Scheduling with Batching and Incompatible Job Families

Yoav Katz
Scheduling with Batching and Incompatible Job Families

Research Thesis

Submitted in Partial Fulfillment of the Requirements for the Degree of Master of Science in Computer Science

Yoav Katz

Submitted to the Senate of the Technion—Israel Institute of Technology
Av 5761 Haifa July 2001
The research thesis was done under the supervision of Doctor Hadas Shachnai in the Department of Computer Science.

I would like to thank Hadas for her ideas and continuous support that were always given in warm manner and with a smile.
# Contents

<table>
<thead>
<tr>
<th>Section</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Abstract</td>
<td>1</td>
</tr>
<tr>
<td>List of Symbols</td>
<td>2</td>
</tr>
<tr>
<td>1 Introduction</td>
<td>4</td>
</tr>
<tr>
<td>1.1 Problem Statement</td>
<td>4</td>
</tr>
<tr>
<td>1.2 Applications</td>
<td>4</td>
</tr>
<tr>
<td>1.3 Main Results</td>
<td>5</td>
</tr>
<tr>
<td>1.4 Related Work</td>
<td>6</td>
</tr>
<tr>
<td>1.4.1 Real-Time Scheduling</td>
<td>6</td>
</tr>
<tr>
<td>1.4.2 Batch Scheduling</td>
<td>7</td>
</tr>
<tr>
<td>1.4.3 Multiple Knapsack</td>
<td>9</td>
</tr>
<tr>
<td>1.5 Organization of this Work</td>
<td>10</td>
</tr>
<tr>
<td>2 Preliminaries</td>
<td>11</td>
</tr>
<tr>
<td>2.1 Definitions and Notation</td>
<td>11</td>
</tr>
<tr>
<td>2.2 Performance Ratios</td>
<td>12</td>
</tr>
<tr>
<td>2.3 Hardness of the Family Batching Problem</td>
<td>13</td>
</tr>
<tr>
<td>2.4 Relation to MWIS on Interval Graphs</td>
<td>14</td>
</tr>
<tr>
<td>3 Properties of Optimal Schedules</td>
<td>16</td>
</tr>
<tr>
<td>4 Fixed Number of Families</td>
<td>20</td>
</tr>
<tr>
<td>4.1 Bounded Batch</td>
<td>20</td>
</tr>
<tr>
<td>4.2 Unbounded Batch</td>
<td>22</td>
</tr>
</tbody>
</table>
5 Optimal Algorithms for Arbitrary Number of Families 24
  5.1 The Feasibility Problem ................................. 24
  5.2 No Release Dates and Unbounded Batch .................. 24
  5.3 Constant Number of Release Dates and Due Dates ........ 25

6 Approximation Algorithms for Arbitrary Number of Families 31
  6.1 Greedy Algorithms ........................................ 31
    6.1.1 The Unweighted Problem ............................. 32
    6.1.2 The Weighted Problem ............................... 32
  6.2 Local Ratio Algorithms .................................... 35
    6.2.1 Overview of the Local Ratio Technique ............... 35
    6.2.2 The Modified Local Ratio Technique ................... 36
  6.3 A Polynomial Time Approximation Scheme .................. 39

7 Multiple Identical Machines 41
  7.1 Fixed Number of Families .................................. 41
  7.2 Unit Length Jobs ......................................... 43
  7.3 Local Ratio Algorithms .................................... 43

8 Summary and Open Problems 44

Bibliography 45
List of Figures

1.1 Relations between different batch scheduling models .................................. 9

6.1 A worst case instance for Rev-Greedy ...................................................... 34
List of Tables

1.1 Summary of known results for real-time and batch scheduling problems . . . 6
Abstract

We consider the problem of batch scheduling of jobs from incompatible families, which arises in production systems, in multimedia-on-demand services, and in many real-life situations. Suppose that $n$ jobs need to be scheduled on $m$ batching machines, that is, machines that can run several jobs simultaneously. Each job is given by its family, release date, due date, and non-negative weight; jobs from the same family have the same processing time. The machines can process the jobs in batches of size at most $b$, with the restriction that jobs from different families cannot be batched together. Our objective is to find a schedule of the jobs, which maximizes the overall weight of on-time jobs (i.e., jobs that are scheduled before their due dates). We call this problem the $f$-batch problem. Our problem is a generalization of classical real-time scheduling; thus, it is strongly NP-hard.

Our first contribution is a detailed characterization of optimal schedules, which enables the development of efficient polynomial time algorithms for our problem. Specifically, we formulate a set of structural properties satisfied by optimal schedules. We then use these properties to obtain optimal solutions for several important classes of instances.

Our second contribution is a technique for transforming the $f$-batch problem to a variant of classical real-time scheduling. This transformation is used for deriving the first combinatorial approximation algorithm for the $f$-batch problem. Our transformation yields a simple $(2 + \epsilon)$-approximation algorithm for general instances with unbounded batch size, based on the Local Ratio technique of Bar-Yehuda and Even [BE-85]. The algorithm can be applied, with the same performance ratio, to a system of $m$ identical machines, for any $m > 1$. 


List of Symbols


$m$ ............... number of machines
$b$ ............... maximal batch size
$F$ ............... number of job families
$f$ ............... index of family
$p_f$ ............... length of jobs in family $f$
$J$ ............... a job
$J, S, X, Y$ ....... a set of jobs
$n$ ............... number of jobs
$j$ ............... job index
$J_j$ ............... job number $j$
$n_f$ ............... number of jobs of family $f$
$q_f$ ............... number of jobs scheduled from family $f$
$U_j$ ............... unit penalty of job $J_j$
$L_j$ ............... lateness of $J_j$
$L_{max}$ .......... maximum lateness of jobs
$f_j$ ............... family of job $J_j$
$p_j$ ............... length of job $J_j$
$P$ ............... sum of the lengths of all jobs
$R$ ............... number of distinct release dates
$r_j$ ............... release date of job $J_j$
$k$ ............... index of release dates
$D$ ............... number of distinct due dates
$d_j$ ............... due date of job $J_j$
l ............... index of due dates
$w_j$ ............... weight of job $J_j$
$s_j$ ............... slack time of job $J_j$
$S$ ............... sum of the slack times of all jobs
$t_j$ ............... scheduling time of job $J_j$
<table>
<thead>
<tr>
<th>Symbol</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \Gamma )</td>
<td>a batch of jobs</td>
</tr>
<tr>
<td>( t_\Gamma )</td>
<td>scheduling time of batch ( \Gamma )</td>
</tr>
<tr>
<td>( p_\Gamma )</td>
<td>length of batch ( \Gamma )</td>
</tr>
<tr>
<td>( f_\Gamma )</td>
<td>family of batch ( \Gamma )</td>
</tr>
<tr>
<td>( B )</td>
<td>a set of batches</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Definition</th>
</tr>
</thead>
<tbody>
<tr>
<td>PTAS</td>
<td>Polynomial Time Approximation Scheme</td>
</tr>
<tr>
<td>( I )</td>
<td>instance of a problem</td>
</tr>
<tr>
<td>OPT</td>
<td>an optimal offline algorithm</td>
</tr>
<tr>
<td>( \mathcal{S}_0 )</td>
<td>an optimal solution</td>
</tr>
<tr>
<td>( OPT )</td>
<td>the value of an optimal solution</td>
</tr>
<tr>
<td>( A )</td>
<td>an algorithm</td>
</tr>
<tr>
<td>( \mathcal{S}_A )</td>
<td>a solution output by ( A )</td>
</tr>
<tr>
<td>( W(\mathcal{S}) )</td>
<td>the weight of job set ( \mathcal{S} )</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

1.1 Problem Statement

Suppose that \( n \) jobs \( \{J_1, \ldots, J_n\} \) need to be scheduled on a set of \textit{batching machines}, that is, machines that can run several jobs simultaneously. There are \( F \) different job families; all the jobs in the family \( f \) have the same length, \( p_f \), \( 1 \leq f \leq F \). Each job, \( J_j \), is associated with a family, \( f_j \), a release date, \( r_j \), a due date, \( d_j \), and a weight, \( w_j \). There are \( m \) identical batching machines. The machines can process the jobs in batches of at most \( b \) jobs, with the restriction that all the jobs in a batch belong to the same family. Our objective is to find a schedule of the jobs which maximizes the overall weight of on-time jobs (i.e., jobs that are scheduled before their due date).

1.2 Applications

The \( f \)-batch problem often arises in production systems. For example, the fabrication of integrated circuits requires a \textit{burn-in} operation. In this operation the semiconductors are exposed to high temperatures, in a fixed capacity oven, in order to identify the chips susceptible to failure. Chips of different types require different exposure times and cannot be processed simultaneously. Given the delivery requirements, we must decide how to schedule the use of the oven, as to maximize the number of chips processed on time.

The \( f \)-batch problem arises also in \textit{Video on Demand (VOD)} systems \cite{AG+95}. In these systems, client requests to view video programs are sent to a centralized video server. The system has a fixed number of channels, through which the programs are transmitted to the clients; using a multicast facility, the server can batch several clients (who wish to view the same program) to use the same communication channel. The server needs to decide which of the requests will be serviced, and in which order, such that the revenue is maximized (Note
that clients must have some degree of “patience”, in order to allow requests to be batched together).

Finally, the f-batch problem has many real-life applications. Consider, for example, the following optimization problem. A travel agency organizes tours of different lengths and participation costs. The agency has a fixed number of guides, who can lead the tours. Each tour can accommodate a limited number of participants. Requests for the tours are made by individuals that arrive at different times and have due dates, which specify the latest starting time of the tour. The agency has to decide which of the various tours would be organized, and how these tours should be scheduled, so as to maximize its revenue.

1.3 Main Results

We focus on the f-batch problem on a single machine. In Chapter 7 we discuss some extensions of our results to the multiple machine case. We characterize variants of our problem as either polynomially solvable (for which we give exact polynomial time algorithms), or NP-hard (for which we develop approximation algorithms). The following are the main results of this work:

- We show that the f-batch problem is polynomially solvable for the following classes of instances:
  - The number of families is fixed.
  - The number of release dates and due dates is bounded by a constant, all jobs have unit lengths and uniform weights, and the batch size is also constant. Note that, if we allow jobs of different families to have different lengths, the problem becomes weakly NP-complete [GJ-79].
  - All jobs have small slack.
  - Unbounded batch size and no release dates.
- We develop a polynomial time approximation algorithm (PTAS) for instances with uniform weighted jobs, no release dates, and \( F = O(\log n) \).
- We give a simple \((2 + \epsilon)\)-approximation algorithm based on the Local Ratio technique for
  - General instances in which the batch size is unbounded.
  - Uniform jobs with no release dates (and arbitrary batch size).

\(^1\)See in Chapter 2
• For the special case, where there are no release dates, and all the jobs have the same length, we show that a natural greedy algorithm yields a 2-approximation for our problem.

In Table 1.1 we summarize the previous results for problems related to the f-batch problem and the contribution of this work. (Abbreviated notation for describing scheduling problems is given in Section 2.1).

