general lower bounding technique in terms of a combinatorial game. Results of this work is that no fixed priority algorithm can solve the Shortest Path problem with any (constant) approximation ratio (in contrast with the Dijkstra's algorithm). For the case of the Weighted Vertex Cover there is a lower bound of 2 on the approximation ratio of a (general) priority algorithm, where for the case of Metric Steiner Tree the approximation ratio lower bound is $\frac{13}{11}$. For the Maximum Independent Set the presented lower bound is $\frac{3}{2}$ even when the graph is of degree at most 3. The authors use their definitions of the memory-

less algorithms to prove that no memoryless adaptive priority algorithm can achieve an approximation ratio better than 2 for the Weighted Independent Set problem on cycles with weights 1 and k and in the same time they give an upper bound by presenting a $\left(1+\frac{2}{k-1}\right)$-approximation adaptive priority algorithm for the same problem. This shows a separation between the memoryless and with memory classes of priority algorithms for this problem. The second chapter of this thesis discusses analogous questions for the Job Scheduling case, where the proofs are based on different concepts than those used in [7]. In this work there is also a straightforward fact derived from the definition of memoryless priority algorithms.

Proposition 1.5.1. *Memoryless algorithms can define a new ordering on the possible input elements only after casting an accepting decision.*

In [1, 2] there is a discussion about different representations of the problems considered in these works. For different representations of the input elements the lower bounds on the approximation ratios are also different. A similar remark is also made in [7]. We also note that all the different representations of the inputs, presented in these works, are polytime equivalent. The sensitivity in the way we represent the input elements is a consequence of both the fact that the algorithm does not have access to the whole input and that it makes irrevocable decisions. Thus it seems that the Priority Algorithms model is not very robust. From a complexity theoretic perspective it seems that the whole study can be confined to the question whether $\rho > 1$. In order to achieve this task we must provide a universal restriction to the representational size of the input elements (for example each data element should be of length $O(\log n)$, where n is the length of the input) and argue about the possible (polytime bijective mappings) between different representations of the input.

The separation between classes of graph theoretic problems, appeared in [7], is based on a separation based on the approximation ratio achievable by algorithms belonging to different classes. This separation introduces a very strong notion, not applicable in this thesis. For example, for the one machine environment Interval Scheduling problem (with proportional profits) the lower bound on the approximation ratio is 3, for the class PRIORITY. This bound is tight since LPF $\in$ Memoryless-Greedy-Fixed-PRIORITY has approximation ratio 3. Notice that an algorithm at the "lowest level of the hierarchy" achieves the lower bound of the stronger class. Thus it is impossible to base a separation on similar arguments. Our approach, presented in the second chapter, is based on the

definitions of some relations that capture the notion of "not-feasible-simulation".

1.6 Definitions and Preliminaries from Complexity Theory

The definitions and the preliminary results of this section can be found in almost any complexity theory textbook, such as [17].

1.6.1 Class characterizations by Turing Machines

Definition 1.6.1. Let $t, s : \mathbb{Z}^{\geq 0} \to \mathbb{Z}^{\geq 0}$ and Σ a finite alphabet.

- $TIME(t(n))$ is the class of all languages $L \subseteq \Sigma^*$ decided by a $O(t(n))$ time bounded Turing Machine.
- $NTIME(t(n))$ is the class of all languages $L \subseteq \Sigma^*$ decided by a $O(t(n))$ time bounded Non-deterministic Turing Machine.
- $SPACE(s(n))$ is the class of all languages $L \subseteq \Sigma^*$ decided by a $O(s(n))$ space bounded Turing Machine.
- $NSPACE(s(n))$ is the class of all languages $L \subseteq \Sigma^*$ decided by a $O(s(n))$ space bounded Non-deterministic Turing Machine.

Definition 1.6.2.

- $P = \bigcup_k TIME(n^k)$
- $NP = \bigcup_k NTIME(n^k)$
- $L = SPACE(log(n))$
- $NL = NSPACE(log(n))$
- $ESPACE = SPACE(2^{O(n)})$

1.6.2 Circuit complexity classes and preliminary results

We provide some definitions from circuit complexity. Our notation is compatible with the one used in [21]. The definitions and lemmata from Descriptive Complexity can be found in [11].

Definition 1.6.3. Let B be a basis, and let $s, d : \mathbb{Z}^{\geq 0} \to \mathbb{Z}^{\geq 0}$. We define the following complexity classes.

- $SIZE - DEPTH_B(s, d)$ is the class of all sets $A \subseteq \{0,1\}^*$ for which there is a circuit family C over basis B of size $O(s)$ and depth $O(d)$ that accepts A.
- $FSIZE - DEPTH_B(s, d)$ is the class of all functions $f : \{0,1\} \to \{0,1\}^*$ for which there is a circuit family C over basis B of size $O(s)$ and depth $O(d)$ that computes f.

In this thesis we are mainly concerned with circuits over the *standard* AND, OR, NOT basis. Whenever we omit B then this means that we have the standard (bounded fan-in) basis. When we refer to circuits of unbounded fan-in we mean that AND and OR have unbounded fan-in. In this case we may omit B and just write $UnbSIZE - DEPTH(s, d)$ or $UnbFSIZE - DEPTH(s, d)$. Say that $\mathtt{F}$ is a class of functions and $SIZE(\mathtt{F}) = \cup_{f \in \mathtt{F}} SIZE(f)$.

Definition 1.6.4. For $i \geq 0$, define

$$NC^i = SIZE{-}DEPTH\left(n^{O(1)}, (log(n))^i\right) \text{ and } AC^i = UnbSIZE{-}DEPTH\left(n^{O(1)}, (log(n))^i\right)$$

Also

$$NC = \bigcup_{i \geq 0} NC^i \quad and \quad AC = \bigcup_{i \geq 0} AC^i$$

As far as it concerns the uniformity conditions we use logspace uniformity for every NC^i, AC^i where $i \geq 1$. For the case of the AC^0 we use $DLOGTIME$ uniformity. We refer to the uniform classes (that concern this thesis) as $U_L - NC^2$ and as $U_D - AC^0$. Especially for the case of $U_D - AC^0$ we use an equivalent logical characterization, where $U_D - AC^0 = FO[<, bit]$. The following paragraph is a brief introduction to $FO[<, bit]$. Details can be found in the excellent textbook written by Immerman [11].

$FO[<, bit]$ is the set of languages that can be expressed through first-order formulae. We fix a specific structure for these formulae which has the standard interpretation of

$<$ (over the integers), a predicate B, where $B(i)$ is true iff the i-th bit of the input is 1, a predicate $bit \subseteq \mathbb{N} \times \mathbb{N}$, $bit(n, i)$ is true iff the i-th bit of the binary representation of integer n is 1 and a finite universe which contains the integers from 1 to n where n is the length of the string. It can be shown that in this structure we can also define the predicates $LENGTH$, $PLUS$, $MULTIPLY$, $EQUAL$, $LESS-OR-EQUAL$, where $LENGTH(i)$ is true if the input length is i and the rest of the predicates have the obvious meaning. Note that these are *predicates* that have nothing to do with the *problems* of definition 1.6.5.

Definition 1.6.5.

- – Problem: ADD (addition)
  - – Input: Two numbers in binary of n bits each
  - – Output: the sum of the input numbers in binary
- – Problem: MAX
  - – Input: n numbers in binary of n bits each
  - – Output: the maximum of these numbers in binary
- – Problem: EQUAL
  - – Input: two n bit numbers in binary
  - – Question: are the two numbers equal?
- – Problem: LESS
  - – Input: two n bit numbers a and b in binary
  - – Question: is a smaller than b?

Below we provide some well-known results from the circuit complexity that can be found in, almost, every text or survey of the field.

Lemma 1.6.1. $U_L - NC^1 \subseteq L \subseteq NL \subseteq U_L - NC^2 \subseteq U_L - NC \subseteq P$.

Lemma 1.6.2.

- $ADD \in FSIZE - DEPTH\left(n^{O(1)}, \log(n)\right)$
- $MAX \in FSIZE - DEPTH\left(n^{O(1)}, \log(n)\right)$

- $EQUAL \in U_D - AC^0$.
- $LESS \in U_D - AC^0$.

Chapter 2

Separating Classes of Priority Algorithms

This chapter studies the relations among classes of Priority Algorithms. Whenever we mention the term "priority algorithm" we refer to a priority algorithm for the Job Scheduling Problem or to restrictions of Job Scheduling, such as Interval Scheduling. We compare every two classes of Priority Algorithms for the Job Scheduling problem. In this context the term class corresponds to a set of algorithms sharing the same constraints on their power (see section 1.3). In section 2.2.3 we give tables for the comparison of any two classes of priority algorithms for the possible settings of the Job Scheduling problem. In section 2.3 we define natural variations of the memoryless property and we show that these variations result computational models that differ in power. Recall that the memoryless property (as defined in the first chapter) corresponds to Priority Algorithms where the function that decides whether to schedule a job, does not depend on the jobs rejected so far. We define machines that have bounded memory. The main theorem of this section is that any two classes that differ in at least one memory cell (that is, one of the classes has memory for one more rejected job than the other class) are different.

2.1 Definitions

In [4] a lower bound of 3 is presented on the approximation ratio of any priority algorithm $A \in$ PRIORITY for the Interval Scheduling problem with proportional profits and a machine environment of $m = 1$ machine. It has also been shown that this bound is tight, since the LPT (Longest Processing Time first) priority algorithm has approxima-

tion ratio 3. The fact that LPT $\in$ Memoryless-Greedy-Fixed-PRIORITY makes it impossible to separate classes of priority algorithms for the Interval Scheduling, proportional profit and one processor problem, based on the approximation ratio. Thus, for the case of this problem and other variations of Job Scheduling we adopt an approach that shows separation by proving the existence of a priority algorithm in a class such that every algorithm in another class cannot achieve at least the same profit (as the previously mentioned algorithm) on every input. That is, we show that it is not feasible to do a simulation with at least the same profit. We give some definitions regarding the notation for comparing classes of priority algorithms.

Definition 2.1.1.

- 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 $\mathcal{S}$ such that $profit(A(S)) > profit(A'(S))$.

- If for every $A \in C_1$ there exists an $A' \in C_2$, such that for every input S, $profit(A(S)) \leq profit(A'(S))$ then we write $C_1 \leq C_2$. Note that $C_1 \leq C_2$ means that C_1 can be simulated by C_2, that is $\neg D(C_2, C_1)$ holds.

- If $C_1 \leq C_2$ and $C_2 \leq C_1$ then we write $C_1 \approx C_2$.

- If $D(C_1, C_2)$ and $C_1 \leq C_2$ 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 $C_1 \perp C_2$.

One thing to notice is that writing the relation $D(C_1, C_2)$ in infix notation "$C_1\, D\, C_2$", where C_1 and C_2 are classes of priority algorithms, then we read "C_1 *is dominated by* C_2" (and not the opposite).

It is easy to see that the relation D is not transitive, whereas the relations $\leq$ and $<$ are. It is obvious that the relation $<$ defines a partial order since it is also anti-symmetric. It is also obvious that $\perp$ and $\approx$ are symmetric.

We make extensive use of the proposition below, directly derived from the definitions.

Proposition 2.1.1. *Let C_1, C_2, C_1', C_2' be classes of priority algorithms. If $C_1' \leq C_1$ and $C_2 \leq C_2'$ and $D(C_1, C_2)$ then $D(C_1', C_2')$.*

We also make use of all the other directly derived corollaries of proposition 2.1.1. For example, lemma 2.2.6 states that D(M-PRIORITY,G-F-PRIORITY) which implies that D(M-G-F-PRIORITY,G-F-PRIORITY), since M-G-F-PRIORITY $\leq$ M-PRIORITY. One more obvious proposition is the following.

Proposition 2.1.2. *If C and C' are classes of priority algorithms then there exists a unique $R \in \{<, >, \approx, \perp\}$, such that $C \ R \ C'$ (where $>$ has the obvious definition).*

We have studied all the possible relations $<$ and $\perp$ among classes of priority algorithms, in the sense that all the others trivially follow from the transitivity of $<$. The main results of this chapter are: (i) G-PRIORITY $<$ PRIORITY and (ii) M-F-PRIORITY $<$ M-PRIORITY $<$ G-PRIORITY.

From the paper of Borodin, Nielsen and Rackoff [4] it is known:

Lemma 2.1.3. M-G-PRIORITY $\approx$ M-PRIORITY.

This result signifies that in case of a memoryless algorithm, greediness[1] is not a real restriction. This result, as well as those that trivially follow from it, are the only ones where greediness is not a restriction. The proof of this lemma is based on a (general) simulation argument. Hence, this lemma holds for every variation of the Job Scheduling problem considered here, such as the Interval Scheduling and for restrictions on the number of processors.

Also note that in the case of one machine environments Interval Scheduling G-F-PRIORITY $\approx$ M-G-F-PRIORITY (lemma 2.2.7)since in the case of G-F-PRIORITY for one machine environment an algorithm has no decision (thus, memory cannot help).

For the rest of this chapter, memoryless and adaptive priority algorithms determine the job ordering by having access to all the information of a scheduled job (including its identifier) and on which processor it has been scheduled.

[1] We use the term "greedy" so as to be consistent with the previous works. What greediness stands for is that all the accepting decisions are made first.

2.2 Relations between classes of priority algorithms

2.2.1 Partially dominated classes of priority algorithms - Main lemmata

This sub-section concerns the main contribution of this chapter. The sub-section is divided into two parts. The first part contains the "partial domination" results for the case of one machine Interval Scheduling. The second contains lemmata that differentiate among the cases of (i) Interval Scheduling on one machine, (ii) Interval Scheduling on more than one machine and (iii) Job Scheduling. D is a two-place predicate. Note that all the D-lemmata cannot be strengthen in the sense that removing a constraint from the first place of the predicate or adding a constraint to the second place makes the relation not to hold.

Partially dominated classes of priority algorithms for one processor Interval Scheduling

In this part we present results concerning partially dominated classes of priority algorithms, where we have the case of Interval Scheduling with proportional profit on one machine (processor).

Remark 2.2.1. It is important to notice that the results for the partially dominated classes, derived for the one processor case, can easily be extended in the case of any (constant) number of processors m. In this chapter every proof that corresponds to a partially dominated result, is constructive by giving an example of an algorithm and a (specific) finite set of inputs. These results can be extended to the case of multiple processors, by adding $m-1$ jobs where each of them overlaps with every other job in the input set considered in the initial proof. For all the results that follow we observe that the simulation arguments hold for the arbitrary profit case whereas for the separation D-relations it suffice to consider the proportional profit case; thus, making our results stronger. Moreover, note that the results hold for every input size. We can pad the separation examples with any number of non-overlapping intervals and the proofs remain the same but minor technical details.

Remark 2.2.2. The following lemmata hold in case where a class of priority algorithms is "partially dominated" by a priority algorithm from another class even when this "domination" happens for only one input instance. A small technical modification shows that

w.r.t. these lemmata we can show a stronger argument where the "partial domination" holds for an infinite number of input instances. In the proofs of the following lemmata we just add any number of intervals not overlapping any other interval or job in the input set.

When you read the proofs: We use the terms "accept a job" and "schedule a job" interchangeably. In what follows the algorithms do *not* have a full description. We only provide the part of the algorithm that it is of interest for our proof. Whenever, we have an adaptive algorithm and we do not describe what happens to the determined order of the jobs, then we implicitly mean that in the subsequent steps the order remains the same. Whenever we determine a new ordering for the jobs then we do so only for the jobs of our interest w.r.t. the proof. Finally, for two distinct jobs $J_i = (i, r, d, p, w), J_j = (j, r, d, p, w)$, where i and j are not equal, we may abuse the notation by writing $J_i = J_j$, meaning that the release, the deadline, the processing time and the profit (weight) of the two jobs are all pairwise equal, but the identifiers differ.

One of the main characteristics of the Priority Algorithms model is the limited knowledge of the future input elements (except what is implied by ordering information). All the separation results of this document are based on a "promise" that given that specific jobs have already been read, from the input, then specific subsets of the remaining jobs will follow. The separation arises from the fact that the "stronger" class of algorithms can identify this "promise" (for example, if it has memory) where the "weaker" cannot.

***Gadgets* used throughout the constructions of the proofs:** All the D-relation proofs presented in this chapter are constructive. Specifically, in order to prove $D(C_1, C_2)$ we give an algorithm in C_2, and a specific 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. We borrow the computational complexity term "gadget" used in the proofs of the hardness results to represent a similar but different notion of a repeated pattern.

