浙江大学

博 士 学 位 论 文

论文题目 通讯网络中排序问题的
若干在线和高性能算法

作者姓名 叶 德 仕

指导教师 张 国 川 教授

克劳斯·扬森 教授

学科(专业) 运筹学与控制论

所在学院 理学院 数学系

提交日期 2005 年 1 月

浙江大学申请博士学位论文

通讯网络中排序问题的若干在线和高性能算法

姓　　名：叶 德 仕

学科专业：运筹学与控制论

指导教师：张 国 川　教 授

克 劳 斯 · 扬 森　教 授

浙江大学　数学系

2005 年 1 月

摘 要

本文主要研究来源于通讯网络的排序(scheduling)问题。我们首先研究的是并行工件（parallel jobs）的排序，每个并行工件可能需要一台或多台机器同时加工这个工件。我们考虑机器之间存在的各种网络拓扑结构对排序结果的影响。具体的网络拓扑结构可以是：完全图结构（PRAM），线（Line），网格（Mesh），超立方体（Hypercube）等等。若干机器可以同时加工一个工件仅当这些机器构成的生成子网络是连通的。也就是说，仅当机器间存在通讯路径时才能够共同工作。并行工件需要一台或多台机器来加工。这些机器必须满足工件所要求的数目和拓扑结构。除了机器之间具有网络拓扑结构之外，我们对工件的各种不同的在线到达模式进行分类和研究，如工件按先后序到达（on dependencies），工件按释放时间到达（over time），工件按列表逐个到达（over list）。我们的目标是对给定的工件集合进行排序以满足各种约束条件并使某种目标值最优化。常见的的目标函数有极小化最大完工时间（minimize the makespan）或者是最大化输出量（maximize the throughput)。

我们对这些排序问题提出了相应的在线算法，并采用国际上通用的算法评价标准竞争比（competitive ratio）来衡量、分析给出的算法。通过研究“坏的”实例，我们给出各种问题的下界（lower bound）。从各种模型的分析来看，不同的网络拓扑结构和工件的不同在线到达模式均对最后的排序结果产生重要的影响。本文对每个具体问题如何设计有效算法进行了深入的研究，从而达到对已有算法的重要改进和对新问题提出高性能的算法。同时我们还研究，事先获得工件的部分信息对算法的影响。对于并行工件的排序问题，我们还研究了几个离线模型，即工件的所有信息都已知。我们或者证明该问题是多项式可解的或者是对 NP 难问题给出多项式时间近似方案。

我们还研究一些经典排序问题，这里每个工件只需要一台机器来加工。我们首先考虑的是机器总加工时间可扩展的排序问题（scheduling with extendable working

time problem)。每台机器有一个初始设定的正规加工时间，但机器的总加工时间可以根据需要决定是否延长。每台机器的工作时间定义为正规加工时间和总加工时间(真正加工时间) 之间的较大值。我们的目标是安排所有的工件，使得所有机器工作时间的总和最小。我们分别对所有的机器都有相同的正规加工时间，和每个机器的正规加工时间可以不相同的情况进行深入的研究。

最后，我们讨论经典排序问题中，事先获得工件的部分信息的半在线模型。我们已知最后加工的工件的加工时间最长，并且当最后一个工件到达的时候，将被通知这就是最后一个工件。我们分别对小数目的平行机和同类机进行了探讨。

全文共分九章。第一章介绍本文研究的问题、模型及其应用背景，以及问题之间的分类，并简要介绍本文的工作。一些基本概念，基础知识和分析问题的工具也在本章中介绍。第九章总结本文详细的成果以及对未来研究方向的展望。其余七章分成两部分。第一部分由二到六章构成，主要研究并行工件的排序问题。第二部分由第七、第八章组成，探讨一些与经典排序相关的问题。具体说明如下：

第二章详细综述并行工件在线排序的历史文献以及相关问题的研究结果。

第三章研究的问题是：在线带先后序约束 (on dependencies) 的并行工件在二维网格上的排序问题。所有工件的加工时间为单位时间，每个并行工件需要一组在二维子网格上的机器同时加工。工件之间存在先后序约束。一个并行工件到达并可以开始加工当且仅当这个工件的所有前序工件都已经完工。但是，工件之间的这种序的关系事先不知道。这意味着，不同策略的安排将可能导致不同的工件到达顺序。我们研究的目标是极小化最大完工时间。我们设计了一个有效的在线算法，其竞争比不超过 5.25。 同时我们也证明了，任何在线算法的竞争比都不可能小于 3.859。这个结果改进了原有的下界 3.25 和上界 46/7。 另外，当并行工件可以旋转 90 度的情况。我们证明了这个问题的上界最多为 4.25， 同时也证明了，不存在在线算法其竞争比小于 3.535。

第四章讨论并行工件列表在线 (over list) 的排序问题。并行工件的到达顺序已事先安排好在一个列表中。但是，安排者并不知道这个已经安排好的顺序，也不知道到底有多少个工件。当且仅当当前工件已经安排好下一个工件才到达 (如果有的话) 。工件一旦安排好之后，就不允许改动。本章讨论的问题基于机器的网络结构是完全图结构，也就是可以忽略网络的拓扑结构，而只需要满足相应的机器数即可。这个问题的目标仍然是极小化最大完工时间。我们给出了一个竞争比为 7 的算法，极大的改进了原先的竞争比为 12 的算法。我们进一步研究三种特殊情况：工件按加工时间非升的顺序到达；工件按工件的尺寸（所需的机器数）非升的顺序到达；工件的最大加工时间事先已知。对第一种情况，我们对贪婪算法进行了分析并证明其上界最多为 2。对第二种情况，我们也对贪婪算法进行了分析并证明这个算法的竞争比在 2.75 与 2.5 之间。对最后一种情况，贪婪算法就不再有效，我们给出了一个算法其紧界为 4。最后，我们研究工件在加工当中允许中断，当每个工件到达的时候，我们要决定何时开始加工该工件，是否对这个工件中断以及每次中断的时刻，安排好之后也就不再允许修改。据我们所知，这是第一个研究可中断并行工件列表在线排序问题的工作。 我们给出了一个竞争比为 2－1/m 的在线算法。

第五章研究可延展的并行工件（malleable parallel job）排序问题。每个工件的加工时间是所使用机器数的函数。每个工件加工所需的机器数事先并不固定，但在安排的时候必须确定，而且一旦确定了，就不能再更改。机器之间的网络拓扑结构为二维网格。我们考虑并行工件在满足单调性（monotonic）的情况下的排序。这里单调性指的是一个工件的加工时间在机器数目减少的情况下，其加工时间不会变短，而它的功（指的是机器数乘加工时间）不会增加。我们的目标是极小化最大完工时间。对这个问题，当机器数目为充分大的时候我们给出一个渐近完全多项式方案(AFPTAS)。也就是，对于任意给定的一个小量 ε，我们给出的算法都能在多项式时间内保证算法所得的解不超过最优解的 $1+\varepsilon$ 倍（加上一个常数）。

第六章我们研究问题的目标是极大化输出量 (throughput)，即极大化完工工件的

个数。首先我们讨论的是在超立方体上的排序问题，每个并行工件均需一个子超立方体来同时加工。并行工件释放时间（release time）相同，但是有各自的交工期(due date)。容易证明这个问题是 NP 难的。因此，我们讨论了单位加工时间的情况。证明该问题是 P 问题。我们进一步讨论一个公开问题，即并行工件在两台机上的排序问题，工件有各自的释放时间，同时每个工件的加工时间为单位时间，其目标仍是极大化输出量。我们证明了该问题也是 P 问题。

第七章讨论机器加工时间可扩展的排序问题。给定 m 台机器以及每台机器的正规加工时间 $b_1, b_2, \cdots, b_m$。 机器的总加工时间可以根据需要决定是否延长。每台机器的工作时间定义为正规加工时间和总加工时间(即安排在该机器上的所有工件的加工时间之和) 之间的较大值。工件是按列表在线到达的。目标是安排所有的工件在给定的机器上，使得所有机器工作时间的总和最小。对于所有机器的正规加工时间相同情况，我们研究了机器数目较少的的情况，详细深入的分析已有算法并证明其算法具体的竞争比。对于两台机器，我们给出了竞争比为 7/6 的最优在线算法。对于三台机器，我们给出了问题的下界 $(17+\sqrt{577})/36$ 和上界 7/6。对于四台机器，我们给出了下界 9/8，并证明其上界不超过 19/16。接着，我们对机器的正规加工时间不同的情况进行了研究。假设最小的正规加工时间不小于最大的工件加工时间。对任意的 m 台机器，我们对贪婪算法进行详细的分析，并完全刻划了其紧界与机器数 m 之间的关系。然后，我们对两和三台机器的情况，给出了改进的算法以及问题的下界。在没有上述假设的情况下，我们对两台机器进行了研究。最后，我们对加工工件是并行工件而且是离线的情况进行了研究，提出了一个算法并证明其紧界是 2。

第八章我们研究经典排序问题中的半在线模型。工件是按列表在线。同时我们已知最后加工的工件的加工时间最长，并且当最后一个工件到达的时候，将被通知这就是最后一个工件。我们分别对小数目的同型机和同类机进行了探讨。目标是极小化最大完工时间。对两台和三台同型机，我们给出了最优的在线算法，其竞争比分别为 $\sqrt{2}$ 和 3/2。接着，我们研究了两台同类机问题，对每个工件一台机

器的加工速度为 1，另一台的加工速度为 1/s，我们分别就不同的速率 s 给出竞争比分析，并证明了对于大部分 s，所给的算法都是最优的列表半在线算法。最后，我们对这个问题的另外一个目标函数：极大化最小完工时间进行了讨论。证明了绝大多数情况下，对于给定的 s，我们所给的算法是最优的。

关键字： 排序，并行工件，网络结构，在线算法，竞争比。

On-Line and Efficient Algorithms for Scheduling Problems in Communication Networks

Deshi Ye

A Dissertation

Presented to the Faculty of the Graduate School

of Zhejiang University

in Partial Fulfillment of the Requirement for the Degree of
Doctor of Philosophy

Supervisors: Prof. Dr. Guochuan Zhang

and Prof. Dr. Klaus Jansen

Hangzhou, January 2005

On-Line and Efficient Algorithms for Scheduling Problems in Communication Networks

Deshi Ye

Submitted for the degree of Doctor of Philosophy
January 2005

Abstract

In this thesis, we deal with a number of scheduling problems which arise in communication networks. This thesis consists of two parts. In the first part, we focus on parallel jobs scheduling, where a job may require more than one machine simultaneously. The network topologies between machines are considered how to influence the schedule. The concrete network topologies could be PRAM, Line, Mesh, Hypercube and so on. A subset of machines can work together only if they are connected. Parallel jobs require some part of the machines (e.g. a mesh of a smaller size), and can be processed simultaneously on any subset of machines with that specific network topology. Different variants on on-line scheduling are explored, such as the model where jobs arrive on dependencies and the model where jobs arrive over list. Our goal is to schedule a given set of jobs so that all the constraints are satisfied and a certain objective is optimized, such as makespan minimization or throughput maximization.

We present on-line algorithms for these scheduling problems and adopt the standard performance measure - competitive ratio to analyze them. We also provide lower bounds with some specific instances. It is showed that the complexity of the network topologies and various variants of on-line models impose a big influence on the schedule. On the other hand, we show that it is quite helpful if we know some partial information of the jobs in advance in order to improve the quality of the schedule. We also design efficient algorithms for some off-line problems, where we know the full information on jobs beforehand, such as the malleable parallel jobs

scheduling problem where the processing time of a malleable job is a function of the set of machines allotted to it, and the problem to maximize the throughput of parallel jobs with release dates.

In the second part, we study some classical scheduling models in which jobs require exactly one machine. In the problem with extendable working time, each machine has a regular working time which can be extended if necessary. The (final) working time of a machine is defined to be the larger value between the original regular working time and the total processing time of the jobs assigned to it. The goal is to assign all the jobs to the machines such that the total working time of the machines is minimized. Both cases of the identical machines and of the machines with unequal regular working times are explored.

Finally, we study the classical on-line scheduling model in which jobs require only one machine and some partial information of jobs is known in advance. Jobs arrive over list, however, it is known that the last job is the job with longest processing time and the on-line scheduler will be informed upon arrival of the last job. Our goal is to minimize the makespan or maximize the minimum machine completion time. In addition to identical machines, uniform machines where machines have different speeds are also considered.

Key words: Scheduling, Multiprocessor tasks, Network topology, On-line algorithm, Competitive ratio.

Acknowledgements

The research work of this thesis was carried out at Department of Mathematics of Zhejiang University from September 2001 until March 2005, and at Department of Computer Science and Applied Mathematics, University of Kiel, from October 2003 until September 2004. It was supervised by Prof. Dr. Guochuan Zhang of Zhejiang University and Prof. Dr. Klaus Jansen of Universität Kiel.

This thesis would have not been written without the help and support of many people.

First and foremost, I am grateful to my supervisor Guochuan Zhang for his insights and unconventional ideas in research, for guiding me to study on-line scheduling problems and bin packing problems, the design and analysis of algorithms, for outstanding and pleasure collaborating. He also helped me in my career and everything else, such as to improve my English. I express also millions of gratefulness to Klaus Jansen for many valuable discussions, suggestions and collaborations in my work, for providing me a wonderful research environment during my stay in Kiel.

I am grateful to my coauthors, Janka Chlebíková, Leah Epstein, Hu Zhang for many valuable discussions and nice cooperation.

I am grateful to the staffs, Jihuang Ding, Ling Gai, Yongqiang Shi from the research group Algorithmic Theory and Application at Zhejiang University for their help, friendliness and valuable discussions.

I would like to express my thankfulness to Prof. Enyu Yao and to Guanting Chen, Shiping Chen,Yong He, Qingxue Huang, Hao Shen, Zhiyi Tan, Qifang Yang from the research group Operations Research at Zhejiang University for the help and assistance they have offered me in the past, and also to the other staffs in this group, I can not list them all here.

I would also like to thank the staffs, Susanne Burfeind, Florian Diedrich, Aleksei Fishkin, Olga Gerber, Ute Iaquinto, Gitta Marchand, Parvaneh Karimi Massouleh, Ralf Thöle from the research group Theory of Parallelism at the University of Kiel for their help and friendliness.

I much appreciate Prof. Denis Trystram for his kind help and valuable discussion during my visit in Grenoble. Thanks also to Pierre-Francois Dutot, Grégory Mounié for their kind help during my visit.

I would also like to thank the many other friends who made my life more pleasant and enjoyable, Jianli Chen, Xufeng Chen, Yi Chen, Jing Fan, Lin Fang, Jiangtao Hu, Jiwei Huang, Min Ji, Meizhen Jiang, Lu Lu, Yinxuan Luo, Wei Meng, Xiangfang Men, Yifeng Wu, Ningyi Xu, Jing Ye, Chengxiang Yuan, Chunyu Zhang, Min Zhuo.

I am grateful to my relatives for their support, help and love.

I would especially like to thank my parents and my siblings for everything they have done for me and never-ending love.

The research work was supported in part by a DAAD program (Sandwich type), National 973 Fundamental Research Project of China, NSFC(19801032), NSFC(10231060), EU-Project APPOL II and DAAD-Project Procope.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This thesis is a theoretical study of scheduling problems in communication networks. We study on-line scheduling of parallel jobs. Various factors, such as the network topology, the manner that jobs arrive, the restriction on processing times, the presence of dependencies between jobs, permission of preemption, partial information on future jobs, are considered how to affect the length of schedules. Then we study some extended classical on-line scheduling problems, which also arise in some communication networks (e.g. TDMA).

A scheduling problem is specified by a resource requirement (e.g. network topology), a set of jobs and a certain objective. *A scheduling algorithm* is to build a schedule in which the jobs are assigned to one or more machines with a specific communication configuration according to the variant of scheduling so that a certain objective is optimized.

As we know, a large number of scheduling algorithms preclude the possibility to develop acceptable fast (polynomial time) algorithms that produce optimal solutions. For the alternatives, it is advantageous to develop the fast approximation algorithms, which do not produce optimal solutions, but will produce solutions that are provably close to optimum.

One approach to design and analysis of these algorithms always assumes that an algorithm accesses the complete knowledge of the whole input. However, many *on-line algorithms* arise in the realistic scenario, in which we have to make a schedule without knowing the complete information an the input. In practice, the input of

some problems are only partially available since some relevant input data arrive in the future or are not accessible at present. Then an on-line algorithm is required to react to the new request with only partial information and must generate an output without knowledge of the entire input.

In this thesis we mainly concentrate on the development of fast approximation algorithms for those problems that are computationally intractable and the study of on-line algorithms for those optimization problems with incomplete information.

1.1 Preliminaries and notations

The scheduling theory is one of the main areas of practical and theoretical computer science. For a long time the classical scheduling theory assumed that a *job* (or *task*) can be executed only by one *machine* (or *processor*) at a time, see [27,118,110,3] for excellent surveys. However this assumption is too restrictive in the case of parallel computer system and modern production systems where jobs (e.g. programs) can be processed on several processors in parallel. Therefore, the model of parallel jobs has gained considerable attention recently [45,49,57,58,59,60,125,117,104]. Parallel jobs may require more than one machine simultaneously.

The terms *parallel job* and *machine* can be used interchangeably with terms *multiprocessor task* and *processor*, respectively.

1.1.1 Network topologies

In many applications, a network topology is considered, in which jobs can only be executed on particular machines [93,112,15,16,61,62,117,35,98]. Only those machines connected may execute a job together simultaneously. Parallel machines with a specific network topology can be viewed as a graph where each node represents a machine and each edge represents the communication link between the two nodes. The concrete topology of networks can be *PRAMs*, *Lines*, *Hypercubes*, *Meshes*, and so on. For any given set $M = \{M_1, M_2, \ldots, M_m\}$ of machines, similar as in [62], some definitions of network topologies are stated as follows.

1. **PRAM:** A parallel system whose underlying network topology is a complete graph of m nodes or m-clique. Therefore, each subset of machines is also a PRAM. So there is no preference in scheduling of certain combinations of machines (see Fig 1.1 (a)).

2. **Line:** A line (one-dimensional mesh) is a parallel system consisting of m machines where machine M_i is directly connected with processors M_{i+1} and M_{i-1} if they exist. A parallel job with *size* s_i (the number of requested machines) can be processed on s_i consecutive machines under this network (see Fig 1.1 (b)).

3. **Hypercube:** An N-dimensional hypercube consists of $m = 2^N$ machines where two machines are directly connected if and only if their binary representations differ by exactly one bit. Available job types include any d-dimensional subcube for $d \leq N$ (see Fig 1.2 (a)).

4. **Mesh:** A two-dimensional mesh $n_1 * n_2$ is a parallel system consisting of $m = n_1 * n_2$ machines $\{M_{i,j} | 0 \leq i \leq n_1, 0 \leq j \leq n_2\}$, where machine $M_{i,j}$ is directly connected to machines $M_{i,j\pm1}$, $M_{i\pm1,j}$ if they exist. Available job types include all $a_i * b_i$ submeshes, where $a_i \leq n_1, b_i \leq n_2$. Higher dimensional meshes and jobs can be defined analogously (see Fig 1.2 (b)).

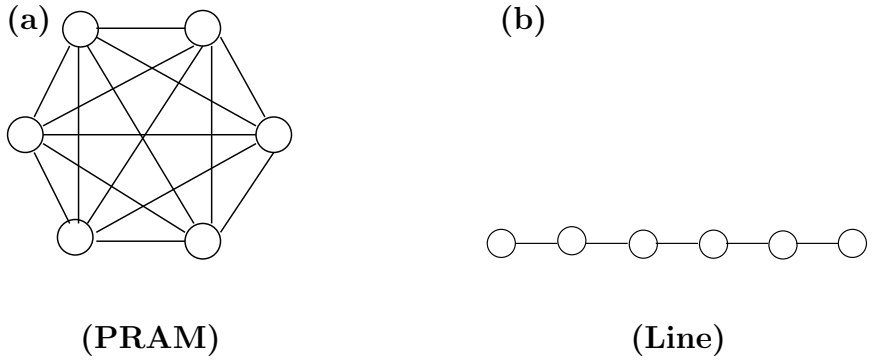

Figure 1.1: The PRAM and line topology

Other than the above parallel models, if each job requires the simultaneous use of a prespecified (fixed) set of machines, we get the *dedicated* variant.

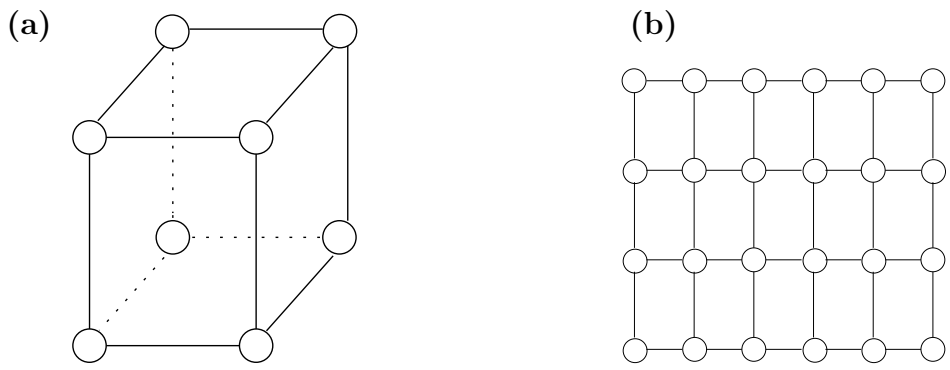

Figure 1.2: The 3-dimensional hypercube and 2-dimensional mesh topology

1.1.2 General scheduling models

The *number of machines* is always denoted by m, and n stands for the number of jobs. In general, the machine scheduling problems that we consider can be described as follows. There is a set $M = \{1, 2, \ldots, m\}$ of m machines that are used to execute a set $J = \{J_1, J_2, \ldots, J_n\}$ of n parallel jobs. Each job $J_i \in J$ has a machine requirement Δ_i (we also call it the *size* of a job), a processing time $p_i(\Delta_i)$, a release date r_i, a due date d_i and a weight w_i. In addition, there might be precedence constraints on the jobs, i.e. some special order in which jobs have to be started (they can be expressed in term of chains, trees, series parallel orders and so on).

A schedule is *feasible* if each machine executes at most one job at a time, and each job $J_j \in J$ does not start earlier than its release date r_j and complete before its due date d_j and it is executed on a sub-system of appropriate type with given size and subject to the precedence constraints.

Preemption may be permissible in some schedule, the processing of any job can be interrupted and resumed at a later time on the same or on different machines.

For the parallel system with PRAM network topology, we deal with the following basic variants of machine requirement Δ:

- **Non-malleable model** $size_j$: each job $J_j \in J$ requires a fixed number of machines for its processing. It is not possible to run a job on fewer machines and it will not decrease the processing time with more machines.

- **Malleable model** $fctn_j$: for each job $J_j \in J$ the processing time p_j is a function of the number of processors assigned to J_j.

1.1.3 Classification of on-line fashions

The classification of the on-line scheduling problem dependents on which part of the problem is given on-line. We distinguish on-line scheduling problems by the manner that jobs arrive. In general, there are three different versions.

1. **Jobs arrive over list**. Jobs are ordered in some list and arrive one by one according to the list. The next job appears only after the current one has been scheduled. We are allowed to assign the jobs to arbitrary time slots, thus a job can start processing later than the successive jobs in the list. However, once we see the successive jobs we can not change the assignment of the previous jobs.

2. **Jobs arrive over time**. Each job J_i has a release date r_i, which is not known before J_i appears.

3. **Jobs arrive on dependencies**. Each job appears only after all its predecessors are completed, i.e., there is some special order in which jobs have to be started (they can be expressed in term of chains, trees, series parallel orders, interval order and so on). As soon as a job appears its size is known. However, we are not aware of the dependencies among jobs in advance.

For each of the above three versions, upon arrival of a job, we may also assume that the job processing time becomes known (called *known processing time* or *clairvoyant*) or unknown until they are completed (called *unknown processing times* or *non-clairvoyant*).

1.1.4 Objective functions

Given a schedule σ, we compute for task $J_j \in J$ the following values: the *completion time* $C_j(\sigma)$; the *flow time* $F_j(\sigma) = C_j(\sigma) - r_j$; the *throughput* $\bar{U}_i(\sigma) = 1$ if $C_j(\sigma) \leq d_i$, and $\bar{U}_i(\sigma) = 0$ otherwise; the *unit penalty* $U_j(\sigma) = 1$ if $C_j(\sigma) > d_j$, and $U_j(\sigma) = 0$

otherwise; the *lateness* $L_j(\sigma) = C_j(\sigma) - d_j$; the *earliness* $E_j(\sigma) = \max\{d_j - C_j(\sigma), 0\}$; the *tardiness* $T_j(\sigma) = \max\{C_j(\sigma) - d_j, 0\}$. If there is no ambiguity about the schedule under consideration, we write $C_j, F_j, \bar{U}_j, U_j, L_j, E_j, T_j$ respectively. Thus some commonly used optimality criteria involve the minimization of: maximum completion time, or makespan $C_{\max} = \max_j C_j$; the *total (weighted) completion time* $\sum_j (w_j)C_j$; the *total (weighted)* flow time (also called response time) $\sum_j (w_j)F_j$; the *(weighted) number of late jobs* $\sum_j (w_j)U_j$; the *maximum lateness* $L_{\max} = \max_j L_j$; the *maximum earliness* $E_{\max} = \max_j E_j$; the *total (weighted) tardiness* $\sum_j (w_j)T_j$ The optimality criteria involve of the maximization of: *throughput* $\sum \bar{U}_i$.

The most common objective function is *makespan*. If the makespan is small, the utilization of machines is high. Then it benefits the owner of machines when the profit are proportional to the work done. Nowadays, the objective of *throughput* becomes popular to meet various quality-of-service (QoS) demands from the user, e.g. offer best-effort service in a network. The objective functions of total completion time and total flow time arise also in the QoS, since users may concentrate on the time it takes to finish individual jobs. Thus it is unacceptable if many short jobs are postponed after some long job even if the makespan is optimal.

1.1.5 Three-field notations

For simplicity of denotations of the analyzed scheduling problems, we adopt the standard three-field notation $\alpha|\beta|\gamma$ proposed by Graham et al. in [74]. It has been enhanced by several people, e.g. Blazewicz, Veltman, Drozdowski [21,126,45].

The scheme $\alpha|\beta|\gamma$ in its three fields describes the machine system (α), the job system (β) and the optimality criterion (γ).

The first field contains up to two symbols $\alpha = \alpha_1, \alpha_2$. The detail describe of notation α_1 and α_2 are illustrated in Table 1.1. The possible machine environment in the α field is as follows:

- Single machine (1): There is only one machine in the system.
- Parallel and identical machine (Pm): There are m identical machines in parallel.

- Uniform machines (Qm): There are m machines in parallel, but the machines have different speeds.
- Unrelated machines (Rm): There are m machines in parallel, but each machine can process different jobs at a different speed.
- Open shop (Om): Each job needs to be processed exactly once on each of the machines. But the order of processing can be arbitrary.
- Flow shop (Fm): The machines are linearly ordered and the jobs all follow the same route (from the first machine to the last machine).
- Job shop (Jm): Each job has its own predetermined route to follow. It may visit some machines more than once and it may not visit some machine at all.

If m is omitted, then the number of machines is not fixed in the definition of the problem, i.e., the number of machines will be specified as a parameter in the input.

Table 1.1: The machine environment in $\alpha|\beta|\gamma$ scheme

α	*Values*	*Meaning*
α_1	1	single machine
	P	identical machines
	Q	uniform machines
	R	unrelated machines
	O	open shop system
	F	flow shop system
	J	job shop system
α_2	$\emptyset$	the number of machines is not fixed in the definition of the problem
	m	the number of machines fixed to m

The second field $\beta = \beta_1, \ldots, \beta_7$ defines the job system. Table 1.2 show the characteristics of jobs. It partly concentrates on the on-line or off-line, the resource requirement of machines, precedence constraints, the admissibility of preemptions, release dates, due dates and bounded processing times.

Table 1.2: The job characteristics in $\alpha|\beta|\gamma$ scheme

β	*Values*	*Meaning*
β_1	$\emptyset$	off-line, the complete information of jobs are known.
	on-line-list	jobs arrive over list
	on-line-time	jobs arrive over time
	on-line-prec	jobs arrive on dependencies
β_2	$\emptyset$	each job requires exactly one machine
	$size_j$	PRAM model, non-malleable version
	$fctn_j$	PRAM model, malleable version
	$cube_j$	Hypercube model
	$line_j$	Line model
	2-d mesh	2-dimensional mesh model
	fix_j	dedicated model
β_3	$\emptyset$	independent jobs
	prec	arbitrary precedence constraints
β_4	$\emptyset$	no preemption allowed
	pmtn	preemptions are allowed
β_5	$\emptyset$	all jobs are released at time zero
	r_j	release time may be specified for each job
β_6	$\emptyset$	no due dates between jobs
	$d_j = D$	common due dates
	d_j	arbitrary due dates
β_7	$\emptyset$	the processing times are known but arbitrary.
	non-clairvoyant	unknown processing time model
	$p_j = 1$	all the processing times of jobs are unit
	$p_j = p$	constant processing times for all jobs
	$\underline{p} \leq p_j \leq \bar{p}$	bounded processing times

The third field $\gamma = \gamma_1$, where $\gamma_1 \in \{C_{\max}, L_{\max}, E_{max}, \sum (w_j)T_j, \sum (w_j)F_j, \sum (w_j)C_j, \sum (w_j)U_j, \sum \bar{U}_j\}$ denotes the optimality criterion, see Table 1.3.

Table 1.3: The objective function in $\alpha|\beta|\gamma$ scheme

γ	*Definition*	*Quantity name*
$C_{\max}$	$\max_j C_j$	makespan (maximum completion time)
$L_{\max}$	$\max_j L_j$	maximum lateness
$E_{\max}$	$\max_j E_j$	maximum earliness
$\sum (w_j)T_j$	$\sum_j (w_j)T_j$	total (weighted) tardiness
$\sum (w_j)F_j$	$\sum_j (w_j)F_j$	total (weighted) flow time
$\sum (w_j)C_j$	$\sum_j (w_j)C_j$	total (weighted) completion time
$\sum (w_j)U_j$	$\sum_j (w_j)U_j$	(weighted) number of late jobs
$\sum (w_j)\bar{U}_j$	$\sum_j (w_j)\bar{U}_j$	(weighted) throughput, (weighted) number of early jobs

1.1.6 Performance measure

We adopt the standard measure *competitive ratio*, introduced by Sleator and Tarjan [121], to evaluate an on-line algorithm. For any instance I, let $C_A(I)$ and $C^*(I)$ be the objective value (e.g. makespan) given by an on-line algorithm A and the objective value (e.g. makespan) produced by an optimal off-line algorithm, respectively. The *competitive ratio* of algorithm A is defined as

$$R_A = \sup_I \{C_A(I)/C^*(I)\}.$$

Algorithm A is called ρ-competitive if $C_A(I) \leq \rho C^*(I)$ holds for any instance I. A lower bound R of a problem means that any on-line algorithm has competitive ratio of at least R.