<table>
<thead>
<tr>
<th>Problem</th>
<th>Approximation</th>
<th>Reference</th>
</tr>
</thead>
<tbody>
<tr>
<td>1|L_{\text{max}}</td>
<td>O(n \log n)</td>
<td>EDD rule [M-68]</td>
</tr>
<tr>
<td>1|\sum U_j</td>
<td>O(n \log n)</td>
<td>[G-79]</td>
</tr>
<tr>
<td>1</td>
<td>r_j | L_{\text{max}}</td>
<td>Strongly NP-hard</td>
</tr>
<tr>
<td>1</td>
<td>r_j | \sum w_j \bar{U}_j</td>
<td>Weakly NP-hard</td>
</tr>
<tr>
<td>1</td>
<td>r_j, p_j = p | \sum w_j \bar{U}_j</td>
<td>Strongly NP-hard, (2 + \epsilon) approx.</td>
</tr>
<tr>
<td>1</td>
<td>p-batch, b \geq n | L_{\text{max}}</td>
<td>O(n^2)</td>
</tr>
<tr>
<td>1</td>
<td>p-batch, b = 2 | L_{\text{max}}</td>
<td>Strongly NP-hard</td>
</tr>
<tr>
<td>1</td>
<td>p-batch, b \geq n | \sum w_j \bar{U}_j</td>
<td>O(n^3)</td>
</tr>
<tr>
<td>1</td>
<td>p-batch | \sum w_j \bar{U}_j</td>
<td>Weakly NP-hard O(n^2 \log n)</td>
</tr>
<tr>
<td>1</td>
<td>p-batch, r_j, p_j = p | \sum w_j \bar{U}_j</td>
<td>Strongly NP-hard</td>
</tr>
<tr>
<td>1</td>
<td>f-batch | L_{\text{max}}</td>
<td>O(n \log n)</td>
</tr>
<tr>
<td>1</td>
<td>f-batch | \sum \bar{U}_j</td>
<td>open, (2 + \epsilon)-approx.</td>
</tr>
<tr>
<td>1</td>
<td>f-batch, r_j, b &gt; n_f | \sum w_j \bar{U}_j</td>
<td>Strongly NP-hard, (2 + \epsilon)-approx.</td>
</tr>
<tr>
<td>1</td>
<td>f-batch, s_j &lt; p_j | \sum w_j \bar{U}_j</td>
<td>O(S \log S)</td>
</tr>
<tr>
<td>1</td>
<td>f-batch, p_f = p | \sum w_j \bar{U}_j</td>
<td>open, 2-approx., O(n^2 \log n)</td>
</tr>
<tr>
<td>1</td>
<td>f-batch, p_f = p, b \geq n_f | \sum w_j \bar{U}_j</td>
<td>O(n + F^2)</td>
</tr>
<tr>
<td>1</td>
<td>f-batch, r_j, p_j = p | \sum \bar{U}_j</td>
<td>open, (2 + \epsilon)-approx.</td>
</tr>
<tr>
<td>1</td>
<td>f-batch, r_j, p_j = 1, b, D, R = \text{const} | \sum \bar{U}_j</td>
<td>O(n^{6D \cdot R + 1})</td>
</tr>
<tr>
<td>1</td>
<td>f-batch, F = \text{const} | \sum \bar{U}_j</td>
<td>O(n^{F+1} + 1)</td>
</tr>
<tr>
<td>1</td>
<td>f-batch, r_j, F = \text{const} | \sum w_j \bar{U}_j</td>
<td>O(n^{(F+1)^2 \cdot F+2} \log n)</td>
</tr>
<tr>
<td>1</td>
<td>f-batch, F = O(\log n) | \sum \bar{U}_j</td>
<td>PTAS</td>
</tr>
</tbody>
</table>

Table 1.1: Summary of known results for real-time and batch scheduling problems

1.4 Related Work

1.4.1 Real-Time Scheduling

The classical real-time scheduling problem is a special case of the f-batch problem in which there is a single job from each family. In real-time scheduling, we are given a set of machines
and set of jobs with specified release dates, due dates and weights, and the goal is to find a schedule that maximizes the weight of the jobs that meet their due date.

It is known that even the problem of determining whether all jobs can be scheduled on a single machine is strongly NP-hard [GJ-79]. The best known algorithm, due to Bar Noy et. al. [BB+00], yields a \((2 + \epsilon)\)-approximation algorithm, based on the Local Ratio technique [BE-85]. A pseudo-polynomial 2-approximation algorithm, based on rounding an optimal fractional solution of an LP formulation of the problem, was given in [BG+99]. A slightly modified version gives a 3-approximation algorithm that runs in polynomial time. If all jobs have the same weight, a greedy algorithm that schedules the first available job yields a 2-approximation.

Lawler and Moore [LM-69] showed that when jobs have no release dates, the problem can be solved exactly in pseudo polynomial-time; they developed an FPTAS based on scaling the weights. When jobs are uniform (i.e. of equal weights) and with no release dates the problem is solvable in \(O(n \log n)\) steps by Moore’s algorithm [M-68]. Baptiste [B-99b] showed that the real-time scheduling problem is polynomially solvable if all jobs have identical length.

A related problem, where each job can be scheduled at one of \(k\) possible time points is called JISPk. Speksma showed in [S-99] that JISPk is MAX SNP-hard even for \(k = 2\); the paper gives a 2-approximation greedy algorithm for the problem.

### 1.4.2 Batch Scheduling

There has been significant research on extending the classical real-time scheduling problem to support grouping of jobs. Several models were proposed, for different applications. Figure 1.1 describes the relation between these scheduling models.

In the family scheduling (or changeover times) model [MP-89], each job is associated with a family; jobs from distinct families that are scheduled consecutively incur a setup delay. The setup times may depend on the families of the jobs. In this model, a batch is a maximal set of jobs that is scheduled contiguously on a machine; setup is required only for the first job in each batch. Large batches have the advantage of high machine utilization, however, they may delay the processing of high priority jobs that belong to other families. A classical application of this model is the production of different colors of paints on the same machine. A setup time for cleaning the machine is required between the production of different paints. The model has two time variants. In batch availability, the completion time of each job is equal to the completion of the batch in which it was scheduled. In item availability, the completion time of a job is equal to its actual completion time. The serial batch processing model (s-batch) [HL-94] is a special case of the family scheduling model with batch availability, in which there is a single family and a fixed setup time.

In the parallel batch processing model (p-batch) [BG+98], each machine is capable of batching several jobs simultaneously. Unlike our model, jobs are not partitioned into families;
thus, any subset of jobs can be batched together. However, jobs may have different \textit{lengths}. A batch is completed when the longest job in the batch is completed. Baptiste [B-99a] showed that, when all jobs have the same length, the p-batch problem is solvable in polynomial time. Note that this is a special case of the f-batch problem, where \( F=1 \).

A special case of the p-batch problem involves instances where jobs belong to \( m \) types (or families) and all jobs from the same type have the same length. Note that unlike the f-batch problem, in this case, jobs from different types can be batched together, and the length of the batch is the length of the longest job type in the batch. This model is often referred to as “batching with families”, in contrast to “batching with incompatible job families”. When the objective is to minimize flow time, the problem is polynomially solvable for fixed number of types (see [HL-97]).

Batching with incompatible job families was studied previously in the context of operational research where the goal was to find optimal schedules for various performance measures. Uzsoy [U-95] examined the problem of scheduling batch processors where the objective is to minimize maximum lateness, the weighted sum of completion times or the makespan, in the case where all jobs have the same length. He shown some properties of optimal solutions and provided exact and heuristic algorithms for several problems involving single and parallel batch processors.

Dobson and Nambimadom [DN-99] considered a generalization of the f-batch problem, where each job can have arbitrary volume (or bandwidth) and the sum of the volumes of all jobs in the batch is specified. This model reduces to our model when all jobs have unit volumes. The paper shows that finding a schedule that minimizes the total weighted flow time is NP-hard, even if there are no release dates. Azizoglu and Webster [AW-01] extended this work and proposed a branch and bound procedure, for the weighted sum of completion times problem.

Mehta and Uzsoy [MU-98] developed exact and heuristic algorithms for the problem of family batching with the objective of minimizing total tardiness. Fowler et al. [DF+00] examined the case where the objective is to minimize total weighted tardiness. This problem is NP-hard even in the case where there are no release dates; their paper presents various heuristic approaches for tackling this problem.

Our research differs from these previous studies in at least two perspectives. First, under the above optimization criteria, all jobs need to be scheduled and the algorithms need only to determine the order of the jobs, and the way the jobs are batched. In the maximum weighted on-time jobs problem, we first need to determine the subset of jobs that will be scheduled. Then we can proceed to specify the order among job batches. Secondly, while these works studied \textit{heuristics} for NP-hard instances, our objective is to develop approximation algorithms with performance guarantees.

Naor [N-00] considered the f-batch problem studied in this work. He gave an LP-based algorithm, which yields \((2 + \epsilon)\)-approximation for general instances with unbounded batch
size.

**Recent developments** Following the results in this work, the modified Local Ratio technique has been developed in [BG+01] to yield a \((2 + \epsilon)\)-approximation, for general instances, with arbitrary batch size.

### 1.4.3 Multiple Knapsack

When all jobs have the same length, our problem reduces to a special case of the class-constrained multiple knapsack problem [ST-01]. Specifically, we can partition the time axis into slots (knapsacks), and try to assign jobs (items) into slots such that only jobs from the same family (class) are assigned to a slot; the number of jobs assigned to a slot is bounded by \(b\); jobs are assigned only to time slots in which they are available. Chekuri and Khanna gave in [CK-00] a 2-approximation algorithm for Multiple Knapsack. In Chapter 6, we adopt some ideas from [CK-00], for deriving a 2-approximation algorithm for the f-batch problem.

![Diagram](Image.png)

*Figure 1.1: Relations between different batch scheduling models*
1.5 Organization of this Work

In Chapter 2 we give some notation and define the performance measures used in this study. Then, we present results that relate the f-batch problem to other scheduling and graph theoretic problems. In Chapter 3 we study the basic properties of optimal solutions for our problem.

In Chapter 4 we give an optimal offline algorithm for the general f-batch problem with a fixed number of families, and in Chapter 5 we develop optimal algorithms for other classes of instances.

In Chapter 6 we present approximation algorithms for the f-batch problem: we apply several approximation techniques, including the greedy approach, local ratio based algorithms, and approximation schemes.

In Chapter 7 we extend some results for the single batching machine case to instances with multiple identical batching machines.

We conclude in Chapter 8 with a summary and open problems.
Chapter 2

Preliminaries

2.1 Definitions and Notation

We first give some graph-theoretic definitions. Let $G = (V, E)$ denote a graph, where $V$ is the set of vertices and $E$ is the set of edges.

**Definition 2.1** An interval graph is a graph for which there exists a mapping from vertices to intervals on the real line, such that two vertices share a common edge iff their corresponding intervals overlap.

**Definition 2.2** Given a graph $G = (V, E)$, an independent set (IS) in $G$ is a subset of vertices $V' \subseteq V$, such that for any $u,v \in V'$, $(u,v) \not\in E$, i.e., non of the vertices in $V'$ are neighbors in $G$.

**Definition 2.3** The maximum independent set problem is to find in $G$ an IS of maximum size. In the weighted case, each vertex $v \in V$ is associated with some positive weight. The maximum weighted IS (MWIS) problem is to find in $G$ an IS of maximum total weight.

In the following we extend the standard $\alpha|\beta|\gamma$ notation, used for describing scheduling problems (see, e.g., [B-98]), to family batch scheduling. We describe an instance of our problem as

$$\alpha_1|f - batch, \beta_1, \beta_2, \beta_3, \beta_4| \sum w_j \bar{U}_j$$

- $\alpha_1$ specifies the machine environment. If $\alpha_1 = P$, then we have a set of identical batching machines; the size of this set is given as part of the input. If $\alpha_1 = P_m$, there is a fixed number of $m$ identical batching machines. If $\alpha_1 = 1$, there is a single batching machine.
• \( \beta_1, \beta_2, \beta_3, \beta_4 \) specify the job characteristics.
  
  - If \( \beta_1 = r_j \), then the jobs may have different release dates; otherwise all jobs are available at time \( 0 \).
  
  - \( \beta_2 \) indicates bounds on the number of job families. Interesting cases are \( F = 1 \), \( F = \text{const} \), \( F = O(\log n) \), where the number of job families is one, constant, or logarithmic in \( n \), respectively.
  
  - \( \beta_3 \) gives restrictions on the possible number of job lengths. If \( \beta_3 \) is \( 'p_f = p' \), all job families have the same length. If \( \beta_3 \) is \( 'p_f = 1' \) then all jobs have unit processing requirements.
  
  - \( \beta_4 \) indicates possible bounds on the batch size. If \( \beta_4 \) is \( 'b = \text{const}' \), then the batch size is a fixed constant. If \( \beta_4 \) is \( 'b \geq n_f' \), there is no limit on the batch size (unbounded batch).

• \( \sum w_j \bar{U}_j \) indicates that our objective is to maximize the total weight of on-time jobs.

**Example 2.1.1** The instance of the \( f \)-batch problem with a single batching machine, arbitrary release dates and constant number of families is denoted \( 1|f\text{-batch}, r_j, F = \text{const}| \sum w_j \bar{U}_j \).

We call the problem \( \alpha_1 | \beta_1, \beta_2, \beta_3, \beta_4 | \sum w_j \bar{U}_j \), the classical real-time scheduling problem, while \( \alpha_1 | f - \text{batch}, \beta_1, \beta_2, \beta_3, \beta_4 | \sum w_j \bar{U}_j \) is the family batching problem. The available jobs at time \( t \) are jobs that can be scheduled in a batch that starts at time \( t \). Finally, the slack time of a job \( J_j \), denoted by \( s_j \), is defined by \( d_j - r_j - p_j + 1 \). This is the number of possible scheduling time points of the job.

Let \( S = \sum_j s_j \) denote the sum of the slack times of all the jobs; \( P = \sum_j p_j \) is the sum of the lengths of all the jobs.

### 2.2 Performance Ratios

As we show below, for many cases the \( f \)-batch problem is NP-hard. For these instances we develop approximation algorithms.