- *"Dummy jobs" and "switches".* A "dummy job" and a "switching-gadget" do not have exactly the same concept. A "dummy job", is a job (or interval) of negligible profit with respect to the total profit of the inputs under consideration. Due to the inherent lack of knowledge about the whole input, the presence or not of a "dummy job" gives the algorithm a way of *a priori* knowing something about the rest of the input jobs. For example, such a knowledge might give advantage to an adaptive algorithm over all fixed priority algorithms. The reason is that an

adaptive algorithm can use this knowledge to reorder the (initially) ordered jobs. On the other hand, the "switches" are mainly used to give advantage to algorithms with memory over algorithms without memory. Usually, the "switching gadget" is a job or a set of jobs where their rejection or non-presence signals something about the rest of the input set. Notice that a memoryless algorithm cannot distinguish between the absence from the input, or presence and rejection of a job. In most of the proofs where "switching gadgets" are used, we see a set of identical (apart from their identifier) jobs that help the algorithm with memory to distinguish among different states (of non-presence or rejection of the jobs) and therefore to gain a specific knowledge on the input set. Intuitively, we use the switching gadget so as to implement bits of memory.

- *"Blockers"*. These are jobs used in constructions where the algorithm has to schedule the job by choosing its starting time or when the algorithm has to choose on which processor to schedule a job/interval. The structure of the input set will be such that scheduling "blockers" prevents other jobs from being scheduled. Such gadgets are used in the proofs of the lemmata 2.2.5, 2.2.6 and of the theorem 2.3.5. The "blocking gadgets" are used together with other gadgets that provide knowledge on the input set, such as "dummy jobs" or "switches". The "blocking gadgets" help a fixed priority algorithm to overcome part of its fixed structure, or help a greedy algorithm to overcome part of its greediness.

The main contribution of this chapter are the following lemmata of partially dominated classes. All proofs only use jobs that are intervals with proportional profit.

Lemma 2.2.1. 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 a "dummy 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).

Remark 2.2.3. Remark: Note that A is not optimal on every subset of S. This is a main common characteristic of every lemma of this sub-section. In chapter 3 we show that

classes of priority algorithms cannot be separated w.r.t. optimal priority algorithms; that is, priority algorithms that are optimal on every subset of an input set. This is due to the fact that if such an optimal algorithm exists then there also exists an optimal algorithm from M-G-F-PRIORITY.

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 $\{J_1, J_3, J_4\}$) $profit(A'(S')) \leq 3 < 4 = profit(A(S'))$. Else if J_4 precedes J_2 and J_3 then on input $S' = \{J_2, J_3, J_4\}$ $profit(A'(S')) \leq 3 < 4 = profit(A(S))$. □

Lemma 2.2.1 and the following lemma show that the classes M-G-PRIORITY and F-PRIORITY are essentially incomparable.

Lemma 2.2.2. D(GREEDY-PRIORITY, FIXED-PRIORITY)

Proof. Let $S = \{J_1, J_2, J_3, J_4\}$, $J_2 = (0, 2), J_3 = (2, 4), J_4 = (1, 4)$ and $J_1 = (2 - \epsilon, 2 + \epsilon)$, for a fixed $\epsilon < 1$ (J_1 is a "dummy job"). Consider the following algorithm $A \in$ FIXED-PRIORITY, with ordering $J_1 > \mathbf{J_4} > J_2 > J_3$:

- If J_1 is read from the input then reject it, and schedule greedily (schedule as many intervals as possible).
- 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.

Let $A' \in$ G-PRIORITY. If J_1 is the first interval in the initial ordering then on input $S' = \{J_1, J_2\}$, $profit(A'(S')) = 2\epsilon < 2 = profit(A(S'))$. If J_2 is the first interval in the initial ordering then on input $S' = \{J_2, J_4\}$, $profit(A'(S')) = 2 < 3 = profit(A(S'))$. Similarly for J_3. If the first interval is J_4 then on input S, $profit(A'(S)) = 3 < 4 = profit(A(S))$. □

Lemma 2.2.3. D(GREEDY-FIXED-PRIORITY, MEMORYLESS-FIXED-PRIORITY)

Proof. Let $S = \{J_1, J_2, J_3, J_4\}$, where $J_1 = (0, 1), J_2 = (1, 3), J_3 = (3, 5), J_4 = (2, 5)$. Intuitively, J_1 is a "dummy" job. Consider the following algorithm $A \in$ M-F-PRIORITY, with fixed ordering $J_1 > J_4 > J_2 > J_3$:

- If J_1 is present then schedule it, reject J_4 (if present) and schedule the rest greedily.
- Else schedule greedily.

Let $A' \in$ G-F-PRIORITY. If in the ordering of A', J_2 or J_3 precedes J_4 then on input $S' = \{J_2, J_4\}$ (or on $\{J_3, J_4\}$) $profit(A'(S')) = 2 < 3 = profit(A(S'))$. Else if J_4 precedes J_2 and J_3 then on input S $profit(A'(S)) = 4 < 5 = profit(A(S))$. □

The comparison of PRIORITY and M-PRIORITY follows as a corollary of the following lemma concerning the separation of G-PRIORITY and M-G-PRIORITY. Recall that M-G-PRIORITY ≈ M-PRIORITY.

Lemma 2.2.4. D(MEMORYLESS-GREEDY-PRIORITY,GREEDY-PRIORITY)

Proof. 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 $J_3 = (1, 3), J_4 = (3, 5), J_5 = (2, 5)$. Let $A \in$ G-PRIORITY, with initial ordering $J_1 > J_2 > J_3 > J_4 > J_5$. A has the following description:

- If J_1 and J_2 are both present then determine the ordering $J_5 > J_3 > J_4$.
- Else schedule greedily keeping the initially determined ordering.

Let $A' \in$ MEMORYLESS-GREEDY-PRIORITY. If J_5 is the first interval in the initial ordering of A', then on input $S' = \{J_1, J_3, J_4, J_5\}$, $profit(A'(S')) = 4 < 5 = profit(A(S'))$. If J_3 is the first interval in the initial ordering then on input $S' = \{J_1, J_2, J_3, J_5\}$, $profit(A'(S')) = 3 < 4 = profit(A(S'))$. Similarly for J_4. Hence, the first interval in the ordering is $U \in \{J_1, J_2\}$. Also, let $U' \in \{J_1, J_2\} \backslash \{U\}$. Consider input sets containing U. Since A' is memoryless whenever it reads U' it rejects it and the ordering remains as determined in the previous round. If J_3 or J_4 precedes in the new ordering J_5 then on input $S' = \{J_1, J_2, J_3, J_5\}$ (or $S' = \{J_1, J_2, J_4, J_5\}$), $profit(A'(S')) = 3 < 4 = profit(A(S'))$. Else, if J_5 precedes J_3 and J_4 in the new ordering, then on input $S' = \{U, J_3, J_4, J_5\}$, $profit(A'(S')) = 4 < 5 = profit(A(S'))$. □

Lemmata that are different for the case of one processor Interval Scheduling, for multiple machines Interval Scheduling and for the case of the Job scheduling Problem

The following lemma corresponds to the case of $m = 2$ processors, interval scheduling with proportional profits problem.

Lemma 2.2.5. D(MEMORYLESS-PRIORITY, GREEDY-FIXED-PRIORITY), *for proportional profit Interval Scheduling on two-machine environments.*

Proof. Say that we have 2 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). (see figure 2.1) 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:

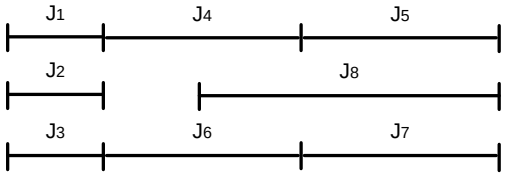

Figure 2.1: Graphical representation of the input for the proof of the lemma 2.2.5

- Schedule greedily from $\{J_1, J_2, J_3\}$.
- If all J_1, J_2 and J_3 are present then schedule J_4 at P_1 and J_5 at P_2.
- Else schedule both J_4 and J_5 at P_1.
- 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 for the case of the Job Scheduling problem for one processor. The following lemma appears to have a more complicated proof than the rest of this chapter.

Lemma 2.2.6. D(MEMORYLESS-PRIORITY, GREEDY-FIXED-PRIORITY), *for proportional profit Job Scheduling on one-machine environments.*

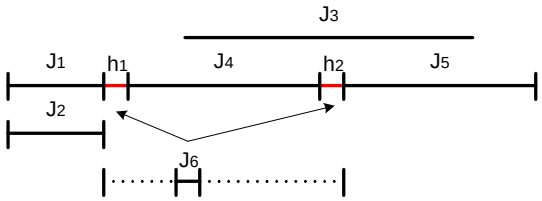

Figure 2.2: Graphical representation of the input for the proof of the lemma 2.2.6

Proof. The representation of the input is in figure 2.2. 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 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:

- 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.

- 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 the case of Job Scheduling or Interval Scheduling on more than 1 processor.

Lemma 2.2.7. 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. □

2.2.2 Almost robust hierarchies according to memory, greediness and adaptiveness criteria for the Job Scheduling Problem

This section contains corollaries concerning the comparison of classes of priority algorithms when only one characteristic (memory, greediness, adaptiveness) differs for every pair of classes of priority algorithms. We focus on the more general case of the Job Scheduling problem with multiple processors (see remark 2.2.1). A complete comparison between classes of priority algorithms for the variations on the types of jobs and number of processors can be found in section 2.2.3

Memory

In this section we provide the comparison according to the memory characteristic.

M-F-PRIORITY $<$ F-PRIORITY
M-PRIORITY $<$ PRIORITY
M-G-F-PRIORITY $<$ G-F-PRIORITY
M-G-PRIORITY $<$ G-PRIORITY

Table 2.1: Comparison of classes of Priority Algorithms, for the Job Scheduling problem on multiple identical machines, according to the memory criterion.

From lemma 2.2.2 we have the following corollary.

Corollary 2.2.8. M-F-PRIORITY $<$ F-PRIORITY

Proof. It is known that M-F-PRIORITY $\leq$ F-PRIORITY. Because, M-F-PRIORITY $\leq$ M-PRIORITY $\approx$ M-G-PRIORITY $\leq$ G-PRIORITY, from lemma 2.2.2 we have that D(M-F-PRIORITY, F-PRIORITY). □

From the lemmata 2.1.3 and 2.2.4 we have the following corollary:

Corollary 2.2.9. M-PRIORITY $<$ PRIORITY

Proof. M-PRIORITY $\leq$ PRIORITY, M-PRIORITY $\approx$ M-G-PRIORITY $<$ G-PRIORITY and G-PRIORITY $\leq$ PRIORITY which implies that D(M-PRIORITY, PRIORITY). □

Directly from lemmata 2.2.6 and 2.2.4 we have:

Corollary 2.2.10. M-G-F-PRIORITY $<$ G-F-PRIORITY

Corollary 2.2.11. M-G-PRIORITY $<$ G-PRIORITY

Greediness

G-F-PRIORITY $<$ F-PRIORITY
G-PRIORITY $<$ PRIORITY
G-M-F-PRIORITY $<$ M-F-PRIORITY
G-M-PRIORITY $\approx$ M-PRIORITY

Table 2.2: Comparison of classes of Priority Algorithms, for the Job Scheduling problem on multiple identical machines, according to the greediness criterion.

Lemma 2.2.2 implies the following corollary.

Corollary 2.2.12. G-F-PRIORITY $<$ F-PRIORITY
G-PRIORITY $<$ PRIORITY

Proof. Notice that G-F-PRIORITY $\leq$ G-PRIORITY and F-PRIORITY $\leq$ PRIORITY. □

Directly from lemma 2.2.3 we have:

Corollary 2.2.13. M-G-F-PRIORITY $<$ M-F-PRIORITY

The above consequence makes clear some things concerning the proof of lemma 2.1.3. Throughout the proofs we have made extensive use of lemma 2.1.3, because it appears to be very powerful. It can be applied on every type of the scheduling problems considered in this work, such as Interval or Job Scheduling on one or multiple processors. As we

will see in section 2.3, it can also be applied to variations of the memoryless property. Despite all this power, which intuitively relies on the (strong) simulation argument, it cannot be extended for the case of the fixed priority algorithms. The reason is that this simulation heavily relies on the fact that in the case of memoryless priority algorithms we can compensate greediness for adaptiveness. When we loose this property it seems plausible that no such further assertions can be made.

Adaptiveness

F-PRIORITY $<$ PRIORITY
F-G-PRIORITY $<$ G-PRIORITY
F-M-PRIORITY $<$ M-PRIORITY
F-M-G-PRIORITY $<$ M-G-PRIORITY

Table 2.3: Comparison of classes of Priority Algorithms, for the Job Scheduling problem on multiple identical machines, according to the adaptiveness criterion

The lemma 2.2.1 implies the following corollary.

Corollary 2.2.14. F-PRIORITY $<$ PRIORITY

Proof. It is known that F-PRIORITY $\leq$ PRIORITY. From lemma 2.2.1 we have D(F-PRIORITY, M-G-PRIORITY. Furthermore, M-G-PRIORITY $\leq$ PRIORITY, which implies D(F-PRIORITY, PRIORITY) □

From lemma 2.2.1 we have:

Corollary 2.2.15. G-F-PRIORITY $<$ G-PRIORITY

From lemma 2.2.1 the following corollary holds.

Corollary 2.2.16. M-F-PRIORITY $<$ M-PRIORITY

Proof. It is known that M-F-PRIORITY $\leq$ M-PRIORITY. From lemma 2.2.1 we have D(F-PRIORITY, M-G-PRIORITY) which together with M-F-PRIORITY $\leq$ F-PRIORITY and M-G-PRIORITY $\approx$ M-PRIORITY implies that D(M-F-PRIORITY, M-PRIORITY). □

A corollary of lemma 2.2.2 follows.

Corollary 2.2.17. M-G-F-PRIORITY $<$ M-G-PRIORITY

Proof. It is known that M-G-F-PRIORITY $\leq$ M-G-PRIORITY. From lemma 2.2.1 it follows that D(F-PRIORITY, M-G-PRIORITY), and also M-G-F-PRIORITY $\leq$ F-PRIORITY, that is D(M-G-F-PRIORITY, M-G-PRIORITY) which proves the corollary. □

Incomparable classes of priority algorithms

Up to this point we followed a systematic approach and we proved that the hierarchy defined is most of the times robust. That is, adding adaptiveness, going from greedy to non-greedy, and from memoryless to priority algorithms with memory, the corresponding classes in most of the cases are "stronger", w.r.t. algorithm profit on all input sets. This study was restricted by looking at just one characteristic at a time. The question is what happens in the case that we examine classes of algorithms where "mixtures" of different characteristics are present. Considering all of the above relations between classes of priority algorithms we provide a few examples concerning the comparison of F-PRIORITY versus the G-PRIORITY and M-G-PRIORITY. In section 2.2.3we provide a complete comparison between any two classes of priority algorithms, where there are more things to notice about the incomparable class pairs.

From the lemmata 2.2.2 and 2.2.1 we have the following corollary.

Corollary 2.2.18. F-PRIORITY $\perp$ G-PRIORITY

Also from lemma 2.2.2 we have the corollary: D(M-G-PRIORITY, F-PRIORITY) (because M-G-PRIORITY $<$ G-PRIORITY). Therefore, we also have the following corollary.

Corollary 2.2.19. F-PRIORITY $\perp$ M-G-PRIORITY $\approx$ M-PRIORITY

2.2.3 Tables comparing classes of priority algorithms

This sub-section contains a comparison between classes of priority algorithms. We have three cases: (i) job scheduling for any number of processors, (ii) interval scheduling for a number of processors $m \geq 2$ and (iii) interval scheduling for one processor ($m = 1$). Since, all the "D-relations" that hold for the job scheduling ($m \geq 1$) hold also for interval scheduling with $m \geq 2$, then the same relations between priority algorithms classes hold

in these two cases. Thus Table 2.4 can be used to compare any two classes of priority algorithms for cases (i) and (ii) under the relations $<$, $\approx$ and $\perp$. Table 2.5 concerns the case of interval scheduling where $m = 1$. It is obvious that the general case is (i), that is Job Scheduling with an arbitrary number of processors. Relation $>$ is defined in the obvious way. If the table entry (r, c) contains the relation R, the class which corresponds to the r-th row is C and the class which corresponds to the c-th column is C', then we read "$C \ R \ C'$ is true". We have only filled the upper triangular part of the table; since all the other relations trivially follow. Also notice that the column $M - PR$ could be missing as well, because M-PRIORITY $\approx$ M-G-PRIORITY. PR is a short-name for PRIORITY. In the tables, below each relation we write the lemmata used to derive the relation.