Asymptotic competitive ratio of an on-line algorithm A is the least α satisfying the inequality

$$C_A(I) \leq \alpha C^*(I) + \beta,$$

for any instance I, where β is a constant. Note that when $\beta \leq 0$, the asymptotic competitive ratio becomes the competitive ratio.

The above definitions based on the objective is to minimize a certain cost. For the problem with objective to maximize a certain value, the competitive ratio of algorithm A is defined as

$$R_A = \sup_I \{C^*(I)/C_A(I)\}.$$

Algorithm A is called ρ-competitive if $C^*(I) \leq \rho C_A(I)$ holds for any instance I. The asymptotic competitive ratio is the least α satisfying the inequality

$$C^*(I) \leq \alpha C_A(I) + \beta,$$

for any instance I, where β is a constant.

In the off-line case, instead of *competitive ratio* we use *worst-case ratio* to evaluate an approximation algorithm. The definition is similar.

The goal in any problem is to find an algorithm with a competitive ratio as small as possible. But how well any on-line algorithm can perform for a given problem? This question can be addressed by examining a general *lower bound* on the on-line problem. A lower bound is usually derived by providing a set of specific instances on which no on-line algorithms can perform better than it. If an on-line algorithm has a competitive ratio matching the lower bound, it is optimal (best possible). For simplicity, in the following instead of $C^*(I)$ and $C_A(I)$, we usually use C^* and C_A to denote the corresponding objective values (e.g. makespans) without causing any confusion.

1.2 Problems considered in this thesis

In this section, we expose the problems considered in this thesis. The problems related to parallel jobs scheduling are given in Subsection 1.2.1. Some extended classical scheduling problems, where each job requires exactly one machine, are presented in Subsection 1.2.2. Scheduling parallel jobs on dedicated machines is beyond the scope of this thesis. We recommend the surveys [50,46] for the malleable jobs scheduling and the survey [45] for parallel jobs on dedicated machines. Throughout this thesis, we assume that, at any time, a machine can handle at most one job.

1.2.1 Scheduling of parallel jobs

We study an on-line problem of scheduling UET (Unit Execution Time) parallel jobs on 2-dimensional meshes, denoted by $P|$ *on-line-prec*, *2-d mesh*, $p_j = 1|C_{\max}$. Parallel jobs arrive dynamically according to the dependencies between them, which are unknown before the jobs appear. Each job is requested to be scheduled on a submesh of the machines which are located on a 2-dimensional submesh, i.e., a job must be scheduled on a rectangle of given dimension. The objective is to minimize makespan. We deal with an UET job system, in which all job processing times are equal. We show a lower bound of 3.859 and present a 5.25-competitive algorithm. It significantly improves a previous lower bound 3.25 and a previous upper bound of 46/7. We consider also the rotated 2-dimensional mesh, in which the rotations of all the parallel jobs are feasible. A lower bound 3.535 is proved and an on-line algorithm with competitive ratio of at most 4.25 is derived.

We address on-line scheduling of parallel jobs, where jobs arrive over list, i.e. the problem $P|$ *on-line-list*, *size*$_j|C_{\max}$. A parallel job may require a number of machines for its processing at the same time. Upon arrival of a job, its processing time and the number of requested machines become known, and it must be scheduled immediately without any knowledge of future jobs. We present a 7-competitive on-line algorithm, which improves the previous upper bound of 12. Furthermore, we investigate three special cases. For the case that jobs arrive in non-increasing order of processing times, the upper bound of algorithm greedy is at most 2. For the case that jobs arrive in non-increasing order of sizes, the competitive ratio of the algorithm greedy is in between 2.5 and 2.75. For the case that the longest processing time of jobs is known in advance, an on-line algorithm with tight bound 4 is presented. Finally, we introduce preemption to our problem, i.e. the problem $P|$ *on-line-list*, *size*$_j$, *pmtn*$|C_{\max}$. Upon arrival of a job, we must make the decision of preemption immediately and we can not change it later. An on-line algorithm is shown to have a competitive ratio of $2 - 1/m$. As far as we know it is the very first result in this topic.

Furthermore, we consider efficient algorithms for the off-line scheduling problems, where all the characters of jobs are known. We study the malleable parallel jobs

scheduling problem, in which, the job processing time is a function of the number of machines allotted to the job. Note that the number of machines to execute the parallel job is not fixed but must be determined before execution, and it cannot be changed until the completion of the job. Unlike the definition in this thesis, some authors use "moldable" instead of "malleable", see, e.g., [55,50,46]. The malleable parallel jobs are called *monotonic*, if $p_j(l)l \leq p_j(l')l'$ and $p_j(l) \geq p_j(l')$ holds for all job J_j and pair $l \leq l'$, where $p_j(l)$ is a function $p_j : l \to \mathbb{R}^+$. We study the monotonic variant malleable parallel jobs on a 2-dimensional mesh and present an asymptotic fully polynomial time approximation scheme when the number m of machines is sufficiently large. Equivalently, for any given $\varepsilon > 0$, we compute a schedule of length at most $(1+\varepsilon)$ times the optimum (maybe plus an additive term), which has running time polynomially in n and $1/\varepsilon$.

Concerning the objective to maximize the throughput, we study the problem of scheduling n independent parallel jobs on hypercubes. A parallel job is required to be scheduled on a subcube. Each job is associated with a due date and available at time zero. It is easy to show the NP-hardness of this problem. We provide an optimal algorithm with polynomial running time for the unit processing time job system. Then we turn to the problem where each job has a release time. We propose an optimal polynomial algorithm for the unit processing time job system and the dimension of hypercube is limited to 1, i.e. the two parallel identical machines case.

1.2.2 Extended classical scheduling problems

We study the scheduling problem on parallel machines with extendable working times: there are m machines $B_1, B_2, \ldots, B_m$ with original regular working time $b_1, b_2, \ldots, b_m$, respectively. The original regular working time can be extended if needed. The (final) working time of a machine is defined to be the larger value between the original regular working time and the total processing time of the jobs assigned to it. Jobs $\{J_1, J_2, \ldots, J_n\}$ arrive over list, which are not known in advance. When job J_i arrives it must immediately and irrevocably be assigned to one of the machines and the next job J_{i+1} becomes known only after J_i has been assigned. The processing time of job J_i is p_i. The goal is to assign all the jobs to the machines

such that the total working time of the machines is minimized.

The regular working times of the machines can be seen as a given capacity, thus this problem is a variant of bin packing: the machines correspond to bins and the jobs to items.

We first consider the problem on a small number of identical machines, i.e. all the regular working times are equal to 1. The on-line algorithm H_x, designed by Speranza and Tuza [122], is carefully analyzed for $m = 2, 3, 4$. The best choices of parameter x is determined and the competitive ratios are shown. H_x is a best possible on-line algorithm for two machines. A new algorithm A_α is derived, which has a competitive ratio of $7/6$ and is better than H_x for three machines. Unfortunately, algorithm A_α does not work for $m > 3$. For $m = 3$ and 4, we present a lower bound $(17 + \sqrt{577})/36$ and $9/8$, respectively. Then we deal with the general problem with unequal regular working times. The overall competitive ratio (see Section 7.1) are introduced.

With the assumption (7.1) that any job processing time is not greater than any machine regular working time (7.1), we analyze algorithm LS (List Scheduling) more carefully and prove the competitive ratios for each m and each collection $\mathcal{B}$ of regular working times. The ratios differ for an even number of machines and an odd number of machines. It is also shown that the competitive ratio of LS for the identical machines is exactly $5/4$ when m is even and $5/4 - 1/(4m^2)$ when m is odd. We then present an improved on-line algorithm for $m = 2$ and $m = 3$. We also discuss the problem without the assumption (7.1). A lower bound $6/5$ for the overall competitive ratio is presented. We prove the competitive ratio of LS for the two-machine case and give an improved on-line algorithm. Finally, we extend our results to the off-line version for parallel jobs, where a job may require more than one machine simultaneously. An upper bound of 2 is presented.

Finally, we consider a variant of on-line classical scheduling, where partial information on future jobs is known beforehand. Assume that the last job is the longest (with the longest processing time). At the moment the last job appears, the adversary will inform that this is the last one. The objective is to minimize the makespan. We provide best possible on-line algorithms with competitive ratios $\sqrt{2}$ and $3/2$, re-

spectively, for $m = 2$ and 3, where m is the number of machines. Then we consider the uniform machines, where machines have different speeds. We present algorithms relying on the speed ratio s. Most of them are best possible. We also explore the problem to maximize the minimum completion time on two uniform machines. The competitive ratios depend also on the speed s. The proposed algorithms are best possible for almost all the value of s.

1.3 Practical motivations

The applications of parallel jobs scheduling arise widely in various realistic scenarios. Many applications of the motivation behind parallel jobs model were given by Drozdowski [45,46]. We present some applications mainly based on [45,46] as follows.

Communication An example of application of parallel jobs scheduling problem arise in wavelength-division-multiplexing (**WDM**) network [127,124]. Speaking the terminology of WDM, processor correspond to *wavelength* (channel), task correspond to *data packet* which have to be transmitted through, and processing time correspond to *transmission time*. Each packet is divided into many parts and all these parts should be transmitted simultaneously. Optical networks employing WDM are now a viable technology for implementing a next-generation network infrastructure that will support a diverse set of existing, emerging, and future applications [69].

Parallel jobs appear in lots of the dynamic bandwidth allocation problem [14,41,99]. The bandwidth allocation problem arises in communication media that should have guaranteed bandwidth policy (e.g., **ATM**(Asynchronous Transfer Mode) [48]). For example, delivering compressed video, require certain level of quality of service (QoS) in the form of guaranteed bandwidth. The applications submit requests for communication channel bandwidth. The scheduling algorithm has to determine the assignment of bandwidth to fulfill the submitted requirements. Here, bandwidth can be regarded as processors. Thus, bandwidth allocation problem can be seen as the assignment of application requests in the bandwidth $\times$ time space. Furthermore, If the communication allows preemption and change the bandwidth assignment, then

it can be profitable both for the utilization of the channel [14] and for the number of transferred messages [41].

Another application for parallel jobs can be scheduling file transfers [32]. A file transfer requires at least two processing terminals, sender and receiver, simultaneously. Simultaneously transfers on multiple buses can also be considered as multiprocessor tasks [82].

Parallel applications Parallel computer applications are based on parallel jobs. A number of massively parallel computer systems (MPP) divide their processors into partitions [22]. The main idea of a partition is to give an exclusive access to a number of processors to one application only. The operation system is responsible for managing the partitions, granting the access to them etc. Note that from the viewpoint of partition manager the applications are multiprocessor task (parallel jobs) because they occupy all the processors in the partition at the same moment of time. Parallel processing is expected to bring a breakthrough in the increase of computing speed, which arises from the applications such as weather forecasting, real time image processing, molecular modelling (computing the structures and properties of chemical compounds, biological macromolecules, polymers, and crystals), physical simulations (such as simulation of airplanes in wind tunnels) etc. The real systems are, e.g., IBM RS/6000 Scalable Parallel Computer, Sun Enterprise 10000 or HPC Cluster, and so on. There also exists commercially parallel computers which use a hypercube network topology. Ametek, Floating Point Systems, Intel Scientific Computers, NCUBE and Thinking Machines are some of the vendors of hypercube and modified hypercube computers. Feitelson [56] proposed a nice review about the studies on the workload of massively parallel processing computer systems.

The parallel jobs scheduling problem also appears in the situation that scheduling of tasks with shared resource involves two dimensions, the resource and the time.

A parallel application can also be found in a parallel database system, e.g. a parallel query execution plan generated by the query optimizer of a parallel database management system [111,68].

Reliable computing Except the efficiently and fast running, the reliability of computer systems is also very important. The reliability plays a crucial role on control systems in factory, avionics, automotive, medical and life support systems.

It is also a goal of the fault-tolerant system, which detect errors, or recover from errors. The required reliability software systems can be obtained by executing redundant replicas of software components and comparing their outputs [5,80,97]. The copies of a computer program must work in parallel on different processors in order to guarantee the required reliability. Furthermore, the redundant copies must visit the same states in about the same time to maintain replica determinism [97]. Thus, the replicated copies of a reliable software system plays the role of a multiprocessor task.

Applications of other problems In this thesis, we also study some classical scheduling problem, where each job requires exactly one machine.

For the extensible scheduling problem, i.e. the extensible bin packing problem, it arises in a wide variety of contexts on communication networks, from the packet scheduling problem to the bandwidth allocation problem. Consider, for example, a slotted **TDMA** (Time Division Multiple Access) [100,113] channel which is used both for the real-time and best-effort traffic. Packets of real-time flows which have high priority are allocated in fixed time slots. The left slots are available for best-effort packets with lower priority. The scheduling problem is to assign best-effort packets with equal sizes or variable sizes into the free slots after the allocation of real-time packets. Sometimes, the system discards some real-time packets in favor of the best-effort traffic, however the total value of them should small.

The extensible scheduling problem has also many applications in bin packing, storage allocation and scheduling problems. Consider, for instance, a set of workers is given with regular working time. In case of needed, the worker can do some overtime works. The problem is to assign duties to the workers in such a way that the total overtime works is minimized. For the parallel jobs version of this problem, which arise in the applications that many workers need to cooperate to finish a single work.

1.4 Outline

The remainder of this thesis is organized as follows. Chapter 2 gives the history overview of on-line scheduling of parallel jobs problems. In Chapter 3 we study the on-line scheduling of UET parallel jobs with dependencies on 2-dimensional mesh. The rotated 2-dimensional mesh topology is also considered. Chapter 4 explores the on-line scheduling where jobs arrive over list on PRAM. Several special cases are investigated and the preemptive schedule is also studied. Chapter 5 studies the malleable parallel jobs scheduling on 2-dimensional mesh. Chapter 6 studies the problem to maximize the throughput. In Chapter 7 the on-line scheduling problem with extendable working times is investigated. Both the identical machines and the machines with unequal regular working times are explored. In Chapter 8 we study the problem where partial information is known in advance. Conclusions and open problems are presented in Chapter 9.

Part I.

Scheduling of parallel jobs

Chapter 2

History overview

2.1 Introduction

In this chapter, we present a history overview for the problems of parallel jobs scheduling considered in Subsection 1.2.1.

The closely related problems to the parallel jobs scheduling are scheduling with resource constraints [66,18] and rectangle packing problem [9,33,96,123]. Parallel jobs scheduling differs from scheduling with resource constraints in the following sense: scheduling with resource constraints distinguishes between machines and resources while parallel jobs scheduling does not. For the makespan minimization problem of parallel jobs scheduling under the line network topology, it can be regarded as the *strip packing* problem [96,123]. In strip packing a finite set of rectangle is to be packed into a rectangular bin of finite width but infinite height, so as to minimize the total height used. Thus, the parallel jobs scheduling problem can be viewed as the strip packing problem, where the width of the rectangular bin corresponds to the machines and the height of the rectangular bin corresponds to the time axis. The size $size_j$ and processing time p_j of a parallel job are corresponding to the width and height of a rectangle that are requested to be packed.

It is worthy to notice that such similarities are often subject to minor but crucial differences in the definitions of the problems.

The rest of this chapter is organized as follows. In Section 2.2 we overview the known results on on-line non-preemptive scheduling to minimize makespan. The

known results on on-line preemptive scheduling are given in Section 2.3. Section 2.4 outlines the historical results on throughput maximization.

2.2 Non-preemptive scheduling on makespan objective

In this section, we concentrate on the problem where no preemption is allowed. Unless specified, the model with known job processing times is addressed, and the parallel jobs are non-malleable types in default. The objective is to minimize makespan. To compare with the (classical) on-line scheduling problems of non-parallel jobs, we give a short paragraph to survey the main results on non-parallel jobs when each on-line model is discussed.

In the following, we briefly introduce some off-line results, before we get into on-line models.

2.2.1 Off-line

The first papers on parallel jobs scheduling mainly investigated variants with unit execution times. Lloyd [105] showed that the problems $P|\, size_j, p_j = 1|C_{\max}$ and $P3|\, size_j,\ prec, p_j = 1|C_{\max}$ with $size_j \in \{1, 2\}$ both are NP-hard, whereas the problem $P2|\, size_j,\ prec, p_j = 1|C_{\max}$ can be solved polynomially. Blazewicz et al. [20] have proved that the problem $P|\, size_j, p_j = 1|C_{\max}$ is strongly NP-hard. Du and Leung [49] showed that the problem $P2|\, size_j|C_{\max}$ and $P3|\, size_j|C_{\max}$ are NP-hard, but are pseudopolynomialy solvable. On the other hand, they proved that $P5|\, size_j|C_{\max}$ is strongly NP-hard. It is still open whether problem $P4|\, size_j|C_{\max}$ is strongly NP-hard. For an overview of results on computational complexity and approximation algorithms, we refer to [45,46]. For latest development, see the website [24] maintained by Brucker and Knust.

2.2.2 Jobs arrive over list

Non-parallel jobs The classical scheduling problem where one job requires exactly one machine has been extensively studied, see [118] for a review. For the problem $P|\,\textit{on-line-list}|C_{\max}$, the first competitive ratio of $2-1/m$ was due to Graham [72]. Albers [2] presented an algorithm with competitive ratio of 1.923 when $m \geq 2$ and also proved that no deterministic on-line algorithm can have a competitive ratio better than 1.852. Fleischer and Wahl [64] improved the competitive ratio to 1.9201 for $m \to \infty$, and it is better than Albers's algorithm when $m \geq 64$. The lower bound was improved to 1.85358 by Gormley et al. [71]. As far as we know, the best known lower bound is 1.88 by Rudin [114].

Parallel jobs In the following, we study the problem $P|\,\textit{on-line-list}, \textit{size}_j|C_{\max}$. Upon arrival of a job, its size $\textit{size}_j$ and processing time p_j are known. The objective is to minimize the makespan.

As we have pointed out before the strip packing problem is equivalent to the problem $P|\,\textit{size}_j|C_{\max}$ under the line network, i.e. the problem $P|\,\textit{line}_j|C_{\max}$. The off-line strip packing problem was studied extensively. It was first studied by Baker et al. [9]. An algorithm with asymptotic worst-case ratio of 5/4 was proposed by Baker et al. [7] in 1981. After 15 years the bound was improved by Kenyon and Remila [96] who designed an asymptotic fully polynomial-time approximation scheme (AFPTAS). Steinberg [123] presented an approximation algorithm with absolute worst-case ratio of 2, which is the best known result in terms of the absolute worst-case analysis.

There are few on-line algorithms for this problem. The very first algorithms - shelf algorithms were developed by Baker and Schwartz [10], which achieve a competitive ratio of 6.99. However the competitive ratio of these shelf algorithms was given under the assumption that the processing times of all jobs are bounded by 1 [38]. Meanwhile the asymptotic worst-case ratio of these shelf algorithms was proved to be arbitrarily close to 1.7. An improved shelf algorithm shelf algorithm, based on harmonic algorithms was given by Csirik and Woeginger [37]. They showed that it is the best possible shelf algorithm for on-line strip packing, which has asymptotic

worst-case ratio of $h_\infty \approx 1.69103$.

A lower bound 2 on the competitive ratio of any on-line strip packing algorithm was given by Brown et al. in [23]. It is easy to verify the same lower bound for PRAM networks.

All the above results are also valid for the parallel jobs scheduling on PRAM networks. Obviously, we can apply all the algorithms for strip packing to PRAM networks. Take a look at the proofs of algorithms in strip packing. The lower bound of an optimal off-line algorithm always ignores the network topology. Then the upper bound of the problem on line network is also an upper bound of the problem on PRAM networks.

In fact, it was Johannes [90] who is the first one to design an on-line algorithm for the scheduling problem on PRAM networks with arbitrary job processing time. The competitive ratio of her algorithm is no more than 12. She also showed a lower bound of 2.25 with an enumerating program.

The results are summarized in Table 2.1.

Table 2.1: Results for $P|$ *on-line-list*, $\beta_2|C_{\max}$

network topology (β_2)	*upper bound*	*lower bound*
Non-parallel job ($\emptyset$)	1.9201 [64]	1.88 [114]
PRAM and $p_j \leq 1$	6.99 [10]	2 [23], 2.25 [90]
PRAM	12 [90]	2 [23], 2.25 [90]
Line (one dimensional mesh) and $p_j \leq 1$	6.99 [10]	2 [23], 2.25 [90]

2.2.3 Jobs arrive over time

Non-parallel jobs It has been studied for the on-line scheduling problem where jobs arrive over time (see [118] as a survey). Graham [72] presented a list scheduling algorithm with competitive ratio of $2 - 1/m$ for $P|$ *on-line-list*$|C_{\max}$. Gusfield [75], and Hall and Shmoys [76] observed that Graham's result holds for the problem with release dates $P|$ *on-line-time*$|C_{\max}$ too. To our best knowledge, the best known upper bound for the problem $P|$ *on-line-time*$|C_{\max}$ was obtained with the simple algorithm

*O*n-Line LPT which always schedules the available job with the longest processing time if there is a machine idle. The competitive ratio of this algorithm is 1.5 [28]. In the same paper, the best known lower bound 1.347 was provided. For the two-machine case, Noga and Seiden [108] presented an algorithm with competitive ratio $(5-\sqrt{5})/2$, which matches the lower bound presented in [28] .

Parallel jobs There are only a few results on on-line scheduling of parallel jobs, where jobs arrive over time. For the model with unknown processing times, Feldmann et al. [62] assumed that $r_j = 0$ for all jobs and proved that the list scheduling algorithm [65] is $2 - 1/m$-competitive. At any time the list scheduling algorithm always starts an available job if its size is at most the number of the idle machines. Shmoys et al. [119] showed that there is no deterministic on-line algorithm with a better competitive ratio than $2-1/m$ for the model with unknown processing times, even if any job requires exactly one machine and arrives at time zero. It implies that the list scheduling is optimal. Naroska and Schwiegelshohn [109] analyzed a list scheduling heuristic for the problem with arbitrary release times and showed that the competitive ratio of $2 - 1/m$ remains ture.

For the known processing times model where the processing time of a job is known upon arrival of the job, clearly the above upper bound $2-1/m$ still holds for list scheduling. However it might not be a lower bound on any on-line algorithms. In particular if all jobs are available at time zero, the problem becomes off-line. For the general case, as far as we know, there is no any result.

Table 2.2 summarizes the results.

Table 2.2: Results on $P|$ *on-line-time*, $\beta_2, \beta_3|C_{\max}$

network topology (β_2)	processing time (β_3)	*upper bound*	*lower bound*
Non-parallel job ($\emptyset$)	Known proc. time($\emptyset$)	1.5 [28]	1.347 [28]
PRAM	Known proc. time($\emptyset$)	$2-1/m$ [65,119,109]	1.347 [28]
PRAM	Unknown proc. time	$2-1/m$ [109]	$2-1/m$ [119]

2.2.4 Jobs arrive on dependencies

Non-parallel jobs Graham [72,73] showed that a greedy algorithm can achieve a competitive ratio of $2 - 1/m$ for the problem $P|$ *on-line-prec*, *non-clairvoyant*$|C_{\max}$. Epstein [51] showed that this is optimal even if the processing times are known and preemption is allowed.

Parallel jobs On-line scheduling of parallel jobs with or without precedence constraints was studied by Feldmann et al. [62,61] and Sgall [117]. For the sake of completeness we also state below their results without dependencies.

No dependencies and unknown processing times Feldmann et al. [62] and Sgall [117] studied the on-line scheduling of independent parallel jobs with unknown processing times on various networks. The size of a parallel job is known in advance, however, the processing time of the job is unknown before its completion. There are no dependencies between jobs. For PRAMs, any greedy algorithm which schedules a job whenever there are some machines available is optimal. Its competitive ratio is $2 - 1/m$. For hypercubes, an optimal on-line algorithm with competitive ratio of $2 - 1/m$ was derived. For the one dimensional mesh (line) model, a lower bound of $2-1/m$ for any deterministic on-line algorithm and an upper bound of 2.5 were presented. For 2-dimensional meshes, no on-line deterministic algorithms can achieve competitive ratio less than $\Omega(\sqrt{\log \log m})$ and an algorithm with competitive ratio of $O(\sqrt{\log \log m})$ was provided. In their papers, they also studied the problem on d-dimensional meshes. A lower bound of $\Omega(\sqrt{\log \log m})$ for any deterministic on-line algorithm and an upper bound of $O(2^d \log d \sqrt{\log \log m} + 2^d (d \log d)^d)$ were provided.

With dependencies and unknown processing times For the problem $P|$ *on-line-prec*, *size*$_j$, *non-clairvoyant*$|C_{\max}$, in [61,117] it is shown that the worst-case performance of any deterministic or randomized on-line algorithm for scheduling parallel jobs with precedence constraints and unknown processing times is rather dismal (the ratio is m), even if the precedence constraints among the jobs are known in advance. If there is restriction on the size of jobs, i.e. *size*$_j \leq \lambda m$, where $0 < \lambda < 1$

and $1 \leq j \leq n$. An optimal on-line algorithm with competitive ratio of $1 + \frac{1}{1-\lambda}$ was given.

A technique called *virtualization* [81] was introduced for this problem, where a parallel job can be scheduled on fewer machines than it requests. A parallel job J_j with size s_j and processing time p_j can be scheduled on s'_j machines, where $s'_j < s_j$. However the processing time of job J_j becomes $p_j * \frac{s_j}{s'_j}$. In this case, the work of a job is preserved and the parallel job is called *ideally malleable job*, as opposed to the non-ideally malleable jobs where the work is not preserved.

The problem with virtualization was investigated in [61,117]. For PRAMs, a best possible on-line algorithm with competitive ratio of $2+\phi \approx 2.618$ was achieved, where ϕ is the golden ratio. Similarly, for the case that $size_j \leq \lambda m$, they showed a best possible algorithm with competitive ratio of $2 + \frac{\sqrt{4\lambda^2+1}-1}{2\lambda}$. For hypercubes, only an upper bound of $O(\frac{\log m}{\log \log m})$ was provide. No nontrivial lower bound is known yet. For the line model, they provided an on-line algorithm with a lower bound of $\Omega(\frac{\log m}{\log \log m})$ for any deterministic on-line algorithm and an upper bound of $O(\frac{\log m}{\log \log m})$. For d-dimensional meshes, a lower bound of $\Omega(\frac{\log m}{\log \log m})$ for any deterministic on-line algorithm was shown. An upper bound of $O((\frac{\log m}{\log \log m})^d)$ was achieved.

Parallel jobs with runtime restrictions Since the performance ratio of the on-line scheduling of parallel jobs with dependencies and unknown processing times is rather dismal. The approaches to improve the performance are to restrict the maximum size of a job to λm machines and to use the virtualization. However, there exists drawbacks. Restricting the maximum size of a job can severely restrict the problem size that can be solved on a particular machine. Virtualization may be impossible or prohibitively expensive if such memory limitations exist.

Another possibility to improve the performance is restricting the processing time. Bischof et al. [16,15] studied the case that there is some a priori knowledge about the processing time of the individual jobs but the dependencies are unknown to the scheduler. They first considered the problem that the job processing times are equal. Such a job system is denoted by UET. For the PRAM and line topologies, they presented a 2.7-competitive algorithm and showed a lower bound of 2.691 for

any deterministic on-line algorithms. For the hypercube network, they gave a best possible on-line algorithm with competitive ratio of 2. For two-dimensional meshes, they derived a 46/7-competitive algorithm and a lower bound of 3.25 for any deterministic on-line algorithm. Then they considered the model with runtime ratio restriction (the quotient of the longest and shortest processing times) for PRAMs. When the shortest processing time is known, a family of job systems with runtime ratio $T_R \geq 2$ was given that bounds the competitive ratio of any deterministic on-line algorithm by $(T_R + 1)/2$ from below. An on-line algorithm with competitive ratio of $T_R/2 + 4$ was provided for the job system with runtime ratio $\leq T_R$. If the assumption that the shortest processing time is known is dropped, a modified algorithm with competitive ratio of $T_R/2 + 5.5$ was given.

Table 2.3 summarizes the results of the on-line problem without dependencies and unknown processing time. The results of on-line problem with dependencies and unknown processing time or with processing time restrictions are concluded in Table 2.4.

Table 2.3: Results on $P|\beta_1$, *non-clairvoyant*$|C_{\max}$

network topology (β_1)	*upper bound*	*lower bound*
Non-parallel job ($\emptyset$)	$2 - 1/m$ [119]	$2 - 1/m$ [119]
PRAM	$2 - 1/m$ [62,118]	$2 - 1/m$ [62,118]
Line	2.5 [62,118]	$2 - 1/m$ [62,118]
Hypercube	$2 - 1/m$ [62,118]	$2 - 1/m$ [62,118]
2-dimensional mesh	$O(\sqrt{\log \log m})$ [62,118]	$O(\sqrt{\log \log m})$ [62,118]
d-dimensional mesh	$O((\frac{\log m}{\log \log m})^d)$ [62,118]	$\Omega(\frac{\log m}{\log \log m})$ [62,118]

2.3 Preemptive scheduling with makespan objective

In the section, we focus on the schedules where preemption is allowed. Preemptive scheduling of non-parallel job problem has been extensively studied, see [27,118].

Table 2.4: Results on $P|\,\textit{on-line-prec}, \beta_1, \beta_2|C_{\max}$

processing time (β_2)	*network topology* (β_1)	*upper bound*	*lower bound*
non-clairvoyant	Arbitrary network	m [61,117]	m [61,117]
UET job system	PRAM or Line	2.7 [16,15]	2.691 [16,15]
$p_j = 1$	Hypercube	$2 - 1/m$ [16,15]	$2 - 1/m$ [16,15]
	2-dimensional mesh	$\frac{46}{7}$ [16,15]	3.25 [16,15]
Runtime ratio $\leq T_R$	PRAM	$T_R/2 + 5.5$ [16,15]	$(T_R + 1)/2$ [16,15]

Non-parallel jobs The off-line version of this problem $P|\,\textit{pmtn}|C_{\max}$ was solved to optimality in polynomial time [106]. Chen et al. [29] presented an optimal on-line algorithm with competitive ratio of $1/(1-(1-1/m)^m)$, which approaches $\frac{e}{e-1} \approx 1.58$ as $m \to \infty$, for the the problem $P|\,\textit{on-line-list}, \textit{pmtn}|C_{\max}$.

It was proved by Hong and Leung [79] and by Gonzales and Johnson [70] that the problem $P|\,\textit{on-line-time}, \textit{pmtn}|C_{\max}$ can be solved optimally, i.e., it is 1-competitive.