**Definition 2.4** The approximation ratio of an offline deterministic algorithm, \( \mathcal{A} \), for maximization problem is \( R_{\mathcal{A}} \), if for every finite input instance \( \mathcal{I} \),

\[
\frac{A(\mathcal{I})}{OPT(\mathcal{I})} \geq \frac{1}{R_{\mathcal{A}}},
\]

where \( A(\mathcal{I}) \) and \( OPT(\mathcal{I}) \) are the profits of \( \mathcal{A} \) and the optimum on \( \mathcal{I} \), respectively.
Definition 2.5 A polynomial time approximation scheme (PTAS) is an algorithm, $\mathcal{A}$, which takes as input both an instance $\mathcal{I}$ and an error bound $\epsilon$, and has the performance guarantee

$$ R_\mathcal{A}(\mathcal{I}, \epsilon) \leq (1 + \epsilon). $$

The running time of $\mathcal{A}$ is polynomial in $|\mathcal{I}|^{1/\epsilon}$.

In the online case, we use competitive analysis (see, e.g., [BE-98]) to obtain performance bounds for the proposed algorithms.

Definition 2.6 An online deterministic algorithm $\mathcal{A}$, is $c$-competitive, if there is a constant $\alpha$, such that for all finite input sequences $\mathcal{I}$,

$$ A(\mathcal{I}) \geq \frac{1}{c} OPT(\mathcal{I}) - \alpha, $$

where $A(\mathcal{I})$ and $OPT(\mathcal{I})$ are the profits of $\mathcal{A}$ and the optimum on $\mathcal{I}$.

Definition 2.7 A randomized online algorithm $\mathcal{A}$, is $c$-competitive, if there is a constant $\alpha$, such that for all finite input sequences $\mathcal{I}$,

$$ E[A(\mathcal{I})] \geq \frac{1}{c} OPT(\mathcal{I}) - \alpha, $$

where $E[A(\mathcal{I})]$ is the expected profit of $\mathcal{A}$ on $\mathcal{I}$, and $OPT(\mathcal{I})$ is the profit of the optimum on $\mathcal{I}$.

2.3 Hardness of the Family Batching Problem

In Chapter 4 we show that the f-batch problem with fixed number of families is polynomially solvable. When the number of families can be as large as the number of jobs, the f-batch problem is NP complete. This follows immediately from the next observation.

Observation 2.1 Any instance $\alpha|\beta|\gamma$ of the real-time scheduling problem can be transformed to an instance of the scheduling problem with incompatible job families.

Proof: Given an instance of $\alpha|\beta|\gamma$, generate an instance of $\alpha|f-batch, \beta|\gamma$ with $n$ families, each containing a single job from the original instance. 

Corollary 2.2 $1|f-batch, r_j| \sum w_j \bar{U}_j$ is strongly NP-complete and $1|f-batch| \sum w_j \bar{U}_j$ is weakly NP-complete.
2.4 Relation to MWIS on Interval Graphs

We now turn our attention to the special case of the f-batch problem, in which the slack time of each job is smaller than its length. We denote this problem as $1|f \text{- batch}, s_j < p_j| \sum w_j \bar{U}_j$. While this case may seem quite restricted, it does fit real-life applications, such as VOD scheduling. We show that in this case our problem can be solved via the relation to the MWIS problem.

**Theorem 2.3** The problem $1|f \text{- batch}, s_j < p_j| \sum w_j \bar{U}_j$ is equivalent to the MWIS on interval graphs.

**Proof:** We show reductions between the f-batch problem and MWIS on interval graphs.

Given an instance $\mathcal{I}_{f \text{- batch}}$ of the scheduling problem, we generate the following instance of MWIS, $\mathcal{I}_{IS}$. Let $S_{f,t}$ be the set of jobs from family $f$ which are available at time $t$ (that is, for any job $J_j \in S_{f,t}$, $r_j \leq t$ and $t + p_j \leq d_j$). Let $b = \min(|S_{f,t}|, b)$, and denote by $S'_{f,t} \subseteq S_{f,t}$ the subset of $b$ most profitable jobs in $S_{f,t}$. For each job set $S'_{f,t}$, the interval graph will contain an interval, $I_{S'_{f,t}}$, whose start-time is equal to $t$ and whose end-time equals to $t + p_f$. The weight of $I_{S'_{f,t}}$ is equal to the sum of the weights of the jobs in $S'_{f,t}$.

Next we show that an optimal solution for $\mathcal{I}_{IS}$ induces an optimal solution for $\mathcal{I}_{f \text{- batch}}$. This is done by showing that (i) any feasible solution for $\mathcal{I}_{IS}$ can be converted to a feasible solution for $\mathcal{I}_{f \text{- batch}}$ with the same weight; (ii) there exists a feasible solution for $\mathcal{I}_{IS}$, whose weight equals to the weight of $OPT(\mathcal{I}_{f \text{- batch}})$.

(i) Given a solution for $\mathcal{I}_{IS}$, for any interval $I_{S'_{f,t}}$ that was selected for the IS, we schedule the jobs in $S'_{f,t}$ as a batch at time $t$. This schedule is legal, since at most $b$ jobs (from the same family) are scheduled together, and the solution guarantees that no batches overlap. While some jobs may have appeared in two sets $S'_{f,t_1}$ and $S'_{f,t_2}$, and thus potentially selected in two batches, this cannot occur: if both intervals corresponding to jobs from $S'_{f,t_1}$ and $S'_{f,t_2}$ were selected, for $t_1 < t_2$, their start dates must be at least $p_f$ time units apart (to prevent overlap). However, the small slack constraint implies that no job that is available at time $t_1$ is available at time $t_2$, since $t_2 - t_1 \geq p_f$.

(ii) Assume that $OPT(\mathcal{I}_{f \text{- batch}})$ schedules a batch of family $f$ at time $t$. Recall that for any $J_j \in S_{f,t}$, $s_j < p_j$; thus, no other batch can contain jobs from $S_{f,t}$ (since these jobs are not available at times $t' \leq t - p_f$ or $t' \geq t + p_f$, where other batches from family $f$ can be scheduled). Moreover, an optimal solution that schedules a batch from family $f$ at time $t$, will select the jobs from $S'_{f,t}$, since these jobs have the largest weight in $S_{f,t}$. Thus, the solution for $\mathcal{I}_{IS}$, which contains $I_{S'_{f,t}}$ for every batch of family $f$ scheduled at time $t$ by $OPT(\mathcal{I}_{f \text{- batch}})$, will have the same weight as $OPT(\mathcal{I}_{f \text{- batch}})$. 

14
The reduction from MWIS to the $f$-batch problem is easy. Given an instance of the MWIS we define an instance of the scheduling problem as follows. For each interval $I$ with start-time $s(I)$, end time $e(I)$ and weight $w(I)$, we define a new job family with length $e(I) - s(I)$, and a single job of that family with a release date $s(I)$, a due date $e(I)$ and a weight $w(I)$. A solution to the scheduling problem can be converted to a solution with the same weight for the MWIS problem and visa versa, by a trivial mapping.

**Corollary 2.4** The $1|f - \text{batch}, s_j < p_j| \sum w_j \bar{U}_j$ can be solved offline in $O(S \log S)$ steps where $S = \sum_j s_j$ is the total slack time of all jobs.

**Proof:** The above reduction, from the $f$-batch problem with small slack to MWIS on interval graphs, generates an interval for any point of time $t$, in which jobs can be scheduled. Each job can be scheduled at $d_j - r_j - p_j + 1$ time points; hence, the total number of intervals in the reduction is $O(S)$. The MWIS problem on interval graphs can be solved offline in $O(n \log n)$ steps using a greedy EDD algorithm, where $n$ is the number of intervals in the graph.

Given an instance $\mathcal{I}_{f\text{-batch}}$ of the $f$-batch scheduling problem, denote by $w_{\min}(w_{\max})$ the minimal (maximal) weight of any job in $\mathcal{I}_{f\text{-batch}}$, and let $p_{\min}(p_{\max})$ be the minimal (maximal) length of any family.

**Corollary 2.5** The problem $1|f - \text{batch}, s_j < p_j| \sum w_j \bar{U}_j$ has $\frac{b_{\max}}{w_{\min}} \frac{p_{\max}}{p_{\min}}$ strongly competitive deterministic online algorithm, and a $2 \log \left( \frac{b_{\max}}{w_{\min}} \right) \log \left( \frac{p_{\max}}{p_{\min}} \right)$ competitive randomized online algorithm.

The above reduction generates intervals, such that the ratio between the minimal and maximal weight intervals is bounded by $\frac{b_{\max}}{w_{\min}}$, and the ratio between the length of the intervals is bounded by $\frac{p_{\max}}{p_{\min}}$. Using the relation of MWIS to call admission on the line, it can be shown that a deterministic online algorithm cannot be better than $\frac{b_{\max}}{w_{\min}} \frac{p_{\max}}{p_{\min}}$ competitive. Any greedy online algorithm will obtain this ratio. Using the “Classify and Random Select” paradigm as in [AB+94], a $2 \log \left( \frac{b_{\max}}{w_{\min}} \right) \log \left( \frac{p_{\max}}{p_{\min}} \right)$ competitive random online algorithm can be derived. Also, the lower bound for randomized online algorithms is $\Omega \left( \max(\log \frac{b_{\max}}{w_{\min}}, \log \frac{p_{\max}}{p_{\min}}) \right)$.
Chapter 3

Properties of Optimal Schedules

In this chapter we prove several properties of optimal solutions for the f-batch problem. We first discuss the case where all release dates are 0. Uzsoy considered this case in [U-95] and showed the following.

**Proposition 3.1** There exists an optimal solution for $1|f - \text{batch}| \sum w_j \bar{U}_j$, such that

(i) The schedule is left shifted, i.e., the first batch is scheduled at time 0, and any other batch is scheduled once the previous batch has completed.

(ii) All on-time jobs in the same family are ordered by the EDD rule.

(iii) All the batches with on-time jobs, except (maybe) the last batch from each family, are full.

(iv) Batches of different families are ordered by the minimal due date of the jobs in the batches.

**Proposition 3.2** There exists an optimal solution for $1|f - \text{batch}| \sum \bar{U}_j$ such that if $q_f$ jobs from family $f$ are scheduled in the solution, then these jobs are the $q_f$ jobs with largest due dates from that family.

**Proof:** Let $J_1$ be the job with the minimal due date, $d_1$, among the jobs that were scheduled from the family $f$, for some $1 \leq f \leq F$; let $J_2$ be the job with maximal due date, $d_2$, among the jobs that were not scheduled from this family. Then while $d_2 > d_1$, we replace $J_1$ by $J_2$. Obviously, the schedule remains feasible. Hence, we can keep replacing jobs, until we get that the set of scheduled jobs consists of those with the largest due dates in $f$. ■
**Proposition 3.3** The problem $1|f - \text{batch}, p_f = p| \sum w_j \bar{U}_j$ is equivalent to $1|f - \text{batch}, p_f = 1| \sum w_j \bar{U}_j$.

**Proof:** Let $\mathcal{I}$ be an instance of $1|f - \text{batch}, p_f = p| \sum w_j \bar{U}_j$. Generate an instance $\mathcal{I}'$ of $1|f - \text{batch}, p_f = 1| \sum w_j \bar{U}_j$, where all families have unit lengths and the due date of each job is $d'_j = \lfloor d_j/p \rfloor$. By Proposition 3.1, there exists an optimal schedule for $\mathcal{I}$ that is left shifted, and which contains only batches whose starting times are integral multiples of $p$. We transform a solution of $\mathcal{I}$ to a solution of $\mathcal{I}'$, by scheduling the same batches at the times $t'_j = t_j/p$. This solution induces a legal solution for $\mathcal{I}'$, with the same weight, since

$$d'_j = \lfloor d_j/p \rfloor \geq \lfloor \frac{t_j + p}{p} \rfloor = \lfloor \frac{p \cdot t'_j + p}{p} \rfloor = t'_j + 1.$$

On the other hand, any solution of $\mathcal{I}'$ can be converted to a legal solution of $\mathcal{I}$ with the same weight, by scheduling the same batches at times $t_j = p \cdot t'_j$

Since $t'_j + 1 \leq d'_j$, we have that

$$p \cdot t'_j + p \leq p \cdot d'_j$$

or

$$t_j + p \leq p \cdot \lfloor \frac{d_j}{p} \rfloor \leq d_j$$

The reverse direction is trivial. \hfill ■

In Propositions 3.4 - 3.7 we refer to the more general case, in which the jobs have release dates. The next result slightly modifies the definition of left shifted schedules to apply to this case.

**Proposition 3.4** There exists an optimal solution for $1|f - \text{batch}, r_j| \sum w_j \bar{U}_j$ that is left shifted, i.e., every batch is scheduled at a release date or at the end of another batch.

**Proof:** Let $\mathcal{S}_O$ be an optimal schedule in which the proposition holds for all batches scheduled up to time $t_{opt}$, where $t_{opt}$ is maximal.