	F-PR	G-PR	G-F-PR	M-PR	M-F-PR	M-G-PR	M-G-F-PR
PR	$>$ 2.2.1	$>$ 2.2.2	$>$ 2.2.1	$>$ 2.2.4	$>$ 2.2.4	$>$ 2.2.4	$>$ 2.2.4
F-PR		$\perp$ 2.2.1,2.2.2	$>$ 2.2.2	$\perp$ 2.1.3,2.2.1 2.2.2	$>$ 2.1.3,2.2.2	$\perp$ 2.2.1,2.2.2	$>$ 2.1.3,2.2.2
G-PR			$>$ 2.2.1	$>$ 2.2.4	$>$ 2.1.3,2.2.1	$>$ 2.2.4,2.1.3	$>$ 2.2.4,2.1.3
G-F-PR				$\perp$ 2.2.1,2.2.5 or 2.2.6	$\perp$ 2.2.3,2.2.5 or 2.2.6	$\perp$ 2.2.1,2.2.5 or 2.2.6	$>$ 2.2.5 or 2.2.6
M-PR					$>$ 2.2.1	$\approx$ 2.1.3	$>$ 2.2.1
M-F-PR						$<$ 2.1.3,2.2.1	$>$ 2.2.3
M-G-PR							$>$ 2.2.1

Table 2.4: Comparison between classes of priority algorithms for the cases of (i) the, general, Job Scheduling ($m \geq 1$) and (ii) Interval Scheduling where $m \geq 2$ problems

	F-PR	G-PR	G-F-PR	M-PR	M-F-PR	M-G-PR	M-G-F-PR
PR	$>$ 2.2.1	$>$ 2.2.2	$>$ 2.2.1	$>$ 2.2.4	$>$ 2.2.4	$>$ 2.2.4	$>$ 2.2.4
F-PR		$\perp$ 2.2.1,2.2.2	$>$ 2.2.2	$\perp$ 2.1.3,2.2.1 2.2.2	$>$ 2.1.3,2.2.2	$\perp$ 2.2.1,2.2.2	$>$ 2.1.3,2.2.2
G-PR			$>$ 2.2.1	$>$ 2.2.4	$>$ 2.1.3,2.2.1	$>$ 2.2.4,2.1.3	$>$ 2.2.4,2.1.3
G-F-PR				$<$ 2.2.1,2.2.7	$<$ 2.2.3,2.2.7	$<$ 2.1.3,2.2.1 2.2.7	$\approx$ 2.2.7
M-PR					$>$ 2.2.1	$\approx$ 2.1.3	$>$ 2.2.1
M-F-PR						$<$ 2.1.3,2.2.1	$>$ 2.2.3
M-G-PR							$>$ 2.2.1

Table 2.5: Comparison between classes of priority algorithms for the one processor, proportional profit, interval scheduling

2.3 Memory and Priority Algorithms

We begin this section with the following obvious fact.

Fact 2.3.1. *Say that $A \in C$, where C is a class of priority algorithms with memory. Then, there exists an algorithm A' which, on every input, has the same profit as A and it is realized through a function that depends only on the current sets of the scheduled and rejected jobs and can remember all the information of a job.*

Proof. Given an algorithm, the job J and the sets of scheduled and rejected jobs at step i, we can retrieve the order in which the jobs were read from the input. Since the algorithm is deterministic, this can be trivially done by taking the initial priority function and checking the context of these sets. Then we inductively proceed by simulating the computation of A and by this simulation we can "retrieve" the context of the memory and see what decision A would take. □

2.3.1 Variations of the memoryless restriction

In this sub-section we consider variations of the memoryless property, regarding the information that an algorithm can "remember" about a *scheduled* job. Our treatment on this subject is far from being complete. We define some natural variations of the memoryless restriction and we give a few lemmata showing that these variations result in priority algorithms that are not of the same power.

Up to this point we considered as memoryless, priority algorithms that do not base their decisions on the rejected jobs. We made the assumption that a priority algorithm (even a memoryless one) can retrieve all of the information from each scheduled job , that is the processor where it is scheduled, its starting time and (id, r, d, p, w). Now we extend this study by restricting the things that a memoryless algorithm can retrieve from a scheduled job. In what follows we use the term "tag" to refer to a job (r, d, p, w) without an integer identifier.

1. Memoryless (M_{full} or M) with integer identifiers. This is the model we considered in this chapter up to this point.

2. Memoryless with tags (M_{tags}). For each scheduled job the algorithm can read the processor where it is scheduled, its starting time, its release, deadline and processing time and as well as its profit (weight).

3. Memoryless without tags (M_{weak}). This is the weakest model since the only thing that we can retrieve from each scheduled job is its starting and its processing time and the processor where it is scheduled.

Thus we introduced two new types of memoryless algorithms. We now describe a new type of the memoryless restriction, called M_{new}. The initial model is modified by making decisions on the ordering at the end of each iteration. M_{new} is a modification of M_{weak} where the algorithm can also re-order the input elements after the rejection of a job. The algorithm, temporarily, remembers all the information of the currently read job. We use this model only for the case of the memoryless algorithms. Our treatment is not going to be as thorough as in the previous sections. With this plausible modification in the model, proposition 1.5.1 does not hold.

In what follows we demonstrate some cases where the above mentioned definitions, on the memoryless property, define classes of algorithms that are of different power. We do not give full proofs of the lemmata as the proofs are pretty-similar to the ones presented in the previous sections. It's useful to notice that for any class (with memory) C of priority algorithms $M_{weak} - C \leq M_{tag} - C \leq M - C$. The following lemma proves that the memoryless fixed priority algorithms with tags dominate over the weak-memoryless fixed priority algorithms. All of the following lemmata hold for the proportional profit case, where the machine environment consists of one processor.

Remark 2.3.1. The constructions of the following lemmata are not based on the "gadgets" used in the previous sections. The arguments are still of the same flavor. We use the extra information provided by one memoryless type in order to construct algorithms that use such information so as to "understand" what follows in the input set.

Lemma 2.3.2. $D(M_{weak} - F - PRIORITY, M_{tag} - F - PRIORITY)$.

Proof. Let $S = \{J_1, J_2, J, J_3, J_4, J_5\}$ where $J_1 = (0, 1, 1, 1)$, $J_2 = (1, 2, 1, 1)$, $J = (0, 2, 1, 1)$, $J_3 = (2, 4, 2, 2)$, $J_4 = (4, 6, 2, 2)$ and $J_5 = (2, 5, 3, 3)$. Let also $A \in M_{tag} - F - PRIORITY$, with initial ordering $J_1 > J_2 > J > J_5 > J_3 > J_4$. A is as follows:

- Schedule greedily from $\{J_1, J_2, J\}$, where J is scheduled at the earliest available time.

- If only one from $\{J_1, J_2, J\}$ is scheduled *or* if (J_1, J_2) is scheduled, then reject J_5 and schedule the rest greedily.

- Else schedule greedily.

Let $A' \in M_{weak}-FIXED-PRIORITY$. We show that J_1, J_2, J have higher priority than J_3 and J_4 and J_5. Suppose that $U \in \{J_1, J_2, J\}$ has lower priority than J_3 or J_4 or J_5. (a) J_3 has higher priority than J_4, J_5. On input $\{U, J_3, J_4, J_5\}$ A' accepts $\{U, J_3, J_4\}$. If we remove J_4 and U, A' still accepts J_3 (since J_3 is the first job that A' reads) and thus it performs worse than A. (b) Similarly for J_4. (c) J_5 has higher priority than J_3, J_4. On input $\{J5\}$ A' gains zero profit (where A accepts J_5).

Here we assume that J_1, J_2, J are the jobs of higher priority and since A' is in $M_{weak} - FIXED - PRIORITY$, on inputs $S_1 = \{J_1, J_2, J_3, J_4, J_5\}$ and $S_2 = \{J_1, J, J_3, J_4, J_5\}$, A' takes the same decisions regarding J_3, J_4, J_5 and schedules $\{J_3, J_4\}$. If we remove from S_2 the interval scheduled last among $\{J_3, J_4\}$ then A' still rejects J_5 (since it accepts the one scheduled first among $\{J_3, J_4\}$). Thus A' gains less profit than A. □

Using the same proof techniques, one can show the corresponding thing for M and M_{tag}.

Lemma 2.3.3. $D(M_{tag} - F - PRIORITY, M - F - PRIORITY)$

Proof. Let $S = \{J_1, J_2, J_3, J_4, J_5\}$, where $J_1 = (1, 0, 1, 1, 1)$, $J_2 = (2, 1, 2, 1, 1)$, $J' = (3, 1, 2, 1, 1)$, $J_3 = (5, 2, 4, 2, 2)$, $J_4 = (6, 4, 6, 2, 2)$ and $J_5 = (7, 3, 6, 3, 3)$. Let $A \in M - F - PRIORITY$), with initial ordering $J_1 > J_2 > J > J_5 > J_3 > J_4$. The description of A follows:

- Schedule greedily from $\{J_1, J_2, J\}$.
- If only one from $\{J_1, J_2, J\}$ is scheduled *or* if (J_1, J_2) is scheduled, then reject J_5 and schedule the rest greedily.
- Else schedule greedily.

The rest of the proof is similar to the proof of the previous lemma. □

From lemmata 2.3.2 and 2.3.3 we have that $M_{weak} - F - PRIORITY < M_{tag} - F - PRIORITY < M - F - PRIORITY$, which shows that (in general) different types of the memoryless property define classes of algorithms that can achieve different profits. In what follows we provide an indication that the M_{new} property defines algorithms that can perform better than the previous types. Observe that for any class C of priority algorithms with memory $M_{weak} - C \leq M_{new} - C$.

Lemma 2.3.4. $D(M - PRIORITY, M_{new} - G - PRIORITY)$

Proof. All the jobs in this proof are intervals of proportional profit. Let $S = \{J_1, J_2, J_3, J_4\}, J_1 = (2-\epsilon, 2+\epsilon)$, for a fixed $\epsilon < \frac{1}{2}, J_2 = (0,2), J_3 = (2,4), J_4 = (1,4)$. Let $A \in M_{new} - G - PRIORITY$, with initial ordering $J_1 > J_4 > J_2 > J_3$ and the following actions:

- If J_1 is read on the input then reject, determine the following ordering for the rest of the jobs $J_2 > J_3 > J_4$ and accept greedily.
- Else accept greedily.

Let $A' \in M - PRIORITY$. If the first job, in the ordering of A', is J_2 and A' accepts it, then on input $S' = \{J_4, J_2\}$, $profit(A'(S')) = 2 < 3 = profit(A(S'))$. If it rejects it then on input $\{J_2\}$ it has zero profit, where A does not. Similarly for J_3. If J_4 is the first job in the ordering and A' accepts it, then on input S, $profit(A'(S)) = 3 < 4 = profit(A(S))$. If it rejects it then on input $\{J_4\}$, A' has zero profit. Thus J_1 must be the first job in the input. If A' accepts it then on input $S' = \{J_1, J_2\}$ $profit(A'(S')) = 1 < 2 = profit(A(S'))$. If A' rejects it then it cannot determine a new ordering and thus the same argument as the previous one can be applied for the remaining $\{J_2, J_3, J_4\}$ jobs about the second job in the initial ordering. □

Remark 2.3.2. If we consistently replace the memoryless property by the two weaker memoryless properties (M_{weak} and M_{tag}) then the lemmata 2.2.1-2.2.7 still hold. We believe that the same thing holds for the M_{new} property also.

2.3.2 Robust memory hierarchies

In this section we define the bounded memory notion which implies a natural memory hierarchy that we show to be robust most of the times.

Bounded memory model - Bounded memory hierarchy.

Definition 2.3.1. We write for a priority algorithm A, that $A \in M(k) - C$ where C is a class of priority algorithms with memory, and we mean that A makes its (irrevocable) decisions by remembering any set of rejected jobs of cardinality less than or equal to $k \in Z^{\geq 0}$. 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 2.3.3. Looking at the proof of lemma 2.2.2 we see that lemma 2.1.3 cannot be extended even when we have memory for only one rejected job. Actually, the same proof (as the one for lemma 2.2.2) shows that D(M(1)-GREEDY-PRIORITY,M(1)-FIXED-PRIORITY).

Notice that in definition 2.3.1 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 2.3.5.

(i) Let S be a set of intervals. The machine environment consists of one processor and C stands for any one among {FIXED-PRIORITY, GREEDY-PRIORITY, PRIORITY}. *(ii) Let S be a set of jobs and we have a machine environment of $m \geq 1$ identical processors, or S be a set of intervals and we have a machine environment of $m \geq 2$ identical processors. Let C stand 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 2.2.7 is M(k)-G-F-PRIORITY $\approx$ G-F-PRIORITY for the one machine environment and every $k \in \mathbb{Z}^{\geq 0}$. Theorem 2.3.5 is a corollary of the three following lemmata (also recall Remark 2.2.1).

Lemma 2.3.6. *For the two machine environment Interval Scheduling,* D(M(k)-PRIORITY,M($k+1$)-G-F-PRIORITY).

Proof. We do not give the full proof of this lemma since it is very similar to the proof of lemma 2.2.5. 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) $J_{i_1} = J_{i_2} = \ldots = J_{i_{k+3}} = (0,1)$ 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 $A \in$ G-F-PRIORITY with initial ordering $J_{i_1} > J_{i_2} > \ldots > J_{i_{k+3}} > J_1 > J_2 > \mathbf{J_5} > J_3 > J_4$. 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 2.2.5. □

Lemma 2.3.7. *For the one machine environment Job Scheduling,*
$D(\text{M}(k)\text{-PRIORITY},\text{M}(k+1)\text{-G-F-PRIORITY})$.

Proof. This proof is also very similar to the proof of lemma 2.2.6. Instead of having 2 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 2.2.6 with all the modifications made in the obvious way. □

Actually, we can take lemmata 2.2.5 and 2.2.6 as corollaries of the above two.

Lemma 2.3.8. *For the one machine environment Interval Scheduling,*
(i) $D(\text{M}(k)\text{-PRIORITY},\text{M}(k+1)\text{-F-PRIORITY})$,
(ii) $D(\text{M}(k)\text{-PRIORITY},\text{M}(k+1)\text{-G-PRIORITY})$.

Proof. (i) Let $S = \{J_{i_1}, J_{i_2}, \ldots, J_{i_{k+2}}\} \cup \{J_1, J_2, J_3\}$, where $J_{i_1} = J_{i_2} = \ldots, = J_{i_{k+2}} = (0,1)$, $J_1 = (1,3)$, $J_2 = (3,5)$, $J_3 = (2,5)$. Let $A \in \text{M}(k+1)\text{-F-PRIORITY}$ with ordering $J_{i_1} > J_{i_2} > \ldots, > J_{i_{k+2}} > \mathbf{J_3} > J_1 > J_2$ 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 $A' \in \text{M}(k)\text{-PRIORITY}$. 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, $profit(A'(S'')) = 3 < 4 = profit(A(S''))$.

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

Theorem 2.3.5 holds also for the bounded memory model where the algorithm is allowed to update its memory context. In the next section we show a separation for such memory models between classes of algorithms having $k+1$ memory bits and algorithms having k memory bits, for every $k \in \mathbb{Z}^{\geq 0}$.

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

Definition 2.3.2. 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.

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

Theorem 2.3.9.

(i) Let S be a set of intervals. The machine environment consists of one processor and C stands for any one among {FIXED-PRIORITY, GREEDY-PRIORITY, PRIORITY}. *(ii) Let S be a set of jobs and we have a machine environment of $m \geq 1$ identical processors or S be a set of intervals and we have a machine environment of $m \geq 2$ identical processors. Let C stand 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 lemmata 2.3.6, 2.3.7, 2.3.8. 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 lemmata 2.3.6, 2.3.7, 2.3.8).

Lemma 2.3.10. $D(\mathrm{M}_b(k)\text{-PRIORITY}, \mathrm{M}_b(k+1)\text{-FIXED-PRIORITY})$

Proof. Fix $k \in \mathbb{Z}^{\geq 0}$. Consider 2^{k+2} intervals of identical release and deadline time. Let $S' = \{J_1, \ldots, J_{2^{k+2}}\}$, $S'' = \{J', J'', J\}$ and $S = S' \cup S''$, where $J_1 = J_2 = \ldots = J_{2^{k+2}} = (0,1)$, $J' = (1,3)$, $J'' = (3,5)$, $J = (2,5)$. Consider the following algorithm $A \in \mathrm{M}_b(k+1)$-FIXED-PRIORITY with ordering $J_1 > J_2 > \ldots > J_{2^{k+2}} > \mathbf{J} > J' > J''$:

- Accept greedily from S'. Use the $k+1$ memory bits to store the value of a counter. When rejecting an interval from S', if the decimal value of the counter is less than $2^{k+1}-1$ then increase the counter by 1.

- If the counter is $2^{k+1}-1$ then reject J (if present) and schedule the rest greedily.