Graham [72,73] showed that a greedy (non-preemptive) algorithm achieves a competitive ratio of $2 - 1/m$ for the problem $P|\,\textit{on-line-prec}, \textit{non-clairvoyant}|C_{\max}$. Epstein [51] pointed out that this is optimal even if the processing times are known and preemption is allowed. Thus, the optimal on-line algorithm for the problem $P|\,\textit{on-line-prec}, \textit{pmtn}|C_{\max}$ is $(2 - 1/m)$-competitive.

Parallel jobs Unlike the non-parallel jobs system, the off-line problem for the parallel jobs, i.e., $P|\,\textit{size}_j, \textit{pmtn}|C_{\max}$, is NP-hard for arbitrary number of machines [44]. Problem $Pm|\,\textit{size}_j, \textit{pmtn}|C_{\max}$ can be solved in polynomial time [20]. Jansen and Porkolab [84] further studied the problem with fixed m by giving a linear time algorithm.

The off-line preemptive scheduling of parallel jobs on hypercube topology was also well studied [46]. Problem $P|\,\textit{cube}_j, \textit{pmtn}|C_{\max}$ can be solved polynomially [31,78,1,43].

For the preemptive on-line scheduling of parallel jobs, there are only few results. As far as we know, there is no results on the preemptive on-line scheduling where

jobs arrive over list.

There are some results on PRAMs where jobs arrive over time. In [8] Baker et al. provided an on-line algorithm with the makespan bounded by $r_n+5H+1.7OPT$, where r_n is the largest release dates, H is largest processing time over all the jobs and OPT is the optimal value. Johannes [90] presented an algorithm with competitive ratio at most $2-1/m$.

We summarize the results on on-line preemptive scheduling in Table 2.5.

Table 2.5: Results on $P|\beta_1, \beta_2, pmtn|C_{\max}$

network topology (β_1)	*on-line model* (β_2)	*upper bound*	*lower bound*
Non-parallel job ($\emptyset$)	*on-line-list*	1.58 [29]	1.58 [29]
	on-line-time	1 [70,79]	1 [70,79]
	on-line-prec	$2-1/m$ [72,73]	$2-1/m$ [51]
	on-line-prec, *non-clairvoyant*	$2-1/m$ [72,73]	$2-1/m$ [51]
PRAM ($size_j$)	*on-line-time*, *non-clairvoyant*	$2-1/m$ [90]	

2.4 Maximizing the throughput

The classical scheduling theory, where each job requests exactly one machine, has been studied extensively for the objective of minimizing the (weighted) number of *late* jobs $(w_i)U_i$, where $U_i = 1$ if job J_i is completed after d_i, and $U_i = 0$ otherwise (see [101,25]). In contrast, previous research for parallel jobs has mainly concentrated on the objectives of minimizing the *makespan* C_{max} and *the sum of completion times* $\sum C_i$. To our best knowledge, the first paper concerning throughput objective for parallel jobs is due to Fishkin and Zhang [63]. It was shown that $P|size_i, p_i = 1|\sum \bar{U}_i$ and $P|fix_i, p_i = 1, d_i = D|\sum \bar{U}_i$ both are strongly NP-hard, where fix_i denotes the *dedicated* variant and $d_i = D$ denotes the common due date variant. They proposed an algorithm with worst-case ratio bounded by $3/2-1/(2m)(m$ is odd) and $3/2-1/(2m-2)$ (m is even) for PRAMs, where m is the number of machines.

For the *dedicated* variant, they showed that a greedy algorithm can achieve worst-case ratio bounded from above by $\sqrt{m}+1$, and the problem can not be approximated within a factor of $m^{1/2-\varepsilon}$ for any $\varepsilon > 0$, unless $NP = ZPP$.

Jansen and Zhang [86] studied the maximizing throughput problem on rectangle packing. A set of rectangles is given, each of which is associated with a profit, and one is asked to pack a subset of the rectangles into a bigger rectangle so that the total profit of rectangles packed is maximized. Regarding the bigger rectangle as a two-dimensional mesh and smaller rectangles as the jobs which have unit processing times, profits, sizes (submeshes) and a common due date, the rectangle packing problem can be transferred into the problem of scheduling parallel jobs on 2-dimensional meshes. They presented a $(2 + \varepsilon)$-approximation algorithm for any given $\varepsilon > 0$. Furthermore, they [87] considered packing a set of rectangles into a bigger rectangle so that the objective is to maximize the number of packed items. They proved that there exists an asymptotic FPTAS, and also a PTAS, for packing squares into a rectangle. They also gave an approximation algorithm with asymptotic ratio of 2 for packing rectangles, and further showed a simple $(2+\varepsilon)$-approximation algorithm.

Chapter 3

On-line scheduling of parallel jobs with dependencies on 2-dimensional meshs

3.1 Introduction

We study an on-line problem of scheduling parallel jobs on 2-dimensional meshes, i.e. the problem $P|$ *on-line-prec, 2-d mesh*$, p_j = 1|C_{\max}$. Parallel jobs arrive dynamically according to the dependencies between them, which are unknown before the jobs appear. Each job may need more than one processor simultaneously and is required to be scheduled on a submesh of the processors which are located on a 2-dimensional mesh, i.e., a job must be scheduled on a rectangle of given dimensions. We deal with an UET job system, in which all job processing times are equal. The objective is to minimize the makespan.

Recall that on-line scheduling of parallel jobs with precedence constraints was studied by Feldmann et al. [62,61] and Sgall [117]. In [61,117] it was shown that the worst-case performance of any deterministic or randomized on-line algorithm for scheduling parallel jobs with precedence constraints and unknown processing times is rather dismal, even if the precedence constraints among the jobs are known in advance. Bischof et al. [15,16] studied the case that there is some a priori knowledge about the processing times of the individual jobs but the dependencies are unknown

to the scheduler. They consider the problem that the job processing times are equal. Such a job system is denoted by UET. For the *PRAM* topology, they presented a 2.7-competitive algorithm and a lower bound of 2.691 for any deterministic on-line algorithms. For the hypercube network topology, they gave a best possible on-line algorithm with competitive ratio of 2. For two-dimensional meshes, they derived a 46/7-competitive algorithm and a lower bound of 3.25 for any deterministic on-line algorithm.

Note that for an UET job system there are already optimal or nearly optimal on-line algorithms if the network topology is a *hypercubes* or a *PRAM*. However, there is a big gap between the best known lower bound and the upper bound for the two-dimensional mesh topology. In this chapter, we are concerned with on-line scheduling of an UET job system on two-dimensional meshes and improve the previous results by Bischof [15]. A two-dimensional mesh $N_1 * N_2$ is a parallel system consisting of $N_1 \times N_2$ processors $\{M_{ij} | 0 \leq i < N_1, 0 \leq j < N_2\}$ where processor M_{ij} is directly connected with processors $M_{i,j\pm1}$, $M_{i\pm1,j}$ (if they exist). Each job has a unit processing time and must be scheduled on a rectangle of given dimensions. Job J_i is characterized by (a_i, b_i), meaning J_i requires an $a_i * b_i$ submesh. We call this topology a *normal 2-d mesh*. On the other hand, it is also reasonable to assume that parallel jobs can be rotated, which means that job J_i can also be scheduled on a $b_i * a_i$ submesh (in this case we assume that $\max_i\{a_i, b_i\} \leq \min\{N_1, N_2\}$), where the processors in each node of the mesh are identical. Such a topology is called a *rotated 2-d mesh*. There are precedence constraints among jobs. A job is available if and only if its predecessors have been completed. The precedences are unknown in advance and an on-line algorithm is only aware of available jobs and has no knowledge about their successors. The goal is to minimize the maximum job completion time (the makespan).

For normal 2-d meshes we give a lower bound of 3.859 for any deterministic on-line algorithms and present a 5.25-competitive algorithm which adopts Steinberg's algorithm [123] as a subroutine. It significantly improves a previous lower bound of 3.25 and a previous upper bound of 46/7. Then we consider the rotated 2-d mesh topology. A 4.25-competitive algorithm is given. Slightly revising the instance

borrowed from [52] shows that the competitive ratio of any on-line algorithm is at least 3.535.

The remainder of this chapter is organized as follows. Section 3.2 gives preliminaries. In Section 3.3, we give lower bounds for any on-line algorithm on normal 2-d meshes and on rotated 2-d meshes. On-line algorithms are presented in Section 3.4.

3.2 Preliminaries

To evaluate an on-line algorithm we adopt the standard measure - *competitive ratio*, which is defined in Section 1.1.6.

Regarding the two-dimensional mesh as a rectangular bin and the jobs (with unit processing times) as rectangles (items), the problem of scheduling a set of available jobs at each unit time interval can be regarded as a 2-dimensional bin packing problem. A job (a_i, b_i) has a width a_i and a height b_i. The makespan of a schedule is just the number of bins used for packing all jobs. Note that the on-line issue is only with respect to the precedences: a job is available if and only if its predecessors have been completed. We distinguish the jobs by levels. The jobs in level 1 are available at the beginning. A job belongs to level i if its predecessor(s) falls in level $i-1$, for $i \geq 2$.

We adopt Steinberg's algorithm [123] as a subroutine of our algorithm. In 2-dimensional bin packing, we are given a list of rectangles $R = (R_1, \cdots, R_l)$, where rectangle R_i has a width a_i and a height b_i. Let $a_L = \max\{a_i | 1 \leq i \leq l\}$, $b_L = \max\{b_i | 1 \leq i \leq l\}$, $s_i = a_i b_i$ and $S_L = \sum_{i=1}^{l} s_i$. Then we get the following lemma.

Lemma 3.2.1 [123] If the following inequalities hold

$$a_L \leq u, \quad b_L \leq v \quad \textit{and} \quad 2S_L \leq uv - (2a_L - u)_+(2b_L - v)_+$$

then it is possible to pack the rectangles of R into a rectangle with a width u and a height v by Steinberg's algorithm, where $x_+ = \max(x, 0)$.

Remark. Note that if the height (the width) of any rectangle is at most $v/2$ (at most $u/2$), the rectangles with total area at most $(uv)/2$ can be packed into a rectangle (bin) with a width u and a height v.

3.3 Lower bounds

In [16,15], Salzer numbers [115] are used to construct an instance for a lower bound of 2.691 for the $PRAM$ network topology. We extend the idea to the normal 2-d mesh topology. The Salzer number t_i is defined as follows. $t_1 = 2$, $t_{i+1} = t_i(t_i - 1) + 1$ for $i \geq 1$. Define $h_\infty = \sum_{i=1}^{\infty} \frac{1}{t_i - 1} > 1.69103$.

Theorem 3.3.1 No on-line algorithms can have competitive ratios lower than $h_\infty^2 + 1 > 3.859$.

Proof. Let $k > 0$ be an arbitrarily large integer. From the definition of Salzer numbers, we have $\sum_{i=1}^{k} 1/t_i + 1/(t_{k+1} - 1) = 1$, $\prod_{i=1}^{k} t_i = t_{k+1} - 1$. We choose N such that $N > (k+1)(t_{k+2} - 1)$. Consider a mesh $N * N$. Let $A_i = \lfloor \frac{N}{t_i} \rfloor + 1$, where t_i's are Salzer numbers, for $i = 1, \ldots, k$. Set $A_{k+1} = N - \sum_{i=1}^{k} A_i - 1$. Then $N/(t_{k+1} - 1) > A_{k+1} \geq N/(t_{k+1} - 1) - (k+1) > 0$.

The job system consists of l levels, where $l \geq k^3$. The $(i + (k+1)(j-1))$-st level consists of $l - (i-1) - (j-1)(k+1)$ jobs with size (A_i, A_j), $i = 1, \cdots, k+1$ and $j = 1, \cdots, k+1$. The last $l - (k+1)^2$ levels form a chain of $l - (k+1)^2$ jobs with size (1, 1). At each level, one of the jobs is the predecessor of all jobs of the next level. Note that jobs at the same level have the same size.

Dependencies are assigned dynamically by the adversary. In the optimal solution, we first process the available job which is the predecessor of the next level. Then we divide in height of the mesh into $k+2$ shelves. The j-th shelf has a height of A_j, $1 \leq j \leq k+1$, and the $(k+2)$-nd (the last) shelf has a height of one. We assign (A_i, A_j) into shelf of height A_j and a job of the chain is assigned into a shelf with height 1. It results in a schedule with length of l. The schedule is illustrated as

follows.

Time intervals	Jobs scheduled
$(0,1]$	$\{(A_1,A_1)\}$
$(1,2]$	$\{(A_1,A_1),(A_2,A_1)\}$
$(2,3]$	$\{(A_1,A_1),(A_2,A_1),(A_3,A_1)\}$
...	...
$((k+1)^2-1,(k+1)^2]$	$\{(A_1,A_1),(A_2,A_1),\ldots,(A_{k+1},A_{k+1})\}$
$((k+1)^2,(k+1)^2+1]$	$\{(A_1,A_1),(A_2,A_1),\ldots,(A_{k+1},A_{k+1}),(1,1)\}$
...	...
$(l-1,l]$	$\{(A_1,A_1),(A_2,A_1),\ldots,(A_{k+1},A_{k+1}),(1,1)\}$

Figure 3.1 shows that the optimal schedule in a unit time interval when $k=2$.

Contrary to the optimal schedule, the job which is the predecessor of all jobs in the next level must be scheduled last among the jobs in the same level by any on-line scheduler, since all the jobs in a level have the same size and the on-line scheduler can not distinguish them. Note that $N/(t_{k+1}-1) > A_{k+1} \geq N/(t_{k+1}-1)-(k+1)$. It implies that at most $(t_i-1)(t_{k+1}-1)$ jobs of size (A_i,A_{k+1}) can be scheduled together on the mesh. It is also easy to check that at most $(t_i-1)(t_j-1)$ jobs of size (A_i,A_j), $(1 \leq i,j \leq k+1)$ can be scheduled together on the mesh by any on-line scheduler. Thus the length generated by the on-line scheduler is at least

$$\begin{aligned} & \sum_{j=1}^{k+1}\sum_{i=1}^{k+1} \lceil \frac{l-(i-1)-(j-1)(k+1)}{(t_i-1)(t_j-1)} \rceil + l - (k+1)^2 \\ \geq & \sum_{j=1}^{k+1}\sum_{i=1}^{k+1} \frac{l-(k+1)^2}{(t_i-1)(t_j-1)} + l - (k+1)^2 \\ = & ((\sum_{i=1}^{k+1} \frac{1}{t_i-1})^2+1)(l-(k+1)^2) \end{aligned}$$

Recall that the optimal schedule has a length of l and $l \geq k^3$. For $k \to \infty$, the competitive ratio of any on-line algorithm can be arbitrarily close to h_∞^2+1. □

Epstein [52] showed that the competitive ratio of any bounded space on-line algorithm for two dimensional bin packing is at least 2.535 if the items can be

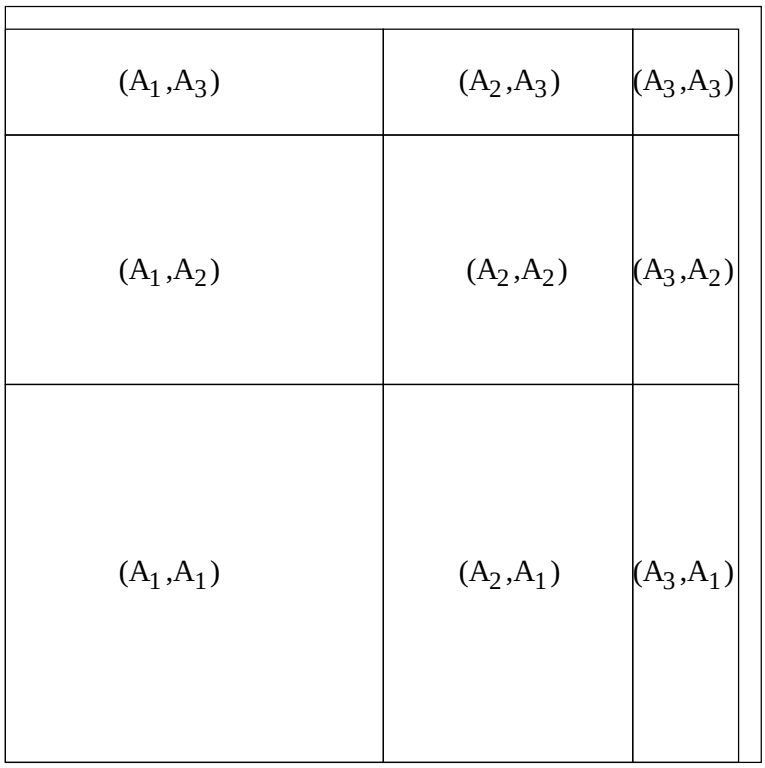

Figure 3.1: Optimal schedule when $k = 2$ in a unit time interval

rotated. In the instance eight types of items are introduced. Let $\delta > 0$ be a sufficiently small constant. Consider square bins of side length 1. Items of type 1 are squares of side length $1/2 + \delta$; items of type 2 are squares of side length $1/3 + \delta$; items of type 3 are rectangles of width $2/3 - \delta$ and height $1/3 + 2\delta$; items of type 4 are rectangles of width $2/3 - 2\delta$ and height $1/3 + 3\delta$; items of type 5 are rectangles of width $11/21 - 4\delta$ and height $10/63 + 2\delta$; items of type 6 are rectangles of width $32/63 - 4\delta$ and height $31/189 + 2\delta$; items of type 7 are squares of side length $1/7 + \delta$; items of type 8 are tiny squares with total area $361/47628 - \Theta(\delta)$. Seven items each from the first 7 types and all tiny squares from type 8, can fit in a bin since the items are rotatable. The following figure (Figure 3.2) illustrates how they can be packed into a bin.

The largest number of rectangles of the same type which can fit into a bin was shown in [52].

- Type 1: A bin can only contain at most 1 such rectangle.

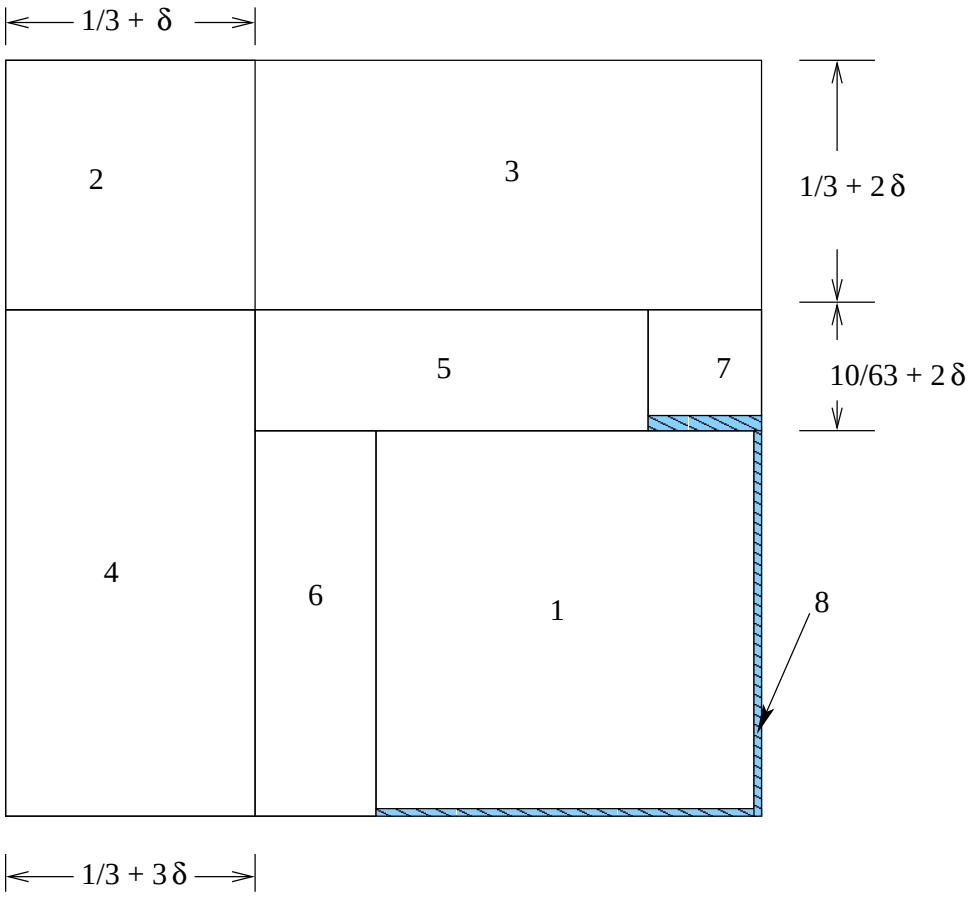

Figure 3.2: An illustration of packing items from the 8 types

- Type 2: A bin can only contain at most 4 such rectangles.
- Type 3: A bin can only contain at most 2 such rectangles.
- Type 4: A bin can only contain at most 2 such rectangles.
- Type 5: A bin can only contain at most 8 such rectangles.
- Type 6: A bin can only contain at most 8 such rectangles.
- Type 7: A bin can only contain at most 36 such rectangles.

We now slightly modify the above instance and apply it to our problem. We update type 8 by removing one tiny square. The removed square is set to be type 9. The squares of type 8 form a set, which is regarded as a single item (but being packed separately). Add precedence constraints among the jobs: A (critical) job of type i is the predecessor of all jobs of type $i+1$ for $i = 1, \ldots, 8$, and the job of type 9 form a chain. There are $l - i + 1$ jobs of Type i, for $i = 1, \ldots, 9$, where $l > 0$ is an arbitrarily large integer. We can assume that the last scheduled job of Type i ($i = 1, \ldots, 8$) by any on-line algorithm is the critical job. It is not difficult to prove the following lower bound.

Theorem 3.3.2 No on-line algorithms can have competitive ratio lower than 3.535 for scheduling on a rotated 2-d mesh.

Proof. It is easy to check that the optimal schedule has a length of l. The length generated by on-line scheduler is at least

$$l + \frac{l-1}{4} + \frac{l-2}{2} + \frac{l-3}{2} + \frac{l-4}{8} + \frac{l-5}{8} + \frac{l-6}{36} + \frac{361}{47628}(l-7) + l - 8.$$

Then the competitive ratio can be arbitrarily close to 3.535 when $l \to \infty$. □

3.4 On-line algorithms

For convenience, in this section we normalize an $N_1 * N_2$ mesh as a unit square (bin). A job J_i, denoted also by (a_i, b_i), has a width $a_i \leq 1$ and a height $b_i \leq 1$. The *work* of a job is defined as the number of requested processors divided by $N_1 \times N_2$. In

other words, the work of job J_i, is the area $a_i b_i$ of job J_i. The *efficiency* of a schedule at any time t is defined to be the number of busy processors at time t divided by $N_1 \times N_2$. Therefore, the efficiency of a schedule at any time can be viewed as the total work of the jobs in a bin. It is also called the efficiency of the bin. For any time unit, assigning jobs to the processors on a mesh can be regarded as packing rectangles into a square bin without any overlap. The resulting makespan by a schedule is exactly the number of bins used for packing the jobs.

We divide the jobs into *big*, *long*, *wide* and *small* jobs. A job is called big if both its width and its height are larger than 1/2. A job is called small if both its width and its height are at most 1/2. A job is long if its height is larger than 1/2 but its width is at most 1/2. A job is wide if its width is larger than 1/2 but its height is at most 1/2.

In the following we first present an (off-line) algorithm, which schedules available jobs into bins. Then applying this algorithm to the jobs level by level, we get an on-line algorithm.

Algorithm RP**.**

1. Group the jobs. In this step we just group the jobs into different bins. The jobs assigned to a bin are packed in the next step.

   - Put big jobs each in a bin.
   - Put long jobs with First-Fit to the partially filled bins (by big jobs) if the total width of the jobs is at most 1. If a long job can not fit in any of the partially-filled bins, open a new bin for it.
   - Open new bins for wide jobs and pack them with First-Fit. A job can be put to a bin if the total height of the jobs is at most one.
   - Consider all partially filled bins with total work of jobs packed less than 1/2. Put small jobs into them with First-Fit as long as the total work of jobs is at most 1/2. If a small job can not fit in any of the partially filled bins (i.e., the total work will exceed 1/2 if the small job is put into the bins), open a new bin for it.

2. Pack the jobs.

- If a bin contains no small jobs, these jobs can be easily packed into a bin since either the total height of them or the total width of them is no more than 1.
- If a bin contains no big job but some small jobs, the total work of the jobs in this bin is at most 1/2 and either all jobs are not wide or all jobs are not long. By Steinberg's algorithm, these jobs can be packed into a bin.
- If a bin contains a big job as well as some small jobs, this bin can be packed as follows. Let x be the total width of the big job and the long jobs (if any). Clearly, $x < 1$. Otherwise, the total work of them is over 1/2, and no small jobs can be accepted in the step for grouping jobs. Among the small jobs, let T_1 be the ones with width larger than $1 - x$ and let T_2 be the ones with width at most $1 - x$. Place the big job to the leftmost bottom of the bin. Put long jobs one by one upon the big job to the left. Put jobs of T_1 one by one on the right of the big job. Put jobs of T_2 to the free space above the long jobs by Steinberg's algorithm. This free space is exactly a rectangle with width $1 - x$ and height 1. The following figure (Figure 3.3) gives an illustration of the packing.

Lemma 3.4.1 The above packing is feasible.

Proof. We only need to consider the last case that a bin contains a big job as well as some small jobs. Let $a_{\max}$ and $b_{\max}$ be the width and the height of the big job, respectively. Let $s(T_1)$ be the total work of jobs in T_1. Assume that the total height of jobs in T_1 is larger than $1 - b_{\max}$. Then $s(T_1) > (1 - b_{\max})(1 - x)$. On the other hand, the total work of the big job and the long jobs is at least $a_{\max}b_{\max}+(x-a_{\max})/2$. Let $s(B)$ be the total work of the jobs in the bin.

$$\begin{aligned} s(B) &> a_{\max}b_{\max} + (x - a_{\max})/2 + (1 - b_{\max})(1 - x) \\ &= 1/2 + a_{\max}(b_{\max} - 1/2) + (1 - x)(1/2 - b_{\max}) \\ &= 1/2 + (a_{\max} + x - 1)(b_{\max} - 1/2) \\ &> 1/2 \end{aligned}$$

It gives a contradiction. Therefore, the total height of jobs in T_1 is at most $1 - b_{\max}$. These jobs can be packed to the right of the big job .

We turn to T_2. Let $s(T_2)$ be the the total work of jobs in T_2. $s(T_2) < 1/2 - x/2 = (1-x)/2$. In other words, the total work of jobs in T_2 is less than the half of the area of a rectangle with width $1 - x$ and height 1. The height of any job in T_2 is at most $1/2$ (they are small jobs) and their width is no more than $1 - x$. Using Steinberg's algorithm the jobs can be packed into the free space of the bin (a rectangle with width $1 - x$ and height 1). □

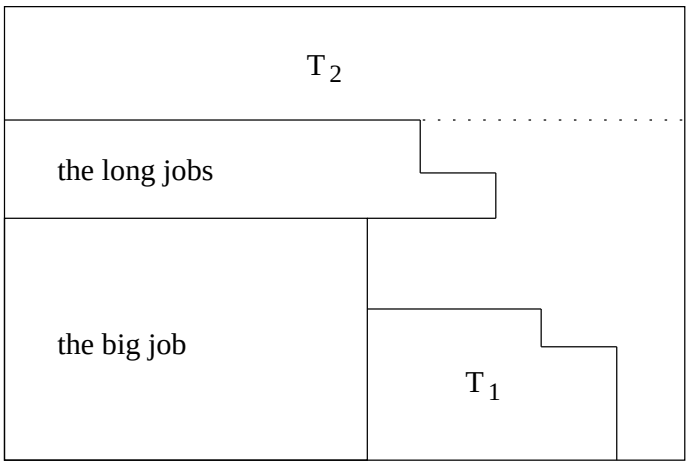

Figure 3.3: An illustration of the packing with a big job

In a packing given by algorithm RP, a bin is called a W-bin if it contains wide jobs, an L-bin if it contains a big job or long jobs, an S-bin if it contains only small jobs.

Lemma 3.4.2 At a level if $m \geq 3$ bins are used, then the total work of the jobs is at least $m/3 - b/12 - 1/2$, where b is the number of big jobs of the level.

Proof. We divide the m bins into two groups. G_1 consists of L-bins and S-bins, while G_2 consists of W-bins. Let $p = |G_1|$ and $q = |G_2|$. $m = p + q$. We first assume that $p \neq 1$ and $q \neq 1$. Consider G_1.

Case 1. There are no big jobs, i.e., $b = 0$. Except the last bin there are at most one bin with efficiency less than $1/3$. To observe this, if a bin B_i has efficiency less

than 1/3, then all jobs assigned to the succeeding bins have size larger than 1/6 (otherwise they should have been assigned to B_i). And any two jobs can be put into a bin. Therefore except B_i and perhaps the last bin all the other bins have efficiency larger than 1/3. On the other hand, the sum of efficiency of B_i and the last bin is more than 1/2. It follows that the total work of the jobs in G_1 is more than $(p-2)/3+1/2 = p/3-1/6$.

Case 2. $b \geq 1$. Note that each bin containing a big job has efficiency larger than 1/4. If all the bins containing a big job have efficiency at least 1/3, we obtain the same bound for the total work of the jobs as in Case 1. Assume that at least one of such bins has efficiency less than 1/3. Then all the other bins except the last one, which contain no big jobs, have efficiency larger than 1/3. Moreover, the sum of efficiency of a bin with a big job and the last bin is larger than 1/2. It implies that the total work of the jobs is at least $(b-1)/4+(p-b-1)/3+1/2 = p/3-b/12-1/12$.

Now we consider G_2. Since G_2 contains no big jobs, it is easy to obtain, analogously as Case 1, that the total work of the jobs in G_2 is more than $q/3-1/6$.

Thus the total work of the jobs in the level is more than $(p+q)/3-b/12-1/3 = m/3-b/12-1/3$.

If $p = 1$, the total work of the jobs is more than $q/3-1/6 = m/3-1/2$. If $q = 1$, the total work of the jobs is more than $m/3-1/2$ if $b = 0$, and the total work of the jobs is more than $m/3-b/12-5/12$ if $b \geq 1$.

By considering all the cases, we conclude that the total work of the jobs is at least $m/3-b/12-1/2$. □

Algorithm $N2d$**.** Apply algorithm RP to the jobs level by level. As the jobs of level i have been assigned, start a schedule for the jobs of level $i+1$.

Let l be the maximum number of bins used for packing the jobs in a level. Let k_i be the number of levels in which algorithm $N2d$ uses exactly i bins for packing the jobs, for $i = 1, 2, \ldots l$. Denote by b_i the total number of big jobs in the levels using i bins.

Theorem 3.4.3 The competitive ratio of algorithm $N2d$ is at most 5.25.