Let $\Gamma$ the first batch in $\mathcal{S}_O$ scheduled at time $t_{\Gamma} > t_{opt}$. Let $r_{min}$ be the minimal release date of the jobs in $\Gamma$. We shift $\Gamma$ to the time $t_{new} = \max(r_{min}, t_{opt})$, which is either the end of the previous batch, or a release date. We can shift all the jobs in $\Gamma$, since $t_{new} \geq r_{min}$, and the time window from $t_{opt}$ to $t_{\Gamma}$ is unused. We get a schedule which is also optimal, but satisfies the proposition up to time $t_{new} + p_{\Gamma} > t_{opt}$ - a contradiction. \hfill ■

From Proposition 3.4 we can obtain a bound on the number of scheduling points in some optimal solution.
Corollary 3.5 The number of distinct scheduling points in a left shifted schedule for \( 1|f - \text{batch}, r_j| \sum w_j U_j \) is bounded by \( O(n^{F+1}) \).

Proof: Since in any left shifted schedule, all scheduling points are either at a release date or after a consecutive sequence of batches, the set of all possible scheduling points is given by

\[
Q = \left\{ r_k + \sum_{f=1}^{F} q_f p_f \left| \sum_{f=1}^{F} q_f \leq n, q_f \geq 0, 1 \leq k \leq R \right. \right\}
\]

The size of this set is bounded by the number of release points, multiplied by the number possible vectors \((q_1, \ldots, q_F)\). The number of such vectors satisfying \( \sum_{f=1}^{F} q_f \leq n \) and \( q_f \geq 0 \), is equal to the number of possible ways to distribute \( n \) identical balls into \( F + 1 \) cells (a cell for each family and a cell for balls corresponding to jobs that were not scheduled).

\[
|Q| \leq R \cdot \left( \frac{n + F}{F} \right)^F = O(n \cdot (n + F)^F) = O(n^{F+1})
\]

 Proposition 3.6 There exists an optimal solution for \( 1|f - \text{batch}, r_j| \sum w_j U_j \) with the following property. Let \( \Gamma \) be a batch of jobs from family \( f \) that was scheduled at time \( t_1 \) and contains a job with a maximum due date \( d_1 \); then no job from family \( f \) with release date \( r_2 < t_1 \) and with due date \( d_2 < d_1 \) is scheduled after time \( t_1 \).

Proof: Let \( S_\circ \) be an optimal schedule, in which the number of jobs that do not satisfy the above property is minimal. Let \( \Gamma_1 \) be a batch scheduled at time \( t_1 \) and \( \Gamma_2 \) be a batch scheduled at time \( t_2 > t_1 \). Let \( J_1 \) be the job with the maximal due date in \( \Gamma_1 \), denoted by \( d_1 \), and \( J_2 \) be a job in \( \Gamma_2 \) with release date \( r_2 \) and due date \( d_2 \), such that \( r_2 < t_1 \) and \( d_2 < d_1 \).

Since \( r_1 \leq t_1 < t_2 \) and \( d_1 > d_2 \geq t_2 + p_f \), we can schedule \( J_1 \) in \( t_2 \). Since \( r_2 < t_1 \) and \( d_2 \geq t_2 + p_f > t_1 + p_f \), we can schedule \( J_2 \) in \( t_1 \). Therefore, we can interchange \( J_1 \) and \( J_2 \) and obtain an optimal schedule, in which the number of jobs that do not satisfy the property is smaller – a contradiction.

Note that if some jobs share the same due date, we can transform the instance to an instance in which there is strict order of the due dates, by fixing the order between these jobs arbitrarily.

 Proposition 3.7 There exists an optimal solution for \( 1|f - \text{batch}, r_j, b > n_f| \sum w_j U_j \) with the following property. Given that a batch of family \( f \) was scheduled at time \( t \), all the jobs, \( J_j \), from family \( f \) with release dates \( r_j \leq t \) are not scheduled after time \( t \).
**Proof:** A job with \( r_j \leq t \) and \( d_j < t + p_f \) cannot be scheduled after time \( t \) and still meet its due date. Any job with \( r_j \leq t \) and \( d_j \geq t + p_f \) that was scheduled after time \( t \), can be moved to the batch scheduled at time \( t \). \( \blacksquare \)
Chapter 4

Fixed Number of Families

In this chapter we examine the f-batch problem in the special case where the number of families is a fixed constant. We show that this case is solvable in polynomial time.

4.1 Bounded Batch

We first consider the case where the maximum batch size is bounded (the bound can be either a fixed constant or a function of $n$).

**Theorem 4.1** The problem $1|f\text{-batch}, r_j, F = \text{const}|\sum w_j U_j$ can be solved optimally in $O(n^{F^2+3F+3}\log n)$ steps.

We show below that an optimal schedule can be constructed recursively, using dynamic programming. Our solution uses the structure described in Proposition 3.6.

Let $W(t, \tau_1, \ldots, \tau_F, \delta_1, \ldots, \delta_F)$ be the maximal weight of a schedule starting at time $t$, where the last batch of jobs from family $f$ was scheduled at time $\tau_f < t$ and $\delta_f$ is the largest due date of any scheduled job from family $f$, $1 \leq f \leq F$. Also, denote by $\text{nextarrival}(t) = \min_{r_j > t} r_j$. Then

$$\alpha_t = W(\text{nextarrival}(t), \tau_1, \ldots, \tau_F, \delta_1, \ldots, \delta_F) \quad (4.1)$$

is the maximal weight of a schedule, which starts at the time of the first arrival of some job after $t$, and in which the last batch of family $f$ started at $\tau_f < t$ and $\delta_f$ is the largest due date of any scheduled job from family $f$, $1 \leq f \leq F$.

Given a partial schedule that ends by the time $t$, in which the last batch from family $f$ was scheduled at time $\tau_f$, and the latest due date among the scheduled jobs from $f$ is $\delta_f$,
let $\delta'_f > \delta_f$ be the due date of some job from family $f$. We define $\mathcal{J}'_f$ to be a collection of subsets of jobs from family $f$ satisfying several inequalities (see below).

$$\mathcal{J}'_f = \{ \mathcal{J} | \mathcal{J} \leq b, \forall J_j \in \mathcal{J} f_j = f, r_j \leq t, d_j \geq t + p_f, d_j \leq \delta'_f, r_j > \tau_f \lor \delta_f < d_j \}$$

The maximum weight of any subset $\mathcal{J}$ in $\mathcal{J}'_f$ is given by

$$\text{maxweight}(f, t, \tau_f, \delta_f, \delta'_f) = \max \{ \sum_{j \in \mathcal{J}} w_j \}.$$  \hspace{1cm} (4.2)

Finally, given the scheduling time of the last batch from each family, $\tau_1, \ldots, \tau_{f-1}, t, \tau_{f+1}, \ldots, \tau_F$, for some $1 \leq f \leq F$, and the latest due date of any scheduled job, $\delta_1, \ldots, \delta_{f-1}, \delta'_f, \delta_{f+1}, \ldots, \delta_F$, we define

$$\beta_{f, \delta'_f} = W(t + p_f, \tau_1, \ldots, \tau_{f-1}, t, \tau_{f+1}, \ldots, \tau_F, \delta_1, \ldots, \delta_{f-1}, \delta'_f, \delta_{f+1}, \ldots, \delta_F) + \text{maxweight}(f, t, \tau_f, \delta_f, \delta'_f)$$

to be the sum of the maximal total weight of a schedule that starts at time $t + p_f$, and the maximum weight of a batch from family $f$ scheduled at the time $t$, in which the maximal due date of any job is $\delta'_f$.

**Lemma 4.2** $W(t, \tau_1, \ldots, \tau_F, \delta_1, \ldots, \delta_F)$ can be calculated by the following recursion:

$$W(t, \tau_1, \ldots, \tau_F, \delta_1, \ldots, \delta_F) = \max(\alpha_t, \max_{f, \delta'_f} \beta_{f, \delta'_f})$$

**Proof:** The recursion is derived from the following observation. Given that we have scheduled batches up to time $t$ and that the last batch from family $f$ was scheduled at time $\tau_f$, we can proceed in one of the following ways:

1. Select a family $1 \leq f \leq F$ and schedule a batch from family $f$ at time $t$. In this case, we need to determine the set of jobs that will be scheduled. Given that the maximum due date of a job in the batch is $\delta'_f$, the selection is uniquely determined. The batch can only contain jobs $J_j$ from family $f$ that can be scheduled (i.e. $f_j = f$, $r_j \leq t$), $d_j \geq t + p_f$, have a due date $d_j \leq \delta'_f$, and were not scheduled in previous batches. By Proposition 3.6, among the jobs that were not scheduled (from family $f$), we can select either new jobs, which arrived after the start time of the last batch of family $f$ ($r_j > \tau_f$), or jobs with a due date which is larger than the maximum due date among the scheduled jobs (i.e. $d_j > \delta_f$). Note that these jobs will not be considered again as we proceed in the recursive calculation. Now, we can define the collection of subsets of at most $b$ jobs that can be scheduled. Since the selection will not affect the remaining computation, clearly, the best is to select a subset $\mathcal{J}$ with maximal weight.

21
2. Move to the next possible batch starting point, which is (by Proposition 3.4) the time of closest job release date.

3. If neither of the above is possible, then no more jobs can be scheduled.

Proof of Theorem 4.1: Let DP be a dynamic programming algorithm which calculates $W$ using the above recurrence relation. The weight of an optimal schedule can be found by calculating $W(0, 0, \ldots, 0, 0, \ldots, 0)$. This is the maximum weight of a schedule starting at time 0, where no batches were scheduled, or alternately, where dummy batches, containing jobs with zero weight and zero length, were scheduled at time zero for each family. We can obtain the actual schedule by storing the decision made in each step in a separate table.

It may seem that the running time and space requirements of the algorithm depend on the maximum length of the schedule. However, since (by Proposition 3.5), the number of possible scheduling points is $O(n^{F+1})$, and the number of possible due dates is bounded by $n$, the total number of points for which $W$ is calculated is bounded by

$$O ((n^{F+1})(F+1) \cdot n^F) = O(n^{(F+1)^2 + F}).$$

Finding the optimal batch for a specific scheduling point can be done by examining all possible new maximum due dates: for each we find the set of feasible jobs, sort them by weights, and select up to $b$ jobs with maximal weight from each family. This can be done in $O(n \cdot n \log n)$ steps. Hence, we get that the overall running time is $O(n^{F^2 + 3F + 2} \log n)$.

4.2 Unbounded Batch

Consider the case where the size of the batch is unbounded, that is, any number of jobs from the same family can be batched at the same time, assuming that the release date and due date conditions are met. For this case, we can derive a simpler and slightly more efficient algorithm.

Theorem 4.3 The problem $1|f-batch, r_j, b > n_f, F = const| \sum w_j \bar{U}_j$ can be solved optimally in $O(n^{(F+1)^2 + 1})$ steps.

Definition 4.1 Let $W(t, \tau_1, \ldots, \tau_F)$ be the maximal weight of a schedule starting at time $t$, where the last batch from family $f$ was scheduled at time $\tau_f$, for $1 \leq f \leq F$. 

22
Lemma 4.4  \( W(t, \tau_1, \ldots, \tau_F) \) can be calculated by the following recursion:

\[
W(t, \tau_1, \ldots, \tau_F) = \left\{ \begin{array}{ll}
0 & \forall f, \text{waiting}(\tau_f, t, f) = 0, \text{nextarrival}(t) = \infty \\
\max \left( W(\text{nextarrival}(t), \tau_1, \ldots, \tau_F), \right.
\max_f W(t + p_f, \tau_1, \ldots, \tau_{f-1}, \tau_{f+1}, \ldots, \tau_F) + \text{waiting}(\tau_f, t, f) \bigg) & \text{otherwise}
\end{array} \right.
\]

where

\[
\text{nextarrival}(t) = \min_{r_j > t} r_j
\]

\[
\text{waiting}(\tau_f, t, f) = \sum_{J_j : f_j = f, \tau_j < r_j \leq t, t + p_f \leq d_j} w_j
\]

Proof: The recursion is derived from the following observation. Given that we have scheduled batches up to time \( t \), and that the last batch from family \( f \) was scheduled at time \( \tau_f \), we can proceed in one of the following ways:

1. Select a family \( f \) and schedule a batch from family \( f \) at time \( t \): the batch will contain all the jobs from family \( f \) that accumulated since the last schedule of a batch from this family. These are jobs \( J_j \) with \( f_j = f, \tau_j < r_j \leq t \), \( t + p_f \leq d_j \).