- Else schedule greedily.

Note that A rejects J if the input set contains at least 2^{k+1} intervals from S' (one scheduled interval and at least $2^{k+1}-1$ rejected). Regarding a round in a run of an algorithm on a subset of S, we say that the algorithm is in a "valid state" if (a) there are at least 2^{k+1} intervals read from S' or (b) the input contains less than 2^{k+1} intervals from S'. We show that every algorithm A', having at least the same profit as A is in valid state before reading an interval from S''. Suppose the contrary. That is, A' reads an interval from S'' having read less than 2^{k+1} intervals from S' and the input contains at least 2^{k+1} intervals from S'. Say that the input contains $\hat{S} \subseteq S'$, where $|\hat{S}| \geq 2^{k+1}$. Say that $\hat{S}' \subset \hat{S}$ is the set of intervals considered so far by A'. If A' reads J and rejects it then $profit(A'(\hat{S}' \cup \{J\})) \leq 1 < 4 = profit(A(\hat{S}' \cup \{J\}))$. If A' schedules J then $profit(A'(\hat{S} \cup S'')) \leq 4 < 5 = profit(A(\hat{S} \cup S''))$. If A' reads J' and rejects it then $profit(A'(\hat{S}' \cup \{J'\})) \leq 1 < 3 = profit(A(\hat{S}' \cup \{J'\}))$. If A' schedules J' then $profit(A'(\hat{S}' \cup \{J, J'\})) \leq 3 < 4 = profit(A(\hat{S}' \cup \{J, J'\}))$. Similarly for J''.

Consider a run of $A' \in \mathrm{M}_b(k)$-PRIORITY. We call as "memory state of A' in a round" the context of the time interval $(0,1)$ and the k memory bits. We argue that if A' has two identical memory states and it is not in a valid state then there exists an input where A' gains less profit than A. Consider a run of A' on $S'_a \subset S'$ where there exists a memory state M which appears twice and A' is not in a valid state. Say that between the two occurrences of M the intervals $\{J'_1, \ldots, J'_m\}$, $m \geq 1$ have been read. Extend S'_a by adding intervals from S' (not considered yet). Say that $S'_a \subset S'_b \subset S'$, $|S'_b| = 2^{k+1}$ is such a set. If A' has at least the same profit as A (on every input) then on $S_1 = S'_b \cup \{J', J'', J\}$, A' rejects J. Let $U \in \{J', J''\}$ be the one scheduled last among $\{J', J''\}$. A' also rejects J on $S_2 = S_1 \backslash \{U\}$. M occurs twice and therefore on $S_3 = S_2 \backslash \{J'_1, \ldots, J'_m\}$, A' also rejects J. Thus, $profit(A'(S_3)) = 3 < 4 = profit(A(S_3))$. Therefore, there exists an input where A' gains less profit than A.

Fix an $\hat{S} \subset S'$, $|\hat{S}| = 2^{k+1}$. Consider the run of $A' \in \mathrm{M}_b(k)$-PRIORITY on $\hat{S} \cup S''$. If A' has the same profit as A, then when reading the 2^{k+1} intervals (from $\hat{S}'$) it updates

its memory each time it reads one interval. During this process A' does not have two identical memory states. Since we have 2^{k+1} intervals and only 2^k memory states this is not possible. □

Remark 2.3.4. We do not consider memory models where the priority algorithms store and update job descriptors in "memory cells". There are some questions that someone may ask for such a model. Which job descriptors do we allow to store in memory? Can a job descriptor appear more than once in memory? Thus, it seems hard to define a natural memory model of this type. Furthermore, considering an instance of the memory context as a string we observe that the memory alphabet depends on the jobs in the input. Therefore, the memory size (w.r.t. a fixed alphabet) is not constant and thus the definition of such memory models seems even less natural.

Chapter 3

Complexity of languages related to the optimality of Priority Algorithms

The field of Priority Algorithms is motivated by the derivation of unconditional lower bounds and it does not study the problems with respect to the complexity classes they belong to. In this chapter we study the complexity of languages related to Priority Algorithms in the classic complexity theoretic context; that is, we derive containment and hardness results for such languages. From this perspective the recognition complexity of problems related to Priority Algorithms seems to be orthogonal to the research direction of Priority Algorithms themselves.

Our main goal is to characterize optimal inputs for priority algorithms and to study the complexity of deciding such optimal sets. That is, we want to look at questions similar to the research conducted in other characterizations of optimal greedy algorithms such as the matroids, greedoids and matroid embeddings. We study the complexity of two main types of questions. We want to decide whether a given priority algorithm is optimal, or whether there exists an optimal priority algorithm for a given finite set of jobs or intervals. Recall that given a set of jobs S, "optimal priority algorithm" (w.r.t. S) means an algorithm that is optimal on every subset of S. The main results of this chapter are: (i) given a finite set of intervals we can decide within polynomial whether a given greedy-fixed priority algorithm (for the one machine environment) is optimal and (ii) the problem of deciding, given a finite set of intervals, whether there exists an optimal priority algorithm (for the one machine environment), is also in P. In general, the definition of

the first type of question requires us to agree on a specific way of describing a priority algorithm. This fact makes the first type of question less elegant and apparently less interesting than the second one. For the case of Interval Scheduling on one processor, it is natural to consider that a greedy-fixed priority algorithm is fully described by a total ordering on the finite set of intervals. This is the only case where a question of the first type has an elegant definition. When the algorithm is allowed to make decisions, if we have jobs or more than one processor, then we have to describe a set of actions. In this case the problem does not seem to have an elegant definition, since we have to agree on a specific encoding of the algorithm; for example, the algorithm is realized by an efficient model of computation (say through a logspace bounded Turing Machine). More importantly, it is easy to observe that in this case the problem has trivial answers[1] and thus we do not provide any results concerning such languages.

The other technical contribution of this chapter concerns the study of optimal priority algorithms on sets of jobs and on multiple-machine environments. We derive normal forms and properties for optimal priority algorithms on such sets.

This chapter is organized as follows. In section 3.1 we provide the definitions of the languages we are studying. Section 3.2 states one of the main theorems used throughout this chapter, by giving a characterization of optimal priority algorithms. We use this characterization so as to state a polynomial time algorithm for deciding whether a given greedy-fixed algorithm for the Interval Scheduling problem on one processor is optimal on a given finite set of intervals. Section 3.3 discusses a characterization of finite sets of intervals that permit optimal priority algorithms for one-machine environments. In this section we give a polytime algorithm for the related problem and we show that if such an optimal algorithm exists then there also exists an optimal algorithm in MEMORYLESS-GREEDY-FIXED-PRIORITY. That is, in contrast to our separation results in chapter 2, it appears that it is not possible to separate classes of optimal priority algorithms. In section 3.4 we strengthen our previous results by showing that the languages of our main interest are in uniform NC^2 and thus unlikely to be P-hard (w.r.t. NC^1 or many-to-one logspace reductions) unless $P = U_L - NC^2$. In section 3.6 we show that deciding whether a greedy-fixed priority algorithm is optimal on a set of intervals of equal profit is in uniform AC^0 (that is, $FO[<, bit]$). In section 3.7 we show the NL-completeness for the

[1]For example, if the priority algorithm is realized by a logspace bounded Turing Machine then we can show that problem is NP-complete even when the input set has intervals and only one job.

Interval Scheduling problem where the intervals have profits bounded by a polynomial in the number of intervals. We conclude by mentioning some open problems in this area (section 3.9). In section 3.8 we study optimal priority algorithms on sets of jobs. In section 3.8 we show normal forms and properties for optimal priority algorithms. Although, the number of optimal priority algorithms on sets of distinct profit jobs may be super-exponential w.r.t. the cardinality of the set, we show that every optimal priority algorithm on such sets schedules the same set of jobs. We also show that if a set admits an optimal priority algorithm then it also admits a memoryless-greedy-priority algorithm and we derive an invariant for sets of jobs of arbitrary profit.

3.1 Definitions

We presume the existence of a finite alphabet Σ where the languages are subsets of Σ^*. Unless stated otherwise, the language corresponding to the problem is structured in the obvious way, where all numbers are represented in binary. Whenever we write the term "optimal algorithm for a given finite set of jobs" we mean a priority algorithm that is optimal on every subset of the input set. In the same way whenever we write "optimal set", we refer to a set of intervals where there exists a priority algorithm which performs optimally on every subset of this set. The following languages can be divided into two categories: (a) problems where the priority algorithm has a "simple" encoding and (b) problems where the priority algorithm is not part of the input. By the term "simple" we mean that the description of the algorithm is just a total ordering of the given set of intervals. Thus, in this category there are algorithms only from the Greedy-Fixed class for Interval Scheduling on a single processor.

Given a priority algorithm, with "simple" description, and a finite set of jobs decide whether the algorithm is optimal

We are restricting to the case of interval scheduling on single machine environments. In this case the algorithm is determined only by a fixed ordering on the given set of intervals. The language corresponding to the following problem (defined in Garey and Johnson style) leads to some interesting complexity results and motivates our further study.

Problem 1. NON-OPT-GF-INTERVAL($m = 1$)
Instance: Say that S is a finite set of $n \in \mathbb{Z}^+$ *intervals* $S = \{J_1, \ldots, J_n\}$ and the description of an algorithm $A \in$ G-F-PRIORITY is determined by a total ordering T on S.
Question: Does there exist a subset $S' \subseteq S$ where A is not optimal on S'?

In order to avoid overwhelming the reader with definitions we just say that all the relevant languages follow the same notational pattern as the one mentioned above. Following, the same pattern as previously, NONOPT-GF-INTERVAL is the same language as in problem 1 with the difference that we do not restrict the machine environment to one processor. In this case the number of identical processors is part of the instance. When we have jobs instead of intervals, we write NONOPT-GF-GENERAL.

The following language can be view as a problem where every interval is of unit profit[2].

Problem 2. NONOPT-GF-INTERVAL-EQUAL-PROFIT($m = 1$)
Instance: Say that S is a finite set of $n \in \mathbb{Z}^+$ *intervals* $S = \{J_1, \ldots, J_n\}$ of the same profit. The description of an algorithm $A \in$ G-F-PRIORITY is determined by a total ordering T on S.
Question: Does there exist a subset $S' \subseteq S$ where A is not optimal on S'?

Given a finite set of jobs decide whether there exists an optimal priority algorithm

To simplify our study we restrict to the case of machine environments *consisting only of one machine.*

Problem 3. EXISTS-OPT-GENERAL($m = 1$)
Instance: Say that S is a finite set of $n \in \mathbb{Z}^+$ jobs $S = \{J_1, \ldots, J_n\}$.
Question: Does there exist a priority algorithm that it is optimal on every subset of S?

It is easy to see that the general problem can be decided within time $O(2^n)$, where n is the length of the input; that is EXISTS-OPT-GENERAL($m = 1$)$\in ESPACE$.

We restrict our attention to problems that only involve intervals in their inputs and the scheduling is done on a single machine. These problems appear to have interesting and somehow counter-intuitive associated complexity results.

[2]We already know that there exist optimal priority algorithms for this problem. The question here concerns whether a given priority algorithm is optimal.

Problem 4. EXISTS-OPT-INTERVAL($m = 1$)
Instance: Say that S is a finite set of $n \in \mathbb{Z}^+$ *intervals* $S = \{J_1, \ldots, J_n\}$.
Question: Does there exist a priority algorithm $A \in$ PRIORITY, where A is optimal on every subset $S' \subseteq S$?

Restrictions on the values of the profit

We consider restrictions on the size of the numbers involved in the problem. If we restrict the values of the profits to be a polynomial in the number of intervals or jobs then we get the following variations.

Problem 5. POLY-NONOPT-GF-INTERVAL($m = 1$)
Instance: Say that S is a finite set of $n \in \mathbb{Z}^+$ *intervals* $S = \{J_1, \ldots, J_n\}$, where the value of the profit of each interval is a polynomial in n. The description of an algorithm $A \in$ G-F-PRIORITY is determined by a total ordering T on S.
Question: Does there exist a subset $S' \subseteq S$ where A is not optimal on S'?

Problem 6. POLY-EXISTS-OPT-INTERVAL($m = 1$)
Instance: Say that S is a finite set of $n \in \mathbb{Z}^+$ *intervals* $S = \{J_1, \ldots, J_n\}$, where the value of the profit of each interval is a polynomial in n.
Question: Does there exist a priority algorithm $A \in$ PRIORITY, where A is optimal one every subset $S' \subseteq S$?

3.2 Characterizations of optimal Priority Algorithms for Interval Scheduling

The following lemma characterizes non-optimal greedy-fixed priority algorithms for one-machine environments.

Lemma 3.2.1. *Say that the machine environment consists of one processor, $A \in$ G-F-PRIORITY, and let S be a set of intervals. A is not optimal iff there exists an interval $J \in S$ and there exists a $T \subseteq S$, where all of the following are true:*

i. every interval in T overlaps J

ii. every two intervals in T do not overlap each other

iii. $profit(T) > profit(\{J\})$

iv. for all $J' \in T$, J *has higher priority than* J'.

Proof. ($\Rightarrow$) We prove the contrapositive argument. More precisely, let A be a priority algorithm. Assume that for all $J \in S$ and for all $T \subseteq S$, if (i), (ii) and (iv) hold then $profit(T) \leq profit(\{J\})$. Then we want to show that A is optimal. The following argument is based on a well-known "exchange lemma" type of argument, used to prove optimality for greedy algorithms. Fix an arbitrary set $S' \subseteq S$. We do induction on the predicate which asserts that the first k steps the algorithm, on S', results in a set of scheduled intervals S_k which can be extended to an optimal schedule for S'. For $k = 0$, it trivially holds. Assume, that it holds for $k = m$; we want to prove that it holds for $m+1$. Let $S_m = \{J_{i_1}, J_{i_2}, \ldots, J_{i_l}\}$. Let OPT be an optimal schedule for S' that extends S_m. If J_{m+1} overlaps any interval from S_m then the assertion is trivially true. If $J_{m+1} \in OPT$ then it is also true. Suppose that J_{m+1} does not overlap any interval from S_m and $J_{m+1} \notin OPT$. Let $T = \{J_{j_1}, \ldots, J_{j_p}\}$ be the set of intervals in OPT that overlap J_{m+1} and no two intervals in T overlap each other. Obviously, no interval from T can overlap any interval from S_m and therefore every interval from T has lower priority than J_{m+1}. Thus, $profit(T) \leq profit(\{J_{m+1}\})$. Hence, the schedule $OPT' = OPT \backslash T \cup \{J_{m+1}\}$ is also of optimal profit.

($\Leftarrow$) is even more straightforward. If there exists an interval $J \in S$ and $T \subseteq S$, where any two intervals in T do not overlap each other, and $profit(T) > profit(\{J\})$ and J has a higher priority than every $J' \in T$ which overlaps J, then on input $\{J\} \cup S'$ A is obviously not optimal. □

The above lemma specifies a necessary and sufficient condition for a G-F-PRIORITY (in one-machine environments) to be non-optimal. Now, the following theorem is straightforward.

Theorem 3.2.2. NONOPT-GF-INTERVAL$(m = 1) \in P$ *(and thus* OPT-GF-INTERVAL$(m = 1) \in P$*).*

Proof. It is well-known that given a set of intervals we can compute within polynomial time an optimal schedule. This can be achieved with a well-known application of a Dynamic Programming technique. For every interval in the input set we compute (within polynomial time) the set S' of its overlapping intervals which have lower priorities. Then

we compute (in polytime) the optimal schedule for S' (using dynamic programming) and we compare it with the profit of the interval we have initially considered. If the optimal schedule (of these intervals) is greater than the interval under consideration we accept. If we have not accepted, we choose the next interval in the input and we repeat. At the end, if all intervals have been considered (and we have not accepted) we reject. The algorithm works in (deterministic) polynomial time. □

In section 3.4 we show a stronger result, that is NONOPT-GF-INTERVAL$(m = 1) \in U_L - NC^2$.