Proof. Clearly, $C_{N2d} = \sum_{i=1}^{l} ik_i$ and

$$C^* \geq \sum_{i=1}^{l} k_i. \tag{3.1}$$

By Lemma 3.4.2, the total work of all jobs is at least $\sum_{i=3}^{l}(k_i(i/3 - 1/2) - b_i/12)$. Then

$$C^* \geq \sum_{i=3}^{l}(ik_i/3 - b_i/12 - k_i/2). \tag{3.2}$$

On the other hand, any two big jobs can not be processed at the same time. It implies that

$$C^* \geq \sum_{i=3}^{l} b_i. \tag{3.3}$$

Summing up the inequalities (3.1) multiplying by 2, (3.2) multiplying by 3, (3.3) multiplying by 1/4, we have

$$\begin{aligned}(2+3+1/4)C^* &\geq \textstyle\sum_{i=1}^{l} 2k_i + \sum_{i=3}^{l}(ik_i - b_i/4 - 3k_i/2) + \sum_{i=3}^{l} b_i/4 \\ &\geq \textstyle\sum_{i=1}^{l} ik_i = C_{N2d}.\end{aligned}$$

It follows that $C_{N2d} \leq 5.25C^*$. □

Corollary 3.4.4 If there are only small jobs and long jobs (or wide jobs), the competitive ratio of algorithm $N2d$ is at most 4.

Proof. At a level if $i \geq 2$ bins are used, the total work of the jobs at this level is larger than $(i-2)/3+1/2 = i/3-1/6$. Adopt the same terminology as in the proof of Theorem 3.4.3. We have $C^* \geq \sum_{i=2}^{l}(ik_i/3 - k_i/6)$. Then

$$3C^* \geq \sum_{i=2}^{l}(ik_i - k_i/2). \tag{3.4}$$

Summing inequalities (3.1) and (3.4) we get

$$4C^* \geq \sum_{i=1}^{l} k_i + \sum_{i=2}^{l}(ik_i - k_i/2) \geq \sum_{i=1}^{l} ik_i = C_{N2d}.$$

□

Corollary 3.4.5 If there are no wide jobs (or no long jobs), the competitive ratio of algorithm $N2d$ is at most 4.25.

Proof. At a level if two bins are used, the total work of the jobs at this level is larger than 1/2. Similarly as the proof of Theorem 3.4.3, we have

$$C^* \geq k_2/2 + \sum_{i=3}^{l}(ik_i/3 - b_i/12 - k_i/3). \tag{3.5}$$

Summing up the inequalities (3.1), (3.5) multiplying by 3, (3.3) multiplying by 1/4, we have $C_{N2d} \leq 4.25C^*$. □

Finally we consider the problem of scheduling parallel jobs on a rotated 2-d mesh. For job $J_i = (a_i, b_i)$, we can schedule it on a submesh $a_i * b_i$ or a submesh $b_i * a_i$, where the processors at each node are identical.

Algorithm *R2d***:** Rotate the jobs such that $a_i \geq b_i$. Then apply algorithm $N2d$.

Theorem 3.4.6 The competitive ratio of algorithm R2d is at most 4.25.

Proof. It follows directly from Corollary 3.4.5. □

Chapter 4

On-line scheduling of parallel jobs in a list on PRAMs

4.1 Introduction and preliminaries

In this chapter we study an on-line problem of scheduling parallel jobs, i.e. problem $P|\,\textit{on-line-list}, \textit{size}_j|C_{\max}$. Parallel jobs are characterized by two parameters, processing time and size (the number of requested machines). Jobs $\{J_1, J_2, \cdots, J_n\}$ arrive over list, which are not known in advance. Upon arrival of job J_i, its processing time and the number of machines required become known, and it must immediately and irrevocably be scheduled. The next job J_{i+1} appears only after job J_i has been assigned. The goal is to minimize the *makespan*.

Baker and Schwartz [10] presented shelf algorithms, which have a competitive ratio of 6.99 under the assumption that the processing times of all the jobs are bounded from above by one. A lower bound 2 on the competitive ratio of any on-line strip packing algorithm was given by Brown et al. in [23]. It is easy to verify that it is also valid for PRAM networks.

The first study of the problem $P|\,\textit{on-line-list}, \textit{size}_j|C_{\max}$ was by Johannes [90], A 12-competitive algorithm was presented and a lower bound 2.25 was proved with an enumerating program.

Our contribution In this chapter, we will improve the upper bound given by Johannes [90]. We present an on-line algorithm with competitive ratio of 7. Then we study some special cases, in which we know some a priori information about the job sizes or job processing times. The first case assumes that *jobs arrive in non-increasing order of processing times*. We obtain an upper bound of 2 by employing a greedy algorithm, which schedules jobs as early as possible. Note that if all job processing times are equal to one, it becomes the classical bin packing problem, for which the best known lower bound and upper bound are 5/3 [23,135] and 7/4 [120], respectively. Next we deal with the second case that *jobs arrive in non-increasing order of sizes*. We show that the greedy algorithm has a competitive ratio of at most 2.75. If all job sizes are equal to one, it is the classical on-line parallel machine scheduling problem, for which the best known lower bound 1.85358 and upper bound 1.9201 were given by Gormley et al. [71] and, Fleischer and Wahl [64], respectively. Finally, we study the case that *the longest processing time is known in advance*, without loss of generality, it is assumed to be 1. We present an on-line algorithm with competitive ratio of 4.

We also investigate the schedule where preemption is allowed. we present an on-line preemptive algorithm for the scheduling of parallel jobs on PRAMs and Lines. A tight bound $2 - 1/m$ is proved, where the number of preemptions of this algorithm is $O(n^2)$. As far as we know, this is the first result on the problem $P|\, \textit{on-line-list},\ \textit{size}_j,\ \textit{pmtn}|C_{\max}$.

Preliminaries An instance I of our problem consists of a list of parallel jobs $\{J_1, J_2, \cdots, J_n\}$ and m identical machines. Each job J_j is characterized by size s_j, the number of required machines, and processing time p_j. Once job J_j appears, its size s_j and processing time p_j become known. We use (s_j, p_j) to denote job J_j. The *efficiency* of a schedule at any time t is defined to be the number of busy machines at time t divided by m. The average efficiency in disjoint intervals (a_i, b_i), $i = 1, \ldots, l$, is thus defined by

$$\frac{\sum_{i=1}^{l} \int_{a_i}^{b_i} (\text{ total number of busy machines at time } t)\, dt}{m \sum_{i=1}^{l} (b_i - a_i)}.$$

We adopt the standard measure *competitive ratio* (see Section 1.1.6) to evaluate an on-line algorithm.

When job J_j is coming, one can schedule it in a time interval which has a duration at least p_j and during which the number of idle machines is at least s_j. The simplest on-line algorithm is the following greedy algorithm.

Algorithm *Greedy*: As a job is coming, schedule it as early as possible.

As shown in [90] algorithm *Greedy* has a competitive ratio of m. The bound can be obtained with the following example of $2m$ jobs. Let $\varepsilon > 0$ be a sufficiently small number. The $(2i-1)$-st job is $(1, m + (i-1)\varepsilon)$, while the $2i$-th job is (m, ε), $i = 1, \ldots, m$. Algorithm *Greedy* processes the job one after another and gives a makespan of $m^2 + m(m-1)\varepsilon/2 + m\varepsilon$. However, one can first process the m jobs of size m and then process the m jobs of size 1 each on a machine. It gives a makespan of $m + (m-1)\varepsilon + m\varepsilon$. The competitive ratio goes to m as ε tends to zero. Figure 4.1 shows the schedule generated by algorithm *Greedy* and an optimal schedule.

The example tells us that it is not wise to be so greedy. To improve the bound we may leave some space for the future jobs in case that the current schedule is not in a "tight" manner. Johannes [90] designed an on-line algorithm alone this line, in which the time axis is partitioned into intervals. The length of the first interval I_1 depends on the processing time of the first job, and the next intervals I_i always have a double length of the interval I_{i-1}. The jobs are scheduled in different way in some intervals. It was shown that the competitive ratio of the algorithm is at most 12. Note that in her algorithm the partition of the time interval is almost independent on the current schedule. In the next section we propose an improved algorithm which partitions the time axis into intervals relying on the current schedule. We prove that the proposed algorithm has a competitive ratio 7.

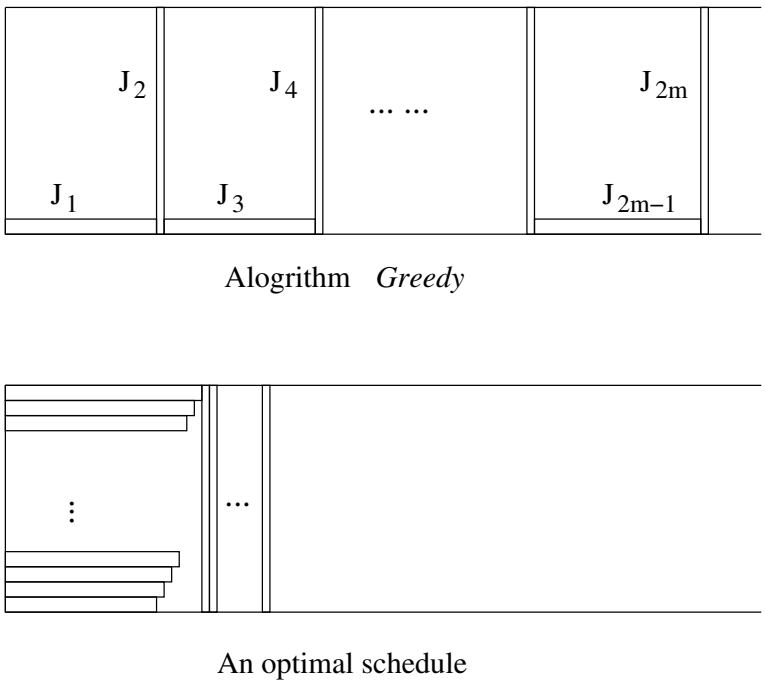

Figure 4.1: An illustration of algorithm *Greedy* and an optimal schedule

4.2 An improved on-line algorithm

A job is called *big* if its size is larger than $m/3$, otherwise it is called *small*. No two big jobs can be processed at the same time. To present the algorithm more clearly we show first the structure of the schedule given by the algorithm. The schedule looks like an aligned house, which consists of *rooms* and *walls*. Rooms and walls appear alternatively. The rooms are constructed by small jobs, while the walls are constructed by big jobs. A big job is assigned to an old wall or builds a new wall by itself. A small item is always assigned to a room. The schedule starts from a wall and ends at a wall. We assume that there is a wall at time zero with a length (thickness) of zero. In case that the last room has no wall on the right side after all jobs are scheduled, we add a dummy wall with length of zero starting at the completion of the last completed job. Therefore the schedule can be described as

$$W_1 \cup R_1 \cup W_2 \cup R_2 \cup \cdots \cup W_N \cup R_N \cup W_{N+1},$$

where W_i denotes the i-th wall and R_i denotes the i-th room. The length of wall W_i is the total processing time of the big jobs involved in the wall. The length of room R_i is the distance between two walls, more precisely, the time difference between

the starting time of the wall W_{i+1} and the ending time of the wall W_i. The length of the schedule is the total length of the rooms and the walls.

A wall is open if there are no small jobs scheduled after it and a room is called open if there is no wall on the right side.

Algorithm *DW* **(Dynamic Waiting)**

1. Create a wall (W_1) of length zero at time zero.

2. If the incoming job J_i is big, i.e., $s_i > m/3$, schedule J_i immediately after the open wall if such a wall exists. If there is no open wall, then there must be an open room. Let h be the length of the interval of efficiency less than 2/3 in the open room and let L be the current length of the schedule. Start job J_i at time of $L + h$.

3. If the incoming job J_i is small, i.e., $s_i \leq m/3$, find the first room which can accommodate J_i and schedule J_i as early as possible. If such a room does not exist, we create a new room for J_i following the present latest wall.

4. If no job comes, add a dummy wall if needed and stop.

An example: There are 10 machines and 9 jobs, where $J_1 = (3, 1)$, $J_2 = (3, 1)$, $J_3 = (2, 3)$, $J_4 = (4, 1)$, $J_5 = (1, 4)$, $J_6 = (1, 6)$, $J_7 = (1, 7)$, $J_8 = (7, 1)$, $J_9 = (6, 1)$. The schedule by algorithm *DW* is shown in Fig. 4.2, where $W_1 = [0, 0]$, $R_1 = [0, 5]$, $W_2 = [5, 6]$, $R_2 = [6, 20]$ and $W_3 = [20, 22]$.

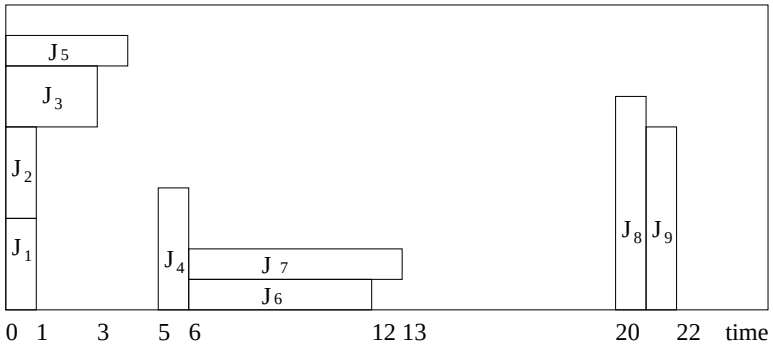

Figure 4.2: An illustration of algorithm *DW*

Lemma 4.2.1 Let S be a schedule by an algorithm A. If the length of the schedule S can be divided into two parts, one with average efficiency no less than α, $0 < \alpha \leq 1$, and another with total length of at most β times the optimal makespan, then the competitive ratio of algorithm A is at most $1/\alpha + \beta$.

Proof. Denote by x the length of the part with average efficiency no less than α and denote by y the length of another part. Thus $C_A = x + y$. Note that $C^* \geq \alpha x$ and $y \leq \beta C^*$. We have $C_A = x + y \leq (1/\alpha + \beta)C^*$. □

Theorem 4.2.2 The competitive ratio of algorithm DW is at most 7.

Proof. Let $\mathcal{L}$ be the final schedule by algorithm DW, i.e., $\mathcal{L} = W_1 \cup R_1 \cup \cdots \cup W_N \cup R_N \cup W_{N+1}$. By the algorithm, all rooms consist of small jobs. We divide room R_i into two time intervals E_i and F_i, where in interval E_i the efficiency of the schedule is at least 2/3, and $F_i = R_i - E_i$. Since R_i consists of small jobs, it is easy to see that E_i precedes F_i.

The idea to prove the bound of 7 is partitioning the time interval of the schedule $\mathcal{L}$ into two classes. In the first class the average efficiency of the intervals is at least 1/3 and in the second class the total length of the time intervals is bounded above by 4 times the longest job processing time. The theorem thus follows from Lemma 4.2.1.

Note that the efficiency of a wall is larger than 1/3. We only need to consider the rooms. If all $F_i = \emptyset$, the efficiency of the schedule at any time is at least 1/3, which implies that the competitive ratio is at most 3.

Assume that $F_k \neq \emptyset$ and $F_j = \emptyset$ for all $j < k$. If $k = N$, all intervals but F_N have efficiency at least 1/3. During the time interval F_N each machine executes at most one job. Otherwise the last job can start earlier since the efficiency of the schedule is less than 2/3 and all jobs in the room are small. By the algorithm, the length of F_N is at most twice the longest job processing time. It proves that the competitive ratio is at most 5 by Lemma 4.2.1.

Now we assume that $k < N$. Note that all time intervals before F_k have efficiency of at least 1/3. We consider the rooms R_j, $j = k, \ldots, N$. For any two adjacent rooms R_i and R_j $(j > i)$, the jobs scheduled in R_j must have processing time larger

than $|F_i|$, the length of F_i. Otherwise, they could have been assigned to room R_i. Therefore, either $E_j = \emptyset$ or $|E_j| > |F_i|$.

Find those $E_j \neq \emptyset$ for $j \geq k+1$. For each of such E_j, we have $|E_j| > |F_i|$, for $k \leq i < j$. We want to determine some F_i $(i \leq j)$ such that the average efficiency of E_j and F_i is at least 1/3. Then the pair of E_j and F_i is called a *match* and such an F_i is called *matched*. Before we start this procedure, all F_i's are *un-matched* for $i \geq k$. Start from the first non-empty E_j $(j > k)$ and continue the following matching procedure until all non-empty E_j's are matched. If $|E_j| \geq |F_j|$, we put E_j and F_j as a match. Otherwise, $|F_j| > |F_i|$ holds for all $k \leq i < j$. We put E_j and the un-matched F_i with the largest index $i < j$ together as a match, and thus F_i is matched. For each match, it is obvious that the average efficiency is at least 1/3.

We do the matching interval by interval starting from R_k (we do not count E_k) until all nonempty E_j's are matched for $j > k$. We will show that one of the following two assumptions must hold during the matching procedure.

1. There is only one un-matched F_i.

2. Re-index the un-matched F_i's and denote them by $\bar{F}_1, \bar{F}_2, \cdots$. Then $|\bar{F}_{j+1}| > 2|\bar{F}_j|$.

Recall that $F_k \neq \emptyset$. We start from intervals R_k. Clearly, the assumption 1 holds. Now we assume that one of the two assumptions is true when we deal with interval R_p $(p \geq k)$. Now we consider interval R_{p+1} with the following two cases.

Case 1. $E_{p+1} = \emptyset$. Consider the last un-matched $F_q \neq \emptyset$ preceding interval R_{p+1}. The small jobs in interval R_{p+1} have processing times greater than $|F_q|$ and the length of interval R_{p+1} is larger than twice of $|F_q|$ when the next wall W_{p+2} is created. Then $|F_{p+1}| > 2|F_q|$, where $q \leq p$.

Case 2. $E_{p+1} \neq \emptyset$. If there is only one un-matched F_i so far, then either F_{p+1} or F_i matches E_{p+1}. Thus there is only one un-matched interval after R_{p+1} is considered. The assumption 1 still holds. If there are more than one un-matched F_i, without loss of generality, re-index them as $\bar{F}_1, \bar{F}_2, \cdots, \bar{F}_q$. By the assumption 2, we have $|\bar{F}_{j+1}| > 2|\bar{F}_j|$, $1 \leq j \leq q-1$. If $|F_{p+1}| > |E_{p+1}|$, then $\bar{F}_q$ and E_{p+1} are matched and F_{p+1} is un-matched. We have $|F_{p+1}| > |\bar{F}_q| > 2|\bar{F}_{q-1}|$ and now F_{p+1} becomes $\bar{F}_q$.

If $|F_{p+1}| \leq |E_{p+1}|$, F_{p+1} and E_{p+1} are matched. The un-matched intervals remain unchanged.

Finally we get a list of un-matched intervals. Let x be the processing time of the longest job in the last un-matched interval $\bar{F}_t$. Thus $2x \geq |\bar{F}_t|$ by the algorithm. $\sum_{j=1}^{t} |\bar{F}_j| < 2|\bar{F}_t|$. Since $C^* \geq x$, we get that the total length of these intervals is bounded above by $4C^*$. Therefore the theorem follows. □

In the following, we will give an instance to show that a lower bound of algorithm DW is 7, i.e. the bound of this algorithm DW is tight. The instance consists of big jobs, sequential jobs and block jobs, where a sequential job has a size of one and a block job has a size of m.

Theorem 4.2.3 The competitive ratio of the algorithm DW is at least 7.

Proof. We assume m is sufficiently large number, without loss of generality, total number of machines generalized to 1 and all jobs request size no greater than 1. Let k be a sufficiently large integer and let $\varepsilon, \delta < 1/2^{k+2}$ be small positive numbers and $\delta > 2\varepsilon + 4/m$. The first group G_0 of jobs consists of 2^{k+1} big jobs, each with size of $1/3 + \varepsilon$ and processing time 1. Let $t = 2^k$. The next t job groups $G_1, \cdots, G_t$, each consists of 4 jobs $\{(1/3 - \delta, (1 - \varepsilon/i)/2), (1, \varepsilon), (1/3 - \delta, (1 - \varepsilon/i)/2), (4\delta, 1 - \varepsilon/i)\}$

The last group G_{t+1} of jobs consists of $3k - 1$ jobs $J_1, J_2, \cdots, J_{3k-1}$, where J_{3i-1} is a big job of $(1, \varepsilon)$ and J_{3i-2} is a small job of $(\varepsilon, 2^{i-1} + (2^{i-1} - 1)\varepsilon)$ and J_{3i} is a small job of $(\varepsilon, 2(2^{i-1} + (2^{i-1} - 1)\varepsilon))$, for $1 \leq i \leq k$. Clearly, $p_{3i+1} = 2p_{3i-2} + \varepsilon$, for $1 \leq i \leq k - 1$. In the schedule generated by algorithm DW we have $t + k$ rooms and $t + k + 1$ walls. The total length of the walls is $2^{k+1} + (t + k)\varepsilon$. The length of the room i is less than one, for $i = 1, \ldots, t$, while the length of the $t + i$-th room is $2p_{3i-2}$, for $i = 1, \ldots, k$. Therefore, $C_{DW} = 2^{k+1} + t + 2^{k+2} - 2 + O(\varepsilon)$ (see Fig. 4.3).

In an optimal schedule, we can process two G_0 jobs at the same time and assign the small jobs $J_1, J_4, \ldots, J_{3k-5}$ to a machine and the sequential job J_{3k-2} to one machine, $J_3, J_6, J_{3i}, \ldots, J_{3k-6}$ to one machine and J_{3k-3} to one machine. the sequential jobs of groups $G_1, \ldots, G_t$ to the left machines between time interval $[0, 2^k]$. The big jobs require size 1 are scheduled one by one following the sequential jobs (see Fig. 4.4). The above schedule is feasible, since $1/3 - \delta + 2/3 + 6\varepsilon \leq 1$ and $4\delta t \leq 1$.

Thus, it shows that $C^* \leq 2^k + 1 + O(\varepsilon)$.

$$C_{DW}/C^* \geq \frac{(2^{k+1} + 2^{k+2} + t - 2 + O(\varepsilon))}{(2^k + 1 + O(\varepsilon))} \rightarrow 7$$

as k tends to infinity. □

Theorem 4.2.3 shows that an asymptotic lower bound of algorithm DW is 7.

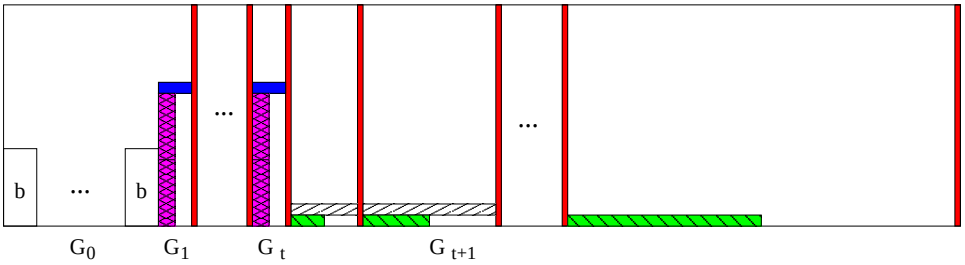

Figure 4.3: The schedule by algorithm DW

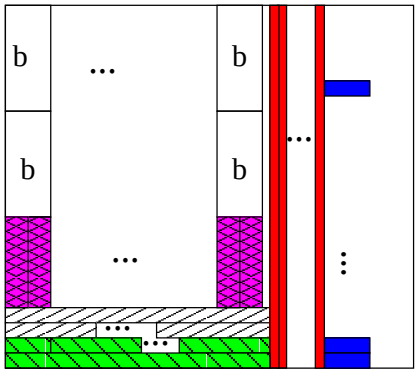

Figure 4.4: An optimal schedule

4.3 Semi on-line problems

In this section, we study the on-line scheduling of parallel jobs problems while some partial information is known in advance. We will see how the partial information can affect the schedule.

4.3.1 Non-increasing processing times

In this subsection, we study the case that the jobs arrive in non-increasing order of processing times. Note that if all the job processing times are equal to one, the scheduling problem is equivalent to the classical on-line bin packing problem. To see this, we can always assume that an on-line algorithm starts any job at an integral time. As far as we know, the best known lower bound and upper bound of classical on-line bin packing are 5/3 [23,135] and 7/4 [120], respectively. To make the thesis self-contained, we give an instance to show the lower bound of 5/3. Let $k > 0$ be a sufficiently large integer and let $m = 6k$. There are three groups, each of which consists of 6 jobs. The jobs of first group I_1 are of $(k-2, 1)$, the jobs of the second group I_2 are of $(2k+1, 1)$, and the jobs of the last group I_3 are of $(3k+1, 1)$. Clearly, $C^*(I_1) = 1$, $C^*(I_1 \cup I_2) = 3$ and $C^*(I_1 \cup I_2 \cup I_3) = 6$. For any on-line algorithm A, it is easy to verify that

$$\max\{\frac{C_A(I_1)}{C^*(I_1)}, \frac{C_A(I_1 \cup I_2)}{C^*(I_1 \cup I_2)}, \frac{C_A(I_1 \cup I_2 \cup I_3)}{C^*(I_1 \cup I_2 \cup I_3)}\} \geq 5/3.$$

In the following, we analyze algorithm *Greedy* for the special problem. Note that *Greedy* is just FF (*First Fit*) bin packing algorithm if all job processing times are equal. It is known that the (absolute) competitive ratio of FF for bin packing is in between 1.7 [92] and 7/4 [120].

Here we define jobs as big ones if its size is larger than $m/2$ and small ones otherwise.

Theorem 4.3.1 The competitive ratio of algorithm *Greedy* for the on-line parallel jobs scheduling with non-increasing processing times is at most 2.

Proof. Denote by $\mathcal{L}$ the time interval of the final schedule by *Greedy*. We partition it into parts according to the efficiency. Let $\mathcal{L}$=$B_1, L_1, B_2, \cdots, L_k, B_{k+1}$, where B_i

is the continuous time interval, at anytime of which the efficiency is larger than 1/2, and L_i is the continuous time interval, at anytime of which the efficiency is at most 1/2. To avoid trivial cases we assume that $L_i \neq \emptyset$, for $i = 1, \ldots, k$ and $B_i \neq \emptyset$, for $i = 2, \ldots, k$. Note that it is possible that $B_{k+1} = \emptyset$. The jobs processed during an interval $L_i = [a_i, b_i]$ are small. No jobs start at any time $t_i \in (a_i, b_i)$. If such a job exists, it would have been scheduled at time a_i since at that time there are at least $m/2$ machines available. It shows that the length of each L_i is at most the longest processing times among all small jobs. Thus $C^* \geq |L_i|$, for $i = 1, 2, \ldots, k$.

If $B_1 = \emptyset$, any job scheduled after L_1 is big (with a size larger than $m/2$); Otherwise, it could have been assigned to interval L_1. Let T be the total length of the (big) jobs scheduled after L_1. $C^* \geq T$ and $C^* \geq |L_1|$, while $C_{Greedy} = |L_1| + T$. It follows that the competitive ratio of *Greedy* is at most two. In the following, we assume that $B_1 \neq \emptyset$.

Consider L_i, $1 \leq i < k$. B_{i+1} must start with a big job, since, otherwise, a small job can be scheduled in L_i. We denote the big job by J_b with processing time of p_b. The small jobs scheduled in L_{i+1} arrive after J_b; Otherwise it would be scheduled in L_i since at that moment at least $m/2$ machines are idle. The processing time of any small job in L_{i+1} is at most p_b but larger than $|L_i|$. It shows that $p_b > |L_i|$. Then the average efficiency over the intervals L_i and B_{i+1} is at least 1/2.

Now we consider the interval L_k together with B_1. Denote the longest job in L_k by J_n with processing time p_n. Thus J_n is a small job and $|L_k| \leq p_n$. At the moment that J_n comes, J_n can not be assigned to B_1. Note that the jobs arrive in non-increasing order of processing times. The jobs, which have been assigned to start at time zero before J_n comes, have processing times at least p_n. Then $|B_1| \geq p_n \geq |L_k|$. We have the average efficiency over B_1 and L_k of at least 1/2.

Combining all the cases considered above, we conclude that the average efficiency of the schedule is at least 1/2, which implies the competitive ratio of algorithm *Greedy* is at most 2. □

4.3.2 Non-increasing job sizes

In this section, we deal with the case that jobs arrive in non-increasing order of sizes. The classical on-line scheduling problem which jobs require exactly one machine is a special problem of our case, if we assume that the job sizes are equal to one. To our knowledge, the currently best lower bound and upper bound are 1.85358 [71] and $1+\sqrt{\frac{1+\ln 2}{2}} < 1.9201$ [64], respectively. In the following, we first analyze algorithm *Greedy*, which is identical to *LS* (List Scheduling) for the classical schedule problem. We show that it is 2.75-competitive and present a lower bound of 2.5.

Theorem 4.3.2 Algorithm *Greedy* is 2.75-competitive.

Proof. Consider the schedule generated by algorithm *Greedy*. Let x be the length of the schedule during which a big job is processed. Then $C^* \geq x$. Denote by t the starting time of the last job J_n with processing time p_n, which determines the makespan. If $t = x$, we get that $C_{Greedy} \leq 2C^*$. Consider the time interval $[x, t)$. The jobs starting in this interval are small. If the efficiency at some time g in $[x, t)$ is less than $2/3$, any job starting at or later than g must have a size larger than $m/3$. Then the size of any earlier job starting in the time interval $[x, g)$ is larger than $m/3$ too. It shows that two such jobs can be scheduled together and the efficiency must be larger than $2/3$. It is a contradiction. Therefore, the efficiency in the interval $[x, t)$ is at least $2/3$. Let $y = t - x$. We have $C^* \geq x/2 + 2y/3$ and $C^* \geq p_n$. On the other hand, $C_{Greedy} = x + y + p_n$. The theorem is proved with the following cases:

Case 1: $y \leq \frac{3}{4}x$. In this case, $C_{Greedy} \leq \frac{7}{4}x + p_n \leq \frac{11}{4}C^*$.

Case 2: $y > \frac{3}{4}x$. In this case, $\frac{7}{4}C^* \geq \frac{7}{8}x + \frac{7}{6}y = x + y + \frac{1}{6}(y - \frac{3}{4}x) > x + y$. Combining with $C^* \geq p_n$, we get $C_{Greedy} \leq \frac{11}{4}C^*$. □

Theorem 4.3.3 The competitive ratio of algorithm *Greedy* is at least 2.5.

Proof. Let $k > 0$ be a sufficiently large integer. Let $m = 3 \cdot 2^k$ and $N = 2^{k-2}$. We are given 6 job groups $\{G_1, G_2, \cdots, G_6\}$. Group G_1 consists of N jobs of $(3 \cdot 2^{k-1} + N, 1), (3 \cdot 2^{k-1} + N - 1, 1), \cdots, (3 \cdot 2^{k-1} + 1, 1)$. G_2 consists of two identical jobs of $(3 \cdot 2^{k-1}, 1/N)$. G_3 has N jobs of $(3 \cdot 2^{k-1} - 1, 1/N), (3 \cdot 2^{k-1} - 2, 1/N), \cdots, (3 \cdot 2^{k-1} - N, 1/N)$. G_4 has N identical jobs of $(2^k + 1, 1)$. G_5 consists of three identical jobs

of $(2^k, 1)$. Finally, G_6 has only one job of $(1, N + 1/N)$. Clearly the jobs arrive in non-increasing order of sizes.