2. Move to the next possible batch starting point, which is the closest job release date.

3. If neither of the above is possible, then no more jobs can be scheduled.

Proof of Theorem 4.3: Let DP be a dynamic programming algorithm, which calculates \( W \) using the above recurrence relation. The weight of an optimal schedule can be found by calculating \( W(0, 0, \ldots, 0) \). Since every coordinate of \( W(t, \tau_1, \ldots, \tau_F) \) is a possible scheduling point, the total number of calculated points is bounded by \( O(n^{(F+1)^2}) \). Finding the weight of the waiting jobs from each family at any time can be done in \( O(n) \) step using bucketing. Thus, the total running time is

\[
O \left( \frac{n^{(F+1)^2}}{n} \right) \cdot \frac{n}{O(n)} = O(n^{(F+1)^2+1})
\]

number of calculation points calculation per point
Chapter 5

Optimal Algorithms for Arbitrary Number of Families

5.1 The Feasibility Problem

In the feasibility problem, $1|f - \text{batch}, r_j|L_{\text{max}} > 0$, we need to determine whether all jobs can be scheduled and meet their due dates. With no release dates, the problem $1|L_{\text{max}} > 0$ is solvable in $O(n \log n)$ by the EDD rule, while the problem with release dates is strongly NP-hard, by reduction from 3-PARTITION [LRK+77]. Uzsoy showed in [U-95] that the family batching feasibility problem with no release dates is solvable in $O(n \log n)$ steps, using the following variant of algorithm EDD.

Algorithm Family-EDD
Sort the jobs in each family by due dates.
While there are still unscheduled jobs
    Select the family $f$ with the job with the minimum due date;
    Schedule (up to) $b$ jobs with earliest due dates from family $f$
    as a new batch.

5.2 No Release Dates and Unbounded Batch

Consider the case where there are no release dates, all the jobs have the same length, and the batch size is unbounded.

Theorem 5.1 The problem $1|f - \text{batch}, p_r = p, b > n_f|\sum w_j \bar{U}_j$ can be solved in $O(n + F^3)$ steps.
**Proof:** We first note that, by Proposition 3.3, we need to consider only the case of unit processing times. Furthermore, under these conditions, Proposition 3.1 implies that a single batch is scheduled from each family. This batch contains all the jobs from family $f$ that met their due date at the batch completion time. Also, there exists an optimal left-shifted schedule, in which the maximum starting time of any batch is $F - 1$. Since each time slot can contain at most one batch, and each family can be scheduled at only one time slot, we can reduce the problem to the following instance of the assignment problem.

Let $X = \{1 \ldots F\}$, $T = \{0 \ldots F - 1\}$. Let $G = (X, T, E)$ be a complete bipartite graph, in which the weight of any edge $e = (f, t)$ is given by $w(f, t) = \sum_{j: f_{j} = f, \tau_{j} \leq t + 1} w_{j}$, for all $1 \leq f \leq F$ and $0 \leq t \leq F - 1$. Find a maximum weight matching in $G$. This assignment problem can be solved in $O(F^3)$ steps [K-55]. Thus, the running time of the algorithm is $O(n + F^3)$ steps.

\[\square\]

### 5.3 Constant Number of Release Dates and Due Dates

We now consider the special case, where the number of distinct release dates, $R$, and number of distinct due dates, $D$, are some fixed constants. Also, the batch size is fixed, and all jobs have the same (unit) job length. We give an exact polynomial time algorithm for this problem.

Note that with slight modification, this problem becomes hard to solve. In particular, if we allow jobs of different families to have different lengths, the problem becomes weakly NP-complete, since the classical scheduling problem with release dates and due dates is weakly NP-complete, even when the release dates and due dates each take only two values [GJ-79].

We describe a polynomial time algorithm that operates in two phases. The algorithm partitions the jobs into a fixed number of groups. In the first phase, the algorithm schedules greedily as many full batches as possible from each group. At the end of the first phase, only a fixed number of jobs from each group remain to be scheduled. In the second phase, the algorithm uses dynamic programming to optimally schedule the remaining jobs.

The first phase of the algorithm is called Reverse-Full-Greedy (RFG). It first partitions the jobs to groups by their release dates, due dates and families. Then, RFG iteratively schedules batches ‘backwards’: from the maximum due date to time zero. The jobs are scheduled using time windows. Let $d_{0} = 0, d_{1} \leq \ldots \leq d_{D}$ be the set of due dates of the given instance; then, the $l$-th windows is the time interval $[d_{l-1}, d_{l}), 1 \leq l \leq D$.

In the $l$-th window, RFG schedules as many full batches as possible from each group that meets the window’s maximal due date, $d_{l}$. In doing this, RFG gives higher priority to groups with large release dates. If the $l$-th window is large enough, then at most $b - 1$ unscheduled jobs will remain from each of the groups. If the window is not large enough (i.e. the window
is completely filled with full batches), then the remaining jobs from each group are moved to the group from the same family and release date with maximal due date $d_{i-1}$. If such a group does not exist, we do not schedule these jobs.

The following notation will be used in the algorithm RFG.

- $X_{f,k,l}$ denotes the set of jobs from family $f$ with release date $r_k$ and due date $d_i$.
- $Z_{f,k,l}$ is the set of jobs from family $f$ with release date $r_k$ that are candidates for scheduling in the time window $[d_{i-1}, d_i]$; initially, $Z_{f,k,l} = X_{f,k,l}$.
- $\gamma_{f,k,l}$ is the number of full batches of family $f$ with release date $r_k$ that are scheduled in the time window $[d_{i-1}, d_i]$;
- $S_{f,k,l}$ is the set of jobs contained in batches of family $f$ with release date $r_k$ scheduled in the time window $[d_{i-1}, d_i]$; thus $|S_{f,k,l}| = b \cdot \gamma_{f,k,l}$.
- $Y_{f,k,l}$ is the set of jobs from family $f$ with release date $r_k$ that were not scheduled at the time window $[d_{i-1}, d_i]$, and were not shifted to an earlier due date.

**Phase 1: Algorithm RFG**

for $l = 1$ to $D$
  for $k = 1$ to $R$
    for $f = 1$ to $F$
      $Z_{f,k,l} = X_{f,k,l}$

  for $l = D$ down to $1$
    $\delta_l = d_i - d_{i-1}$
    for $k = R$ down to $1$
      for $f = F$ down to $1$
        if $r_k \leq d_i$ then
          $\gamma_{f,k,l} = \min \{ \lfloor \frac{|Z_{f,k,l}|}{b} \rfloor, \delta_l \}$
          Let $S_{f,k,l}$ be an arbitrary subset of $Z_{f,k,l}$ such that $|S_{f,k,l}| = b \cdot \gamma_{f,k,l}$.
          $Y_{f,k,l} = Z_{f,k,l} / S_{f,k,l}$
          $\delta_l = \delta_l - \gamma_{f,k,l}$
          if $\delta_l = 0$ then
            $Z_{f,k,l-1} = Z_{f,k,l-1} \cup Y_{f,k,l}$
            $Y_{f,k,l} = \emptyset$
          end if
        else
          $Y_{f,k,l} = \emptyset$
          $Z_{f,k,l} = \emptyset$

26
Note that since we schedule full batches until either the window is full or all possible full batches were scheduled, when the first phase of the algorithm terminates, $Y_{f,k,l}$, the number of unscheduled jobs from family $f$, release date $r_k$ and due date $d_l$ that we need to consider is at most $b - 1$.

In the second phase we exhaustively search for the optimal schedule of the remaining jobs in some “holes” that remained in the schedule. Algorithm DP uses the fact that all families with the same number of remaining jobs in each group, $Y_{f,k,l}$ for all values of $l, k$, are equivalent with respect to scheduling possibilities. The state of the algorithm is composed of the holes that need to be filled, and the number of families from each configuration. Two families $f, g$ have the same configuration if $Y_{f,k,l} = Y_{g,k,l}$ for all $1 \leq k \leq R, 1 \leq l \leq D$. When we schedule a batch from a family $f$ at some specific hole, we choose to schedule (up to) $b$ available jobs with minimal due dates from this family. Since this changes the configuration of the family, we modify the state by decreasing the number of families in the original configuration and increasing the number of families in the new configuration.

The next example shows how we update the corresponding configuration vectors when we schedule jobs from a specific family in the second phase.

The vector $\vec{v}_f$ represents the size of each of the groups of the family $f$, at the end of the first phase. Let $[b - 1] = \{0, \ldots, b - 1\}$, and denote by $y_{f,k,l} = |Y_{f,k,l}|$, i.e., the size of the set $Y_{f,k,l}$. Then we define $\vec{v}_f \in [b - 1]^{DR}$ as the vector $(y_{f,1,1}, \ldots, y_{f,1,D}, \ldots, y_{f,R,1}, \ldots, y_{f,R,D})$.

**Example 5.3.1** Assume that $r_k = (k - 1) \ast 5$ for $1 \leq k \leq 3$, $d_j = j \ast 5$ for $1 \leq j \leq 3$, $b = 10$, and for the family $f$, $y_{f,1,1} = 4; y_{f,1,2} = 3; y_{f,1,3} = 6; y_{f,2,2} = 2; y_{f,2,3} = 9; y_{f,3,3} = 3$; then $f$ has the following configuration vector.

$$
\vec{v}_f = \begin{pmatrix}
d_1 = 5 & d_2 = 10 & d_3 = 15 \\
r_1 = 0 & 4 & 3 & 6 \\
r_2 = 5 & 0 & 2 & 9 \\
r_3 = 10 & 0 & 0 & 3
\end{pmatrix}
$$

The entry in row $k$ and column $l$ gives the size of $Y_{f,k,l}$; thus there are, for example, 4 jobs whose release date is 0 and due date is 5. Now, if we schedule in the time window $[5,10)$, then we select the 3 jobs from $Y_{f,1,2}$, the 2 jobs from $Y_{f,2,2}$ and 5 jobs (out of 6) from $Y_{f,1,3}$.

$$
\vec{v}_f' = \begin{pmatrix}
d_1 = 5 & d_2 = 10 & d_3 = 15 \\
r_1 = 0 & 4 & 0 & 1 \\
r_2 = 5 & 0 & 0 & 9 \\
r_3 = 10 & 0 & 0 & 3
\end{pmatrix}
$$

27
Let $Q_\vec{v}$ to be the number of families for which $\vec{v}_f = \vec{v}$ for $\vec{v} \in \{b - 1\}^D$. Let $\text{free}_t$ the number of free time slots at the end of Phase 1 in the time window $[d_{i-1}, d_i)$.

Define

$$H_k = \left( \underbrace{1, 1, \ldots, 1, 1}_{\text{free}_1}, \underbrace{2, 2, \ldots, 2}_{\text{free}_2}, \ldots, \underbrace{D, D, \ldots, D}_{\text{free}_\ell} \right)$$

Let $W(h, Q_{(0,0,\ldots,0)}, Q_{(1,0,\ldots,0)}, Q_{(b-1,0,\ldots,0)}, \ldots, Q_{(b-1,b-1,\ldots,b-1)})$ be the optimal weight of a schedule of batches at times $H_k \ldots H_{|H|}$ where there are $Q_\vec{v}$ families with $\vec{v}_f = \vec{v}$.

**Phase 2: Algorithm DP**

Solve the following dynamic programming problem.

$$W(h, Q_{(0,0,\ldots,0)}, Q_{(1,0,\ldots,0)}, Q_{(b-1,0,\ldots,0)}, \ldots, Q_{(b-1,b-1,\ldots,b-1)}) =$$

$$\max_{\vec{v} \in \{b-1\}^D} \left\{ W(h+1, Q_{(0,0,\ldots,0)}, Q_{\vec{v}-1}, \ldots, Q_{\vec{v}+1}, \ldots, Q_{(b-1,b-1,\ldots,b-1)}) + \text{gain}(H_k, \vec{v}) \right\}$$

where $\text{gain}(h, \vec{v})$ and $\vec{v}'(h, \vec{v})$ are calculated as follows:

$\text{gain}=0$ $\vec{v}' = \vec{v}$

for $l=h$ to $D$

for $k=1$ to $R$

if $r_k \leq d_h$ then

$\text{take} = \min \left( \vec{v}_{k,l}, b - \text{gain} \right)$

$\vec{v}'_{k,l} = \vec{v}_{k,l} - \text{take}$

$\text{gain} = \text{gain} + \text{take}$

end if

end for

end for

The recursion is terminated by

$$W(|H| + 1, *, *, \ldots, *) = 0$$
Return the optimal value

\[ W(1, Q_{(0,0,...,0)}, Q_{(1,0,...,0)}, Q_{(b-1,0,...,0)}, \ldots, Q_{(b-1,b-1,...,b-1)}) \]

**Theorem 5.2** Let \( b, D, R \geq 1 \) be some constants. Then the problem \( \min f - \text{batch}, p_f = 1, b = \text{const}, D = \text{const}, R = \text{const} \sum U_j \) can be solved in \( O(n^{bDR+1}) \).

We prove the optimality for each of the two phases using the next lemmas.