Remark 3.2.1. Using lemma 3.2.1 we can show that POLY-OPT-GF-INTERVAL$(m = 1) \in NL$. Within logarithmic space we can add numbers and compare sums of numbers, when the numbers to be summed are indicated in the input (or they can be determined "efficiently"). Obviously when performing these tasks, one should not write in the working tape any of the numbers (or a sum of numbers). When the numbers have *values* which are at most a polynomial in the cardinality of the given set, it is also trivial to decide the corresponding variation of the $SUBSET-SUM$ or the $PARTITION$ problem[3] within non-deterministic logarithmic space. The restricted variation of OPT-GF-INTERVAL$(m = 1)$ can be decided easily within non-deterministic logspace. This cannot be done by a "global" guess as depicted in the example in figure 3.1. Instead of making a "global

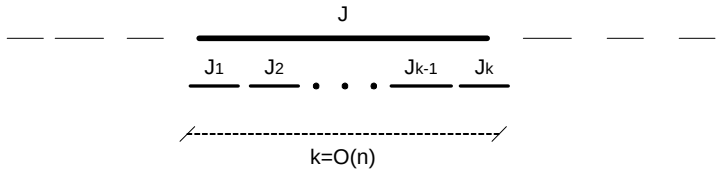

Figure 3.1: J has higher priority than its overlapping jobs. The sum of the profits of $J_1, \ldots, J_k$ is greater than the profit of J, but excluding one among $J_1, \ldots, J_k$ results in a total profit which is less than the profit of J.

guess", the algorithm guesses one by one the overlapping intervals and stores the sum

[3] $SUBSET - SUM$ and $PARTITION$ are well-known NP-complete problems and thus unlikely to be in NL. $SUBSET - SUB$ contains encodings of a set of integers and a distinct integer, such that there exists a subset of the elements of this set where the sum of its elements equals to the distinct number. A string belongs to $PARTITION$, if it is the encoding of a set of integers, such that there exists a partition of this set into two sets where the sum of their elements are equal.

of their profits in the working tape (since the numbers have "small" values). Hence, Poly-Opt-GF-Interval$(m = 1) \in NL$.

3.3 Characterization of optimal sets of intervals

In this section we show a necessary and sufficient condition for the non-existence of a priority algorithm which is optimal on every subset of a given set of *intervals* for one-machine environments. This lemma can be used to verify within polynomial time whether there exists such an optimal algorithm. We study separately the restricted case, where the profits of the intervals are distinct, and the general case where the profits are arbitrary. We assume that all the sets of intervals are "normalized" so as they do not contain intervals of profit zero. Notice, that by the definition of an interval, the release and deadline times cannot coincide. Sometimes, we abuse the notation and we treat the sequences as sets, meaning that instead of a sequence we are considering the set of elements it contains.

3.3.1 The distinct profits case

If for a given set of intervals all profits are unique then, we show the following necessary and sufficient condition for the non-existence of an optimal priority algorithm.

Lemma 3.3.1. *Let S be a finite set of intervals, with distinct profits. There does not exist a priority algorithm which is optimal on every subset of S iff (*) there exists an interval $J \in S$ and a set $T \subseteq S$, such that every interval in T overlaps J and has less profit than J, no two intervals in T overlap each other, and the total profit of intervals in T is greater than the profit of J.*

Proof. Assume (*) holds. Consider $A \in$ PRIORITY. If J has higher priority than every $J' \in T$ then we restrict our attention to the set $S' = \{J\} \cup T$. If A reads J and rejects then on input $\{J\}$ A is non-optimal. If A schedules J then on input S' the algorithm is again non-optimal. If J does not have higher priority than every other interval in T, then assume that $J'' \in T$ is one with higher priority than J. Consider the set $S'' = \{J, J''\}$. If A reads J'' and rejects it, then it performs non-optimally on $\{J''\}$. If it reads J'' and it accepts it then it performs non-optimally on S''.

For the other direction we show the contrapositive argument: Assume (*) does not hold. We define LPRF $\in$ M-G-F-PRIORITY (LPRF stands for "Largest Profit

First") as the algorithm which orders the intervals in decreasing profit order. Since all profits are distinct, the ordering of the algorithm is well-defined. We wish to show that LPRF is optimal on every subset of S. We use lemma 3.2.1. Since (*) does not hold this follows immediately by observing that for every interval $J' \in T$, J has higher priority iff it has greater profit. □

From the proof of the lemma 3.3.1 we take the following corollary.

Corollary 3.3.2. *If there exists a priority algorithm $A \in$ PRIORITY (for the one-machine environment) which is optimal on every subset of a set S of intervals with distinct profits, then the greedy-fixed priority algorithm with the "largest profit first ordering" is also optimal on every $S' \subseteq S$.*

Remark 3.3.1. Recall that MEMORYLESS-GREEDY-FIXED-PRIORITY $<$ PRIORITY. Notice that if the predicate (*) holds, we showed that no optimal algorithm from the class PRIORITY (for the one-machine environment) exists. On the other hand if this combinatorial property does not hold, then we showed that an algorithm from MEMORYLESS-GREEDY-FIXED-PRIORITY (for the one-machine environment) is optimal. Thus, although we have shown, in chapter 2, that greediness, fixed ordering and the memoryless property are weaker (w.r.t. simulation), this is not true for the case of optimal priority algorithms.

3.3.2 The general case

When the intervals of the input set have possibly non-distinct profits, then we need to extend the previous condition so as to make it necessary and sufficient for the non-existence of an optimal priority algorithm. Even though the condition can be strengthened in a simple way, the proof that the new condition is necessary and sufficient is more technical than in the distinct profits case.

Definitions - Preliminaries for sets of intervals

Definition 3.3.1. Say that we are given a finite set of intervals $S = S' \cup \{J_a, J_b\}$, where $S' = \{J_1, \ldots, J_n\}$ is a set of intervals each of which is of profit $p \in \mathbb{Z}^{\geq 0}$, and J_a, J_b have profit less than p. We call a sequence $T = \langle J_a, J_{i_1}, J_{i_2}, \ldots, J_{i_k}, J_b \rangle$, $J_{i_j} \in S'$, $1 < k \leq n$, a *bad chain* if every element of T overlaps only with the previous (if any) and the next

(if any) interval in the sequence. A sequence like T, without J_a and J_b, is called *simple chain.*

Proposition 3.3.3. *Let $S = \{J_1, \ldots, J_n\}$ be a set of intervals. If there exists a bad chain in S then there does not exist a priority algorithm $A \in$ PRIORITY which is optimal on every subset of S.*

Proof. Let $T = \langle J_a, J_{i_1}, J_{i_2}, \ldots, J_{i_k}, J_b \rangle$, $2 \leq k$ be the bad chain. Let $A \in$ PRIORITY having an initial ordering starting with J_a. First consider the case where A reads J_a and rejects it. Then, on the input set $\{J_a\}$ it does not perform optimally. If A accepts J_a, then on input $\{J_a, J_{i_1}\}$ it does not perform optimally. Similarly, for J_b. If the ordering starts with J_{i_1} and rejects, A performs non-optimally on $\{J_{i_1}\}$. If A accepts J_{i_1} then on input $\{J_a, J_{i_1}, J_{i_2}\}$ performs non-optimally. Similarly we reason for the case where the ordering starts with J_{i_j}, $1 \leq j \leq k$. □

Definition 3.3.2. Given a set S of intervals and an interval $J \in S$ we define $c(J, S)$ to be the profit of an optimal schedule among all intervals overlapping J which have smaller profit than J. We call $c(J, S)$ the *overlapping characteristic* (or simply "characteristic") of J (w.r.t. S). If T is a simple chain where all intervals have not only the same profit but the same overlapping characteristic as well, we call T a *simple chain of equal characteristic.*

Obviously, an interval which does not overlap any other interval of smaller profit, has overlapping characteristic 0.

Definition 3.3.3. Let S be a set of intervals. We define the equivalence relation $Q_S \subseteq S \times S$ "reachable through overlapping intervals of the same profit and characteristic" as follows: $(J, J') \in Q_S$ if there exists a sequence $T = \langle J = J_{i_1}, \ldots, J_{i_k} = J' \rangle$ such that $T \subseteq S$, all intervals in T have the same profit and characteristic and for every $1 \leq j < k$ J_{i_j} overlaps at least $J_{i_{j+1}}$.

If no confusion arises we omit the subscript S and we simply write Q.

Proposition 3.3.4. *The relation Q_S, "reachable through overlapping intervals with the same profit and the same characteristic", is an equivalence relation.*

Definition 3.3.4. Let S be a set of intervals, where every $J \in S$ is of the form $J = (r_J, d_J, p_J)$, $r_J < d_J$, $r_J, d_J, p_J \in \mathbb{Z}^{\geq 0}$, where r_J, d_J and p_J are the release time, the deadline time and the profit of interval J respectively.

- $min_r(S)$ is the minimum release time among the intervals in S.
- $max_d(S)$ is the maximum deadline among the intervals in S.
- We write that an *interval J completely overlaps an interval J'* if $r(J) \leq r(J')$ and $d(J) \geq d(J')$.
- We write that an *interval J is on the right of an interval J'* if $r(J) > r(J')$ and $d(J) > d(J')$.
- We write that an *interval J is on the left of an interval J'* if $r(J) < r(J')$ and $d(J) < d(J')$.

Observe that if an interval J completely overlaps another interval J', then J' must have less or equal overlapping characteristic than J.

Proposition 3.3.5. *If S is a set of intervals of the same profit and $[J]_{Q_S}$, $J \in S$, is an equivalence class of Q_S then for every two $J', J'' \in [J]_{Q_S}$, such that neither of J', J'' completely overlaps the other, there exists a simple chain of the form $\langle J', \ldots, J'' \rangle$.*

Proof. Follows from the definition of Q_S, where the construction of the chain is done by choosing intervals having the greatest finishing time among those that fulfill the properties of forming a simple chain. □

We introduce the following notation. Given an interval $J \in S$, by the set S_J we denote the set of all intervals in S that overlap J and have profit less than the profit of J.

An optimal priority algorithm for optimal sets of intervals with arbitrary profits

In section 3.3.1 we studied the specific structure that an optimal set of intervals must have, when the profits of the intervals are distinct. Notice that lemma 3.3.1 holds for the case where we allow intervals of the same profit, as long as they do not overlap each other. Now, we allow overlapping intervals of equal profit. In this case we want to have a closer look, to focus, on the "window of time" where intervals of equal profit overlap each other. Proposition 3.3.6 states, that in such a "window" we expect to find more "structure" than in the case of lemma 3.3.1. From the following proposition we are only

going to use in the subsequent lemmata part (3). The rest are provided so as to make this proof clearer.

Proposition 3.3.6. *Let S be a set of intervals. Assume that S does not contain a bad chain.*

1. *Let $T = \langle J_1, \ldots, J_k \rangle$, $J_i \in S, k \geq 2$ be a simple chain of equal characteristic. Then, there does not exist an interval $J \in S$ with the same profit as the intervals in T and with bigger overlapping characteristic than the intervals in T, and such that $d_J \leq d_{J_k}$ and $r_J \geq r_{J_1}$ (that is, T completely overlaps J).*

2. *Say that $[J]_Q$, $J \in S$, overlaps an interval $J' \in S$, where J' has greater overlapping characteristic than J and the same profit as J. Let $r' = min_r([J]_Q)$ and $d' = max_d([J]_Q)$. If J' does not completely overlap (r', d') then J' is either on the left or on the right of (r', d') and there exists an interval $\hat{J} \in S$ that overlaps J' and does not overlap any interval from $[J]_Q$ and $\hat{J}$ has lower profit than J.*

3. *Say that $[J]_Q$ overlaps two intervals $J', J'' \in S$, where J', J'' each has greater overlapping characteristic than J and the same profit as J. Let $r' = min_r([J]_Q)$ and $d' = max_d([J]_Q)$. If neither of J', J'' completely overlaps (r', d') then either they both are on the left or they both are on the right of (r', d').*

Proof.
(1) Let $T = \langle J_1, \ldots, J_k \rangle, k \geq 2$ be a simple chain of equal characteristic. Suppose, that there exists an interval $J \in S$ having the same profit as each interval in T and $d_J \leq d_{J_k}$ and $r_J \geq r_{J_1}$, and such that it has greater overlapping characteristic than each of the intervals in T. Recall that S_J denotes the set of intervals each of which overlaps J and such that each interval in S_J has smaller profit than J. Say that OPT_{S_J} is an optimal feasible schedule of S_J. Let$J_L \in OPT_{S_J}$ be the interval in OPT_{S_J} with the smallest finishing time (the "leftmost" one) and $J_R \in OPT_{S_J}$ be the interval with the largest starting time (the "rightmost" one) in OPT_{S_J}. Let $J' \in T$ be the interval with the largest starting time that overlaps[4] J_L and $J'' \in T$ be the interval with the smallest finishing time that overlaps J_R. It cannot be the case that J' overlaps with both J_L and J_R. If this holds then J' is of greater characteristic than we assumed. Therefore $J' \neq J''$. Thus, the sub-chain of T, $T' = \langle J', \ldots, J'' \rangle$ together with J_L and J_R form the bad chain $\langle J_L, J', \ldots, J'', J_R \rangle$.

[4]If there are more than one such intervals then choose the later one in the sequence according to T.

(2) If the interval $(r', d') \in [J]_Q$ then the result is obvious. Otherwise, there exists a simple chain of equal characteristic T that extends from r' to d'. From this point we use (r', d') to refer to the intervals of T. From (1) we know that (r', d') cannot completely overlap J' (that is, $r_{J'} \geq r'$ and $d_{J'} \leq d'$). That is, J' is either on the left or on the right of (r', d'). $S_{J'}$ must contain an interval not overlapping (r', d'); if not then (r', d') overlaps every interval in $S_{J'}$, and we can reason as in (1) that S contains a bad chain. Hence, J' overlaps (r', d') either from the left or from the right and there exists an interval $\hat{J}$ of smaller profit than J' which overlaps J' and does not overlap (r', d').

(3) This is a corollary of (2). Assume that J' is on the left of J''. If J', J'' overlap each other then the result is obvious. Suppose that J' and J'' do not overlap each other. If $(r', d') \in [J]_Q$ then from (2) there exist a $\hat{J}_1 \in S$ of smaller profit than J' that overlaps J' and does not overlap (r', d'). There is also a $\hat{J}_2 \in S$ of smaller profit than J'' that overlaps J'' and does not overlap (r', d'). Therefore, $\langle \hat{J}_1, J', (r', d'), J'', \hat{J}_2 \rangle$ is a bad chain. If $(r', d') \notin [J]_Q$ then there is a simple chain of equal characteristic T that extends from r' to d'. Let $\langle J_{i_1}, \ldots J_{i_l} \rangle, l \geq 1$ be the sub-chain of T where J_{i_1} is the rightmost interval in T that overlaps J' and J_{i_l} is the leftmost interval in T that overlaps J''. There also exist $\hat{J}_1$ and $\hat{J}_2$ as previously. Hence, $\langle \hat{J}_1, J', J_{i_1}, \ldots J_{i_l}, J'', \hat{J}_2 \rangle$ is a bad chain. □

The following algorithm (ELPRF) is a modification of the LPRF. LPRF works optimally (given the preconditions of section 3.3.1). Given the preconditions of lemma 3.3.7 we shall prove that ELPRF is also optimal. Notice that the steps the algorithm performs are "driven/triggered" from the previous mentioned lemmata. Especially, the most complicated steps of the algorithm stem from the structure of the set as described in proposition 3.3.6. Proposition 3.3.6(3) restricts the cases where an interval with greater overlapping characteristic, but the same profit, can overlap an equivalence class $[J]_Q$, given that there is no bad chain in the input set.

Algorithm. ELPRF **(Extended - Largest Profit First)**

Input: A set $S = \{J_1, J_2, \ldots, J_n\}$ of intervals.

Order the intervals in non-increasing profit order. For intervals with the same profit

order them in non-decreasing characteristic. Say that J, J' are two intervals having the same profit and the same characteristic:

1. We order J, J' according to the Q-equivalence class. That is, if $[J]_Q \neq [J']_Q$ and every interval in $[J]_Q$ is to the left of every interval in $[J']_Q$, assign to every interval in $[J]_Q$ higher priority than every interval in $[J']_Q$.

2. Let $I = (min_r([J]_Q), max_d([J]_Q))$. If there exists an interval $\hat{J}$ of the same profit as J, but higher characteristic, such that $\hat{J}$ is on the right of I then order $[J]_Q$ in *non-decreasing* finishing time order. Otherwise, order $[J]_Q$ in order of *non-increasing* starting time.

Remark 3.3.2. It seems interesting to discuss, at a more intuitive level, why we gave such a complicated algorithm. Recall that in section 3.3.1 we showed that the LPRF (Largest Profit First) is optimal (given the preconditions of section 3.3.1) on sets of intervals where each interval is of distinct profit. Therefore, it seems plausible to try to devise an algorithm that first orders the intervals in non-increasing profit order and then deals with intervals of the same profit. The instance in figure 3.2 satisfies the preconditions of lemma 3.3.7 below, and therefore there exists an ordering which is optimal. Let R be the equivalence relation which is defined the same as Q only we ignore the equal overlapping characteristic constraint. That is, the equivalence classes of R contain intervals of the same profit that are "reachable through overlapping". Motivated by algorithms for similar scheduling problems, one might hope that there will be an optimal ordering with the property that each equivalence class of R is ordered either in non-decreasing finishing time or in non-increasing starting time. In figure 3.2 we give an instance where every algorithm that first schedules either J_1 (the interval with the smallest finishing time) or J_4 (the interval with the largest starting time) is not optimal (on every subset of the input). It is also obvious that a similar instance can be created for every number of intervals in the input. Furthermore, we observe that this input instance rules out not only the class of algorithms that order the R-equivalence classes in a "left-to-right" fashion or in a "right-to-left" fashion but also rules out every algorithm that alternates between "left-to-right" and "right-to-left" ordering within an equivalence class.