By algorithm *Greedy*, the jobs in G_1 are processed one by one. Then the two jobs in G_2 start at time N. The i-th job in group G_3 starts at time $N - i$, for $1 \leq i \leq N$. In the time interval $[0, N]$, the longest idle time of the machines is $1 - 1/N$. Then the G_4 jobs are scheduled in the interval $[N + 1/N, 3N/2 + 1/N]$, where two G_4 jobs are processed together at each unit time. The three G_5 jobs start at time $3N/2 + 1/N$. The last job (from G_6) can only start at $3N/2 + 1 + 1/N$. Thus $C_{Greedy} = 5N/2 + 1 + 2/N$ (see Fig. 4.5).

However, we can schedule jobs of G_1, G_4 and G_6 by time $N + 1/N$. Each two of the jobs of size $1/N$ (G_2 and G_3 jobs) can be processed at the same time. The three jobs of G_5 are scheduled together at time $N + 2/N + 1$. It implies that $C^* \leq N + 2/N + 2$ (see Fig. 4.6). Thus,

$$C_{Greedy}/C^* \geq 5/2,$$

as k tends to infinity. □

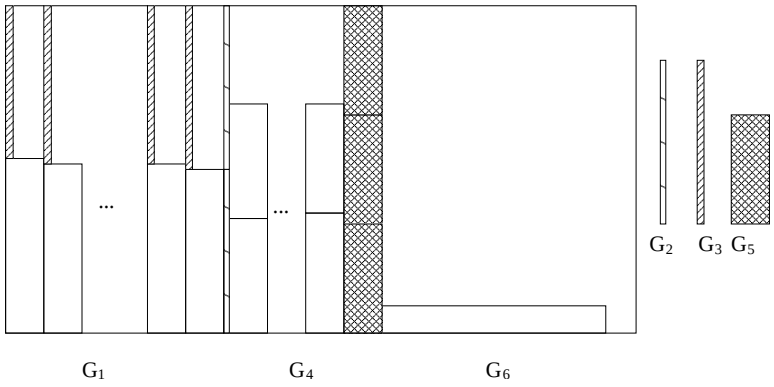

Figure 4.5: The schedule by algorithm *Greedy*

4.3.3 Known longest processing time

In this section, we investigate the problem with known partial information. The longest processing time of all the jobs is known in advance. Without loss of generality, we assume that $p_j \leq 1$, $j = 1, 2, \ldots, n$ and the there exists some $1 \leq k \leq n$ such

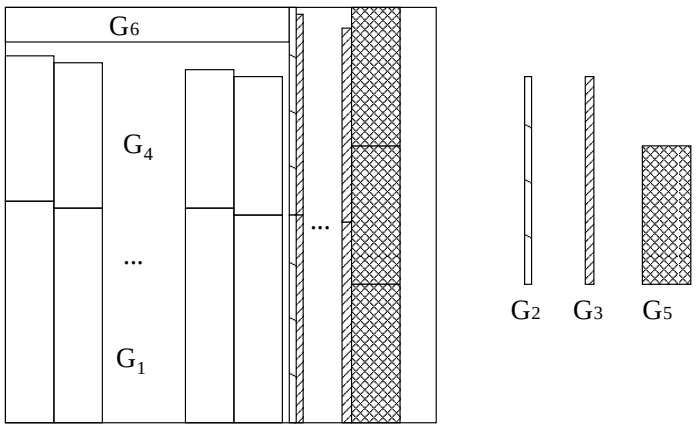

Figure 4.6: An optimal schedule

that $p_k = 1$. We distinguish jobs as big ones and small ones as in the Section 4.2. The definitions of wall and room are also the same as those in Section 4.2.

Algorithm FR (Fixed Room)

1. Create a wall (W_1) of length zero at time zero. Then run the following procedure as a new job is coming.

2. If the incoming job J_i is big, i.e., $s_i > m/3$, schedule J_i immediately following the open wall.

3. If the incoming job J_i is small, i.e., $s_i \leq m/3$, find the first room which can accommodate J_i, and schedule J_i as early as possible. If such a room does not exist, we create a new room for J_i following the present latest wall such that the length of this new room is exactly 1. Add a dummy wall at the end of this new room.

Theorem 4.3.4 The competitive ratio of algorithm FR is at most 4.

Proof. The algorithm is also feasible for small jobs, since the processing times of all the jobs are bounded from above by one, and thus the new room is big enough to accommodate a job. Similarly as the proof of Theorem 4.2.2, let $\mathcal{L}$ be the final schedule by algorithm FR, i.e., $\mathcal{L} = W_1 \cup R_1 \cup \cdots \cup W_N \cup R_N \cup W_{N+1}$. By the algorithm, all rooms consist of small jobs. We divide room R_i into two time intervals E_i and F_i, where in interval E_i the efficiency of the schedule is larger than 2/3, and

$F_i = R_i - E_i$. Since R_i consists of small jobs, it is easy to see that E_i precedes F_i. Note that $|R_i| = 1$, for $1 \leq i \leq N$.

Notice again that the efficiency of a wall is larger than 1/3.

Case 1. $N = 1$. The length of time interval with efficiency less than 1/3 is at most $|R_1|$. Since $C^* \geq 1$, the competitive ratio is at most 4 by Lemma 4.2.1.

Case 2. $N = 2$. Let x be the total length of walls. Thus $C^* \geq x/2$, since no three big jobs can be processed at the same time. On the other hand, we have $C^* \geq 1$ since there exists $p_k = 1$. In this case, $C_{FR} = x + 2$. If $x \leq 2$, we have $C_{FR}/C^* \leq (x+2)/1 \leq 4$. If $x > 2$, then $C_{FR}/C^* \leq (x+2)/(x/2) = 2 + 4/x \leq 4$.

Case 3. $N \geq 3$. Note that if $|E_j| \geq 1/2$, then the average efficiency of the time interval R_j is at least 1/3. For any $1 \leq j < N-1$, we have $|E_j| \neq \emptyset$, otherwise the jobs in room R_N will be scheduled to R_j. For any pair of indices j and $j+1$, the minimum length of jobs in R_{j+1} is at least $|F_j|$, otherwise this job can be assigned to room R_j. Thus we have $|E_{j+1}| > |F_j|$ for any $1 \leq j \leq N-2$, which implies that the average efficiency of these two rooms is at least 1/3. Hence the lemma follows if N is an odd number. Now we assume that $N = 2q$, where $q \geq 2$. Find the first j such that $|E_j| < 1/2$, $1 \leq j \leq N-1$. If no such j exists, the lemma follows immediately. In the following we will show that at most one such time interval exists. Since j is the first time interval such that $|F_j| > 1/2$, then for any job scheduled in room p, where $j < p \leq N-1$, its length is larger than $|F_j|$, which implies that $|E_p| > |F_j| > 1/2$. At the same time, the average efficiency of the two time intervals R_j and R_{j+1} or R_j and R_{j-1} is at least 1/3. Therefore the average efficiency of the time intervals except room R_N is at least 1/3. Then the theorem follows by Lemma 4.2.1. □

Theorem 4.3.5 The competitive ratio of algorithm FR is at least 4. Thus the bound for algorithm FR is tight.

Proof. Consider the following instance. Let ε be a sufficiently small positive number. The first two jobs J_1 and J_2 are of $(\lfloor m/3 \rfloor + 1, 1)$. The next 4 jobs $J_3, \ldots, J_6$ are small. The total size of them is m and each has a processing time of ε. Job J_7 is (m, ε) and the last job J_8 is $(1, 1)$.

By the algorithm, we schedule the first two jobs J_1, J_2 individually, which are

followed by $J_3, \ldots, J_6$. Jobs J_7 is scheduled at time 3. Since the last job can not be assigned to room R_1, J_8 is scheduled at time $3+\varepsilon$. Thus, $C_{FR} = 4+\varepsilon$ (see Fig. 4.7).

In an optimal schedule, we can assign J_1, J_2, J_8 at time zero and the small jobs $J_3, \ldots, J_6$ at time 1. Thus, $C^* = 1 + \varepsilon$ (see Fig. 4.8).

$$C_{FR}/C^* \geq \frac{4+\varepsilon}{1+\varepsilon} \to 4.$$

□

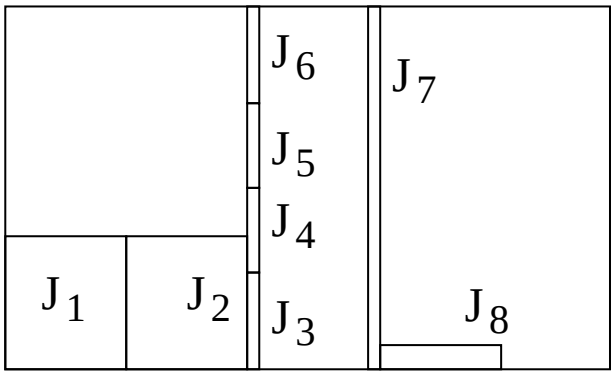

Figure 4.7: The schedule by algorithm FR

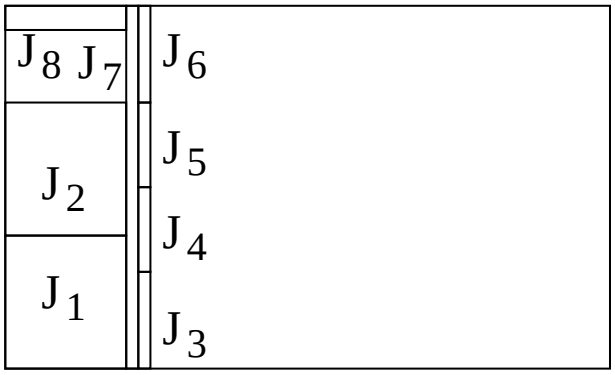

Figure 4.8: An optimal schedule

4.4 Preemptive algorithm on PRAMs and lines

In this section, we will give an algorithm for the preemptive online scheduling of parallel jobs problem on PRAMs. Because we always assign jobs on consecutive machines, the algorithm also valid for the line topology.

Mathematically, the system can be viewed as a 2-dimensional bin with width m and infinity height, where the width (vertical dimension) corresponds to the machines and the height (horizontal dimension) corresponds to a time axis.

Algorithm FFP (First Fit with Preemption)

1. Schedule the first job J_1 at time $t = 0$ to the leftmost on the top.

2. **Scan and Partition:** Scan the current schedule from the left to right, make a snapshot at each time point at which some job is completed. Denote them by $t_1, t_2, \ldots, t_k$ respectively. Then partition the schedule into $k+1$ blocks $B(j)$, $j = \{1, \ldots, k+1\}$, where

   - $B(1) = (0, t_1]$.
   - $B(i) = (t_{i-1}, t_i]$, $1 < i \leq k$.
   - $B(k+1) = (t_k, \infty)$.

3. **First Fit:** For each incoming job J_i of (s_i, p_i), assign it to the available block $B(j)$ of lowest index to the leftmost on the top if there is enough free space and $p_i \leq |B(j)| = t_j - t_{j-1}$. If the processing time p_i is greater than the length of this block, then we do preemption and assign this job with length $|B(j)|$ to the leftmost on the top in $B(j)$. Consider the next blocks and assign the preempted job recursively until it is completely scheduled.

An Example: Let $m = 7$ and the list of jobs be $J = \{J_1, \ldots, J_5\}$, where $J_1 = (1, 1)$, $J_2 = (2, 2)$, $J_3 = (3, 0.5)$, $J_4 = (2, 1)$, $J_5 = (2, 2)$. The schedule by algorithm FFP is shown in Fig. 4.9.

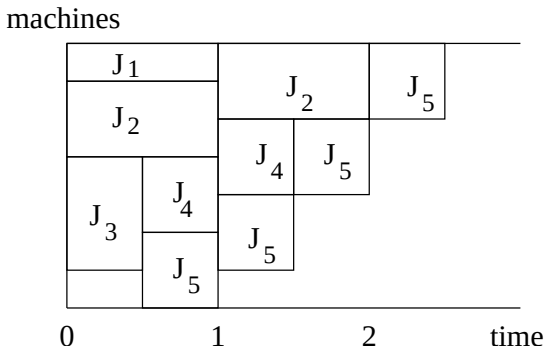

Figure 4.9: An illustration of algorithm FFP

Theorem 4.4.1 The competitive ratio of algorithm FFP is $2 - 1/m$. Moreover, the number of preemptions is $O(n^2)$.

Proof. The lower bound of the algorithm is obvious, since FFP is equivalent to the list scheduling algorithm [72] if all job sizes are one.

Now we prove the upper bound. Denote ALG and OPT to be the makespan produced by algorithm FFP and the makespan given by an optimal off-line algorithm, respectively. Let $s \geq 1$ be the minimum number of busy machines during the whole schedule. If $s \geq (m+1)/2$, then the upper bound $2 - 1/m$ follows. In the following, we assume that $s < (m+1)/2$. Then $2s < m+1$, i.e., $2s \leq m$. It implies that two jobs with size of at most s can be processed simultaneously. Let I_s be the time interval during which there are s busy machines. Let J_k be the job with an earliest starting time among all jobs that are fully or partly executed in I_s. The staring time and completion time of J_k are denoted by t_1 and t_2, respectively. Clearly $I_s \subseteq [t_1, t_2]$. Note that the size of J_k is at most s and thus not larger than $m-s$. Then the number of busy machines is always at least $m-s+1$ before time t_1. Any job starting after I_s must have a size larger than $m - s$. More precisely at any time when J_k is not processed, the number of busy machines is at least $m - s + 1$. The length of the schedule, during which the number of busy machines is at least $m - s + 1$, is $ALG - p_k$.

Clearly, $OPT \geq p_k$. On the other hand, OPT is not less than the total work divided by m. That is

$$OPT \geq \frac{m-s+1}{m}(ALG - p_k) + \frac{s}{m}p_k.$$

If $p_k \geq \frac{m}{2m-1}ALG$, we have

$$OPT \geq p_k \geq \frac{m}{2m-1}ALG,$$

then

$$R_{FFP} = \frac{ALG}{OPT} \leq 2 - 1/m.$$

If $p_k \leq \frac{m}{2m-1}ALG$, it is easy to verify that the minimum value of $\frac{m-s+1}{m}(ALG - p_k) + \frac{s}{m}p_k$ is reached when $s = 1$ and $p_k = \frac{m}{2m-1}ALG$. Thus the competitive ratio is at most $2 - 1/m$.

Finally we show that the number of preemptions is $O(n^2)$. For each job, we need to scan the current schedule and make a snapshot. However, the snapshot made only when some job is completed. Then $k \leq n$ holds in the second step of the algorithm FFP. This implies that there are no more than $O(n^2)$ preemptions during the whole schedule. □

Chapter 5

Scheduling malleable parallel jobs on 2-dimensional meshes

In this chapter we study the malleable parallel jobs scheduling problem. We will present an asymptotic fully polynomial time approximation scheme when the number of machines is sufficiently large.

5.1 Introduction

We study the problem of scheduling a set of independent *malleable parallel jobs* (MPJ) $J = \{J_1, J_2, \ldots, J_n\}$ on a given $m \times m$ 2-dimensional mesh. Each job J_i requires $a_i \times b_i$ submesh to process it simultaneously. The processing time of a job J_i is a function of the set of machines allotted to it, i.e., $p_j : D \to \mathbb{R}^+$, where $D = \{1 \times 1, \ldots, 1 \times m, \ldots, m \times 1, \ldots, m \times m\}$. If $a_i \times b_i$ machines allotted to job J_i, then the processing time is denoted by $p_j(a_j \times b_j)$. The $a_i \times b_i$ machines are required to execute job J_i in union and without preemption, i.e., they have to start processing job J_i at some time τ_i and complete it at time $\tau_i + p_i(a_i \times b_i)$. Our goal is to minimize the makespan, i.e.,

$$\max\{\tau_j + p_j(a_j \times b_j), j = 1, 2, \ldots, n\}.$$

This problem can be denoted by $P|\ 2\text{-}d\ mesh, fctn_j|C_{\max}$.

Assume that the malleable parallel jobs are under the *monotonic* property, which arises from the usual behaviour of parallel programs(cf. Cosnard and Trystram [36]),

and Blayo, Debreu, Mounié, Trystram [17] have shown these assumptions to be realistic in implementing actual parallel applications.

Monotonic malleable jobs:

1. The processing time $p_j(a_j \times b_j)$ of a malleable job J_j is a non-increasing function of the number $s_j \stackrel{\text{def}}{=} a_j * b_j$ of machines executing the job.
2. The work $w_j \stackrel{\text{def}}{=} a_j * b_j * p_j(a_j \times b_j)$ of a malleable job J_j is a non-decreasing function of the number s_j of machines executing the job.

The first monotonic property means that adding more machines for processing a malleable parallel job will not increase the processing time. The second property reflects that the work will not decrease by adding more machines, which may due to the communication between machines.

Known Results For the parallel jobs scheduling problem on PRAM network topology, Jansen and Porkolab [85] studied the case when there is only a constant number of machines and presented polynomial-time approximation schemes (PTAS) for both malleable and non-malleable parallel jobs which run in $O(n)$ time. For the case m is not fixed, Jansen [83] investigated the problem $P|fctn_j|C_{\max}$ on PRAM, Line and Hypercube and presented an asymptotic fully polynomial-time scheme (AFPTAS) for the malleable parallel jobs. Kenyon and Remila [94] provided an asymptotic fully polynomial-time scheme for the strip packing problem under the assumption that the each rectangle has height at most 1.

To our best knowledge, the best currently absolute approximation ratio is $3/2+\varepsilon$ (for any fixed $\varepsilon > 0$) for the malleable parallel jobs scheduling with the monotony assumption [107]. The best known absolute approximation ratios are 2 for the problem of scheduling non-malleable parallel jobs on the network topologies PRAM, Line, and Hypercube [123,116,30].

Lepère, Trystram and Woeginger [103] studied the malleable parallel jobs under precedence constraints. A polynomial time approximation algorithm with worst-case ratio close to $(3+\sqrt{5})/2$ for the special case of series parallel precedence constraints

and for the special case of precedence constraints of bounded width. For the general case, the worst-case ratio is $(3+\sqrt{5}) \approx 5.236$. Jansen and Zhang in [88,89,137] proved the general case with ratio to at most 4.73. They also provided a 3.29-approximation algorithm when the work function is convex in processing time, or when the speedup function (the inverse of processing time) is concave in number of machines.

In the following, we will study the problem when there is a large resource available, i.e., m is sufficiently large. Furthermore, we assume that the largest processing time of any job is bounded from above by 1.

5.2 Preemptive schedules

In this section we study the preemptive version of the problem on 2-dimensional meshes. Each job can be preempted at any time and restarted later possibly on a different set of machines, i.e., the migration is allowed, which is that each job may be assigned to different machine sets during different execution phases.

Given any schedule of the parallel jobs, if we snapshot at instant t, we see a certain set of jobs, which are all being processed at t. Then we consider the configurations, which correspond to collections of the jobs which can be processed simultaneously on the specific network topology at time t. With a linear program similar as [83], our preemptive malleable scheduling problem which denoted by $P|$ *2-d mesh*, $fctn_j$, $pmtn|C_{\max}$ can be formulated as follows.

$$\begin{array}{ll} \min \sum_{f\in F} x_f & \\ \textit{subject to} & \sum_{s_k\in D} \frac{1}{p_j(s_k)} \sum_{f\in F:|f^{-1}(j)|=s_k} x_f \geq 1,\ j=1,\ldots,n \\ x_f \geq 0, & \forall f \in F. \end{array} \tag{5.1}$$

The set F denotes the set of all configurations. The variable x_f denotes the length of configuration f in the schedule and $|f^{-1}(j)|$ is the set of machines allotted to job J_j in the configuration f.

Denote $Lin_M(I)$ to be the optimum value of the linear program (5.1). The same

as [83], we can get a preemptive schedule of length at most $(1+\varepsilon)Lin_M(I)$ for any given $\varepsilon > 0$.

5.3 Converting the preemptive schedule

In this section, similar to Jansen [83], we show how to convert a preemptive schedule into a non-preemptive schedule. Firstly, we compute an unique machine allotment for almost all jobs. Then convert the preemptive schedule to a non-preemptive schedule without losing too much of the quality of the schedule.

Let $p_{\max} = \max_j \max_{l \in D} p_j(l)$. For each job $J_j \in J$, let $x_{j,l} = \frac{1}{p_j(l)} \sum_{f \in :|f^{-1}(j)|=l} x_f \in [0,1]$. Let L_{big} be the set of job J_k with the number of required machines $s_k \geq \varepsilon' m^2$ and the remaining jobs consist L_{small}, where ε' will be specified late. Then we construct an instance of strip packing problem with rectangles $(x_{j,l}p_j(\lfloor l/m \rfloor), \lfloor l/m \rfloor)$ for the set L_{big}. We put the rectangles of L_{big} on a left-justified stack by order of non-increasing width. Let $H := H(L_{big})$ be the height of the stack. Consider the horizontal lines $y = iH/M$ where M is a later specified constant and $i \in \{1, \ldots, M-1\}$. Let a_i and b_i be the smallest and largest width of group i, respectively. Let $W_{j,i}$ be the set of widths in group i corresponding to task J_j. Compute the total volume of the set L_{small}, $Volume_{small} = \sum_{j=1}^{n} \sum_{l \in D, l \leq \varepsilon' m^2} x_{j,l} p_j(l) l$. We put the jobs in L_{small} in group 0. For each group $i \in \{0, 1, \ldots, M\}$ and job J_j let $z_{j,i} = \sum_{w: w \in W_{j,i}} y_{j,i}(w)$ be the fraction of job J_j executed in group i, where $y_{j,i}$ is the fraction of job J_j with width w executed in group i.

Thus, we do the following steps to get unique machine numbers for nearly all the jobs.

1. For each group $i \in \{1, 2, \ldots, M\}$ and job J_j with at least two widths in group i: Let $l_{j,i}$ be the smallest number of required machines among them. Then replace the rectangles corresponding to job J_i in group i by $(z_{j,i}p_j(l_{j,i}), l_{j,i})$.

2. For each job J_j with at least two widths in group 0: replace all rectangles correspond to to job J_j in group 0 by $(z_{j,0}p_j(1), 1)$.

3. Round all jobs over the groups using a general assignment problem:

$$\begin{aligned}
&\textstyle\sum_{j=1}^{n} z_{j,0} p_j(1) \le Volume_{small} && \\
&\textstyle\sum_{j=1}^{n} z_{j,i} p_j(l_{j,i}) \le H/M && i = 1, \ldots, M \\
&\textstyle\sum_{i=0}^{M} z_{j,i} = 1 && j = 1, \ldots, n \\
&z_{j,i} \ge 0 && j = 1, \ldots, n,\ i = 1, \ldots, M
\end{aligned}$$

With same analysis as in [83], one can round the variables $z_{j,i}$ so that there are at most M fractional variables. Then we obtain a rectangle packing instance with a set $L'_{big} = \{(p_j(l_{j,i}), l_{j,i}) | z'_{j,i} = 1, i > 0\}$ of big jobs and a set of $L'_{small} = \{(p_j(1), 1) | z'_{j,i} = 1, i > 0\}$.

We construct two sets of instance $inf(L)$ and $sup(L)$ for a given set L of rectangles [94], which rounding each rectangle in group i up to b_i generates $sup(L)$ and down to b_{i+1} generates $inf(L)$. Thus we adopt the fractional strip packing algorithm as in [83] for the set $sup(L'_{big})$, which has at most M distinct widths. The fractional strip packing can be solved approximately within ratio $(1 + \delta)$ for any given $\delta > 0$. Moreover, we put the big rectangles into the space generated by the non-zero basic variables. For the small rectangles, we just use a modified next fit decreasing height (MNFDH) [94] to place it and schedule the left jobs at the end of the generated schedule one after one.

Finally, the detailed algorithm for our problem works as follows.

1. Solve the linear program (5.1) approximately with the ratio $(1 + \varepsilon)$.

2. Set $M = (1/\varepsilon')^2$, $\varepsilon' = \delta/(\delta + 2)$, $\delta \le \min(1, \varepsilon/8)$.

3. Compute the value $x_{j,l} = \frac{1}{p_j(l)} \sum_{f \in : |f^{-1}(j)| = l} x_f \in [0, 1]$ for easy job j and $l \in D$.

4. Construct an instance of strip packing with rectangles $(x_{j,l} p_j(\lfloor l/m \rfloor), \lfloor l/m \rfloor)$ for $x_{j,l} > 0$, and let L_{big} be the set of job J_k with the number of required machines $s_k \ge \varepsilon' m^2$ and the left jobs consist L_{small}.

5. Use the rounding technique to obtain a strip packing instance $L'_{big} = \{(p_j(l_{j,i}), l_{j,i}) | z'_{j,i} = 1, i > 0\}$, $L'_{small} = \{(p_j(1), 1) | z'_{j,i} = 1, i > 0\}$ and set $\mathcal{F} = \{J_j | z_{j,i} \in (0, 1)\ for\ at\ least\ one\ i \in M\}$.

6. Construct instance $sup(L'_{big})$ with a constant number M of distinct widths.

7. Solve the fractional strip packing problem for $sup(L'_{big})$ with ratio $(1+\delta)$ and round the solution to obtain only M non-zero variable x_j.

8. Place the width rectangles of L'_{big} into the space generated by the non-zero variables x_j.

9. Insert the narrow rectangles of L'_{small} using modified next fit decreasing height.

10. Schedule the jobs in $\mathcal{F}$ at the end.

5.4 Analysis of the algorithm

We prove the correctness of our algorithm and the bound $1+\varepsilon$. It is worth noticing that the crucial difference between our problem and the problem on PRAM network [83] is the way to convert the preemptive schedule into a non-preemptive schedule. Actually, the difference lies in the first step of converting preemptive schedule.

In the following, we show that the method we have proposed is still valid for the converting step. Recall that all the malleable jobs are under the monotonic assumption. We now consider the case that job J_k have two different allotments, $a_i \times b_i$ and $a_j \times b_j$. If the number $s_i = a_i * b_i$ is equal to the number $s_j = a_j * b_j$, i.e., $s_i = s_j$. We have $p_k(a_i \times b_i)a_i * b_i \leq p_k(a_j \times b_j)a_j * b_j$ and $p_k(a_i \times b_i)a_i * b_i \leq p_k(a_j \times b_j)a_j * b_j$, thus $p_k(a_i \times b_i) = p_k(a_j \times b_j)$. This implies that the processing time of job J_k only depends on the number of machines. Therefore, we can change the shapes of rectangle which job J_k is allotted in any form when the area of the rectangle is kept.

For any $s > \varepsilon m^2$ and $m \geq \frac{1+\varepsilon}{\varepsilon^2}$, we have $\frac{s}{s-m} \leq 1+\varepsilon$. Thus, for any job J_k with required number of machines $s_k \geq \varepsilon' m^2$, i.e., the big job. We change their shape into the form $m \times \lfloor s_k/m \rfloor$. Therefore, the processing time of job J_k may increase. However, the extension of processing time of job J_k is bounded by ratio $1+\varepsilon'$, since the number of machines assigned to job J_k shrank at most m, which implies that the extension of the processing time is within the factor $\frac{s}{s-m} \leq$

$1 + \varepsilon'$, due to the monotonic assumption. Thus, our problem can be regarded as the problem scheduling on the PRAM network. However the extension of the total processing time is not larger than the ratio $1 + \varepsilon'$, since for such a problem, Jansen [83] have proposed a nice algorithm that converts the preemptive schedule into a non-preemptive schedule with at most $1 + \varepsilon$ factor of the optimum. Note that, the running time of our algorithm is the same as the one in [83] for PRAMs. Thus,we obtain the following theorem.

Theorem 5.4.1 For any given $\varepsilon > 0$, there exists an asymptotic fully polynomial-time scheme (AFPTAS) for the problem $P|\, fctn_j|C_{\max}$ on an $m \times m$ 2-dimensional mesh with monotonic parallel jobs, where the job processing times are at most 1 and $m = \Omega(1/\varepsilon^2)$. The running time of this algorithm is $O(n^2(\ln n + \varepsilon^{-2}) \ln(n\varepsilon^{-1}) + \varepsilon^{-4} \max\{M(\varepsilon^{-2}), \varepsilon^{-3} \ln(\varepsilon^{-1})\})$.

Chapter 6

Maximizing the throughput

In this chapter we consider off-line problems by dealing with the parallel jobs scheduling where its objective is to maximize the throughput. In Section 6.1, we study the problem of scheduling parallel jobs on hypercubes. In Section 6.2, we study the problem of scheduling parallel jobs with distinct release dates and due dates on two identical machines.

6.1 Scheduling parallel jobs on hypercubes

In this section we address the scheduling problem on hypercubes. The problem can be described as follows. A set $J = \{J_1, J_2, \cdots, J_n\}$ of n jobs and a *m-dimensional* hypercube. Each job J_i has a unit processing time $p_i = 1$, an integral due date d_i and a *size* (the number of requested machines) s_i of *x-dimensional* hypercube, where $0 \leq x \leq m$. A *m-dimensional* hypercube has $N = 2^m$ machines where two machines are directly connected if their binary representations differ by exactly one bit. Available job-types include all *x-dimensional* subcubes for $x \leq m$. Here we assume that all jobs are available at time zero. The objective is to maximize the *throughput* $\sum \bar{U}_i$, where $\bar{U}_i = 1$ if job J_i is completed before or at time d_i, and $\bar{U}_i = 0$ otherwise. A job J_i is called *early* if it meets its due dates d_i ($\bar{U}_i = 1$), and *lost* ($\bar{U}_i = 0$) otherwise. A *lost* job will not be scheduled. Use the standard notation scheme by Graham et al. [74], our problem is denoted as $P|cube_i, p_i = 1| \sum \bar{U}_i$.

Note that for the throughput objective the parallel jobs scheduling problems

have been studied if the network topology is a *PRAM*, *dedicated*, or *2-dimensional meshes*. In this chapter, we concentrate on the network topology of *hypercube*. It is easy to show that $P|cube_i|\sum \bar{U}_i$ is NP-hard, even for the case that there are only two machines and all jobs have a common due date. Therefore, it is significant to study the UET (unit execute time) job system that $p_i = 1$. We present an optimal algorithm for the problem $P|cube_i, p_i = 1|\sum \bar{U}_i$, which runs in time of $O(n^2 \log n)$.

An instance I of our problem consists of a list of parallel jobs $J = \{J_1, J_2, \cdots, J_n\}$ and an m-dimensional hypercube. We use (s_i, d_i) to denote job J_i. Our goal is to schedule all the jobs on the hypercube network to maximize the throughput, i.e., the number of *early* jobs.