**Lemma 5.3** Let \( S \) be the schedule output by RFG. There exists an optimal schedule \( S_O \), which consists of all the batches in \( S \), and some batches formed from jobs in \( Y_{f,k,l} \).

**Proof:** Recall that RFG schedules \( \gamma_{f,k,l} \) full batches from family \( f \) of jobs with release date \( r_k \) in the \( l \)-th time window. Let \( (f, k, l) \) and \( (f', k', l') \) be two tuples, \( 1 \leq f, f' \leq F, 1 \leq k, k' \leq R \) and \( 1 \leq l, l' \leq D \). We say that \( (f', k', l') \succ (f, k, l) \) if (i) \( l' > l \) or (ii) \( l' = l \) and \( k' > k \) or (iii) \( l' = l \) and \( k' = k \) and \( f' > f \).

Let \( S_O \) be an optimal schedule in which \( \gamma_{f', k', l'} \) full batches are scheduled from family \( f' \) with release date \( r'_k \) in the \( l' \)-th time window, for all \( (f', k', l') \succ (f, k, l) \), while less than \( \gamma_{f,k,l} \) batches from \( f \) with release date \( r_k \) are scheduled in the \( l \)-th time window, and \( (f, k, l) \) is minimal. We say that \( S_O \) is the optimal solution which is most similar to \( S \).

Assume that \( S_O \) has less than \( \gamma_{f,k,l} \) full batches from family \( f \) at time window \( l \). Note that jobs from \( X_{f,k,l} \) cannot be scheduled in time window \( l' > l \), since their due date is \( d_l \). Therefore, these jobs can only be scheduled at the time window \( l \) or earlier.

If \( S_O \) does not completely fill time window \( l \), or if there exists an incomplete batch of jobs from \( X_{f,k,l} \), then a full batch can be created by taking of unscheduled jobs and by shifting jobs from earlier time windows to this window.

If \( S_O \) completely fills the \( l \)-th time window, then we pick a batch which does not belong to the \( \gamma_{f', k', l'} \) batches of jobs where \( (f', k', l') \succ (f, k, l) \). If another batch from \( X_{f,k,l} \) was scheduled at an earlier time, we swap the two batches and fill the batch as before. If no earlier batch from \( X_{f,k,l} \) was scheduled, we remove the selected batch completely and replace it by a full batch from \( X_{f,k,l} \).

We repeat the process until we obtain \( \gamma_{f,k,l} \) full batches, and thus reach a contradiction to the assumption that \( S_O \) is the optimal schedule which is most similar to \( S \).

Assume that \( S_O \) contains a batch of jobs which do not belong to \( \{Y_{f,k,l} \cup S_{f,k,l}\} \). These jobs were shifted left, beyond their release dates, because their time window became full. Since only \((d_l - d_{l-1}) \cdot b\) jobs with due date \( d_l \) can be scheduled after time \( d_{l-1} \), the “effective”
due date of the remaining jobs is \( d_{i-1} \). Any algorithm has to choose which jobs to schedule and which to shift left. RFG schedules the jobs with the largest release dates, which can be shown to be optimal using an interchange argument.

\[ \square \]

**Lemma 5.4** The DP algorithm schedules optimally the remaining jobs \( Y_{f,k,l} \) from Phase 1.

**Proof:** The algorithm uses the fact that all families with the same configuration vector are equivalent with respect to scheduling possibilities. In other words, if it is possible to schedule a batch of jobs from family \( f \) at time \( t \), then it is possible to schedule a batch of the same size from family \( f' \) at the same time, provided that \( \bar{v}_{f'} = \bar{v}_f \). Assume that we have a family \( f \) with configuration vector \( \bar{v}_f = \bar{v} \), and we decide to schedule a batch in the time window \([d_{i-1}, d_i]\). Given that all the remaining batches will be scheduled at later time window, we can schedule the (up to \( b \)) jobs with the smallest due dates and a legal release date.

\[ \square \]

**Proof of Theorem 5.2:** By Lemma 5.3, there exists an optimal schedule, \( S \), that schedules batches of the jobs from \( S_{f,k,l} \) the same as RFG, and additional batches of jobs from \( Y_{f,k,l} \). By Lemma 5.4, the schedule produced by RFG can be extended by DP to optimally schedule the jobs from \( Y_{f,k,l} \).

The running time of the algorithm is dominated by the second phase, and is thus given by

\[
O\left( \frac{|H| \cdot F^{(b^{DR})}}{\text{size of the table}} \cdot \frac{(b^{DR}) \cdot DR}{\text{calculation per entry}} \right) = O\left( n \cdot n^{(b^{DR})} \cdot (b^{DR}) \cdot DR \right) = O\left( n^{(b^{DR}+1)} \right)
\]

In order to obtain the optimal schedule, we can store in the table in each step the chosen \( \bar{v} \), in addition to the optimal value. Once we find the optimal solution, we scan the selection that leads to the optimal value \( \bar{v}_1, \bar{v}_2, \ldots, \bar{v}_h, \ldots, \bar{v}|H| \). We can then construct the solution, by scheduling at the time window \( H_k \) a family \( f \) with \( \bar{v}_f = \bar{v}_h \).

\[ \square \]
Chapter 6

Approximation Algorithms for Arbitrary Number of Families

6.1 Greedy Algorithms

In this section we study two natural greedy algorithms for the f-batch problem. We first show an easy reduction from family batching to the real-time scheduling problem.

Proposition 6.1 Let $A$ be a $c$-approximation algorithm for real-time scheduling. Then $A$ is a $cb$-approximation for the $f$-batch problem.

Proof: Let $I_b$ be an instance of the family batching problem with batch size $b$, and let $I$ be the same instance without batching.

Let $S_{\Omega}$ be an optimal solution for $I_b$. We convert $S_{\Omega}$ to $S'$, a feasible solution for $I$, by selecting the single job with maximal weight from each batch. The weight of the solution is decreased by a factor of at most $b$. Let $S_A$ be the solution produced by $A$ for $I$; $A$ is a $c$-approximation for $I$, thus the weight of $S_A$ is within factor $c$ from the weight of any feasible solution for $I$. We conclude that

$$OPT(I_b) = W(S_{\Omega}) < b \cdot W(S') < cb \cdot W(S_A) = cb \cdot A(I)$$

In the following discussion of greedy algorithms, we assume that there are no release dates and that all jobs are of equal lengths.
6.1.1 The Unweighted Problem

The classical scheduling problem, 1 | p_j = 1 | \sum \tilde{U}_j, can be solved in \( O(n) \) steps, using the EDD rule (and bucket sorting). Recall that Family-EDD solves 1\( |L_{\text{max}} > 0 \). However, as we show below, a natural variant of this algorithm to family batching is sub-optimal, when the objective is to maximize the number of on-time jobs.

**Algorithm Family-EDD**
Sort the jobs in each family by due dates.
While there are still jobs that can be scheduled on-time
Select the family \( f \) containing the job with the minimum due date
Schedule (up to) \( b \) jobs with earliest due dates from family \( f \) as a new batch.

**Theorem 6.2** Family-EDD is a \( b \)-approximation for 1\( |f - \text{batch}, p_f = 1 | \sum \tilde{U}_j \), and the bound is tight.

**Proof:** The upper bound follows directly from the optimality of EDD for 1\( |p_j = 1 | \sum \tilde{U}_j \) and from Proposition 6.1.

For the lower bound consider an instance with two families: the first family consists of one job with due date 1, and the second consists of \( b \) jobs also with due date 1. FAMILY-EDD is indifferent to the selection of the family that will be scheduled. It may schedule the job from the first family, and thus obtain the worst case approximation ratio of \( b \).

6.1.2 The Weighted Problem

Now we consider the problem 1\( |f - \text{batch}, p_f = p | \sum w_j \tilde{U}_j \), where all jobs have the same length and no release dates are specified. We first note that by Proposition 3.3, this problem is equivalent to 1\( |f - \text{batch}, p_f = 1 | \sum w_j \tilde{U}_j \). We can also assume that the maximum due date is bounded by \( n \), since left shifted schedules dominate.

We describe a greedy algorithm and analyze its performance, for both the non-batching and batching scenarios. Let \( \mathcal{Y} \) be the set of all jobs. Denote by \( d_{\text{max}} \) the maximal due date of any job in \( \mathcal{Y} \).

**Algorithm Rev-Greedy**
for \( t = d_{\text{max}} \) down to 1
Let \( Y_t \) be the set of jobs with maximum weight among all possible subsets of \( \mathcal{Y} \) of size \( \min(b, |\mathcal{Y}|) \), where the jobs in each subset are from the same family.
\( Y = \mathcal{Y} / Y_t \)
Theorem 6.3 Algorithm Rev-Greedy solves \(1|p_j = 1| \sum w_j \bar{U}_j\) optimally.

Proof: Let \(S\) be the solution produced by the Rev-Greedy algorithm. Let \(S_O\) an the optimal solution in which all jobs from time \(t\) through the end of the schedule are scheduled exactly as in \(S\), and \(t\) is minimal. Let \(J_k\) be the job scheduled by \(S\) at time \(t-1\). There must be a job, \(J_i\), scheduled in \(S_O\) at time \(t-1\). Otherwise, if \(J_k\) was scheduled earlier in \(S_O\) it could be moved to time \(t\) - contradicting \(t\)'s minimality, and if \(J_k\) was not scheduled in \(S_O\) it can be added to \(S_O\) - contradicting \(S_O\)'s optimality. Since Rev-Greedy selected \(J_k\) and not \(J_i\) at time \(t-1\), it follows that \(w_k \geq w_i\). Thus, if \(J_k\) was not scheduled in \(S_O\), then we replace \(J_i\) with \(J_k\) and obtain an optimal solution identical to \(S\) from time \(t-1 < t\) - a contradiction. Finally, if \(J_k\) was scheduled in \(S_O\) at some earlier time, we can interchange \(J_k\) and \(J_i\) in \(S_O\) to obtain an optimal solution identical to \(S\) from time \(t-1 < t\) - a contradiction.

Unfortunately, Rev-Greedy does not perform that well in the family batching environment.

Theorem 6.4 Rev-Greedy is a 2-approximation for \(1|f = \text{batch}, p_f = 1| \sum w_j \bar{U}_j\), and the bound is tight even for \(1|f = \text{batch}, p_f = 1, b = 2| \sum \bar{U}_j\). Its running time is \(O(n^2 \log n)\) steps.

Proof: Let \(S_O\) be an optimal solution for a given instance, \(I\), and let \(S\) be the solution produced by Rev-Greedy for the same instance. Let \(Y_t\) be the set of jobs scheduled in \(S\) at time \(t\), and \(X_t\) be the set of all jobs scheduled in \(S_O\) at time \(t\) that were not scheduled in \(S\).

Since both \(X_t\) and \(Y_t\) were available at time \(t\) and Rev-Greedy chose to schedule \(Y_t\),

\[
W(X_t) \leq W(Y_t)
\]

Therefore,

\[
W(S) = \sum_{t=1}^{n} W(Y_t) \geq \sum_{t=1}^{n} W(X_t)
\]

If \(\sum_{t=1}^{n} W(X_t) \geq \frac{1}{2} W(S_O)\) then we are done; otherwise, assume that

\[
\sum_{t=1}^{n} W(X_t) < \frac{1}{2} W(S_O) \quad (6.1)
\]

We can write

\[
S_O = (S_O \cap \bigcup Y_t) \cup (S_O \cap \bigcup Y_t)
\]

Hence, we get that

33
\[ W(S_0) \leq \sum_{i=1}^{n} W(X_i) + \sum_{i=1}^{n} W(Y_i) \]

or

\[ W(S_0) - \sum_{i=1}^{n} W(X_i) \leq \sum_{i=1}^{n} W(Y_i) = W(S) \]

From Equation (6.1) we conclude that

\[ \frac{W(S_0)}{2} < W(S) \]

We proceed to show that the bound is tight. Consider the following instance. The input consists of jobs from \( F \) families, where each family contains two jobs of unit length. The job \( J^f_1 \) from family \( f \) has due date \( f \). The job \( J^f_2 \) from family \( f \) has due date \( f + 1 \), for any \( 1 \leq f \leq F \) (see Figure 6.1).

\[ \begin{array}{cccccccc}
0 & J^F_2 & J^F_1 & J^2_2 & J^2_1 & J^1_2 & J^1_1 \\
1 & & & & & & & \\
F-1 & & & & & & & \\
F & & & & & & & \\
F+1 & & & & & & & \\
\end{array} \]

Figure 6.1: A worst case instance for Rev-Greedy

Note that Rev-Greedy first schedules \( J^F_2 \) at time \( F \). At time, \( F - 1 \) it can either schedule \( J^F_1 \) or \( J^{F-1}_2 \). Both will yield the same profit. W.l.o.g. we assume that the algorithm selects \( J^{F-1}_2 \). At time \( F - 2 \), Rev-Greedy can either schedule \( J^F_1 \), \( J^{F-1}_1 \) or \( J^{F-2}_2 \). Again, w.l.o.g. we assume that \( J^{F-2}_2 \) is selected. In general, Rev-Greedy always selects \( J^f_2 \) at time \( f \), for \( f \geq 1 \). Finally, \( J^1_1 \) is scheduled at time 0. The total number of jobs scheduled by Rev-Greedy is \( F + 1 \). Note that we can force these selections by assigning the weight \( 1 + \epsilon \) to the jobs \( J^f_2 \) and the weight 1 to the jobs \( J^f_1 \), for \( 1 \leq f \leq F \).