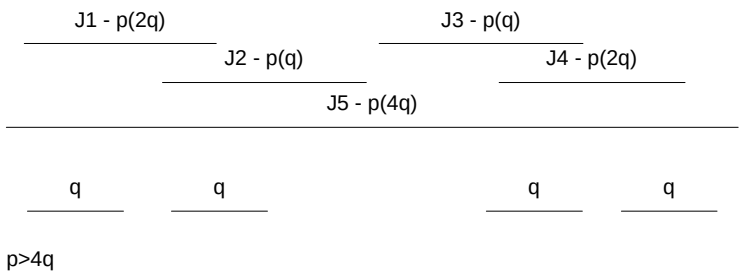

Figure 3.2: Example, where the "class of simple" algorithms fails. p and q, $p > 4q$, are interval profits. $J_1, \ldots, J_5$ are intervals each of profit p. We have written into parenthesis the overlapping characteristic of each interval of profit p. Notice that in every optimal ordering it is necessary that $J_2 > J_1$ and $J_3 > J_4$. For example an optimal ordering is: $J_2 > J_3 > J_1 > J_4 > J_5 >$ (the profit q intervals).

Characterization of optimal sets of intervals of arbitrary profits

Lemma 3.3.7. *Given a finite set of intervals $S = \{J_1, \ldots, J_n\}$, if*
(a) the negation of (*) (from lemma 3.3.1) holds. That is, for every interval $J \in S$ and every set $S' \subseteq S$, where every interval in S' overlaps J and has less profit than J and no two intervals in S' overlap each other, we have that the total profit of intervals in S' is less than or equal to the profit of J,
and
(b) there is no bad chain in S,
then ELPRF *is optimal on every subset of S.*

Proof. We will use lemma 3.2.1. Suppose that ELPRF $\in$ M-G-F-PRIORITY is not optimal. Let $J \in S$ and $T \subseteq S$ and assume (i),(ii),(iii), (iv) of lemma 3.2.1. That is, assume:

i. every interval in T overlaps J

ii. every two intervals in T do not overlap each other

iii. $profit(T) > profit(\{J\})$

iv. for all $J' \in T$, J has higher priority than J'.

If every interval in T has (strictly) smaller profit than J then this contradicts (a). Thus, T contains at least one interval of profit equal to the profit of J. It is sufficient to look

at the cases where $T = \{J', J''\}$, J' is to the left of J'' and $profit(J') = profit(J)$, $profit(J'') \leq profit(J)$.

- Case: $profit(J') = profit(J) = profit(J'')$. Because of the way the algorithm works $c(J, S) \leq c(J', S)$ and $c(J, S) \leq c(J'', S)$.

  - Suppose that J, J', J'' have equal overlapping characteristic. Since they belong to the same equivalence class ELPRF orders them either by increasing finishing time or by decreasing starting time. J' and J'' do not overlap each other and they both overlap J. Trivially, if J completely overlaps either one of J', J'' then it has lower priority than the one it completely overlaps. If J's finishing time is less than or equal to that of J' then it cannot overlap J''. Similarly, J's starting time cannot be bigger than the starting time of J'' (it cannot overlap J'). In both cases J cannot possibly have greater priority than J' or J''.

  - Suppose that J, J', J'' do not have equal overlapping characteristic. If J has smaller overlapping characteristic than each of J', J'' then it cannot completely overlap either one of them. In this case, since J', J'' do not overlap each other there exists a $\hat{J}_1$ that overlaps J', does not overlap J and is of smaller profit than J'. Similarly, there exists a $\hat{J}_2$ that overlaps J'', does not overlap J and is of smaller profit than J''. Therefore, $\langle \hat{J}_1, J', J, J'', \hat{J}_2 \rangle$ is a bad chain. That is, J must have equal characteristic to at least one of J', J''. Say that $c(J, S) = c(J', S)$, so $c(J, S) = c(J', S) < c(J'', S)$. Note that $J' \in [J]_Q$ and J'' is completely to the right of J'. This implies that J'' does not completely overall $[J]_Q$ and it is not to the left of $[J]_Q$. Since there is no bad chain in S, proposition 3.3.6(3) therefore tell us that J'' is to the right of $[J]_Q$. By examining how the algorithm operates, we see that $[J]_Q$ (recall that $J, J' \in [J]_Q$) is ordered in increasing finishing time. Hence, J' has higher priority than J, which is a contradiction. Therefore, we must have $c(J, S) = c(J'', S) < c(J', S)$. As above this means that J' is to the left of $[J]_Q$. By proposition 3.3.6(3) there cannot be a J''' to the right of $[J]_Q$, where J''' is of the same profit as J and has greater overlapping characteristic than J. Therefore, the algorithm will order $[J]_Q$ in decreasing starting time. Hence, J'' has higher priority that J, which is a contradiction.

- Case: $profit(J') = profit(J)$, $profit(J'') < profit(J)$. Because of the way the algorithm works $c(J, S) \leq c(J', S)$. Therefore, there exists an interval $\hat{J} \in S$ that overlaps J', does not overlap J and has smaller profit than J. Hence, $\langle \hat{J}, J', J, J'' \rangle$ is a bad chain.

□

We are ready to give the main lemma of this section, which is a corollary of the previously mentioned lemmata.

Lemma 3.3.8. *Let a finite set of intervals $S = \{J_1, \ldots, J_n\}$. There does not exist an $A \in$ PRIORITY, where A is optimal on every subset of S iff*
(a) (*) (from lemma 3.3.1) holds
or
(b) there exists a bad chain in S.

Proof. We show that if (a) or (b) holds, then there is no optimal priority algorithm. The proof for (a) follows from the same argument in the proof of the lemma 3.3.1. (b) is the proposition 3.3.3. We show the other direction by its contrapositive argument, which is the statement of lemma 3.3.7. □

As a corollary of the above lemma we prove the following theorem. In section 3.4 we prove the stronger theorem EXISTS-OPT-INTERVAL$(m = 1) \in U_L - NC^2$. For this stronger result and for theorem 3.3.10 we use the following lemma.

Problem 7. EXISTS-BAD-CHAIN
Instance: Let S be a finite set of intervals.
Question: Does there exist a bad-chain in S?

Lemma 3.3.9. EXISTS-BAD-CHAIN $\in L$

Proof. The deterministic logspace bounded algorithm that decides whether a bad chain exists is straightforward (as in lemma 3.3.5). Let J_a, J_b and $\hat{J}_a, \hat{J}_b$ be 4 intervals from the input, where $profit(\hat{J}_a), profit(\hat{J}_b) < profit(J_a) = profit(J_b)$. Obviously we can keep 4 indexes in logspace for each of them and counters so as to examine every 4 such intervals. Furthermore we can check in logspace whether 2 intervals overlap each other. For every 4 such intervals we examine whether a bad chain can be formed by checking whether we can find an interval of profit equal to the profit of J_a (and J_b) that "goes farthest". If by

adding this interval a bad chain is not formed (and these intervals can still form a bad chain) we keep the interval we previously added and we repeat by checking whether we can add a new interval (of the same profit as J_a) that "goes farthest". □

Theorem 3.3.10. Exists-Opt-Interval($m = 1$) *is in* P.

Proof. For every interval J in the input set, we compute the set of its overlapping and less profitable intervals S_J. Then, we find the maximum total profit and we use lemma 3.3.8 to prove non-optimality. This works within polynomial time. If we haven't proved non-optimality then we use the algorithm from the proof of lemma 3.3.9 to test whether there exists a bad chain. □

In section 3.4 we show that the following Interval Scheduling problem is in logspace uniform NC^2.

Problem 8. Interval-Scheduling($m = 1$)
Instance: Let $S = \{J_1, \ldots, J_n\}$, $n \in \mathbb{Z}^+$ be a set of intervals and $B \in \mathbb{Z}^+$ a bound.
Question: Does there exist a feasible schedule $S' \subseteq S$, where $profit(S') \geq B$?

Remark 3.3.3. In section 3.4 we show that Opt-Exists-Interval($m = 1$) is in NC^2. It is worth noting that ELPRF is also in $FSIZE - DEPTH(n^{O(1)}, (\log n)^2)$. This is due to the fact that Interval-Scheduling($m = 1$) $\in U_L - NC^2$ and that we can decide in deterministic logarithmic space whether a bad chain exists, even for the arbitrary profits case (lemma 3.3.9). If each profit has value which is polynomial in the number of intervals then we can implement ELPRF in the function analog of NL.

3.4 Languages in uniform NC^2

In this section we show the following theorem.

Theorem 3.4.1. Opt-Exists-Interval($m = 1$), Interval-Scheduling($m = 1$) *and* Opt-GF-Interval($m = 1$) *are in* $U_L - NC^2$.

This theorem follows as a corollary of the following lemmata and lemma 3.3.9.

Remark 3.4.1. We can reduce the problem of computing the optimal schedule of a set of intervals to the problem of computing the longest (vertex) path in a directed acyclic graph where weights have been assigned to the vertices. This can be easily solved by

the use of Dynamic Programming. It is worth noting that in the context of scheduling theory these types of graphs have been studied extensively. These graphs are called *vertex weighted disjunctive graphs* and the longest (vertex) path in such a graph, corresponding to an optimal schedule, is called *critical path.*

In this work we use the term "path in a graph" to denote a "simple path in a graph". We use the term "walk in a graph" to denote a "not-necessarily simple path in a graph". Sometimes we use the term "path" to refer to a "walk" but this is done only for consistency with the literature and where no confusion arises. For example, in the literature the term "path" is used to denote a walk in a digraph w.r.t. the st-connectivity problem.

Problem 9. DAG-LONGEST-PATH
Instance: Let $G = (V, E)$ be a directed weighted acyclic graph and $w : E \to \mathbb{Z}$. Let $B \in \mathbb{Z}$.
Question: Does there exist a directed path of (weighted) length greater or equal to B?

It is obvious that a directed (weighted) longest walk on a DAG is not infinite and coincides with a directed longest path in the same graph. The fact that DAG-LONGEST-PATH is in NC (not necessarily in NC^2) can be taken as a simple corollary of [16, 13] through the theorems that interface the PRAM model and the bounded fan-in circuits. Since these works are mainly interested in tradeoffs between the number of processors and parallel time, they are not quite suitable for our purposes. We need a much simpler result that only deals with the depth of a bounded fan-in circuit. The following lemma can be shown using "standard" ideas used in the literature of parallel algorithms (for example see [15]).

Lemma 3.4.2. DAG-LONGEST-PATH $\in U_L - NC^2$

Proof. Let $G = (V, E), w : E \to \mathbb{Z}$ be a weighted directed acyclic graph. Let $n = |V|$ and each vertex be one element among $\{1, \ldots, n\}$. We denote as $D^G_{n\times n}$ the distance matrix of (G, w), where the $d_{i,j} = w((i, j))$. If $(i, j) \notin E$ then $d_{i,j} = -\infty$. We write D, instead of $D^G_{n\times n}$ if no confusion arises. We first show the following proposition.

Proposition 3.4.3. *Say that D is a distance matrix of a weighted directed graph (G, w), where $G = (V, E), w : E \to \mathbb{Z}$. The $a_{i,j}$ element of D^k corresponds to the longest walk of k edges on (G, w), given that the operations used for the matrix multiplication are the binary operations* $\max, +$, *where $\mathbb{Z} \cup \{-\infty\}$ are monoids under max and $+$, instead of the field $(\mathbb{Z}, \cdot, +)$.*

Proof. Straightforward induction on k. □

We compute D^n by using repeated squaring. Thus, we need $\lceil \log(n) \rceil$ matrix multiplications. Since, $ADD, MAX \in FSIZE - DEPTH\left(n^{O(1)}, \log(n)\right)$ we can compute matrix multiplication over $(\max, +)$ in $FSIZE - DEPTH\left(n^{O(1)}, \log(n)\right)$. Hence, we can compute D^n in $FSIZE - DEPTH\left(n^{O(1)}, (\log(n))^2\right)$. That is, we can compute a longest walk of length n in $FSIZE - DEPTH\left(n^{O(1)}, (\log(n))^2\right)$, since in a weighted DAG the maximum element of D^n corresponds to the longest path in the given weighted directed acyclic graph. If we compare each element with the given B for each matrix entry we can compute 1 iff B is less or equal than the corresponding element. Hence, we can put an unbounded fan-in OR gate and accept if there exists such an element. This OR gate can be trivially simulated with a complete binary tree, of bounded to 2 fan-in OR gates, of depth order of a logarithm in its fan-in. □

In section 3.7 we show that POLY-INTERVAL-SCHEDULING($m = 1$) is complete for NL. We conjecture that INTERVAL-SCHEDULING($m = 1$) is not in NL. Under this conjecture the following theorem seem to be some interest.

Lemma 3.4.4. INTERVAL-SCHEDULING($m = 1$) *is logspace reducible to* DAG-LONGEST-PATH.

Proof. Given an instance x of INTERVAL-SCHEDULING($m = 1$) we reduce it, within logarithmic space, to an instance of DAG-LONGEST-PATH. Let S be a finite set of intervals and B be an integer. The instance for the DAG-LONGEST-PATH is the same integer B and the weighted graph constructed as follows. Order the intervals of S in increasing finishing time. Say that the sequence is $T = \langle J_1, \ldots, J_n \rangle$. We construct a set of vertices $V = \{u_1, \ldots, u_n, u_{n+1}\}$. Each of the first n vertices corresponds to an interval in T. Say that the vertex u_i corresponds to the interval J_i. From a vertex u_i we put an edge to a vertex u_j, if $j > i$ and J_i does not overlap J_j. The weight of (u_i, u_j) is equal to the profit of J_i. Finally, we add an edge from each $u_i \in V \backslash \{u_{n+1}\}$ to u_{n+1} and we assign weight equal to the profit of J_i. Observe that the corresponding graph is a weighted directed acyclic graph. Notice that $\langle u_1, \ldots, u_{n+1} \rangle$ is a topological ordering of its vertices and thus there is no path from a vertex u that corresponds to an interval J to a vertex v that corresponds to an interval J' if J overlaps J'. It is obvious that the above construction, which computes a function $f : \Sigma^* \to \Sigma^*$, works within logarithmic space. We must show that $x \in$ INTERVAL-SCHEDULING($m = 1$) iff $f(x) \in$ DAG-LONGEST-PATH.

Suppose that $x \in$ INTERVAL-SCHEDULING$(m = 1)$. Then there exists a set of non-overlapping intervals (that is, a feasible schedule) $S' = \{J_{i_1}, \ldots, J_{i_k}\} \subseteq S$ where the intervals have total profit greater or equal to B. Each J_{i_j}, $1 \leq j \leq k$ corresponds to a vertex u_{i_j} in the constructed graph. W.l.o.g. $J_{i_1}, \ldots, J_{i_k}$ are the intervals in increasing finishing time order. By construction there exists an edge from u_{i_j} to $u_{i_{j+1}}$, $j < k$ and there also exists an edge from u_{i_k} to u_{n+1}. Also, by construction the total profit of the path equals to $profit(S')$.

Suppose that $f(x) \in$ DAG-LONGEST-PATH. Then there exists a path $\langle u_{i_1}, \ldots, u_{i_k} \rangle$ of total weight $B' \geq B$. Again, by construction, the set $\{J_{i_1}, \ldots, J_{i_{k-1}}\}$ is a feasible schedule of total weight $B' \geq B$. □

We take as a corollary INTERVAL-SCHEDULING$(m = 1) \in U_L - NC^2$. The following lemmata have very similar proofs as the previous one.

Lemma 3.4.5. EXISTS-OPT-INTERVAL$(m = 1)$ *is logspace reducible to* DAG-LONGEST-PATH.

Proof. Check whether there exists a bad chain and for every interval in the given set repeat the construction in the proof of lemma 3.4.4 (for the set of intervals of profit smaller than p that overlap an interval of profit p). □

Lemma 3.4.6. OPT-GF-INTERVAL$(m = 1)$ *is logspace reducible to* DAG-LONGEST-PATH.