An x-dimensional subcube is *normal* if the binary representations of all machines only differ in last x bits, which implies that if two normal subcubes intersect then one of them is a subcube of another. To avoid fragmentation, we only use *normal* subcubes.

The problem of scheduling a set of available jobs at each unit time interval can be regarded as a one dimensional bin packing problem, where jobs (with unit processing time) as items, the number of requested machines as sizes of items, the unit time slot as a bin. Note that all the job sizes and the capacity of the bin are power of 2.

Algorithm *MDS*

1. Sort the jobs in nonincreasing order of due dates (ties broken in favor of the smallest size). Without causing any confusion, we still denote by $J = \{J_1, J_2, \cdots, J_n\}$ the list of jobs after sorting. Then $d_i \leq d_j$ if $i \leq j$, and $s_i \leq s_j$ if $d_i = d_j$ and $i \leq j$.

2. Let $L = \emptyset$ and denote by *sch* the current schedule.

3. Pick the jobs one by one from the ordered list J and schedule them as early as possible. If it fails to schedule job J_i, i.e., it can not be scheduled to complete by its due date d_i, we do following rearrangement. Denote by J_k the job with the largest size in the current schedule *sch*.

   - If $s_i \geq s_k$, then job J_i is called a *rejected job* and is dropped to set L. $L = L \bigcup \{J_i\}$.

- Otherwise, we replace job J_k with job J_i from the schedule. Then job J_k goes to the set L and $L = L \bigcup \{J_k\}$. J_k is called a *removed job* and J_i is called a *replacing job*, respectively. Moreover, we re-schedule jobs in *sch* which start after J_i in the following manner: assign them as early as possible under the same order in J.

Remark. During the step of rescheduling jobs in algorithm MDS, there are no more *lost* jobs produced. The *lost* jobs in L will not be processed.

Note that scheduling jobs as early as possible can be viewed as the FF(*First Fit* [91]) bin packing algorithm if all job processing times are equal to 1.

Let $I_1, I_2, \cdots, I_z$ be the unit time slots in the schedule generated by the algorithm MDS, during which there are still idle room (some machines are idle). Denote by $Idle(I_t)$ the total idle room in I_t, for $t = 1, 2, \ldots, z$. Without loss of generality, we assume that $1 \leq z \leq d_n$, where d_n is the largest due date.

Lemma 6.1.1 For any $1 \leq t \leq z$, if 2^k is the size of the smallest job in the time slot I_t, then $Idle(I_t) \geq 2^k$.

Proof. Besides the job with the smallest size, assume the other jobs in time slot I_t to be $J_{p_1}, J_{p_2}, \cdots, J_{p_q}$ with sizes $2^{k_1}, 2^{k_2}, \cdots, 2^{k_q}$, respectively. We know $k_i \geq k$, $1 \leq i \leq q$. Then, $\sum_{i=1}^{q} 2^{k_i}$ can be divided by 2^k. From $Idle(I_t) = 2^m - 2^k - \sum_{i=1}^{q} 2^{k_i}$, we know $Idle(I_t)$ must be divided by 2^k, which implies the lemma. □

Lemma 6.1.2 For any $1 \leq i < j \leq z$, let job J_p (s_p, d_p) and job J_q (s_q, d_q) to be the jobs with smallest size in time slot I_i and I_j, respectively. Then we have $s_p < s_q$.

Proof. Since $j > i$, by the algorithm MDS, job J_q can not be assigned to time slot I_i, thus, $s_q > Idle(I_i)$. By Lemma 6.1.1, we have $Idle(I_i) \geq s_p$. Then $s_p < s_q$. □

Lemma 6.1.3 For any $1 \leq t \leq z$, if job J_p with size 2^k can not be assigned to time slot I_t, in which the smallest job size is 2^j, then $Idle(I_t) \leq 2^j + 2^{j+1} + \cdots + 2^{k-1}$.

Proof. Clearly $Idle(I_t) < 2^k$. To prove the lemma, we do the following things. Combine the jobs in I_t with same size, e.g., if there are two jobs with same size 2^q, then we regard them as one job with size 2^{q+1}. Repeat these procedures until

all jobs in time slot I_t have distinct sizes, which are assumed to be $2^{j_1}, 2^{j_2}, \cdots, 2^{j_h}$, $j_1 < j_2 < \cdots < j_h \leq m-1$. Since the smallest job size in I_t is 2^j, we have $j \leq j_1$. Note that

$$2^m = 2^j + \sum_{i=j}^{m-1} 2^i$$

and the number of busy machines in I_t is $\sum_{i=1}^{h} 2^{j_i}$. From $j_1 \geq j$ and $j_i > j_{i-1}$, $2 \leq i \leq h$, we have

$$Idle(I_t) = 2^m - \sum_{i=1}^{h} 2^{j_i} = 2^j + \sum_{i=1}^{c} 2^{j'_i},\ where\ j \leq j'_1 < j'_2 < \cdots < j'_c \leq k-1.$$

Notice that $j'_1, \cdots, j'_c$ will not be continuous if $j'_1 = j$, otherwise, we will have $Idle(I_t) \geq 2^k$, which is a contradiction. Then we obtain that $Idle(I_t) \leq 2^j + 2^{j+1} + \cdots + 2^{k-1}$. □

Lemma 6.1.4 Let d_x be the end time point of interval I_t for $1 \leq t \leq z$. If job $J_f(s_f, d_f)$ has size of $s_f > Idle(I_t)$ and due date of $d_f \geq d_x$, then $\sum_{p=1}^{t} Idle(I_p) < s_f$, i.e., the total idle room of the schedule before d_x is smaller than s_f.

Proof. Let $s_f = 2^k$. Assume that 2^{j_i} to be sizes of the jobs with smallest size in time slot I_i, respectively, $1 \leq i \leq t$. By Lemma 6.1.2, we have $0 \leq j_1 < j_2 < \cdots < j_t$. Since jobs in time slot I_p can not be assigned to time slot I_{p-1}, $2 \leq p \leq t$, and by Lemma 6.1.3, we get $Idle(I_t) \leq \sum_{p=j_t}^{k-1} 2^p$ and $Idle(I_q) \leq \sum_{p=j_q}^{j_{q+1}-1} 2^p$, $1 \leq q \leq t-1$. Then $\sum_{p=1}^{t} Idle(I_t) \leq \sum_{p=j_1}^{k-1} 2^p \leq \sum_{p=0}^{k-1} 2^p$. Recall that $2^k = 1 + \sum_{p=0}^{k-1} 2^p$. Therefore $\sum_{p=1}^{t} Idle(I_p) < 2^k = s_f$. □

Lemma 6.1.5 If a removed job J_p (s_p, d_p) in the lost set is replaced by an early job J_q (s_q, d_q), then no job in the interval $(0, d_q]$ of the schedule *sch* has size larger than s_p and the total idle room in the interval $(0, d_q]$ of the schedule *sch* is less than s_p.

Proof. Since J_q is a replacing job, then s_p is no less than the largest size of early jobs in the interval $(0, d_q]$ by the algorithm. Note that J_q need to replace J_p, by the Lemma 6.1.4, we have before this replacement the total idle room x in interval $(0, d_q]$ is less than s_q. Then after the replacement, the total idle room increase at most $s_p - s_q$, which implies that the total idle room after the replacement is at most

$s_p - s_q + x < s_p$. If there is no replacement occurs before d_q, then idle room in interval $(0, d_q]$ will not increase. If there is a replacements happens after job J_q in the interval $(0, d_q]$, we assume that the replacing job is J_r (s_r, d_r) and its removed job is J'_r (s'_r, d'_r), here $d_r \geq d_q$, $d'_r \leq d_q$. Note that $s'_r \leq s_p$ and the total room in interval $(0, d_r]$ at this time is less than s'_r. It implies that whether the replacement happens or not after J_q, the total idle room in the interval $(0, d_q]$ of schedule *sch* is bounded by s_p. Then the lemma follows. □

Theorem 6.1.6 Algorithm MDS produces an optimal solution for the problem $P|cube_i, p_i = 1|\sum \bar{U}_i$ in $O(n^2 \log n)$ time.

Proof. Let *sch* be the final schedule produced by algorithm MDS. Assume $\bar{d}_1, \ldots, \bar{d}_h$ to be the distinct due dates of *lost* jobs. A replacing job J_i is characterized by (t_i, s_i, d_i, J'_i), where t_i denotes the start time of job J_i in the schedule *sch*, s_i and d_i denote the size and the due date of the job, respectively, J'_i denotes the job which is removed by job J_i. Now we partition the schedule *sch* into blocks $B(1), \ldots, B(l)$ as follows:

- The first block $B(1) = (0, d^*_1]$. Find the job J_i with largest due date of replacing jobs in time interval $(0, \bar{d}_1]$. If there are no replacing jobs at all, then $d^*_1 = \bar{d}_1$. Otherwise, find d^*_1 that is the smallest number such that all replacing jobs (if any) in $(0, d^*_1]$ have due dates at most d^*_1.

- Each further block $B(k) = (d^*_{k-1}, d^*_k]$. If there is no replacing job with due date between d^*_{k-1} and the smallest due date $\bar{d}_j$ over all the lost jobs whose due dates are greater than d^*_{k-1}, then $d^*_k = \bar{d}_j$. Otherwise, find d^*_k that is the smallest number such that all replacing jobs (if any) in $(d^*_{k-1}, d^*_k]$ have due dates at most d^*_k but larger than d^*_{k-1}.

We will show the following properties for the blocks. For each block $B(k)$, we have no job in interval $(0, d^*_k]$ with size larger than the smallest size of those lost jobs in this block. The total idle space of interval $(0, d^*_k]$ is less than the smallest size of those lost jobs in block $B(k)$. Then the theorem follows.

For each block $B(k)$, we consider the following cases.

Case 1. There is no replacing job in block $B(k)$. Consider the lost jobs $\bar{J}_i$ in this block, then $\bar{d}_i = d_k^*$ by the rule of construction of block. Then there is no job in interval $(0, d_k^*]$ with size larger than $\bar{s}_i$, otherwise, job $\bar{J}_i$ can replace the job with larger size and shall have been accepted by the schedule *sch*. By Lemma 6.1.4, we have the total idle of interval $(0, d_k^*]$ in schedule *sch* is less than $\bar{s}_i$.

Case 2. There is at least one replacing job in block $B(k)$. Assume $\bar{J}_{k_1}, \ldots, \bar{J}_{k_p}$ to be the rejected jobs in this block in the order of non-decreasing of due dates. Let jobs $J_{k_1}, \ldots, J_{k_q}$ be the replacing job in the order of non-decreasing of due dates and jobs $J'_{k_1}, \ldots, J'_{k_q}$ to be the removed job in the lost set in this block accordingly. Then $s_{k_q} = d_k^*$. We have s'_{k_q} is at least the largest size of those early jobs in this block and the total idle room of this block is at most s'_{k_q} by Lemma 6.1.5. In the following we will show that the size of lost jobs in this block is at least s'_{k_q}. For any removed job J'_{k_j}, $1 \leq j \leq q$, there is at least one replacing job J_{k_w} between t_{k_j} and d_{k_j}, then s'_{k_j} is larger than s'_{k_w}. Continue this analysis, we find that s'_{k_q} is the smallest number. Therefore we know all the removed jobs have size at least s'_{k_q}. For those rejected jobs $\bar{J}_{k_j}$, there is at least a replacing job J_w with $t_w < \bar{d}_{k_j}$, then $\bar{s}_{k_j} \geq s'_{k_j}$ by the algorithm. Hence all the rejected jobs have size at least s'_{k_q}.

In the following, we estimate the running time of algorithm MDS. Recall that scheduling jobs as early as possible can be viewed as the *First Fit* algorithm and the procedure of reassigning jobs can also be viewed as *First Fit* algorithm. Each jobs may need a procedure to reassign jobs and the running time of *First Fit* algorithm is bounded by $O(n \log n)$ [91]. Then algorithm MDS runs in time of $O(n^2 \log n)$. □

Corollary 6.1.7 The common due date version of the problem $P|cube_i, p_i = 1, d_i = D|\sum \bar{U}_i$ can be solved in $O(n \log n)$ time.

Proof. We only need to show that running time of algorithm MDS for this special case. Because for the common due date version of the problem, the rescheduling job procedure is not necessary. The running time is determined by *First Fit* algorithm and sorting jobs. Both of them take at most $O(n \log n)$ time. Hence the running time of algorithm MDS is $O(n \log n)$ time. □

6.2 Scheduling parallel jobs on two identical machines

In this section we study the problem of maximizing the throughput of parallel jobs on two parallel identical machines. A set of unit processing time jobs $J = \{J_1, J_2, \ldots, J_n\}$ are required to be scheduled on two parallel identical machines. Each job requires one machine or two machines at the same time. Each job J_j is associated with release date r_j and due date d_j. The release date r_j is assumed to be an integer. A job is called *small* if it requests only one machine for its processing, otherwise, it is called *big*. The objective is to maximize the *throughput* $\sum \bar{U}_i$. A job J_i is called *early* if it meets its due date d_i ($\bar{U}_i = 1$), and *lost* ($\bar{U}_i = 0$) otherwise. A *lost* job will not be scheduled. The complexity of this problem was open (see the web maintained by Brucker et al. [24]). In the following, we will present an optimal algorithm with running time of $O(n^2)$.

Known results $P2|p_j = 1, size_j|\sum \bar{U}_j$ was solved in $O(n \log n)$ time by Brucker et al. [26]. In the same paper, the more general problem $Pm|p_j = p, size_j|\sum w_j\bar{U}_j$ was shown to be a polynomial problem, while the problem $P|p_j = 1, size_j|\sum \bar{U}_j$ was shown to be strongly NP-hard by Fishkin and Zhang [63]. If we consider the precedence constraints between jobs, the problem $P2|prec, p_j = 1, size_j|\sum \bar{U}_j$ becomes NP-hard [19].

For single machine model, the problem $1|r_j, p_j = p|\sum w_j\bar{U}_j$ is polynomial time solvable by Baptiste in [11]. Furthermore, for the non-parallel job models, the problem $Pm|r_j, p_j = p|\sum w_j\bar{U}_j$ has been shown to be solved in polynomial time by Bapstiste et al. [13]. $P|p_j = p|\sum w_j\bar{U}_j$ admits polynomially, since it can be regarded as an assignment problem. $P|r_j, p_j = 1|\sum w_j\bar{U}_j$ is also polynomially solvable, since it can be reduced to a network flow problem.

For the other objectives, the problem $P2|r_j, p_j = 1, size_j|C_{\max}$ was shown to be polynomially in [102,19]. The problems $Pm|r_j, p_j = p, size_j|C_{\max}$, $Pm|r_j, p_j = p, size_j|\sum C_j$ and $Pm|p_j = p, size_j|L_{\max}$ are easy problems too [12].

6.2.1 Description of the algorithm

Our algorithm works as follows. The basic idea is to schedule small jobs first and then insert big jobs into the schedule.

Algorithm SBR

1. **Schedule small jobs:** Schedule available small jobs as early as possible by EDD (Earliest due date first) rule, in case of ties, take the one with the smallest release time. If it fails, then this small job gets lost (will not be processed).

2. **Schedule big jobs:** Let $\mathcal{BL}$ be the set of big jobs. Let $d_{\max}(\mathcal{BL}) = \max\{d_j | j \in \mathcal{BL}\}$. Let $T = d_{\max}(\mathcal{BL})$.

   (a) If $T = 0$, then go to Step **Relocate**.

   (b) If $T > 0$ and during the time interval $(T-1, T]$ both machines are idle, assign the big job J_k with largest release time among the available jobs to this time slot, if such a job exists. Denote $\mathcal{A}$ to be the set of jobs with due date at least T and the jobs are not available during time slot T. Then $\mathcal{BL} = \mathcal{BL} - \{J_k\} - \mathcal{A}$.

   (c) If $T \leq d_{\max}(\mathcal{BL})$, then let $T = T - 1$, otherwise let $T = d_{\max}(\mathcal{BL})$. Go to the Step **Schedule big jobs** (a).

3. **Relocate:** Denote $\mathcal{SS}$ to be the set of small jobs assigned in the current schedule.

   (a) If $\mathcal{SS} = \emptyset$, the algorithm stops.

   (b) Find job J_k in $\mathcal{SS}$ with largest completion time, then move it to a feasible position as late as possible.Let $\mathcal{SS} = \mathcal{SS} - J_k$. Continue this move until either no small jobs can be moved (in this case, the algorithm stops) or a time slot in which both machines are idle appears. In the latter case, assign the unscheduled big job with largest release time among the available jobs if such a big job exists.

6.2.2 Analysis of the algorithm

We will analyze the algorithm SBR and prove that the output of this algorithm is optimal. The key point is that the number of small jobs accepted by algorithm SBR is the largest one among all algorithms. In an optimal schedule, we assume that it always takes the small job in a certain time slot if either a big job or a small job can be accepted. We first show that there exists an optimal algorithm which accepts the same small jobs as the algorithm SBR does. Then, we prove that the number of big job accepted by algorithm SBR is maximum without dropping any small job accepted in the first step of algorithm SBR. Then it follows that the algorithm SBR is an optimal algorithm.

In the following, when we say a time slot t, it means that the time interval is $(t, t+1]$.

Lemma 6.2.1 At any time t, the schedule produced by algorithm SBR accepts the maximum number of small jobs in interval $[0, t]$.

Proof. Note that the second and the third steps of the algorithm SBR do not add or drop small jobs. Thus, we only need to consider the schedule after the first step of the algorithm SBR. For any lost small job J_k, the due date of any job processed between r_k and d_k in the schedule is at most d_k. If all small jobs processed in $(r_k, d_k]$ have release dates not less than r_k, then no jobs can be scheduled earlier. Once k is accepted, at least one job will get lost. If there is a job J_p with release date $r_p < r_k$, then both machines are busy during $(r_p, r_k]$. Because small jobs are assigned as early as possible, the small jobs assigned in $(r_p, r_k]$ have due dates at most d_p. Continuous the searching, we get an interval $(I_1, d_k]$ in which both machines are busy and all the small jobs have release dates of at least I_1 and due dates of at most d_k. It implies that once J_k is accepted in this schedule, there is at least one small job will get lost. The lemma follows. □

Lemma 6.2.2 There exists an optimal schedule which accepts the same small jobs as the algorithm SBR does.

Proof. For any optimal schedule OPT, we do the following exchanges without

reducing the number of early jobs. Let J_i be a small job which is accepted by algorithm SBR but not in OPT. Assume that job J_i is scheduled at time t_i in the schedule by algorithm SBR. Then we just replace the big job or one of the small jobs in the schedule OPT at time t_i. Note that the number of early jobs does not decrease in the schedule OPT. By Lemma 6.2.1 we know that the algorithm SBR accepts the maximum number of small jobs. So, there exists an optimal schedule that accepts the same small jobs as the algorithm SBR does. □

Lemma 6.2.3 In the final schedule produced by the algorithm SBR, denote by t_i those time slots during which only one machine is busy. If we remove all the small jobs which were moved across t_i in the *Relocate* step, for some i, then no additional small jobs (except those small jobs moved across t_{i+1}) can be accepted in each interval $[t_i, t_{i+1}]$ without dropping a small job.

Proof. Consider the first such interval $(0, t_1]$ in the schedule produced by algorithm SBR. Note that no small jobs scheduled in $(0, t_1]$ can be assigned later than t_1 due to the *Relocate* step in the algorithm SBR. Now we remove all the small jobs moved across time point t_1. If there is no small job scheduled at time t_1 in the first step, then all the remaining small jobs in the schedule should have release times greater than t_1. Thus, no small job still in the schedule could be moved to the interval $(0, t_1]$. It is worthy to notice that if there are no small jobs moved across t_1, then the number of small jobs accepted in this interval is the maximum due to Lemma 6.2.1. If there is a small job J_k assigned in t_1 during the third step of the algorithm (Relocate), i.e., J_k was moved across t_1. Similarly, all the small jobs after t_1 have release times no less than t_1. The remaining case is that there is a small job J_q assigned in t_1 and was reassigned after or at time $t_1 + 1$. Assume that J_q is assigned at time T in the final schedule. Then consider the small job J_w scheduled in the interval $(t_1, T]$. Note that J_k was relocated at T. We see that $d_w \leq T - 1 < d_q$, since J_w is scheduled at $(t_1, T]$ originally. It implies that J_w can not be assigned before t_1. Obviously, the small jobs after T has release time no less than $T + 1$, which shows that no small jobs can be assigned to $(0, t_1]$. So, all the small jobs scheduled after t_1 can not be moved forward to across t_1 without changing the optimal value. Similarly, we can

prove for any other time interval $(t_i, t_{i+1}]$. Thus, the lemma holds. □

Theorem 6.2.4 The problem $P2|r_j, p_j = 1, size_j|\sum \bar{U}_j$ can be solved optimally in $O(n^2)$ time.

Proof. To prove this theorem, by Lemma 6.2.1, we only need to show the algorithm SBR accepts the maximum number of big jobs. Let sch be the final schedule produced by the algorithm SBR. We deal with the following two cases:

Case 1: At some time t, both machines are idle in sch. Obviously any big job J_l which remains in the lost set can not be assigned at this time slot t. It implies that the due date d_l and the release time r_l of the job J_l in the lost set must satisfy either $d_l \leq t$ or $r_l \geq t + 1$. Now assume that an optimal algorithm assigns one big job at time t and accepts a certain big job J_b in the lost set together with all the big jobs accepted in the schedule sch. There exists some big job in the schedule sch, otherwise no big job can be accepted by any algorithm. Assume that a big job J_s is scheduled at time t in the optimal schedule. Without loss of generality, we can assume that J_b is assigned to a position in the optimal schedule, while the position is occupied by J_s in the schedule sch, otherwise, J_b takes the position of J_s after a finite number of exchanges. If J_s is scheduled after time t in the schedule sch. Then $r_b \leq r_s \leq t$ and $d_b \geq t + 1$, it is a contradiction. If J_s is assigned before t in the schedule sch, it is also a contradiction from the third step of the algorithm SBR that J_s can not be schedule at time t.

Case 2: At some time t, at least one job J_k is processed in sch. So, if the schedule sch accepts a big job in the lost set at time t, then J_k must be relocated away from time t. According to Lemma 6.2.3, we know that at least a small job should be rejected in the schedule sch, if we keep job J_k into the schedule sch.

Thus the problem $P2|r_j, p_j = 1, size_j|\sum \bar{U}_j$ can be solved optimally. As for the running time of the algorithm, it takes $O(n \log n)$ time for the first step and $O(n^2)$ time for the other two steps. Hence, the algorithm SBR runs in $O(n^2)$ time. □

Part II.

Scheduling of non-parallel jobs

Chapter 7

On-line scheduling with extendable working time

7.1 Introduction

We consider the following on-line scheduling problem with extendable working time: there are m machines $B_1, B_2, \ldots, B_m$ with original regular working time $b_1, b_2, \ldots, b_m$, respectively. The original regular working time can be extended if needed. The (final) working time of a machine is defined to be the larger value between the original regular working time and the total processing time of the jobs assigned to it. Jobs $\{J_1, J_2, \ldots, J_n\}$ arrive over list, which are not known in advance. When an item J_i arrives it must immediately and irrevocably be assigned to one of the machines and the next job J_{i+1} becomes known only after J_i has been assigned. The processing time of job J_i is p_i. The goal is to assign all the jobs to the machines such that the total working time of the machines is minimized.

The regular working times of the machines can be seen as a given capacity, thus this problem can be regarded as a bin packing problem: the machines correspond to bins and the jobs to items. If the regular working times can be different, the problem is a bin packing problem with unequal bin sizes.

Competitive ratios. An instance of the problem consists of a list of jobs and m machines with original regular working times $b_1, b_2, \ldots, b_m$. Let $\mathcal{B}$ be the collection of

regular working times, i.e., $\mathcal{B} = \{b_1, b_2, \cdots, b_m\}$. For any instance L, let $A_L(m, \mathcal{B})$ and $OPT_L(m, \mathcal{B})$ be the total working time of the machines used by an on-line algorithm A and the total working time of the machines used by an optimal off-line algorithm, respectively. Then the competitive ratio of algorithm A is

$$R_A(m, \mathcal{B}) = \sup_L A_L(m, \mathcal{B})/OPT_L(m, \mathcal{B}).$$

If the regular working times are identical, i.e., $b_1 = b_2 = \cdots = b_m$, the competitive ratio is denoted as $R_A(m)$ (or R_A in short). Further we define the overall competitive ratios as follows.

$$R_A(m, \cdot) = \sup_{\mathcal{B}} R_A(m, \mathcal{B}), \quad R_A = \sup_{m,\mathcal{B}} R_A(m, \mathcal{B}).$$

Known results The special case that all regular working times are equal to one, i.e., the identical machines, has been extensively studied. It was proved that the off-line version of the problem is NP-hard in the strong sense [39]. Actually, it can be easily reduced from the 3-partition problem [67]. Dell'Olmo et al. [39] proved that the worst-case ratio of the LPT algorithm is 13/12. If the number of the bins is fixed, it follows from the results of [128] that there exists a fully polynomial time approximation scheme. If m is not fixed, a polynomial time approximation scheme was provided in [4]. Recently, Coffman and Lueker [34] presented a fully polynomial time asymptotic approximation scheme. They also proved that the problem does not admit a fully polynomial time approximation scheme unless P = NP, when m is not fixed.

The on-line version was studied by Speranza and Tuza [122]. They showed that the competitive ratio of a list scheduling heuristic is 5/4. Then a class of heuristics H_x were presented which depend on a parameter x, $0 < x < 1$, and tend to load partially loaded bins until a limit total load of $1 + x$ is reached on the bin. For any number of bins, the algorithm has an overall competitive ratio bounded above by 1.228. For the lower bound of the on-line problem, they showed that no heuristic can have a competitive ratio smaller than 7/6 by presenting an instance for the case that $m = 2$.

The extensible scheduling problem on machines with unequal regular working

times, i.e. the extensible bin packing problem with unequal bin sizes, was first studied by Dell'Olmo and Speranza [40]. They considered both the off-line case and the on-line case under the assumption that the maximum item size is not larger than the minimum bin size. They showed an upper bound of $4-2\sqrt{2}$ for an LPT algorithm in the off-line case and an upper bound of $5/4$ for a list scheduling algorithm LS in the on-line case. The bound $5/4$ is tight in terms of the overall competitive ratio, i.e., $R_{LS}=5/4$.

A similar problem, called *on-line bin stretching*, was introduced by Azar and Regev [6]. A set of bins of fixed sizes is given. Items arrive on-line while it is known that there exists a valid packing of all items into the given bins. One is asked to pack all items into the bins, which can be overloaded. The goal function is the stretch factor, which is the maximum ratio for any bin between the size to which any bin was stretched (the sum of all items assigned to it) and its original size. Azar and Regev [6] studied the problem of identical bins, i.e. all original bins are of size 1. Epstein [53] considered the two-bin case with unequal bin sizes.

Our results. Firstly, we study the problem on identical machines, i.e. all regular working times are equal to one. We consider the on-line scheduling on a small number of machines. We first prove lower bounds for $m=3$ and $m=4$, respectively. Then we analyze algorithm H_x for $m=2,3$ and 4 and show that the competitive ratios are $7/6$, $11/9$ and $19/16$, respectively. For $m=2$ the algorithm is best possible. Finally, we provide an improved algorithm for $m=3$, whose competitive ratio is $7/6$. It implies that the optimal on-line algorithm for $m=3$ is at least as good as the one for $m=2$ in terms of competitive ratio.

Then we study the problem with unequal regular working times. Firstly we assume, as in [40], that the largest processing time is at most the smallest regular working time, i.e.,

$$\max_{i=1}^{n} p_i \leq \min_{j=1}^{m} b_j. \tag{7.1}$$

We analyze the LS algorithm more carefully and prove the competitive ratios for each m and each collection $\mathcal{B}$ of regular working times. The ratios differ for an even number of machines and an odd number of machines. It also shows that the

competitive ratio of LS for the identical machines is exactly $5/4$ when m is even and $5/4 - 1/(4m^2)$ when m is odd. We then present an improved on-line algorithm for $m = 2$ and $m = 3$. Finally we discuss the problem without the assumption (7.1). A lower bound $6/5$ for the overall competitive ratio is presented. We prove the competitive ratio of LS for the two-machine case and present an improved on-line algorithm.

7.2 On a small number of identical machines

In this section, we consider the extensible scheduling problem on identical machines, i.e. all the regular working time are equal. Without loss of generality, the regular working times are assumed to be 1. For any instance L, let $A(L)$ and $OPT(L)$ be the total working time of the machines in the schedule generated by an on-line algorithm A and the total working time of the machines in the schedule by an optimal off-line algorithm, respectively.

7.2.1 Lower bounds

The total processing time of the jobs assigned to a machine is called the *load* of the machine. If the load of a machine is less than 1, the machine is not completely full. If its load is 1, the machine is full. If the load of a machine exceeds 1, the excess of the machine is defined to be the difference between the load and 1. If the load of a machine is at most 1, the machine has no excess or the excess of the machine is zero.

In [122] a lower bound $7/6$ for any on-line algorithms was given. The worst-case instance was constructed for the case that $m = 2$. In the following we present lower bounds for $m = 3, 4$.

Theorem 7.2.1 No on-line algorithms can have competitive ratios smaller than $(17 + \sqrt{577})/36 > 41/36$ for three machines.

Proof. Let $f = (-19 + \sqrt{577})/12$. Note that $1/3 < f < 1/2$ is the positive root of equation $6x^2 + 19x - 9 = 0$. For any on-line algorithm A, consider the following

instance L. The first two jobs J_1 and J_2 both have processing time f. If algorithm A assigns J_1 and J_2 to different machines, two jobs with processing time 1 are coming. Then $A(L) \geq 3 + f$ and $OPT(L) = 3$. It implies that $R_A \geq 1 + f/3$. If the first two jobs go to the same machine, say Machine 1, the jobs J_3 and J_4 with processing time $(1 + f)/2$ are coming. We consider three cases.

1. Algorithm A schedules J_3 or J_4 on Machine 1. In this case, $A(L) \geq 2 + 2f + (1 + f)/2$ and $OPT(L) = 3$. Hence, $A(L)/OPT(L) \geq (5 + 5f)/6 > 1 + f/3$.

2. Algorithm A schedules J_3 and J_4 together on an empty machine, say Machine 2. In this case, $A(L) = 3 + f$ and $OPT(L) = 3$. Hence, $A(L)/OPT(L) = 1 + f/3$.

3. Algorithm A schedules J_3 and J_4 on different machines (Machine 2 and Machine 3). Then a new job J_5 with processing time 1 is coming. Note that $2f > (1 + f)/2$. Wherever algorithm A assigns J_5, the total excess of the three machines is at least $(1 + f)/2$, i.e., $A(L) \geq 3 + (1 + f)/2$. However, an optimal algorithm assigns J_1 and J_3 on a machine, J_2 and J_4 on a machine, and J_5 on a machine. It follows that $OPT(L) = 2 + 3f$. Therefore, $A(L)/OPT(L) \geq (7 + f)/(4 + 6f) = 1 + f/3$.