34
In contrast, the optimal algorithm can schedule all $2F$ jobs by taking at time $f - 1$ a batch containing $J_f^i$ and $J_f^j$, for $1 \leq f \leq F$. Thus, the approximation ratio of Rev-Greedy is at least $\frac{2F}{F+1} \to 2$.

Each iteration can be performed in three steps: calculating the available jobs; sorting these jobs by weight and family; and selecting the family with maximum batch weight. Thus, the algorithm running time is $O(n^2 \log n)$ steps.

6.2 Local Ratio Algorithms

In this section we apply the Local Ratio technique developed in [BB+00] to family batching problems. We give a polynomial time $(2+\epsilon)$-approximation algorithm for $1|f-batch,r_j,b > n_f|\sum w_j U_j$ and $1|f-batch|\sum U_j$.

6.2.1 Overview of the Local Ratio Technique

Assume we are given a maximization problem, defined by a set of constraints $T$ and a weight function $w$, and our goal is to find a feasible solution of maximum weight for which the constraints are satisfied. In the Local Ratio technique we decompose the weight function into two weight functions, $w_1$ and $w_2$, whose sum is the original weight function. We then find a solution which is an $r$-approximation for the two subproblems $(T, w_1)$ and $(T, w_2)$. By the Local Ratio theorem, this solution is an $r$-approximation for the original problem.

The question that remains is how to decompose the weight function and find a solution that approximates both subproblems. The Local Ratio technique suggests a general scheme for this: decompose the problem such that one problem will have an “easy” weight function, $w_1$, for which any maximal feasible solution is an $r$-approximation, and the second problem will have the remainder weight function, $w_2 = w - w_1$. We recursively solve the problem for $w_2$ and obtain a solution. We greedily extend this solution until we obtain a maximal solution. Now, assuming the recursive solution was an $r$-approximation for $(T, w_2)$, it is now an $r$-approximation for $(T, w_1)$ as well, and thus, by the Local Ratio Theorem, an $r$-approximation for $(T, w)$.

In the discrete real-time scheduling problem, each job can be scheduled in a specified number of time points (called job instances). The decomposition of the problem is done by finding the job instance, $\bar{T}$, that completes first and then setting the weight $w_1$ of all the instances of this job, $G(\bar{T})$, and all instances of other jobs which overlap this instance in time, $N(\bar{T})$, to $\alpha = w(\bar{T})$. The weight of all other instances in $w_1$ is set to zero. Any feasible solution for $w_1$ can have a weight of at most $2\alpha$, since at most one batch can be selected from both $G(\bar{T})$ and $N(\bar{T})$. On the other hand, any maximal solution must contain one instance from $G(\bar{T}) \cup N(\bar{T})$ or else $\bar{T}$ could be added to it. Thus, any maximal solution is
a 2-approximation for \( w_1 \), as required. Any job instance with zero or negative weight can be removed since it will not appear in any solution. Since the weight of at least one job instance is set to zero in \( w_2 \) in each decomposition, the above recursive procedure will terminate.

It was shown in [BB+00], that the algorithm can be implemented to output a 2-approximation solution in \( O(n \log n) \) steps, where \( n \) is the number of job instances. If jobs have time windows, the algorithms can be implemented to yield a \( 2(1 + \epsilon) \)-approximation in \( n^2/\epsilon \) steps, independent of the size of the time windows.

### 6.2.2 The Modified Local Ratio Technique

When we consider the f-batch problem, the above real-time scheduling decomposition does not yield a 2-approximation. Specifically, given the selection returned from the recursive call for \( w_2 \), we may not be able to schedule more than one a job with non-zero weight in \( w_1 \), while a different feasible solution for \( w_1 \) could schedule \( b \) jobs. Thus, we take different approach: rather than solving directly the f-batch problem, we reduce it to another scheduling problem:

**Reduction**

1. Construct all possible batch instances, \( B \), that can appear in a solution.
2. Two batches can be selected for the solution if (i) they do not overlap in time and (ii) the corresponding set of jobs are disjoint.
3. Set the weight of a batch to be the sum of the weights of the jobs it contains.

Note that the constraints in 2 generalize the real-time scheduling constraints (where two job instances can be scheduled if they do not overlap in time and do not belong to the same job). We now solve the problem using the following Local Ratio decomposition.

**Algorithm Local-Ratio**

1. Delete all batch instances with zero or negative profit. If \( B = \emptyset \) then output the empty schedule, else proceed to the next step.
2. Select a batch instance \( \bar{\Gamma} \) with minimal completion time. If several batch instances have the same completion time, select the batch containing the job with smallest due date. Decompose \( w(\Gamma) \) by \( w(\Gamma) = w_1(\Gamma) + w_2(\Gamma) \) as follows. Let \( N(\bar{\Gamma}) \) be the set of all the batches in \( B \) that overlap \( \bar{\Gamma} \) and let \( G(\bar{\Gamma}) \) be the set of all the batches in \( B \) that share a job with \( \bar{\Gamma} \). Finally, let \( \alpha = w(\bar{\Gamma}) \).

\[
w_1(\Gamma) = \begin{cases} \alpha & \Gamma \in G(\bar{\Gamma}) \cup N(\bar{\Gamma}) \\ 0 & \text{otherwise} \end{cases}
\]
3. Solve the problem recursively using \( w_2 \) as the weight function. Let \( S' \) be the resulting schedule.

4. If \( S' \cup \{ \bar{t} \} \) is a feasible solution, return \( S = S' \cup \{ \bar{t} \} \). Otherwise, return \( S = S' \).

We are faced with two problems: the number of batch instances is exponential in \( b \), and we need to show that any maximal solution for \( w_1 \) is a 2-approximation.

Both these issues are resolved by a critical modification of the algorithm. First, we construct only batch instances that can appear in a specific set of optimal solutions. There is a polynomial number of such batch instances. Secondly, we will measure the performance of our approximation algorithm against a restricted optimal algorithm which returns solutions that have a certain structure. In particular, in the restricted optimal schedule, any batch of family \( f \) scheduled at time \( t \) contains all available jobs from family \( f \) that arrived after the time the previous batch of family \( f \) was scheduled. Since by Proposition 3.7, there exists an optimal solution with this property, any approximation ratio shown against the restricted optimal algorithm holds for any optimal algorithm.

**Theorem 6.5 Local-Ratio** is a \((2 + \varepsilon)\) approximation algorithm for the \(1|f-batch, r_j, b > n_f| \sum w_j \bar{u}_j \) problem. Its running time is \( O(n^3/\varepsilon) \) steps.

**Proof:** Denote by \( \mathcal{B} \), the set of all possible batches that can appear in a restricted optimal solution. Let \( \Gamma \) be a batch in \( \mathcal{B} \). Assume that \( \Gamma \) contains jobs from family \( f \), and that it was scheduled at time \( t \). Also, assume that the previous batch of the same family, \( \Gamma' \), was scheduled at time \( t' \). In this case, using the same arguments as in the proof of Theorem 4.3, the set of the jobs in \( \Gamma \) is uniquely defined. Specifically, \( \Gamma \) can only contain jobs, \( J_j \), from the family \( f \) that can be scheduled \(( f_j = f, r_j < t, d_j \geq t + p_f) \), and have a release date that is larger than the scheduling time of the last batch \(( r_j > t') \). The weight of \( \Gamma \) is \( w_\Gamma = \sum_{J \in \Gamma} w_j \).

Let \( H \) be the list of all release dates and due dates ordered by increasing values. Now, for some \( h, H[h] < t < H[h+1] \). Similarly, for some \( h', H[h'] < t' < H[h'+1] \).

Given the choice of \( h \) and \( h' \), the jobs in \( \Gamma \) are uniquely determined. The scheduling time of a batch is chosen from the time window defined by the minimum due date and the maximum release date of the jobs in the batch. This bounds the number of batch instances in \( \mathcal{B} \) by \( O(n^2) \).

In all restricted optimal solutions, at most one batch from \( G(\bar{\Gamma}) \) is selected. Assume that two batches, \( \Gamma_1 \) and \( \Gamma_2 \) from \( G(\bar{\Gamma}) \) were scheduled in some restricted optimal solution at times \( t_1 \) and \( t_2 \) \(( t_1 < t_2) \), respectively. \( \Gamma_2 \) must share a job \( J_j \) with \( \bar{\Gamma} \). This job was available at time \( r_j \leq t_1 \leq t_2 \) and has a due date \( d_j \geq t_2 + p_f > t_1 + p_f \). This implies that \( J_j \) could have been scheduled in \( \Gamma_1 \), contradicting the fact that the solution obeys Proposition 3.7.

37
Thus, a feasible restricted solution for $(B, w)$ can have a weight of at most $2\alpha$, since at most one batch can be selected from both $G(\Gamma)$ and $N(\Gamma)$. On the other hand, since $S$ is maximal, it must contain one batch from $G(\Gamma) \cup N(\Gamma)$. Therefore, $w_2(S)$ is at least $\alpha$.

The algorithm can be implemented so that it outputs a $2(1+\epsilon)$ approximation in $O(B^2/\epsilon)$ steps, independent of the size of the time windows of the batches. Therefore, the running time of Local-Ratio is $O(n^4/\epsilon)$.

**Corollary 6.6** Local-Ratio is a $(2 + \epsilon)$-approximation algorithm for the $1|f - batch,b > n_f|\sum w_j \bar{U}_j$ problem. Its running time is $O(n^2/\epsilon)$.

**Proof:** When the batch size is unbounded and there are no release dates, a single batch is scheduled from each family, i.e., for all batches $t' = 0$. This reduces the number of batches in $B$ to $O(n)$ and the running time of Local-Ratio to $O(n^2/\epsilon)$.

We now use our technique for deriving $(2 + \epsilon)$-approximation algorithm for the $1|f - batch|\sum \bar{U}_j$ problem.

**Theorem 6.7** Local-Ratio is a $(2 + \epsilon)$ approximation algorithm for the $1|f - batch|\sum \bar{U}_j$ problem. Its running time is $O(n^4/\epsilon)$ steps.

**Proof:** As before, we measure the performance of the approximation algorithm against a restricted optimal algorithm. In other words, we consider only feasible solutions that conform to the properties of Propositions 3.1-3.2.

We show below that (i) the number of batch instances we need to generate, to allow any optimal (restricted) solution, is polynomial in $n$; (ii) in the Local Ratio decomposition, any maximal feasible solution is a 2-approximation, compared to restricted optimal solutions.

(i) Let $\Gamma$ be a batch in an optimal restricted schedule. Assume that $\Gamma$ contains jobs from family $f$ and that it was scheduled at time $t$. Also, assume that the previous batch of the same family, $\Gamma'$, was scheduled at time $t' < t$ and the maximum due date of any job in $\Gamma'$ is $d_{max}$. In this case, the batch is uniquely determined. Specifically, $\Gamma$ will contain the next (up to) $b$ available jobs in due date order from the family $f$. A job $J_j$ is available, if it can be scheduled ($f_j = f$, $d_j \geq t + p_j$), and have a due date which is larger than $d_{max}$. The weight of $\Gamma$ is $w_{\Gamma} = |\Gamma|$.

Let $d_1 \ldots d_D$ be the list of all due dates ordered by increasing values; then, for some $l$, $d_l < t < d_{l+1}$. Given the choice of $d_l$ and $d_{max}$, the jobs in the batch are uniquely determined. The scheduling time of a batch is chosen from the time window defined by the minimum due date of the jobs in the batch. This bounds the number of batch instances in $B$ by $O(n^2)$.
(ii) Recall that $\tilde{\Gamma}$ is the batch with minimal completion time. We show that in any restricted optimal solution, at most one batch from $G(\tilde{\Gamma})$ is selected. This would imply that at most two batches can be selected from $G(\tilde{\Gamma}) \cup N(\tilde{\Gamma})$; on the other hand, any maximal solution must contain one batch from $G(\tilde{\Gamma}) \cup N(\tilde{\Gamma})$.