Corollary 3.4.7. EXISTS-OPT-INTERVAL$(m = 1)$, OPT-GF-INTERVAL$(m = 1) \in U_L - NC^2$

3.5 Languages in L

In the "open problems" section 3.9 we conjecture about the complexity of POLY-NON-OPT-GF-INTERVAL$(m = 1)$ and POLY-EXISTS-OPT-INTERVAL$(m = 1)$. When we further restrict these languages to the case of distinct profit intervals then the following lemma holds.

Lemma 3.5.1. POLY-OPT-GF-INTERVAL$(m = 1,$ DISTINCT PROFIT$)$ *and* POLY-EXISTS-OPT-GF-INTERVAL$(m = 1,$ DISTINCT PROFIT$)$ *are in* L.

Proof Sketch. Straightforward application of lemma 3.2.1. Let S be the set of intervals. Consider the intervals in increasing profit order. Iterate through intervals in increasing

profit order and successively build a set S_{OPT}. For every interval simulate LPRF on the set of overlapping intervals. If the profit (of the optimal schedule computed by LPRF) is greater than the interval under consideration then reject. Else continue with the next interval in the order. If every interval has been considered without rejecting then accept. □

3.6 Priority Algorithms for intervals of equal profits

In this section we deal with greedy-fixed priority algorithms for one processor machine environments. Recall (from chapter 2) that for the one machine environments G-F-PRIORITY $\approx$ M-G-F-PRIORITY. Such algorithms are well-defined if we only describe the ordering of the intervals. Lemma 3.6.1 characterizes an optimal priority algorithm for the case where all intervals are of the same profit and follows as a corollary of lemma 3.2.1.

Lemma 3.6.1. *Let S be a set of intervals of the same profit and $T = \langle J_1, \ldots, J_n \rangle$ be a sequence of these intervals. The greedy-fixed priority algorithm that defined through T is not optimal (on every subset of S) iff there exist three intervals $J, J', J'' \in T$, where J overlaps J', J'' and J', J'' do not overlap each other and J has higher priority than both J, J''.*

It is interesting to notice that the well-known optimal priority algorithms, for the interval scheduling problem where the profits are equal, $SFTF$ (Shortest Finishing Time First) and $LSTF$ (Largest Starting Time First) trivially satisfy the optimality condition derived from lemma 3.6.1. Although, we do not intend to characterize every such optimal algorithm we just state a few, easy to show, lemmata that mix the $SFTF$ and $LSTF$ strategies.

In what follows we assume that an instance for NON-OPT-GF-INTERVAL-EQUAL-PROFIT consists of n pairs of intervals, where every pair is of the form (r, d) and every interval is an n-bit long number in binary.

Lemma 3.6.2. NON-OPT-GF-INTERVAL-EQUAL-PROFIT $\in U_D - AC^0$

Proof. It is straightforward to show containment in AC^0. For the case of deterministic logtime uniform AC^0 we use an alternative characterization. It suffices to show

that NON-OPT-GF-INTERVAL-EQUAL-PROFIT $\in FO[<, bit] = U_D - AC^0$. What follows is a straightforward technicality. From section 1.6.2 we have that for $FO[<, bit]$ we can have the predicates $PLUS$ and $TIMES$. We use all kinds of shorthand notation for predicates (e.g. say that P is a predicate of arity 1, then $P(a+b)$ stands for $\exists c\, PLUS(a,b,c) \wedge P(c)$). Since, we have n intervals (pairs) of n-bit long numbers we can obtain n $(\exists m\ (LENGTH(m) \wedge (\exists n\ 2n^2 = m)))$ and thus the release time of the $k-th$ interval starts from the bit $(k-1)n+1$ and the deadline from $kn+1$. Recall that the predicate $B(i)$ is true iff the i-th bit of the input is 1. The following first-order formula checks whether the release time of the k-th interval is greater than the deadline time of the h-th interval:

$\exists i\ (i \leq n) \wedge B((k-1)n+i) \wedge \neg B(hn+i)$

$\wedge\Big(\forall l\big((i < l \leq n) \wedge B(hn+l) \rightarrow B((k-1)n+l)\big) \vee \big(l > n\big)\Big)$

This formula checks whether there exists a bit i of the release time of the k-th interval that it is 1 and the corresponding bit of the deadline of the h-th interval is 0 $(B((k-1)n+i) \wedge \neg B(hn+i))$. Furthermore, every higher than i bit of the deadline must be less than or equal to the corresponding bit of the release time $(B(hn+l) \rightarrow B((k-1)n+l))$. We can trivially write a first-order formula which is satisfied by the standard model for $FO[<, bit]$. The formula checks whether there exist three interval k_1, k_2, k_3 where the deadline time of the second is greater than the release time of the first and the release time of the third is less than the deadline of the first and the deadline of the second is less or equal to the release time of the third (lemma 3.6.1). □

Lemma 3.6.3. *Let S be a set of intervals and*

$T = \langle \underbrace{J_1, J_2, \ldots, J_{i-1}}_{\text{"increasing finishing time"}}, \underbrace{J_i, J_{i+1}, \ldots, J_n}_{\text{"decreasing starting time"}} \rangle$ *be a sequence of these intervals, where every $J \in \{J_i, J_{i+1}, \ldots, J_n\}$ has finishing time greater than or equal to the finishing time of J_{i-1}. The greedy-fixed priority algorithm with ordering T is optimal on every subset of $S = \{J_1, J_2, \ldots, J_n\}$.*

This lemma cannot be extended in an obvious way to a recursive type of lemma without imposing a further restriction in the finishing time of such a partition. Under this restriction we get the following lemma.

Lemma 3.6.4. *Let S be a set of intervals and $T = \langle J_1, \ldots, J_{k-1}, J_k, \ldots, J_l, J_{l+1}, \ldots, J_n \rangle$ be a sequence of these intervals in increasing finishing time order.*

Let $T' = \langle \underbrace{J_{i_1}, J_{i_2}, \ldots, J_{i_a}}_{\text{"increasing finishing time"}}, \underbrace{J_{j_1}, J_{j_2}, \ldots, J_{j_b}}_{\text{"decreasing starting time"}}, \underbrace{J_{h_1}, J_{h_2}, \ldots, J_{h_c}}_{\text{"arbitrary optimal ordering"}} \rangle$,

where $J_{i_u} \in \{J_1, \ldots, J_{k-1}\}$, $J_{j_u} \in \{J_{l+1}, \ldots, J_n\}$ *and* $J_{h_u} \in \{J_k, \ldots, J_l\}$ *and no interval in* $\{J_k, \ldots, J_l\}$ *has greater starting time than an interval in* $\{J_{l+1}, \ldots, J_n\}$. *The greedy-fixed priority algorithm with ordering* T' *is optimal on every subset of* $S = \{J_1, J_2, \ldots, J_n\}$.

3.7 POLY-INTERVAL-SCHEDULING($m = 1$) is complete for NL

Problem 10. POLY-INTERVAL-SCHEDULING($m = 1$)
Instance: Let $S = \{J_1, \ldots, J_n\}$, $n \in \mathbb{Z}^+$ be a set of intervals with profits having values a polynomial in n, and $B \in \mathbb{Z}^+$ a bound.
Question: Does there exist a feasible schedule $S' \subseteq S$, where $profit(S') \geq B$?

We show that POLY-INTERVAL-SCHEDULING($m = 1$) is complete for NL w.r.t. logspace many-to-one reductions. For this result we need the NL-completeness of $STCON(k)$ which has a well-known folklore proof.

Problem 11. STCON(K)
Instance: Let $G = (V, E)$ be a digraph, $s, t \in V$ and $k \in \mathbb{Z}^+$ and $k \leq n$.
Question: Is there a directed (not necessarily simple) path from s to t of k edges?

Lemma 3.7.1. *$STCON(k)$ is $NL-$complete.*

Proof. Trivial. Sketch: It is easy to modify the algorithm for the $STCON$ such that we keep a counter for the number of visited vertices. We can update this counter within logspace. For the hardness direction we reduce $STCON$ to $STCON(k)$. Given the digraph $G = (V, E)$ we construct a new graph by adding a self-loop edge to s. □

Theorem 3.7.2. POLY-INTERVAL-SCHEDULING($m = 1$) *is complete for NL w.r.t. logspace many-to-one reductions.*

Proof. In section 3.2 we show containment of POLY-INTERVAL-SCHEDULING($m = 1$) in NL. We reduce $STCON(k)$ to POLY-INTERVAL-SCHEDULING($m = 1$). Let the instance $G = (V, E)$, $V = \{u_0, \ldots, u_{n-1}\}$, $s = u_0$, $t = u_{n-1}$ and $k \in \mathbb{Z}^+$. We will associate with each of the vertices $k + 1$ points of the time line consisting of the non-negative

integers. First associate the $s = u_0, u_1 \ldots, u_{n-1} = t$ with points (of time) $0, 1, \ldots, n-1$ respectively. Also associate these vertices with points $n, n+1, \ldots, 2n-1$; etc ending with points $kn, \ldots, (k+1)n-1$. Thus, in total we can see that there exist $k+1$ blocks of time points. For each edge in the graph we create k intervals with proportional profit. For each edge (u_i, u_j) and for each $0 \leq l < k$ we create an interval from the point corresponding to u_i in block l to the point corresponding to u_j in block $l+1$; that is, we create the interval $(ln+i, (l+1)n+j)$. The constructed instance for the POLY-INTERVAL-SCHEDULING$(m=1)$ is the set S and the desired bound is equal to $nk-1$.

If there exists a path of length k then we show that there exists a schedule of total profit nk. Let that path be $\langle s = u_0, u_{i_1}, \ldots, u_{k-1} = t \rangle$. Then there exists an interval from the point of s to that of u_{i_1} of the first block. There also exists an interval from point u_{i_1} of the 1st block to the point u_{i_2} of the second. Similarly, we show that we can schedule intervals from s to t: $(0n+0, 1n+i_1), (1n+i_1, 2n+i_2), \ldots, ((k-1)n+i_{k-1}, (k-1)n+n)$. Obviously this schedule does not leave any "gap", and since we consider intervals of proportional profits the total profit equal to the distance between s and t which is $nk-1$.

Conversely, let S be a schedule of profit $nk-1$. Since each interval of S starts from a point which corresponds to a vertex v and finishes at a point which corresponds to a vertex u if the edge $(u, v) \in E$, then there exists a path from s to t which corresponds to the points listed in the shortest finishing time first ordering. This path is of length k since between two successive blocks we can schedule only one interval and an interval is extended exactly between two successive blocks. □

Remark 3.7.1. It is clear that the same reduction works for the language where we consider an exactly equal total profit for the desired bound.

3.8 Optimal sets of jobs

What if we allow jobs and multiple-machine environments? We study normal forms for priority algorithms for the Job Scheduling problem (on possibly multiple processors). We are motivated by the study of the complexity of languages related to the question of whether a given set of jobs admits an optimal priority algorithm. We attempt to restrict the possible forms an optimal priority algorithm may have. As the following example shows the situation is different than in case of interval sets; where ELPRF $\in$

M-G-F-PRIORITY.

Counter-example. The following set of distinct proportional profit jobs is somehow discouraging regarding the existence of a restriction on the possible forms of optimal priority algorithms. This set rules out a possible normal form that an optimal priority algorithm may have: not only there does not exist an optimal priority algorithm considering the jobs in non-increasing profit order; but also there does not exist a fixed-priority algorithm for this set.

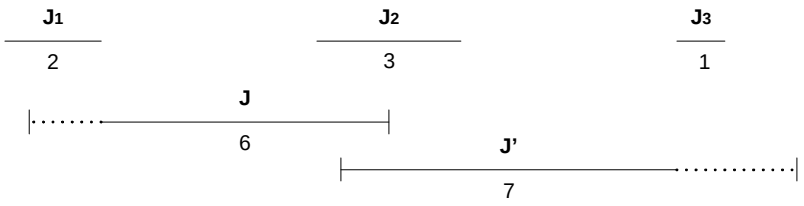

Figure 3.3: Jobs J_1, J_2, J_3 are intervals. J and J' are jobs where their processing time is denoted by the solid horizontal line and their release and deadline times are denoted by the small vertical "line-bounds". The job profits are written below each job.

The following definition associates the "behavior" (computation) of the algorithm with the given set of jobs.

Definition 3.8.1. Let S be a finite set of jobs and A be a priority algorithm on S. Say that $J, J' \in S$. We say that J *causally precedes* J' *when computing on* S, $J \succ_{A,S} J'$ if A reads J (from the input) before it reads J' (when the input set is S). For a fixed set S we write $J \succ_A J'$ if for every $S' \subseteq S$ **where** $\mathbf{J}, \mathbf{J'} \in \mathbf{S'}$ we have $J \succ_{A,S'} J'$. If it is understood from the context we may simply write $J \succ J'$.

Say that $S = \{J_1, J_2, J_3, J, J'\}$ is the set of jobs depicted on figure 3.3. Every job is of distinct proportional profit. Obviously, for an optimal fixed-priority algorithm on a set S, there is a total ordering w.r.t. $\succ$ on S.

We give a MEMORYLESS-GREEDY-PRIORITY algorithm optimal on S and we show that there does not exist a fixed-priority algorithm optimal on S.

Consider the jobs in figure 3.3. The following priority algorithm from MEMORYLESS-GREEDY-PRIORITY is optimal on S.

- The initial ordering is $J_2 > J_1 > J_3 > J > J'$.

- If J_2 is scheduled then determine the ordering $J > J' > J_1 > J_3$ and schedule greedily.
- Else, if J_1 is the first scheduled job then determine the ordering $J > J' > J_3$ and schedule J at the latest available time and J' at the earliest available time. Similarly, if J_3 is the first scheduled job (the ordering is $J > J'$).

Looking into the possible subsets of S, we see that this priority algorithm is optimal on S. Suppose that there exists a fixed-priority algorithm A optimal on S. Therefore, there exists a total ordering on S under the *causal precedence* relation. If $J_1 \succ J$ then A does not gain optimal profit either on: (i) $\{J_1\}$ or (ii) $\{J_2\}$ or (iii) $\{J_1, J\}$ or (iv) $\{J_2, J\}$ or (v) $\{J_1, J_2, J\}$. Hence, $J \succ J_1$. Similarly, $J' \succ J_3$. Suppose that $J_3 \succ J$. Then $J' \succ J_3 \succ J \succ J_1$. On input $\{J', J, J_3\}$, J' is scheduled not overlapping J_3. Then, on input $\{J, J', J_1\}$ A does not gain optimal profit. Similarly when $J_1 \succ J'$. Therefore, J, J' causally precede J_1 and J_3. On input $\{J, J', J_1, J_2\}$, at least one job among $\{J, J'\}$ is scheduled overlapping the corresponding $U \in \{J_1, J_2\}$. Then, on input $\{J, J', U\}$ A does not gain optimal profit.

Properties of optimal priority algorithms Theorems 3.8.2, 3.8.1 and 3.8.3 are properties of optimal priority algorithms. These results hold for machine environments of any number of machines, where the jobs are not necessarily of proportional profit.

Here is a normal form for optimal priority algorithms.

Theorem 3.8.1. *Given a finite set of jobs S and an optimal priority algorithm $A \in$* PRIORITY *on S, there exists $A' \in$* M-G-PRIORITY *which is also optimal on S.*

Theorem 3.8.1 directly follows from lemmata 3.8.6 and 3.8.7.

It is easy to see that there exist infinitely many finite sets of distinct profit jobs, where for each such set the number of (different) optimal priority algorithms is super-exponential w.r.t. the cardinality of the set. In this sense the following theorem is of some importance regarding sets of distinct profit jobs.

Theorem 3.8.2. *Let S be a finite set of* distinct profit *jobs which admits an optimal priority algorithm. Then, every two priority algorithms optimal on S schedule the same set of jobs on every subset of S.*

Theorem 3.8.2 is a corollary of lemmata 3.8.5 and 3.8.8.

There are trivial examples where theorem 3.8.2 fails in the arbitrary profits case. Theorem 3.8.3 is the strongest "invariant" we have obtained, analogous to theorem 3.8.2 for sets of arbitrary profits. This theorem shows that under optimal priority algorithms the profit of the (so far) scheduled jobs is unaffected by the addition of new jobs of smaller profit (in the input set).

Theorem 3.8.3. *Let S be a finite set of jobs and $A \in$ PRIORITY be optimal on S. For every $S' \subseteq S$ consider the optimal schedule $A(S')$. Say that m is the minimum profit among jobs in S'. Say that $\hat{S} \subset S$ where every job in $\hat{S}$ has profit smaller than m. If $A(S' \cup \hat{S}) = S'' \cup \hat{S}'$, where $S'' \subseteq S'$ and $\hat{S}' \subseteq \hat{S}$, then $profit(A(S')) = profit(S'')$.*

Theorem 3.8.3 is a corollary of lemma 3.8.4 (the proof of theorem 3.8.3 is given at the end of this section).