Then it is proved that for any on-line algorithm A, the competitive ratio $R_A \geq 1 + f/3 = (17 + \sqrt{577})/36$. □

Theorem 7.2.2 No on-line algorithms can have competitive ratios smaller than 9/8 for four machines.

Proof. For any on-line algorithm A, consider the following instance L. The first four jobs have processing time 1/4. After the assignment of the four jobs, let $l_{\max}$ be the largest load among the machines. Note that $l_{\max}$ is either at most 1/2 or at least 3/4.

1. $l_{\max} \leq 1/2$. In this case, the last three jobs with processing 1 are coming. The total excess of the three machines which accept a job with processing time 1 is at least 1/2. Therefore, $A(L) \geq 4 + 1/2$ and $OPT(L) = 4$. We have $A(L)/OPT(L) \geq 9/8$.

2. $l_{\max} \geq 3/4$. In this case, four jobs with processing time $3/4$ are coming. There must be one machine with excess at least $1/2$. Hence, $A(L) \geq 9/2$. Since $OPT(L) = 4$, we have $A(L)/OPT(L) \geq 9/8$.

The theorem is thus proved. □

If the total processing time of the jobs in an instance L_1 is less than m, the number of machines, some machines are not completely full in the optimal schedule. We can construct an instance L_2 by adding some new jobs to L_1 such that those incompletely full machines become full (the total processing time of jobs assigned is exactly 1). Note that $OPT(L_1) = OPT(L_2)$ and $A(L_1) \leq A(L_2)$ for any on-line algorithm A. Without loss of generality for proving upper bounds for proposed algorithms, throughout this paper, we only consider those instances in which the total processing time of jobs is at least m.

7.2.2 Competitive analysis of algorithm H_x

Speranza and Tuza [122] presented an approximation algorithm, called H_x, which depends on a parameter x, $0 < x < 1$. The algorithm H_x works as follows.

Algorithm H_x:
Assign the incoming job J_i to the machine with largest possible load such that after the assignment of J_i to the machine, the load of the machine is at most $1 + x$. If there does not exist such a machine, the job is assigned to the machine with the smallest load (ties broken in favor of the smallest index of machines).

In the following, we will analyze the algorithm H_x for a small number of machines. We first present a lower bound for algorithm H_x. Then we prove that the competitive ratio of algorithm H_x with some prespecified number x matches the lower bound, and thus the value of x for the specific number of machines is the best.

The two-machine case

We will show that the competitive ratio of H_x for two machines is exactly $7/6$, when $x = 1/3$. Thus, it is a best possible on-line algorithm.

Theorem 7.2.3 The competitive ratio of algorithm H_x for two machines is 7/6 when $x = 1/3$, and it is a best possible algorithm.

Proof. In the schedule produced by algorithm H_x for any instance L, let x_1 and x_2 be the loads of Machines 1 and 2, respectively. If both x_1 and x_2 are at most one, or neither x_1 nor x_2 is less than one, the schedule is optimal. Without loss of generality, we assume that $x_1 > 1$ and $x_2 < 1$. If $x_1 \leq 1 + x = 4/3$, then $H_x(L)/OPT(L) \leq (1 + 4/3)/2 = 7/6$. Assume that $x_1 > 4/3$. Let J_a, with processing time p_a, be the last job assigned to Machine 1. Clearly, $x_1 - p_a \leq x_2$, otherwise J_a would have been scheduled on Machine 2; and $x_1 - p_a + x_2 > 1 + x = 4/3$, otherwise the jobs on two machines would have been assigned together to one machine. We thus get $x_2 \geq 2/3$. Then

$$\begin{aligned} H_x(L)/OPT(L) &\leq (x_1 + 1)/(x_1 + x_2) \\ &\leq (x_1 + 1)/(x_1 + 2/3) \\ &< (4/3 + 1)/2 = 7/6. \end{aligned}$$

Since no heuristic can have competitive ratio smaller than 7/6 for two machines [122], H_x is a best possible on-line algorithm for $m = 2$ by setting $x = 1/3$. □

The three-machine case

We investigate algorithm H_x for the three-machine case.

Lemma 7.2.4 For any parameter $1 > x > 0$, the competitive ratio of algorithm H_x for three machines is at least 11/9.

Proof. If $x \leq 1/3$, consider the following instance L: the job list is $\{J_1, J_2, J_3, J_4, J_5\}$, and the processing times are $1/3, 1/3, 2/3 + 1/N, 2/3 + 1/N, 1$, respectively, where $N > 0$ is a sufficiently large integer. In an optimal schedule, $\{J_1, J_3\}$, $\{J_2, J_4\}$ and $\{J_5\}$ are assigned to a machine, respectively, whereas H_x will put J_1, J_2 and J_5 together into a machine, and assign J_3 and J_4 each to a machine. Thus, $OPT(L) = 3 + 2/N$ and $H_x(L) = 3 + 2/3$, which follows that $H_x(L)/OPT(L) \to 11/9$ as N tends to infinity.

If $x > 1/3$, consider an instance L as follows. Let $N > 0$ be a sufficiently large integer. The first $N - 1$ jobs are short jobs, each with processing time $1/N$,

which are followed by job J_N with processing time x. Clearly, H_x will assign the N jobs together in a machine. If $x \geq 1/2$, the last two jobs J_{N+1} and J_{N+2} have processing times $1 - 1/N$ and $1 - x$, respectively. They are assigned to a machine. Then $H_x(L) = (1 - 1/N + x) + (1 - 1/N + 1 - x) + 1 = 4 - 2/N$. However, jobs J_N and J_{N+2} can be scheduled on a machine; job J_{N+1} occupies a machine; the first $N - 1$ short jobs go to a machine. Clearly $OPT(L) = 3$. As N tends to infinity, $H_x(L)/OPT(L)$ is arbitrarily close to 4/3. If $x < 1/2$, the last two jobs have processing times $1 - 1/N$ and x, respectively, which are assigned by H_x to a machine. Then $H_x(L) = 2x - 2/N + 3$. However, an optimal algorithm can arrange the jobs such that the load of each machine is at most one. $H_x(L)/OPT(L)$ tends to $(2x + 3)/3 > 11/9$ as N is sufficiently large. Therefore, the competitive ratio of H_x is at least 11/9 when $x \leq 1/3$, at least $(2x + 3)/3$ when $1/3 < x < 1/2$, and at least 4/3 when $x \geq 1/2$. □

By setting $x = 1/3$, we will prove that the competitive ratio of H_x is 11/9.

Theorem 7.2.5 The competitive ratio of algorithm H_x is 11/9 for three machines when $x = 1/3$.

Proof. In the schedule produced by algorithm H_x for any instance L, let x_1, x_2 and x_3 be the total processing time of jobs in the three machines, respectively. For $i = 1, 2, 3$, if all $x_i \geq 1$ or all $x_i \leq 1$, the schedule by algorithm H_x is optimal.

Case 1. Two x_i's are less than 1. Without loss of generality, assume that $x_1 > 1$ and $x_2, x_3 < 1$. $H_x(L) = x_1 + 2$ and $OPT(L) \geq x_1 + x_2 + x_3 \geq 3$. If $x_1 \leq 5/3$, then $H_x(L) \leq 5/3 + 2 = 11/3$ and $OPT(L) \geq 3$. It implies that $H_x(L)/OPT(L) \leq 11/9$. Now we assume that $x_1 > 5/3$. By the algorithm, $x_2 + x_3 > 4/3$. Thus we have $H_x(L)/OPT(L) < (x_1 + 2)/(x_1 + 4/3) < 11/9$.

Case 2. Only one x_i is less than 1. Without loss of generality, we assume that $x_3 < 1$. $H_x(L) = x_1 + x_2 + 1$ and $OPT(L) \geq x_1 + x_2 + x_3 \geq 3$. If $x_1 + x_2 \leq 8/3$, $H_x(L)/OPT(L) \leq (8/3 + 1)/3 = 11/9$. Then we can assume that $x_1 + x_2 > 8/3$. If

$x_3 \geq 1/3$,

$$\begin{aligned} H_x(L)/OPT(L) &\leq (x_1 + x_2 + 1)/(x_1 + x_2 + x_3) \\ &\leq (x_1 + x_2 + 1)/(x_1 + x_2 + 1/3) \\ &< (8/3 + 1)/3 = 11/9. \end{aligned}$$

In the following we will prove that $x_3 > 1/3$. Since $x_1 + x_2 > 8/3$, one of x_1 and x_2 is greater than $4/3$. Assume that $x_1 > 4/3$. Before the last job J_b of Machine 1 is assigned, the load of Machine 1 is greater than $1/3$. By the algorithm, $x_3 > 1/3$. Otherwise, the job J_b should have been assigned to Machine 3. □

Combining Lemma 7.2.4 and Theorem 7.2.5, we conclude that $x = 1/3$ is the best choice for algorithm H_x for three machines and the tight bound of H_x is $11/9$.

The four-machine case

Lemma 7.2.6 For any parameter $0 < x < 1$, the competitive ratio of algorithm H_x for four machines is at least $19/16$.

Proof. If $x < 1/2$, consider an instance L, which consists of 7 jobs with processing time $1/4, 1/4, 1/4, 3/4, 3/4, 3/4, 1$, respectively. In the schedule by algorithm H_x, the first three short jobs will be assigned together to a machine, and the three jobs with processing time $3/4$ each goes to a machine. Then the excess of a machine is $3/4$ after the last job is assigned. Therefore, $H_x(L) = 4+3/4 = 19/4$. However, in an optimal schedule the machines have no excess. $OPT(L) = 4$. Thus, $H_x(L)/OPT(L) = 19/16$.

If $x \geq 1/2$, the first $N - 1$ short jobs, each of which has processing time $1/N$, where $N > 0$ is a sufficiently large integer, are followed by a long job with processing time x. H_x will assign the N jobs to one machine. The last two jobs have processing times $1 - 1/N$ and x, respectively. They also go to a machine. Note that there are two empty machines. Then $H_x(L) = 4 + 2x - 2/N$, while $OPT(L) = 4$. The ratio reaches $1 + x/2 \geq 5/4$ as N tends to infinity. □

In the following, we will prove that the competitive ratio of algorithm H_x can reach the lower bound $19/16$ by setting $x = 1/4$.

Theorem 7.2.7 The competitive ratio of algorithm H_x for four machines is 19/16 when $x = 1/4$.

Proof. Similarly as the proof for $m = 2, 3$, in the schedule given by algorithm H_x for any instance L, let x_i be the total processing time of the jobs assigned to Machine i, for $i = 1, 2, 3, 4$. If each x_i is at most one or each x_i is at least one, H_x produces an optimal schedule.

Case 1. Three x_i's are less than 1. Without loss of generality, assume that $x_1 > 1$ and $x_2, x_3, x_4 < 1$. $H_x(L) = x_1+3$ and $OPT(L) \geq x_1+x_2+x_3+x_4 \geq 4$. If $x_1 \leq 7/4$, $H_x(L)/OPT(L) \leq (7/4 + 3)/4 = 19/16$. If $x_1 > 7/4$, let the last job assigned to Machine 1 be J_c with processing time p_c. Obviously, $x_1 - p_c > 3/4$. Moreover, for $i = 2, 3, 4$, $x_1 - p_c \leq x_i$, and then $x_i > 3/4$. We have $x_2 + x_3 + x_4 > 9/4$. Therefore,

$$\begin{aligned} H_x(L)/OPT(L) &< (x_1 + 3)/(x_1 + 9/4) \\ &< (7/4 + 3)/4 = 19/16. \end{aligned}$$

Case 2. Two x_i's are less than 1. Without loss of generality, assume that $x_3, x_4 < 1$. $H_x(L) = x_1 + x_2 + 2$ and $OPT(L) \geq x_1 + x_2 + x_3 + x_4 \geq 4$. If $x_1 + x_2 \leq 11/4$, then $H_x(L)/OPT(L) \leq (11/4 + 2)/4 = 19/16$. Assume that $x_1 + x_2 > 11/4$. Note that $x_3 + x_4 > 5/4$. We have

$$\begin{aligned} H_x(L)/OPT(L) &< (x_1 + x_2 + 2)/(x_1 + x_2 + 5/4) \\ &< (11/4 + 2)/4 = 19/16. \end{aligned}$$

Case 3. Only one x_i is less than 1. Assume that $x_4 < 1$. $H_x(L) = x_1+x_2+x_3+1$ and $OPT(L) \geq x_1 + x_2 + x_3 + x_4 \geq 4$. If $x_1 + x_2 + x_3 \leq 3/4+3$, then $H_x(L)/OPT(L) \leq (3/4 + 4)/4 = 19/16$. Assume that $x_1 + x_2 + x_3 > 3/4 + 3 = 15/4$. Clearly, $\max\{x_1, x_2, x_3\} > 5/4$. Without loss of generality, suppose that $x_1 > 5/4$. Let J_d be the last job assigned to Machine 1. Before J_d is assigned, the load of Machine 1 is greater than 1/4. Clearly $x_4 > 1/4$. Otherwise, J_d cannot be assigned to Machine 1 but to Machine 4.

$$\begin{aligned} H_x(L)/OPT(L) &< (x_1 + x_2 + x_3 + 1)/(x_1 + x_2 + x_3 + 1/4) \\ &< (15/4 + 1)/4 = 19/16. \end{aligned}$$

By setting $x = 1/4$ the competitive ratio of algorithm H_x reaches the lower bound 19/16, and thus $x = 1/4$ is the best choice for four machines. □

We have proved the best algorithms among the class of algorithm H_x by setting the parameter x. It is interesting to see that the competitive ratio (11/9) for three machines is larger than that (19/16) for four machines. In the next section, we will give an improved algorithm for three machines and prove the competitive ratio 7/6, the same bound as H_x for two machines.

7.2.3 A new algorithm A_α for three machines

In this subsection we present an on-line algorithm characterized by the total excess of all machines, we denote it by A_α, where $\alpha > 0$ is a parameter.

Algorithm A_α:
The incoming job J_i with processing time p_i is assigned to the smallest index machine, if the total excess of the machines is not greater than α; otherwise J_i is assigned to the lowest index Machine j such that $p_i + l_j \leq 1$, where l_j is the current workload of the j-th machine. If such a machine does not exist, assign J_i to the machine with the smallest load, in case of ties, assign it to the machine with smallest index.

Algorithm A_α differs from algorithm H_x in that A_α may allow more excess of one machine but the total excess of the machines is controlled.

Lemma 7.2.8 The competitive ratio of algorithm A_α is at least 7/6 for three machines.

Proof. For any $\alpha > 0$, consider the following instance L. The job list consists of 6 jobs with processing time $1/4, 1/4, 1/4, 3/4, 3/4, 3/4$ respectively. Algorithm A_α assigns the first three jobs to one machine. Whatever the value of α is, the total excess of the three machines is at least 1/2 after the six jobs are assigned. Then $A_\alpha(L)/OPT(L) \geq (3 + 1/2)/3 = 7/6$. □

Theorem 7.2.9 The competitive ratio of algorithm A_α is 7/6 for three machines by setting $\alpha = 1/2$.

Proof. In the schedule given by algorithm A_α for any instance L, let y_i be the total processing time of the jobs assigned to Machine i ($i = 1, 2, 3$). Analogously, we only consider the following two cases.

Case 1. $y_i > 1$, $1 \leq i \leq 3$, and $y_j < 1$ for $j = 1, 2, 3$ and $j \neq i$. $A_\alpha(L) = y_i + 2$ and $OPT(L) \geq y_1 + y_2 + y_3 \geq 3$. If $y_i \leq 3/2$, then $A_\alpha(L)/OPT(L) \leq (3/2+2)/3 = 7/6$. Assume that $y_i > 3/2$. Let p_h be the processing time of the last job J_h assigned to Machine i. Let $\beta = y_i - p_h$, i.e., the load of Machine i immediately before job J_h is assigned. If $\beta \geq 3/4$, then $y_j \geq 3/4, j \neq i$. If $\beta < 3/4$, we have

- $y_j > 3/4$, as $j < i$. Otherwise, the jobs on Machine i assigned before J_h should go to Machine j since $\beta + y_j < 3/2$;
- as $j > i$, any job on Machine j has processing time greater than 3/4 and then $y_j > 3/4$.

Thus, it is always true that $y_j \geq 3/4, j \neq i$. It follows that

$$A_\alpha(L)/OPT(L) \leq (y_i + 2)/(y_i + 3/2) \leq 7/6.$$

Case 2. $y_i, y_j \geq 1$ and $y_k < 1$, $1 \leq i, j, k \leq 3$, and $j \neq i$. $A_\alpha(L) = y_i + y_j + 1$ and $OPT(L) \geq y_i + y_j + y_k \geq 3$. If the total excess of the machines is not greater than 1/2 then the theorem holds, since $A_\alpha(L)/OPT(L) \leq (3 + 1/2)/3 = 7/6$. Now assume that $y_i + y_j > 2 + 1/2 = 5/2$. If $y_k \geq 1/2$, then $A_\alpha(L)/OPT(L) \leq (y_i + y_j + 1)/(y_i + y_j + 1/2) < 7/6$. In the following, we will prove that $y_k \geq 1/2$. Let J_l be the job assigned to Machine i or j such that the total excess is over 1/2 after it is assigned. Before J_l is assigned, let the load of the machine (the one J_l goes to) be u and let the load of Machine k be v. By the algorithm, $u \leq v$. Clearly, $v > 1/2$, otherwise, those jobs in the two machines at that moment must be assigned together in a machine since the total processing time is not greater than 1. We thus get $y_k \geq v > 1/2$. The proof is completed. □

Before closing this section, we add a remark to the algorithms we have analyzed. Observe that both algorithms H_x and A_α may assign a job to a machine whose load

has already reached or exceeded one even if there exists a machine whose load is less than one. Clearly, it is not wise to do so, since it will increase the excess of the machines. We can simply modify the algorithms by putting the condition that if the load of a machine is at least one, it is not assigned a job as far as there exists some machine whose load is less than one. However, the modified algorithms can not achieve a better competitive ratio, because the worst-case instances introduced in the context still apply.

7.3 On machines of unequal regular working times

In this section, we study the extensible scheduling problem on machines with unequal regular working times. Consider any algorithm A. During the execution of algorithm A, if a machine B_j has a load (the total processing time of jobs assigned to it) is over its original regular working time b_j, B_j is called *heavy*; otherwise, it is called *light*. The difference between the load and the working time b_j is *idle space* if B_j is light or *excess* if B_j is heavy.

Consider any instance L where $\sum_{i=1}^{n} p_i < \sum_{j=1}^{m} b_j$. In an optimal packing there must be some machines with idle space. Construct a new instance L' by adding new jobs such that no machines have idle space in the optimal packing without changing the optimal value. Note that $A_{L'}(m, \mathcal{B}) \geq A_L(m, \mathcal{B})$ and $OPT_{L'}(m, \mathcal{B}) = OPT_L(m, \mathcal{B})$. Therefore,

$$A_L(m, \mathcal{B})/OPT_L(m, \mathcal{B}) \leq A_{L'}(m, \mathcal{B})/OPT_{L'}(m, \mathcal{B}).$$

In instance L', the total processing time of jobs is at least the total original regular working time of machines. It implies that we only need to consider the instances in which the total processing time of jobs is at least the total original regular working time of machines to prove an upper bound. In the following, in proving the competitive ratios we always assume

$$\sum_{i=1}^{n} p_i \geq \sum_{j=1}^{m} b_j. \tag{7.2}$$

7.3.1 Tight bound for a list scheduling algorithm

List Scheduling was introduced by Graham [72] for parallel machine scheduling. The algorithm always schedules the current job to the machine with minimum load at this time. Applying list scheduling to extensible scheduling on machines with unequal regular working time may result in different versions. We first consider a list scheduling algorithm, denoted by LS, which always assigns the incoming job to the machine of largest idle space. Dell'Olmo and Speranza [40] proved that $R_{LS} = 5/4$. In this subsection we figure out $R_{LS}(m, \mathcal{B})$. Then we will show that a different version of list scheduling, which always assigns the incoming item to the machine with the least load (the total processing time of jobs in a machine), is not as good as LS in terms of the overall competitive ratio.

Let b_{min} be the smallest regular working time in the collection $\mathcal{B}$ and let p_{max} be the largest processing time. Under the assumption (7.1), we have $p_{max} \leq b_{min}$. Consider the packing given by algorithm LS. For each machine B_j, let l_j be the total processing time of the jobs it accommodates after the schedule is over. If B_j is light, let s_j be its idle space, i.e., $s_j = b_j - l_j$. If B_j is heavy, let r_j be its idle space immediately before it becomes heavy, and e_j be its excess, i.e., $e_j = l_j - b_j$. We re-index the machines so that the first k machines are heavy.

In proving an upper bound for algorithm LS we can assume, without loss of generality, that a machine accepts no more jobs once it becomes heavy. Otherwise, all machines are heavy and the scheduling is optimal.

Lemma 7.3.1 The competitive ratio of the list scheduling algorithm

$$R_{LS}(m, \mathcal{B}) \geq \begin{cases} 1 + \frac{mb_{min}}{4\sum_{j=1}^{m} b_j}, & \text{if } m \text{ is even,} \\ 1 + \frac{(m^2-1)b_{min}}{4m\sum_{j=1}^{m} b_j}, & \text{if } m \text{ is odd.} \end{cases}$$

Proof. Consider the following instances. If m is even, a large number of small enough jobs are coming first. The total processing time of these jobs is $\sum_{j=1}^{m} b_j - mb_{min}/2$ such that after assigning these jobs by LS, each machine has idle space exactly $b_{min}/2$. Then $m/2$ jobs with processing time of b_{min} are coming. Clearly, the optimal value is $\sum_{j=1}^{m} b_j$, while LS generates a total working time $\sum_{j=1}^{m} b_j + mb_{min}/4$. Thus $R_{LS}(m, \mathcal{B}) \geq 1 + \frac{mb_{min}}{4\sum_{j=1}^{m} b_j}$.

If m is odd, the instance is slightly different. A large number of small enough jobs are coming first. The total processing time of them is $\sum_{j=1}^{m} b_j - (m-1)b_{min}/2$, so that after assigning these jobs each machine has idle space exactly $\frac{m-1}{2m}b_{min}$. Then $(m-1)/2$ jobs with processing time b_{min} are coming. Clearly, the optimal total working time is $\sum_{j=1}^{m} b_j$, while LS generates a total working time of $\sum_{j=1}^{m} b_j + (m^2-1)b_{min}/(4m)$. Thus $R_{LS}(m,\mathcal{B}) \geq 1 + \frac{(m^2-1)b_{min}}{4m\sum_{j=1}^{m} b_j}$. The lemma is proved. □

Theorem 7.3.2 The competitive ratio of the list scheduling algorithm

$$R_{LS}(m,\mathcal{B}) = \begin{cases} 1 + \frac{mb_{min}}{4\sum_{j=1}^{m} b_j}, & \text{if } m \text{ is even,} \\ 1 + \frac{(m^2-1)b_{min}}{4m\sum_{j=1}^{m} b_j}, & \text{if } m \text{ is odd.} \end{cases}$$

Proof. We only need to prove that

$$R_{LS}(m,\mathcal{B}) \leq \begin{cases} 1 + \frac{mb_{min}}{4\sum_{j=1}^{m} b_j}, & \text{if } m \text{ is even,} \\ 1 + \frac{(m^2-1)b_{min}}{4m\sum_{j=1}^{m} b_j}, & \text{if } m \text{ is odd.} \end{cases}$$

Consider any instance $L(m,\mathcal{B})$ satisfying (7.2). By the algorithm LS, we know that

$$\frac{\sum_{j=1}^{k} r_j}{k} \geq \min_{j=1,\cdots,k} r_j \geq \max_{j=k+1,\cdots,m} s_j \geq \frac{\sum_{j=k+1}^{m} s_j}{m-k}.$$

Then

$$(m-k)\sum_{j=1}^{k} r_j \geq k \sum_{j=k+1}^{m} s_j \tag{7.3}$$

From (7.2), we have

$$\sum_{i=1}^{k} e_i \geq \sum_{j=k+1}^{m} s_j. \tag{7.4}$$

We add $(m-k)\sum_{j=1}^{k} e_j$ to both sides of the inequality (7.3) and from (7.4), we get

$$\begin{aligned} (m-k)\sum_{j=1}^{k}(r_j+e_j) &\geq k\sum_{j=k+1}^{m} s_j + (m-k)\sum_{j=1}^{k} e_j \\ &\geq k\sum_{j=k+1}^{m} s_j + (m-k)\sum_{j=k+1}^{m} s_j \\ &= m\sum_{j=k+1}^{m} s_j. \end{aligned}$$

It means that $\sum_{j=k+1}^{m} s_j \leq \frac{(m-k)}{m}\sum_{j=1}^{k}(r_j+e_j)$. Note that the competitive ratio

$$R_{LS}(m,\mathcal{B}) \leq (\sum_{i=1}^{n} p_i + \sum_{j=k+1}^{m} s_j)/\sum_{i=1}^{n} p_i.$$

Then

$$R_{LS}(m,\mathcal{B}) \le 1 + \frac{(m-k)\sum_{j=1}^{k}(r_j+e_j)}{m\sum_{i=1}^{n}p_i} \le 1 + \frac{(m-k)\sum_{j=1}^{k}(r_j+e_j)}{m\sum_{j=1}^{m}b_j}.$$

Since $r_j + e_j \le p_{\max} \le b_{min}$, we obtain

$$R_{LS}(m,\mathcal{B}) \le 1 + \frac{(m-k)kb_{min}}{m\sum_{j=1}^{m}b_j}.$$

If m is even, then $m^2 - 4mk + 4k^2 = (m-2k)^2 \ge 0$ and $(m-k)k/m^2 \le 1/4$. It follows

$$R_{LS}(m,\mathcal{B}) \le 1 + \frac{mb_{min}}{4\sum_{j=1}^{m}b_j}.$$

If m is odd, then $m^2 - 4mk + 4k^2 = (m-2k)^2 \ge 1$ and $(m-k)k \le \frac{1}{4}(m^2-1)$. We have

$$R_{LS}(m,\mathcal{B}) \le 1 + \frac{(m^2-1)b_{min}}{4m\sum_{j=1}^{m}b_j}.$$

□

Corollary 7.3.3

$$R_{LS}(m,\mathcal{B}) \le R_{LS}(m) = \begin{cases} \frac{5}{4}, & \text{if } m \text{ is even,} \\ \frac{5}{4} - \frac{1}{4m^2}, & \text{if } m \text{ is odd.} \end{cases}$$

Moreover, $R_{LS} = 5/4$. □

Note that $R_{LS}(m,\mathcal{B}) < R_{LS}(m)$ if $m \ge 2$ and $b_i \ne b_j$ for some i, j. It means that the competitive ratio becomes smaller if the regular working times are unequal.

Before closing this subsection, we consider a different version of list scheduling: assign jobs to the machine of least load. Denote the algorithm as LS'. Consider the following simple instance L for two machines. Let $b_1 = 2b_2$ and let $\varepsilon > 0$ be a sufficiently small number. The first two jobs have processing time of $b_2 - \varepsilon$, which are followed by an job with processing time of $\varepsilon/2$. By algorithm LS', after assigning the first two jobs, each machine has a load of $b_2 - \varepsilon$. If the 3rd job goes to the first machine B_1, a job (the last one) with processing time of b_2 comes, which is assigned to B_2. If the 3rd job goes to the second machine B_2, then the 4th job has a processing time of ε and the 5th job (the last one) has processing time of b_2. The 4th is assigned

to B_1 while the 5th is assigned to B_2. In both cases, $LS'_L(2,\mathcal{B}) = b_1 + b_2 + b_2 - \varepsilon$. In an optimal packing, we can assign the last job to B_2 and the others to B_1. Then $OPT_L(2,\mathcal{B}) = b_1 + b_2$, which shows that $R_{LS'}(2,\mathcal{B}) \to 4/3$ as ε tends to zero. It implies that $R_{LS'} \geq 4/3$.

7.3.2 The two- and three-machine cases

Assume that $b_1 \geq b_2 \geq \cdots \geq b_m \geq p_{max}$. In this section we consider the cases that $m = 2$ and $m = 3$.

Algorithm $A_m(\alpha)$: Assign the incoming job J_i to the lowest indexed machine, if the total excess of the machine is not greater than α; otherwise assign J_i to the machine of largest idle space.

In this algorithm α is a user-specified parameter. Choosing different parameters results in different algorithms.

Lemma 7.3.4 For any parameter $\alpha > 0$, the competitive ratio of algorithm $A_2(\alpha)$ is at least $1 + b_2/(3(b_1 + b_2))$.

Proof. Let N be a sufficiently large integer. If $\alpha < b_2/3$, consider an instance L as follows. The first $\lfloor (b_1 - 2b_2/3)N \rfloor$ jobs are small jobs, each with processing time $1/N$, which are followed by jobs J_1, J_2, J_3 with processing time of $b_2/3, 2b_2/3$ and $2b_2/3$, respectively. Clearly, $A_2(\alpha)$ assigns all the small jobs together with J_1 to machine B_1, and J_2 to machine B_2. Thus no matter where J_3 is assigned, $A_2(\alpha)_L(2,\mathcal{B}) \geq b_1 + b_2 + b_2/3 - 1/N$. For an optimal solution, we can assign J_1 and J_2 to B_2, and the remaining jobs to B_1. Thus $OPT_L(2,\mathcal{B}) = b_1 + b_2$, which follows that $A_2(\alpha)_L(2,\mathcal{B})/OPT_L(2,\mathcal{B}) \to 1 + b_2/(3(b_1 + b_2))$ as N goes to infinity.

If $\alpha \geq b_2/3$, consider the following instance L: the first $\lfloor (b_1 - 2b_2/3)N \rfloor$ jobs with processing time $1/N$, are followed by job J_1 with size b_2. So $A_2(\alpha)_L(2,\mathcal{B}) \geq b_1 + b_2/3 + b_2 - 1/N$. In an optimal packing, we assign J_1 to B_2, and the remaining jobs to B_1. $OPT_L(2,\mathcal{B}) = b_1 + b_2$, which follows $A_2(\alpha)_L(2,\mathcal{B})/OPT_L(2,\mathcal{B}) \to 1 + b_2/(3(b_1 + b_2))$ as N tends to infinity. □

By setting $\alpha = b_2/3$, we prove that the competitive ratio of $A_2(\alpha)$ is $1+b_2/(3(b_1+$

$b_2)$).

Theorem 7.3.5 The competitive ratio of $A_2(\alpha)$ is $1 + b_2/(3(b_1 + b_2))$ if $\alpha = b_2/3$. Moreover the overall competitive ratio is 7/6.