Assume that two batches, $\Gamma_1$ and $\Gamma_2$ from $G(\Gamma)$ were scheduled in some restricted optimal solution at times $t_1$ and $t_2$, $t_1 < t_2$ respectively; $\Gamma_2$ must share a job $J_{j}$ with $\tilde{\Gamma}$. For batch $\Gamma$, let $d_{\max}(\Gamma)$ ($d_{\min}(\Gamma)$) as the maximum (minimum) due date of a job in batch $\Gamma$. From the above discussion, $d_{\max}(\Gamma) \leq d_{\max}(\Gamma_1) < d_{\min}(\Gamma_2) \leq d_{j}$. On the other hand, since $J_{j}$ appears in $\tilde{\Gamma}$, $d_{j} \leq d_{\max}(\Gamma)$. A contradiction.

6.3 A Polynomial Time Approximation Scheme

We now develop a PTAS for $||F = c \log n|| \sum \bar{U}_{j}$ based on the guessing scheme proposed in [CK-00].

**Theorem 6.8** Let $c > 0$ be a fixed constant. There exists a PTAS for $||F \leq c \log n|| \sum \bar{U}_{j}$ with running time $O(n^{c(1 + \frac{1}{c}) + 1} \log^{2} n)$.

**Proof:** The following is a PTAS for our problem.

1. Guess the optimal value $0 \leq \mathcal{O} \leq n$ for the instance $\mathcal{I}$.

2. For $1 \leq f \leq F$, guess the number, $q_{f}$, of jobs from family $f$ that will appear in the optimal solution ($0 \leq q_{f} \leq \mathcal{O}$): choose a positive integer $0 \leq k_{f} \leq \frac{F}{\mathcal{O}}$, such that $q_{f} = k_{f} \frac{\mathcal{O}}{F}$. The error induced by the rounding is $F \frac{\mathcal{O}}{F} = \mathcal{O}$. Then,

$$\sum_{f=1}^{F} q_{f} = \mathcal{O} \Rightarrow \sum_{f=1}^{F} k_{f} \frac{\mathcal{O}}{F} = \mathcal{O} \Rightarrow \sum_{f=1}^{F} k_{f} = \frac{F}{\mathcal{O}}.$$

The number of possible vectors $(k_{1}, \ldots, k_{F})$, satisfying $\sum_{f=1}^{F} k_{f} = \frac{E}{\mathcal{O}}$, is bounded by

$$\left( \frac{\frac{E}{\mathcal{O}} + F - 1}{F - 1} \right) = \left( \frac{F \left( 1 + \frac{1}{\mathcal{O}} \right) - 1}{F - 1} \right) \leq 2^{F \left( 1 + \frac{1}{\mathcal{O}} \right)}.$$

3. From each family, $1 \leq f \leq F$, select the set, $Q_{f}$, of the $q_{f}$ jobs with largest due dates. By Proposition 3.2 if any subset of $q_{f}$ jobs from family $f$ can be scheduled, then $Q_{f}$ can also be scheduled.
4. Solve the f-batch feasibility problem using algorithm Family-EDD (see Theorem ??) in order to determine whether all the jobs in \( \bigcup_{f=1}^{F} Q_f \) can be scheduled on time. If such a schedule exists, then the weight of the schedule is at least \( \mathcal{O}(1 - \epsilon) \). If no such schedule exists for all combinations of \( k_f \), then there is no schedule with weight of \( \mathcal{O}(1 - \epsilon) \) or more.

5. Use binary search on \( \mathcal{O} \) to find the largest \( \mathcal{O} \), such that a schedule with value \( \mathcal{O}(1 - \epsilon) \) exists.

We now compute the running time of the scheme. For each value of \( 1 \leq \mathcal{O} \leq n \) in the binary search, we go over the possible vectors \((k_1, \ldots, k_F)\) and for each vector we solve the feasibility problem.

Hence, when \( F < c \log n \), we get that the overall running time is

\[
\mathcal{O}(\log n) \cdot \mathcal{O}(2^{F(1 + \frac{1}{\epsilon})}) \cdot \mathcal{O}(n \log n) = \mathcal{O}(2^{F(1 + 1/\epsilon)} n \log^2 n) = \mathcal{O}(n^{c(1+1/\epsilon)+1} \log^2 n)
\]
Chapter 7

Multiple Identical Machines

In this chapter we consider the f-batch problem on multiple identical batching machines. Since even the classical two machine feasibility problem \( P2||L_{\text{max}} > 0 \) is NP-hard, we can obtain polynomial time algorithms only for restricted instances of the multiple machine f-batch problem. We focus on instances where either the number of families is fixed or all jobs have the same (unit) length.

7.1 Fixed Number of Families

The dynamic programming algorithms in Chapter 4 can be extended to the multiple machine case.

**Theorem 7.1** The problem \( P_m|f - \text{batch}, r_j| \sum w_j \hat{U}_j \) can be solved optimally in \( O(n^{F+1})(F+m+1)+F+2 \) steps.

**Proof:** Let \( W(t, l_1, \ldots, l_m, \tau_1, \ldots, \tau_F, \delta_1, \ldots, \delta_F) \) be the maximal weight of a schedule starting at time \( t \), where machine \( m' \) completed (or will complete) its last batch at time \( l_{m'} \), \( 1 \leq m' \leq m \), the last batch of jobs from family \( f \) was scheduled at time \( \tau_f \), and \( \delta_f \) is the largest due date of any scheduled job from family \( f \), \( 1 \leq f \leq F \). The weight of the optimal schedule is \( W(0, 0, \ldots, 0, \ldots, 0, \ldots, 0) \). We can calculate \( W \) using dynamic programming.

As in the single machine case, given that we have scheduled batches up to time \( t \), we can proceed in one of the following ways:

1. Select a family to schedule, choose a subset of the jobs and place it (as a batch) on some available machine. Since all machines available for use at some time point \( t \) are
equivalent, we can choose one of these machines arbitrarily. The available machines are machines $m'$ with $l_{m'} \leq t$. The next possible scheduling point is the first time that some machine becomes idle.

2. Move to the next batch starting point, which is the time of the closest release date.

3. If neither of the above is possible, no more jobs can be scheduled.

Therefore,

$$W(t, l_1, \ldots, l_m, \tau_1, \ldots, \tau_F, \delta_1, \ldots, \delta_F) = \max(\alpha_i, \max_{f, \delta'_f} \beta_{f, \delta'_f})$$

where

$$\beta_{f, \delta'_f} = W(\text{nextavail}(f),$$

\begin{align*}
&l_1, \ldots, l_{m'-1}, t + p_f, l_{m'+1}, \ldots, l_m, \\
&\tau_1, \ldots, \tau_{f-1}, t, \tau_{f+1}, \ldots, \tau_F, \\
&\delta_1, \ldots, \delta_{f-1}, \delta'_f, \delta_{f+1}, \ldots, \delta_F) \\
&+ \text{maxweight}(f, t, \tau_f, \delta_f, \delta'_f)
\end{align*}

for some $1 \leq m' \leq m$, where $l_{m'} \leq t$

$$\text{nextavail}(f) = \min(l_1, \ldots, l_{m'-1}, t + p_f, l_{m'+1}, \ldots, l_m)$$

and $\alpha_i$ and $\text{maxweight}(f, t, \tau_f, \delta_f, \delta'_f)$ are defined in Equation 4.1 and Equation 4.2.

Finally, the number of points $W$ in which is calculated is bounded by

$$O\left((n^{F+1})(F+m+1)^2 \cdot n^F\right)$$

Finding the optimal batch for a specific scheduling point can be done in $O(n \cdot n \log n)$ steps, giving an overall running time of $O\left(n^{(F+1)(F+m+1)+F+2 \log n}\right)$.
\section{7.2 Unit Length Jobs}

We now show that the multiple machine problem with unit length jobs is equivalent to the single machine problem with unit length jobs.

\textbf{Theorem 7.2} The problems $1|p_f = 1, r_j|\sum w_j \bar{U}_j$ and $P|p_f = 1, r_j|\sum w_j \bar{U}_j$ are equivalent.

\textbf{Proof:} Let $\mathcal{I}_p$ be an instance of $P|p_f = 1, r_j|\sum w_j \bar{U}_j$ consisting of $n$ jobs with release dates, $r_j$, and due dates, $d_j$. Let $\mathcal{I}$ be an instance of $1|p_f = 1, r_j|\sum w_j \bar{U}_j$ where for each job $J_j$ in $\mathcal{I}_p$ we define a job $J_j$ of the same weight, with release date $r'_j = m \cdot r_j$ and due date $d'_j = m \cdot d_j$.

Let $\mathcal{S}_p$ be an optimal solution for $\mathcal{I}_p$ and $\mathcal{S}$ be an optimal solution for $\mathcal{I}$. We can convert $\mathcal{S}_p$ into a feasible solution for $\mathcal{I}$ of the same weight, by scheduling on the single machine as follows. Any batch $\Gamma$ that was scheduled in $\mathcal{S}_p$ on machine $m'$ at time $t'$ is scheduled at time $t = m \cdot t' + (m' - 1)$. We can convert $\mathcal{S}$ into a feasible solution for $\mathcal{I}_p$ of the same weight, by scheduling a batch $\Gamma'$ that was scheduled in $\mathcal{S}$ at time $t$ to run on machine $m' = t - m \cdot \lfloor \frac{t}{m} \rfloor + 1$ at time $t' = \lfloor \frac{t}{m} \rfloor$. The reverse direction is immediate.

\textbf{Corollary 7.3} The problems $1|p_f = p|\sum w_j \bar{U}_j$ and $P|p_f = p|\sum w_j \bar{U}_j$ are equivalent.

\textbf{Proof:} The proof follows directly from Theorem 7.2 and from Proposition 3.3.

\section{7.3 Local Ratio Algorithms}

The Local Ratio algorithms of Section 6.2 can be applied to the multiple machine case. Using the general paradigm of [BB+00], we modify the weight function as follows. Given the batch $\bar{\Gamma}$, which completes first, we define for any $\Gamma$

$$w_1(\Gamma) = \begin{cases} w(\bar{\Gamma}) & \Gamma \in G(\bar{\Gamma}) \\ \frac{1}{m} \cdot w(\bar{\Gamma}) & \Gamma \in N(\bar{\Gamma}) \setminus G(\bar{\Gamma}) \\ 0 & \text{otherwise} \end{cases}$$

Any feasible solution may contain only one batch from $G(\bar{\Gamma})$ and at most $m$ batches from $N(\bar{\Gamma}) \setminus G(\bar{\Gamma})$. On the other hand, any maximal solution not containing an instance from $G(\bar{\Gamma})$ must contain $m$ batches from $N(\bar{\Gamma}) \setminus G(\bar{\Gamma})$ (otherwise $\bar{\Gamma}$ could be added to the schedule). Thus we obtain the following theorem:

\textbf{Theorem 7.4} Local-Ratio is a $(2 + \epsilon)$ approximation algorithm for the $1|f – batch, r_j, b > \sum w_j \bar{U}_j$ and $1|f – batch|\sum \bar{U}_j$ problems.
Chapter 8

Summary and Open Problems

Contribution of this work In this work we studied the problem of batch scheduling of jobs from incompatible job families, with the objective of maximizing the weighted number on-time jobs. To the best of our knowledge, this is the first comprehensive study of this problem. We showed that the problem is polynomially solvable when the number of families is fixed, as well as for other important sub-classes of instances. We also proposed an efficient \((2 + \epsilon)\)-approximation algorithm for the general unbounded batch instance. This result matches the best known bound for classical real-time scheduling.

Interestingly, our results show that the EDD rule that proved to be efficient in classical real-time scheduling is also essential in obtaining optimal and close approximation algorithms for the f-batch problem.

A main contribution of this work is the extension of the Local Ratio technique: the basic scheme compares the Local Ratio solution to an \emph{optimal} solution. We use the existence of a structured optimal solution in two ways. We reduce the solution set that the Local Ratio algorithm needs to consider; thus, we get a polynomial running time for the algorithm. Also, we compare the performance of the local ratio algorithm to that of a \emph{restricted optimal} algorithm and use the special structure of the optimal solution to obtain the approximation ratio. Our method may be useful in other applications of the Local Ratio technique.

Open problems Several questions remain open for future work. The most puzzling question is whether \(1|f - batch| \sum \bar{U}_j\) is polynomially solvable. Despite the apparent structure of this problem, we were not able to find an optimal polynomial algorithm. Another question involves the hardness of the case where all jobs have the same length but arbitrary release dates, \(1|f - batch, r_j, p_f = p| \sum w_j U_j\). Both problems are polynomially solvable in the setting of real-time scheduling.

Finally, the existence of a PTAS for other (hard) subclasses of the \(f - batch\) problem, and in particular, for the classical real-time scheduling problem, is still unresolved.
Bibliography


45


46