Lemma 3.8.4. *Let S be a finite set of jobs and $A \in$ M-PRIORITY be optimal on S. For every $S' \subseteq S$ consider the optimal schedule $A(S')$. Say that m is the minimum profit among jobs in S'. Say that $\hat{S} \subset S$ where every job in $\hat{S}$ has profit smaller than m. If $A(S' \cup \hat{S}) = S'' \cup \hat{S}'$, where $S'' \subseteq S'$ and $\hat{S}' \subseteq \hat{S}$, then $profit(A(S')) = profit(S'')$.*

Proof. We show the lemma by induction on the cardinality of S'. For the induction basis fix an $S' = \{J\}$. Suppose that the induction predicate does not hold. That is, there exists an $\hat{S}$ s.t. $profit(A(S')) \neq profit(S'')$. Therefore, $S'' = \emptyset$. A is memoryless and thus $A(S' \cup \hat{S}) = A(S' \cup \hat{S}')$. Consider the sequence in which A reads jobs from $\hat{S}'$ (when computing on $S' \cup \hat{S}'$). Also, consider the smallest prefix P of this sequence where $A(P \cup S') = P$. Say that $\hat{J}$ is the last job in this prefix P. Every job from P is scheduled and thus $profit\big(A((P \backslash \{\hat{J}\}) \cup S')\big) > profit\big(P \cup S'\big)$; thus A is not optimal.

Suppose that for every S', where $|S'| < k$ the induction predicate holds. We wish to show that for every S' where $|S'| = k$ the induction predicate is true. Assume the contrary. That is, there exists an S', $|S'| = k$ where $profit(A(S')) \neq profit(S'')$. Since A is optimal we have that $profit(A(S')) > profit(S'')$ (S'' defined as above - in the statement of the lemma). Note that there does not exist $J \in S'$ s.t. $J \notin A(S')$ and $J \notin S''$. If such J exists then by the induction hypothesis and the fact that A is memoryless we reach a contradiction. Consider the sequence $T_1 = \langle J_1^1, J_2^1, \ldots, J_l^1 \rangle$ in which A reads jobs from $\hat{S}'$ ($\hat{S}'$ defined as above). Let $P_j = \{J_1^1, \ldots, J_j^1\}$, $1 \leq j \leq l$. Say that i is minimum s.t. $A(S' \cup P_i) = S''' \cup P_i$ where $profit(A(S')) > profit(S''')$. Clearly, by the induction hypothesis, J_i^1 cannot be the last scheduled job read by A.

Therefore there exists a job from S' scheduled by A after J_i^1. Consider the two runs of A on $(S' \cup P_{i-1})$ and $(S' \cup P_i)$. Up to the point that J_i^1 was read (when A runs on $(S' \cup P_i)$) the same jobs were scheduled, for the two runs of A. Record the sequence of jobs read by A after J_i^1, when A runs on $(S' \cup P_i)$ (notice that these jobs are all from S'). Let this sequence be $T_2 = \langle J_1^2, J_2^2, \ldots, J_m^2, \ldots, J_n^2 \rangle$. We have already argued that the two runs agree only on acceptances. We look for the first job J_m^2 where the two runs of A disagree on the rejection/acceptance of this job. If such a job does not exist then we have a contradiction ($profit(A(S')) = profit(S'')$). Say that all jobs read by A, up to the point that J_i^1 (excluding J_i^1) has been read, is P. Say that $J_m^2 \notin A(S')$ and $J_m^2 \in S'''$. Since A is memoryless, J_m^2 is read after every job in P_i and by the induction hypothesis we have that: $profit\big(A((S'\backslash\{J_m^2\}) \cup P_i)\big) = profit\big(A(S'\backslash\{J_m^2\})\big) + profit\big(P_i\big) = profit\big(P_i\big) + profit\big(A(S')\big) > profit\big(P_i\big) + profit\big(S'''\big) = profit\big(A(S' \cup P_i)\big)$, a contradiction. Hence, $J_m^2 \in A(S')$ and $J_m^2 \notin S'''$. When only P has been considered, we know that all of $\{J_1^2, \ldots, J_{m-1}^2, J_m^2\}$ can be scheduled. Let $profit(A(P)) = s$, $profit(\{J_1^2, \ldots, J_{m-1}^2\}) = s'$ and $profit(\{J_m^2\}) = m > m' = profit(\{J_i^1\})$. Thus, $profit\big(A(P \cup \{J_1^2, \ldots, J_m^2\} \cup \{J_i^1\})\big) = s + s' + m' < s + s' + m = profit\big(A(P \cup \{J_1^2, \ldots, J_m^2\})\big)$ which is a contradiction. □

Lemma 3.8.5. *Let S be a finite set of* distinct profit *jobs which admits an optimal priority algorithm. Then, every two optimal memoryless priority algorithms on S schedule the same set of jobs on every subset of S.*

Proof. By induction on cardinality of $S' \subset S$. The induction basis trivially holds. Assume that for every S', $|S'| < k$ the lemma (induction predicate) is true. We wish to show that it holds for every $|S'| = k$. Suppose that there exists an S', $|S'| = k$ and two memoryless priority algorithms A, A' optimal on S, such that $A(S') \neq A'(S')$. Consider the sequence in which A reads jobs from S': $T_A = \langle J_1, J_2, \ldots, J_k \rangle$. The corresponding sequence for A' is $T_{A'} = \langle J_{i_1}, J_{i_2}, \ldots, J_{i_k} \rangle$. Each job in T_A is associated with an acceptance or rejection decision. A and A' cannot agree on a rejection decision for a job. If this happens for a job $\hat{J} \in S'$ then $A(S') = A(S'\backslash\{\hat{J}\}) = A'(S'\backslash\{\hat{J}\}) = A'(S')$. Therefore, the jobs rejected by A are accepted by A' and vice-versa. Say that $A(S') = S'' \cup S_A$, $A'(S') = S'' \cup S_{A'}$, such that $S_A \cap S_{A'} = \emptyset$.

Say that $\hat{J}$ is the last job in T_A accepted by A and rejected by A'. Remove $\hat{J}$ from S'. Since A' is memoryless $A'(S'\backslash\{\hat{J}\}) = A'(S')$. On $(S' \cup \{\hat{J}\})$ A accepts every job from $S_{A'}$ since $(S'' \cup S_{A'})$ is the unique optimal schedule for a memoryless priority algorithm.

Furthermore, there are no scheduled jobs from S_A. Therefore, when A computes on S' $\hat{J}$ is read before every job from $S_{A'}$. We argue that $\hat{J}$ is the only job in S_A. Suppose that $|S_A| > 1$ (when A computes on S'). Since $\hat{J}$ is the last job accepted by A and rejected by A', then there exists a $\hat{\hat{J}} \in S_A$ read before $\hat{J}$ (when A computes on S'). If we remove $\hat{J}$ then A rejects every job from S_A, which is not true since $\hat{\hat{J}}$ is read (and scheduled) before considering jobs from $S_{A'}$. Hence, $S_A = \{\hat{J}\}$.

Symmetrically we show that $S_{A'} = \{\hat{J'}\}$. We know that $profit(S'' \cup S_A) = profit(S'' \cup S_{A'})$ which implies that $profit(\{\hat{J}\}) = profit(\{\hat{J'}\})$ contradicting the distinct profits assumption. □

Lemma 3.8.6. *Given a finite set of jobs S and an optimal priority algorithm $A \in$* PRIORITY *on S, there exists $A' \in$* G-PRIORITY *which is also optimal on S. Furthermore, for every $S' \subseteq S$ $A(S') = A'(S')$.*

Proof. Straightforward. Fix an arbitrary set $S' \subseteq S$. Suppose that A (preemptively) rejects a job $J \in S'$ which can be scheduled (during the execution of A on S'). Say that $S'' \subseteq S'$ is the set of jobs read up to the point that J was read. Then on $S'' \cup \{J\}$ A does not gain optimal profit if $profit(J) > 0$. Therefore A rejects jobs of positive profit only if they cannot be scheduled. A' is the same as A where in every ordering the jobs of zero profit come last. Furthermore, A' rejects a job only if this job cannot be scheduled. □

Lemma 3.8.7. *Given a finite set of jobs S and an optimal priority algorithm $A \in$* G-PRIORITY *on S, there exists $A' \in$* M-PRIORITY *which is also optimal on S.*

Proof. We show a stronger statement. Given A we show that a family of memoryless algorithms is optimal on S. Say that $R = S \backslash A(S)$. Fix a set $R' \subseteq R$. We show that every algorithm $A_M^{R'} \in$ M-G-PRIORITY constructed as follows is optimal on S: Simulate A by reconstructing the execution history assuming the jobs from R' that **cannot be scheduled** (during the current execution of $A_M^{R'}$) to be in the input set. We prove by induction on the cardinality of $S' \subseteq S$. The induction basis trivially holds. Assume that the lemma (induction predicate) holds for every S', $|S'| < k$. We wish to show that it holds for every S' where $|S'| = k$. Fix an arbitrary S'', $|S''| = k$. Consider the execution of A on S''. If A schedules every job from S'' then (by construction and the optimality of A) $A_M^{R'}$ also schedules every job from S''. Else, let $J \in S''$ be a job rejected by A. Then, $profit(A(S'')) = profit(A(S'' \backslash \{J\}))$. By the induction hypothesis $profit(A(S'' \backslash \{J\})) = profit(A_M^{R'}(S'' \backslash \{J\})) = profit(A_M^{R'}(S''))$, since $A_M^{R'}$ is memoryless. □

Lemma 3.8.8. *Let S be a set of distinct profit jobs that admits an optimal priority algorithm. Then, for every optimal priority algorithm A there exists an optimal memoryless algorithm A_M where for every $S' \subseteq S$ we have that $A(S') = A_M(S')$.*

Proof. Straightforward. Since S is a set of distinct profit jobs every optimal memoryless priority algorithm on S accepts the same set of jobs on every subset of S. From the proof of lemma 3.8.7 we have a family of optimal memoryless priority algorithms. Fix a memoryless priority algorithm A_M optimal on S. For an arbitrary $S' \subseteq S$ consider the set R of jobs rejected by A (when computing on S'). Say that A_M^R is the optimal memoryless algorithm associated with the set of rejected jobs R (as in the proof of lemma 3.8.7). We have that $A_M(S') = A_M^R(S') = A(S')$. □

Here is the proof of theorem 3.8.3.

Proof of Theorem 3.8.3. This is a corollary of lemmata 3.8.7 and 3.8.4. Fix S' and $\hat{S}$ such that $profit(A(S')) \neq profit(S'')$ ($S', S'', \hat{S}$ are defined as in the statement of the theorem). Let R be the set of jobs rejected by A on $(S' \cup \hat{S})$. Consider the optimal memoryless priority algorithm A_M^R (as in the proof of lemma 3.8.7). Say that $A_M^R(S' \cup \hat{S}) = S''_{A_M^R} \cup \hat{S'}_{A_M^R}$ (as in lemma 3.8.4). By lemma 3.8.4 we have that $profit(A_M^R(S')) = profit(S''_{A_M^R})$. Since A_M^R and A are optimal on S we have that $profit(A(S')) = profit(A_M^R(S'))$. Also, A_M^R and A have identical runs on $(S' \cup \hat{S})$. Hence, $S''_{A_M^R} = S''$. Therefore, $profit(A(S')) = profit(S'')$. □

3.9 Open problems - Conjectures

In this section we make a list of what we consider as the most interesting open questions regarding the complexity of languages studied in this chapter.

1. The most important open question concerns the complexity of EXISTS-OPT-GENERAL. The straightforward algorithm shows the problem to be in $ESPACE$. We conjecture that this language is hard for NP. Actually, if the following problem is hard for NP then we can show that EXISTS-OPT-GENERAL is also hard for NP.

   Problem 12. PARTITION (REMOVE MIN)
   Instance: Say that H is a finite set of integers where: $B = \sum_{a \in H} a$, $m = min(H)$ and there exists a partition H_1, H_2 of $H \backslash \{m\}$ such that $\sum_{a \in H_1} a, \sum_{a \in H_2} a < \frac{B}{2}$.
   Question: Does there exist a partition H'_1, H'_2 of H such that $\sum_{a \in H'_1} a = \sum_{a \in H'_2} a$?

2. INTERVAL-SCHEDULING($m = 1$), NON-OPT-GF-INTERVAL($m = 1$) and EXISTS-OPT-INTERVAL($m = 1$): In section 3.7 we show that POLY-INTERVAL-SCHEDULING ($m = 1$) is complete for NL. We showed that these first three languages are in $U_L - NC^2$. We conjecture that they are of the same complexity (w.r.t. logspace reductions or even NC^1 reductions) and they are complete for some (uniform) complexity class between NL and $U_L - NC^2$. Actually, we conjecture that they are all complete for a class which is between $LOGCFL$ and $U_L - NC^2$.

3. POLY-NON-OPT-GF-INTERVAL($m = 1$) and POLY-EXISTS-OPT-INTERVAL($m = 1$): We showed that both languages are in NL. We conjecture that randomization helps for these problems. RL stands for the class of languages decided by one-sided error randomized logspace bounded Turing Machines. Note that $L \subseteq RL \subseteq NL$. We conjecture that these two languages are in RL.

Bibliography

[1] S. Angelopoulos. Randomized priority algorithms. In *Proceedings of the 1st Workshop on Approximation and Online Algorithms (WAOA)*, 2003.

[2] S. Angelopoulos and A. Borodin. On the power of priority algorithms for the facility location and set cover. In *Proceedings of the 5th Workshop on Approximation Algorithms for Combinatorial Optimization (APPROX)*, pages 26–39, 2002
(to appear in a special issue of *Algorithmica* dedicated to papers from APPROX 2002).

[3] E.M. Arkin and E.L. Silverberg. Scheduling jobs with fixed start and end times. *Discrete and Applied Mathematics*, 18:1–8, 1987.

[4] A. Borodin, M.N. Nielsen, and C. Rackoff. (incremental) priority algorithms. In *Proceedings of the 13th Annual Symposium on Discrete Algorithms (SODA)*, pages 752–761, Jan. 2002.

[5] A. Borodin, M.N. Nielsen, and C. Rackoff. (incremental) priority algorithms. *Algorithmica*, 37(4):295–326, 2003.

[6] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C.Stein. *Introduction to Algorithms.* MIT Press, 2002.

[7] S. Davis and R. Impagliazzo. Models of greedy algorithms for graph problems. In *Proc. Symposium On Discrete Algorithms (SODA)*, 2004.

[8] R.L. Graham. Bounds on multiprocessing timing anomalies. *SIAM Journal on Applied Mathematics*, 17(1):416–429, 1969.

[9] R.L. Graham, E.L. Lawler, J.K. Lenstra, and A.H.G. Rinnooy Kan. Optimization and approximation in deterministic sequencing and scheduling: a survey. *Annals of Discrete Mathematics*, 5:287–326, 1979.

[10] P. Helman, B.M.E. Moret, and H.D. Shapiro. An exact characterization of greedy structures. *SIAM J. Disc. Math.*, 6(2):274–283, 1993.

[11] N. Immerman. *Descriptive Complexity.* Springer, 1999.

[12] D. Karger, C. Stein, and J. Wein. *Scheduling Algorithms. Appearing in Algorithms and Theory of Computation Handbook, M.J. Atallah, editor.* CRC Press, 1998.

[13] P.N. Klein and S. Subramanian. A linear-processor polylog-time algorithm for shortest paths in planar graphs. In *Proc. symposium on Foundations of Computer Science (FOCS)*, pages 259–270, 1993.

[14] B. Korte and L. Lovasz. Mathematical structures underlying greedy algorithms. *Lecture Notes in Computer Science*, 177:205–209, 1981.

[15] T. Leighton. *Introduction to Parallel Algorithms and Architectures: Arrays, Trees, Hypercubes.* Morgan Kaufmann Publishers, 1992.

[16] I. Niepel and P. Rossmanith. Uniform circuits and exclusive read PRAMs. In *Foundations of Software Technology and Theoretical Computer Science*, pages 307–318, 1991.

[17] C.H. Papadimitriou. *Computational Complexity.* Addison Wesley - Longman, 1994.

[18] C.H. Papadimitriou and K. Steiglitz. *Combinatorial Optimization : Algorithms and Complexity.* Dover Publications, Inc., 1998.

[19] M. Pinedo. *Scheduling: Theory, Algorithms and Systems.* Prentice Hall, 1995.

[20] O. Regev. Priority algorithms for makespan minimization in the subset model. *Information Processing Letters*, 84(3):153–157, 2003.

[21] H. Vollmer. *Introduction to Circuit Complexity - A Uniform Approach.* Springer Verlag, 1999.