Proof. If both machines are light or both are heavy, the scheduling is optimal. We only need to consider the case that exactly one machine is heavy. From the assumption (7.2), we have $M_1 + M_2 \geq b_1 + b_2$, where M_i is the load of machine B_i, $i = 1, 2$.

Let $\alpha = b_2/3$. Let T be the excess of the heavy machine and let X be the idle space of the light machine. If $T \leq \alpha$, then $R_{A_2(\alpha)}(2, \mathcal{B}) \leq (T + b_1 + b_2)/(b_1 + b_2) \leq 1 + b_2/(3(b_1 + b_2))$. Now assume that $T > \alpha$. We want to prove $X \leq \alpha$.

Case 1. B_2 is light. $X = b_2 - M_2$. Let J_n be the last job assigned to B_1, which makes the excess over α. $B_2 \neq \emptyset$ and before J_n is assigned B_1 is light and has an idle space at least X. Otherwise, a_n should have been assigned to B_2. It implies that the processing time of the jobs in B_2 is larger than $X + \alpha$. Thus $X = b_2 - M_2 < b_2 - (X + \alpha)$. It follows that $X < \alpha$.

Case 2. B_1 is light. $X = b_1 - M_1$. Before the last job is assigned to B_2, B_2 is light and has an idle space at least X. The total processing time of jobs in B_2 before the last job is assigned is at most $b_2 - X$. According to the algorithm, $b_2 - X + M_1 > b_1 + \alpha$. It implies that $X \leq \alpha$.

In both cases we have $R_{A_2(\alpha)}(2, \mathcal{B}) \leq (X + M_1 + M_2)/(M_1 + M_2) \leq 1 + b_2/(3(b_1 + b_2))$. It is easy to verify that $R_{A_2(\alpha)}(2, \cdot) = 7/6$. □

Since no on-line algorithm can have an overall competitive ratio less than 7/6 for two machines [40], $A_2(\alpha)$ is an optimal on-line algorithm for $m = 2$ in terms of the overall competitive ratio, setting $\alpha = b_2/3$.

We will show a lower bound for two bins under the assumption (7.1). Let $b_1 = kb_2/3 + x$, where $0 \leq x < b_2/3$ and $k \geq 3$ (k is an integer).

Lemma 7.3.6 No on-line algorithm can achieve a competitive ratio smaller than

$$R = \begin{cases} 1 + (b_2/3 - x)/(b_1 + b_2), & \text{if } 0 \leq x \leq b_2/6; \\ 1 + x/(b_1 + b_2), & \text{if } b_2/6 < x < b_2/3. \end{cases}$$

Proof. Let A be any on-line algorithm. Consider the following instance L. The jobs with processing time of $b_2/3$ come one by one until $n(b_2) = 2$ or $n(b_1) = k-1$, where $n(b_i)$ is the number of jobs with processing time of $b_2/3$ placed into machine B_i by algorithm A for $i = 1, 2$.

Case 1. $n(b_2) = 0, n(b_1) = k-1$. If $0 \leq x \leq b_2/6$, the next two jobs have processing time of $2b_2/3$. Thus $A_L(2, \mathcal{B}) \geq b_1 + b_2 + b_2/3 - x$.

If $b_2/6 < x < b_2/3$, the next job has processing time of x. If x goes to B_1, two jobs with processing time of $2b_2/3$ come. It follows that $A_L(2, \mathcal{B}) \geq b_1 + b_2 + b_2/3$. If x goes to B_2, a job with processing time of b_2 comes. Then $A_L(2, \mathcal{B}) \geq b_1 + b_2 + x$.

Case 2. $n(b_2) = 1, n(b_1) = k-1$. The next job has processing time of b_2 and then $A_L(2, \mathcal{B}) \geq b_1 + b_2 + b_2/3$.

Case 3. $n(b_2) = 2, n(b_1) = p$, where $0 \leq p \leq k-1$.

Subcase 3a) If $p = k-1$, the next job has processing time of $2b_2/3 + x$, and thus $A_L(2, \mathcal{B}) \geq b_1 + b_2 + b_2/3$.

Subcase 3b) If $p = k-2$, we consider two subcases. If $0 \leq x \leq b_2/6$, the next job has processing time of x. If it is assigned to B_1, a job with processing time of b_2 comes. Then $A_L(2, \mathcal{B}) \geq b_1 + b_2 + b_2/3$. If it is assigned to B_2, a job with processing time of b_2 comes. So $A_L(2, \mathcal{B}) \geq b_1 + b_2 + b_2/3 - x$. If $b_2/6 < x < b_2/3$, the next two jobs have processing time of $b_2/3 + x$ and $2b_2/3$. So, $A_L(2, \mathcal{B}) \geq b_1 + b_2 + x$.

Subcase 3c) If $p = k-3$, the next two jobs have processing time of $2b_2/3 + x/2$. $A_L(2, \mathcal{B}) \geq b_1 + b_2 + b_2/3$.

Subcase 3d) If $p \leq k-4$, let $s = \lceil (k-p)/3 \rceil$-1. The next s jobs have processing time of b_2. If one of such jobs is packed into B_2, then $A_L(2, \mathcal{B}) \geq b_1 + b_2 + 2b_2/3$. Otherwise, all the s jobs go to B_1. The idle space of B_1 becomes $(k-p)b_2/3 + x - (\lceil (k-p)/3 \rceil - 1)b_2$, which is either $b_2/3 + x$ or $2b_2/3 + x$ or $b_2 + x$. Then it is reduced to the above three subcases 3a) -3c).

Note that in any of the above cases, $OPT_L(2, \mathcal{B}) = b_1 + b_2$. Thus, the lemma is proved. □

Lemma 7.3.6 shows that $R_A(2, \mathcal{B}) \geq 1 + \frac{b_2}{6(b_1+b_2)}$ for any on-line algorithm A. It

also shows that algorithm $A_2(\alpha)$ is optimal for the case that $x = 0$ (or $b_1 = kb_2/3$ for some integer k), by setting $\alpha = b_2/3$. Now we consider the three-bin case.

Lemma 7.3.7 For any parameter $\alpha > 0$, the competitive ratio of algorithm $A_3(\alpha)$ is at least $1 + b_3/(2(b_1 + b_2 + b_3))$.

Proof. Let N be a sufficiently large integer. If $\alpha \geq b_3/2$, consider the following instance L. The first $\lfloor b_1N - 1 \rfloor$ jobs with processing time of $1/N$ are followed by a job of processing time $b_3/2 + 1/N$. Clearly, $A_3(\alpha)_L(3, \mathcal{B}) \geq b_1 + b_2 + 3b_3/2 - 1/N$ and $OPT_L(3, \mathcal{B}) \leq b_1 + b_2 + b_3 + 1/N$. It follows that $A_3(\alpha)_L(3, \mathcal{B})/OPT_L(3, \mathcal{B}) \rightarrow 1 + b_3/(2(b_1 + b_2 + b_3))$ as N tends to infinity.

If $\alpha < b_3/2$, consider the following instance L. $\lfloor b_1N - 1 \rfloor$ jobs with processing time of $1/N$ are followed by job J_1 with processing time of $\alpha + 1/N$. Then $\lfloor (b_2 - b_3/2)N \rfloor + 1$ jobs with processing time of $1/N$ are coming, which are followed by the last two jobs J_2, J_3 with processing time of $b_3/2$, $b_3 - \alpha - 1/N$, respectively. Algorithm $A_3(\alpha)$ assigns the first $\lfloor b_1N - 1 \rfloor$ jobs together with J_1 to B_1, $\lfloor (b_2 - b_3/2)N + 1 \rfloor$ jobs with processing time $1/N$ to B_2, and J_2 and J_3 to B_3. It shows $A_3(\alpha)_L(3, \mathcal{B}) \geq b_1 + b_2 + 3b_3/2 - 2/N$. In an optimal packing, we can assign J_1 and J_3 to B_3, the first $\lfloor b_1N - 1 \rfloor$ jobs with processing time $1/N$ to B_1 and the remaining jobs to B_2. It follows that $OPT_L(3, \mathcal{B}) \leq b_1 + b_2 + b_3 + 1/N$. Then $A_3(\alpha)_L(3, \mathcal{B})/OPT_L(3, \mathcal{B}) \rightarrow 1 + b_3/(2(b_1 + b_2 + b_3))$, as N tends to infinity. □

In the following, we prove that the competitive ratio of algorithm $A_3(\alpha)$ can reach the lower bound $1 + b_3/(2(b_1 + b_2 + b_3))$ by setting $\alpha = b_3/2$.

Theorem 7.3.8 The competitive ratio of the algorithm $A_3(\alpha)$ is $1 + b_3/(2(b_1 + b_2 + b_3))$ if $\alpha = b_3/2$.

Proof. Let $\alpha = b_3/2$. In the packing given by $A_3(\alpha)$, let M_i be the total processing time of the jobs scheduled to machine B_i. If all the machines are heavy or all the machines are light, the algorithm gives an optimal packing. Hence, we only need to consider the following cases.

Case 1. Only one machine is heavy. If the excess of the heavy machine is at most α, we can get the competitive ratio $1 + b_3/(2(b_1 + b_2 + b_3))$ immediately. Before the

last job of the heavy machine is assigned, all the three machines are light and the heavy machine has the largest idle space at the moment.

Subcase 1a) B_1 is heavy. Then $M_1 > b_1 + \alpha$. Note that $M_2 + M_3 > b_2 + \alpha$. We have

$$\begin{aligned} R_{A_3(\alpha)}(3, \mathcal{B}) &\leq (M_1 + b_2 + b_3)/(M_1 + M_2 + M_3) \\ &= 1 + (b_2 + b_3 - M_2 - M_3)/(M_1 + M_2 + M_3) \\ &\leq 1 + b_3/(2(b_1 + b_2 + b_3)) \end{aligned}$$

Subcase 1b) B_2 is heavy. $M_2 > b_2 + b_3/2$ and $M_1 + M_3 > b_1 + b_3/2$.

$$\begin{aligned} R_{A_3(\alpha)}(3, \mathcal{B}) &\leq (M_2 + b_1 + b_3)/(M_1 + M_2 + M_3) \\ &= 1 + (b_1 + b_3 - M_1 - M_3)/(M_1 + M_2 + M_3) \\ &\leq 1 + b_3/(2(M_1 + M_2 + M_3)) \\ &\leq 1 + b_3/(2(b_1 + b_2 + b_3)) \end{aligned}$$

Subcase 1c) B_3 is heavy. $M_3 > b_3 + b_3/2$. Let X be the total idle space of M_1 and M_2. Then $X = b_1 + b_2 - (M_1 + M_2)$. Let Y_3 be the load in B_3 before the last job is assigned. Thus

$$b_3 - Y_3 \geq b_2 - M_2,\ b_3 - Y_3 \geq b_1 - M_1.$$

We have $X = b_1 + b_2 - (M_1 + M_2) \leq 2(b_3 - Y_3)$. On the other hand,

$$Y_3 + M_1 \geq b_1 + b_3/2,\ Y_3 + M_2 \geq b_2 + b_3/2.$$

We have $X = b_1 + b_2 - (M_1 + M_2) \leq 2Y_3 - b_3$. So, $X \leq \min\{2(b_3 - Y_3), 2Y_3 - b_3\} \leq b_3/2$. Therefore,

$$\begin{aligned} R_{A_3(\alpha)}(3, \mathcal{B}) &\leq (X + M_1 + M_2 + M_3)/(M_1 + M_2 + M_3) \\ &\leq 1 + b_3/(2(b_1 + b_2 + b_3)). \end{aligned}$$

Case 2. Only one machine is light. If the idle space X of the light machine is at most α or the total excess of the two heavy bins is at most α, we have the competitive ratio immediately. We only need to consider the case that $X > \alpha$ and the total excess exceeds α. However, We will show that it is impossible. Before considering

the following two cases, we assume that $M_i > 0$ for $i = 1, 2, 3$. Otherwise, the total excess is at most α.

Subcase 2a) B_3 is light. $M_3 = b_3 - X < \alpha = b_3/2$. Let Y_i be the load of B_i immediately before it becomes heavy, for $i = 1, 2$. Then $b_i - Y_i \geq X$; otherwise the last item, which makes the total excess of B_1 and B_2 over α, would have been assigned to machine B_3. Thus all the items in B_3 should have been assigned to B_i by the algorithm. It is a contradiction.

Subcase 2b) B_3 is heavy. Recall that any processing time of job is at most b_3. There are at least two jobs in B_3. Assume that J_k is the last job assigned to B_3. Those jobs assigned to B_3 have processing time at least $X > b_3/2$. Thus the idle space of B_3 before J_k is assigned is less than X, which means that J_k would have not been assigned to B_3. It is a contradiction. □

From Theorems 7.3.5 and 7.3.8 we realize that the competitive ratio of algorithm $A_m(\alpha)$ for problem on machines with unequal regular working times is better than that for identical machines.

7.3.3 Without the assumption on processing times

In this subsection, we will discuss the problem without the assumption (7.1) for the two-machine case.

Lemma 7.3.9 Without the assumption (7.1), no deterministic on-line algorithms can achieve an overall competitive ratio lower than 6/5 for two machines.

Proof. Let $b_1 = 3/2$ and $b_2 = 1$. The first job J_1 has processing time of $1/2$. If J_1 is assigned to machine B_2, two jobs J_2, J_3 with processing time of 1 are coming. If J_1 is assigned to machine B_1, item J_2 with processing time of $3/2$ is coming. For both cases, the overall competitive ratio is at least 6/5. □

Lemma 7.3.10 For any on-line algorithm A the competitive ratio

$$R_A(2, \mathcal{B}) \geq \begin{cases} 1 + (4b_2/3 - b_1)/(b_1 + b_2), & \text{if } b_1 \leq 7b_2/6, \\ 1 + (b_1 - b_2)/(b_1 + b_2), & \text{if } 7b_2/6 < b_1 < 4b_2/3, \\ 1 + b_2/(3(b_1 + b_2)), & \text{if } b_1 \geq 4b_2/3. \end{cases}$$

Proof. Let A be any on-line algorithm. Consider first the case that $b_1 \geq 4b_2/3$ with the following instance L. The first two jobs have processing time of $b_2/3$. If both jobs go to B_1, the next job has processing time of b_1; if both jobs go to B_2, two jobs with processing time of $2b_2/3$ and $b_1 - b_2/3$, respectively, are coming; if the first two jobs go to different machines, the next job has processing time of b_1. For any of the above cases, $A_L(2, \mathcal{B}) \geq b_1 + 4b_2/3$ and the optimal value $OPT_L(2, \mathcal{B}) = b_1 + b_2$. It implies that the lemma is true.

Now turn to the case that $b_1 < 4b_2/3$. Let $b_1 = b_2 + x$, where $0 \leq x < b_2/3$. The lemma follows from Lemma 7.3.6 by setting $k = 3$. □

We consider on-line algorithms. Assume that all jobs have processing time of at most b_1 ($b_1 \geq b_2$).

Theorem 7.3.11 The competitive ratio of the list scheduling algorithm LS for two machines is $1 + \min\{b_2, b_1/2\}/(b_1 + b_2)$.

Proof. We first show the upper bound. Without loss of generality, we consider the case only one machine is heavy.

Case 1. $b_1/2 \geq b_2$. Since the LS algorithm assigns jobs to the machine with largest idle space. Recall that the total processing time of jobs is not less than the total regular working time of machines, we get that the idle space is at most b_2. Then, $LS_L(2, \mathcal{B}) \leq \sum p_i + b_2$, $OPT_L(2, \mathcal{B}) \geq \sum p_i \geq b_1 + b_2$, which implies $R_{LS}(2, \mathcal{B}) \leq 1 + b_2/(b_1 + b_2)$.

Case 2. $b_1/2 < b_2$. Let X be the idle space and T be the excess. We only consider the case that both X and T are greater than $b_1/2$. Otherwise, it follows that $R_{LS}(2, \mathcal{B}) \leq 1 + b_1/(2(b_1 + b_2))$. Let p_n be the processing time of the last job assigned to the machine which has an excess over $b_1/2$. Recall that all jobs have processing time of at most b_1. Let Y_i be the load of machine B_i before the last job is assigned. Then $b_i - Y_i \geq X > b_1/2$, since $p_n \leq b_1$. It implies that either X or T is at most $b_1/2$.

The following simple instance shows that the bound is tight. Consider three jobs with processing times of $\min\{b_2, b_1/2\}$, $\max\{0, b_2 - b_1/2\}$, and b_1, respectively. The optimal value is $b_1 + b_2$ while LS costs $b_1 + b_2 + \min\{b_2, b_1/2\}$. □

In the following, we present an on-line algorithm to improve the upper bound in some cases.

Algorithm $A2$**:**

- If $b_1 \leq 4b_2/3$, apply algorithm $A_2(\alpha)$ by setting $\alpha = b_2/3$;
- if $4b_2/3 < b_1 \leq 2b_2$, apply $A_2(\alpha)$ by setting $\alpha = b_1 - b_2$;
- if $b_1 > 2b_2$, apply algorithm LS.

Theorem 7.3.12 The competitive ratio of algorithm $A2$ is

$$R_{A2}(2, \mathcal{B}) \leq \begin{cases} 1 + b_2/(3(b_1 + b_2)), & \text{if } \ b_1 \leq 4b_2/3, \\ 1 + (b_1 - b_2)/(b_1 + b_2), & \text{if } \ 4b_2/3 < b_1 \leq 2b_2, \\ 1 + b_2/(b_1 + b_2), & \text{if } \ b_1 > 2b_2. \end{cases}$$

Proof. Consider the following cases:

Case 1. $b_1 \leq 4b_2/3$. We use $A_2(\alpha)$, by setting $\alpha = b_2/3$. The proof is similar as the one of Theorem 7.3.5.

Case 2. $4b_2/3 < b_1 \leq 2b_2$. We only need to consider the case that exactly one machine is heavy, the excess of the heavy machine is over $b_1 - b_2$, and the idle space of the light machine is larger than $b_1 - b_2$. However, we will show this case is impossible. Before the last job J_n, which makes the excess over $b_1 - b_2$, is assigned, if B_2 is empty, then the excess is at most $b_1 - b_2$ after assigning J_n. Assume that B_2 is not empty before J_n is assigned. Note that B_1 has an idle space larger than $b_1 - b_2$ before J_n is assigned. It means that the total processing time of jobs in B_2 is more than $2(b_1 - b_2)$. Then the idle space of machine B_2 is at most $b_2 - 2(b_1 - b_2) < (b_1 - b_2)$. In this case J_n is assigned to B_1. B_2 is a light machine and the idle space is less than $b_1 - b_2$. It is a contradiction.

Case 3. $b_1 > 2b_2$. It follows from Theorem 7.3.11. □

7.4 Note on parallel jobs

In this section we extend the model to parallel jobs scheduling. A short discussion on the off-line version is given below.

For any instance L, we are given m identical machines, each with a regular working time of one. A set of jobs $J = \{J_1, J_2, \ldots, J_n\}$ are available. Each job J_i is characterized by processing time p_i, $0 < p_i \leq 1$ and size (the number of requested machines) s_i. The objective is to minimize the total working time of the machines.

Similarly as the case for non-parallel jobs, without loss of generality, we assume that $\sum_{i=1}^{n} p_i * s_i \geq m$, since, otherwise, there are some machines not completely full in an optimal schedule and we can add some tiny jobs without changing the optimal value such that all machines are full.

The extensible scheduling problem can be regarded as a strip packing problem. Job J_i corresponds to a rectangle with width s_i and height p_i. It needs to be packed in a strip with width m and an infinite height.

We can apply the Steinberg's algorithm [123] to our problem for packing all the jobs into a rectangle with width m and height $2\sum_{i=1}^{n} p_i * s_i$. It is feasible to do that by Lemma 3.2.1, since $p_i \leq 1$ and $s_i \leq m \leq \sum_{i=1}^{n} p_i * s_i$. Then we have the following theorem.

Theorem 7.4.1 Using Steinberg's algorithm, the worst-case ratio is at most 2.

Proof. Clearly the total working time of machines produced by Steinberg's algorithm is at most $2\sum_{i=1}^{n} p_i * s_i$. On the other hand, any optimal off-line algorithm can not generate a schedule with total working time of machines less than $\sum_{i=1}^{n} p_i * s_i$. Then the theorem follows. □

Chapter 8

On-line scheduling with partial information

8.1 Introduction

In real-world situations, most scheduling problems occur neither as complete off-line nor complete on-line models. Most likely, a problem arises as an on-line model with some partial information. We will study in this chapter whether it can help or not to get a better competitive ratio for the classical scheduling problem, in which jobs require exactly one machine for processing, with some partial information.

In this chapter, we only consider the model where jobs arrive over list. The default objective function of any scheduling problem is makespan. For other objectives, we will point it out explicitly.

Recently, there have been some results on *semi on-line* version [77,95,134], which assume that different partial information is known in advance or given during the scheduling. In the first paper on the worst-case analysis of scheduling heuristics, Graham [72] showed that algorithm LS has a competitive ratio of $2-1/m$, where m is the number of machines. Faigle, Kern and Turan [54] proved that for $m = 2$ and 3, LS is the best possible; in other words, no on-line algorithms can have competitive ratio less than $2 - 1/m$ for $m = 2$ and $m = 3$.

In general, there are three rules associated with on-line scheduling:

(a) No information on future jobs;

(b) a job must be assigned as it appears;

(c) an assigned job cannot be moved later.

If any of the three rules is violated, the problem will not be pure on-line and it is called *semi on-line*. For semi on-line scheduling, a number of problems have been studied (see [77,95,134]). Best possible semi on-line algorithms have been shown to be better than LS for the two-machine case. There are three versions falling into the class in which the on-line rule (a) is relaxed:

P_1 : The total sum of job processing times is known in advance.

P_2 : The longest processing time is known in advance.

P_3 : All job processing times are in between p and rp ($p > 0, r \geq 1$).

In the class in which the on-line rule (b) is violated, the following case has been studied.

P_4 : A buffer of length k is available to maintain k items. An incoming job can be either arranged to a machine or put to the buffer temporarily if the buffer is not full.

In addition, [95] considered a semi on-line problem which belongs to neither of the above classes. Two parallel machines are available which assign each job independently to the partition sets. The best of the two produced solutions is chosen. We denote this problem by P_5. Note that only the two-machine case was studied. Interestingly, except P_3, all other problems have a lower bound 4/3 which can be reached by proposed algorithms. In particular, the best possible algorithm for P_4 only needs a buffer of length one. For the problem P_3, LS is still the best possible with competitive ratio $(r+1)/2$ as $1 \leq r \leq 2$ and $3/2$ as $r \geq 2$.

In this chapter we will consider a new semi on-line version in which a job list will be ended by a longest job, in other words, the last job in the list is one with the longest processing time. We denote this problem by LL (Longest Last). We wonder if such an additional information would help us to find some algorithms with better

competitive ratio than that of LS for two- and three-machine cases. This paper will give an answer.

An adversary is called *mute* if he/she never says one word before the schedule is completed; otherwise, the adversary may give some hints and he/she is called *suggestive*.

Observation 8.1.1 For Problem LL, if the adversary is mute, LS is still the best possible for $m = 2$ and 3, where m is the number of machines.

Proof. It is true because the instances in [54] still work. □

Thus we will concentrate on a case that the adversary is suggestive.

Assumption: *At the same time the last job appears, the adversary will inform that this is the last one.*

Under this assumption, the scheduler knows if an incoming job is the last before assigning it. Throughout this paper the above assumption always holds when Problem LL is mentioned. To design better algorithms, we must save some space for the last job (with longest processing time), i.e., one machine should be reserved in some way to accommodate the last job. This will be the main idea to design the algorithms in the sections below.

Organization of this chapter The problem on two- and three- identical machines are studied in Section 8.2. Then we consider two uniform machines in Section 8.3. Section 8.4 focuses on the problem whose objective function is to maximize the minimum completion time.

8.2 On two- and three- identical machines cases

8.2.1 The two-machine case

Lemma 8.2.1 There does not exist any on-line algorithm with competitive ratio less than $\sqrt{2}$ for Problem LL when $m = 2$.

Proof. Consider any heuristic H with the following instances. The first two jobs have processing times 1. If H assigns the two jobs to different machines, the last

job with processing time 2 comes. In this case H produces a makespan 3 while the optimal makespan is 2. Then the ratio of the makespans reaches 3/2. If H puts the two jobs together to a machine, say Machine 1, then the 3rd job with processing time $\sqrt{2}$ arrives. We have two cases below.

Case 1. H puts the 3rd job to Machine 1. In this case, the current workload of Machine 1 is $2+\sqrt{2}$, while Machine 2 is empty. The last job with processing time $\sqrt{2}$ comes. Then $C_H \geq 2+\sqrt{2}$ while $C^* = 1+\sqrt{2}$. Thus $C_H/C^* \geq \sqrt{2}$.

Case 2. H puts the 3rd job to Machine 2. In this case, the current workload of Machine 1 is 2, while Machine 2 has a length $\sqrt{2}$. The last job with processing time $2+\sqrt{2}$ comes. Then $C_H \geq 2+2\sqrt{2}$ while $C^* = 2+\sqrt{2}$. Thus $C_H/C^* \geq \sqrt{2}$.
The proof of this lemma is complete. □

In the following we will provide an on-line algorithm with competitive ratio $\sqrt{2}$. We denote jobs and their processing times by J_i and p_i $(i = 1, 2, \ldots)$, respectively. Let $M_j(i)$ denote the workload of Machine j immediately before the job J_i is assigned, for $i = 1, 2, \ldots$, and $j = 1, 2$.

Algorithm A_1

Step 0. Let $i = 1$.

Step 1. If J_i is the last job arrange it to Machine 2 and stop. Otherwise, go to Step 2.

Step 2. If $M_2(i) + p_i > (\sqrt{2}-1)(M_1(i)+p_i)$, assign J_i to Machine 1; otherwise assign it to Machine 2. Let $i = i+1$, go to Step 1.

Observation 8.2.2 In the schedule produced by algorithm A_1, $M_1(i) \geq \sqrt{2}M_2(i)$ holds for $1 \leq i \leq n$, where n is the number of jobs.

Proof. Note that $M_1(1) = M_2(1) = 0$. We can assume that $i \geq 2$. Let J_t be the last job assigned to Machine 2 at the moment. From the algorithm, we have $M_2(i) \leq (\sqrt{2}-1)(M_1(i)+p_t)$, where p_t is the processing time of J_t. Note that $p_t \leq M_2(i)$. Thus

$$M_2(i) \leq (\sqrt{2}-1)(M_1(i)+M_2(i)). \tag{8.1}$$

It implies that $M_1(i) \geq \sqrt{2}M_2(i)$. □

Theorem 8.2.3 Algorithm A_1 is a best possible algorithm for Problem LL when $m = 2$.

Proof. We only need to prove that $R_{A_1} \leq \sqrt{2}$. Let y be the processing time of the last job J_n. Then the makespan C_{A_1} produced by A_1 is $\max\{M_1(n), M_2(n) + y\}$. Consider the following two cases.

Case 1. $C_{A_1} = M_1(n)$. Let x be the processing time of the last job assigned to Machine 1. From algorithm A_1, we have $M_2(n)+x > (\sqrt{2}-1)M_1(n)$ (otherwise this job would have been scheduled on Machine 2). Since J_n is a job with the longest processing time, $y \geq x$. We get

$$C^* \geq (M_1(n)+M_2(n)+y)/2 \geq (M_1(n)+M_2(n)+x)/2 \geq (\sqrt{2}/2)M_1(n) = (\sqrt{2}/2)C_{A_1},$$

which implies that $C_{A_1}/C^* \leq \sqrt{2}$.

Case 2. $C_{A_1} = M_2(n) + y$. If $M_2(n) = 0$, $C^* = C_{A_1} = y$. If $M_2(n) > 0$, let z be the processing time of the last job assigned to Machine 2 immediately before J_n. Note that (8.1) holds. If $y \geq M_1(n) + M_2(n)$,

$$C_{A_1}/C^* \leq (M_2(n) + y)/y = 1 + M_2(n)/y \leq 1 + M_2(n)/(M_1(n) + M_2(n)) \leq \sqrt{2}.$$

Now assume that $y < M_1(n) + M_2(n)$.

$$\begin{aligned} C_{A_1}/C^* &\leq 2(M_2(n) + y)/(M_1(n) + M_2(n) + y) \\ &= 2 - 2M_1(n)/(M_1(n) + M_2(n) + y) \\ &\leq 2 - M_1(n)/(M_1(n) + M_2(n)) \\ &\leq \sqrt{2}. \end{aligned}$$

By Lemma 8.2.1, A_1 is a best possible on-line algorithm. □

8.2.2 The three-machine case

For three machines, we first consider the following instance. Two jobs with processing time 1 appear first. If the two jobs are assigned together to a machine, the 3rd

job with the same processing time comes and it is claimed to be the last. Then the produced makespan is twice that the optimal makespan. If the two jobs are assigned to different machines, then two jobs with processing time 2 arrive one by one and the fourth is the last. In this case, it will result in a makespan no better than 3 while the optimal value is just 2. Hence the following lemma holds.

Lemma 8.2.4 There does not exist any on-line algorithm with competitive ratio less than 3/2 for Problem LL when $m = 3$.

For the three-machine case, different from the two-machine case, a very simple algorithm is proved to be best possible.

Algorithm LSw (*List Scheduling* with a *waiting* machine)

Step 0. Set Machine 3 to be a waiting machine.

Step 1. If the incoming job is the last job, arrange it to Machine 3 and stop.

Step 2. Arrange the incoming job to the one of Machines 1 and 2 with less workload at the moment.

Note that algorithm LSw applies LS to all but the last job with two machines and schedules the last (the longest) job on the 3rd machine (the waiting machine).

Theorem 8.2.5 The worst-case ratio of algorithm LSw is not greater than 3/2. Moreover, it is a best possible on-line algorithm.

Proof. Let M_i be the workloads of the three machines in the schedule produced by algorithm LSw. Observe that the difference between M_1 and M_2 is not greater than the processing time of the last job, i.e., not greater than M_3. If the makespan C_{LSw} of the schedule is M_3, then it is optimal. Without loss of generality, assume that the makespan is determined by M_1, i.e., $C_{LSw} = M_1$. Then, $C^* \geq (M_1 + M_2 + M_3)/3 \geq 2M_1/3$. It implies that $C_{LSw}/C^* \leq 3/2$. □

8.3 On two uniform machines

In this section, we deal with the problem on two uniform machines. Denote by $1/s$ ($s \geq 1$) the speed of the slow machine. Without loss of generality, the speed of fast

machine is 1. s is called *speed ratio*. If a job J with processing time p is scheduled on a machine with speed α, then it needs time of p/α to complete.

We denote jobs and their processing times by J_t and p_t $(t = 0, 1, 2, \ldots)$, respectively. Let $M_i(t)$ denote the workload of Machine i immediately after the job J_t is assigned, for $t = 0, 1, 2, \ldots$, and $i = 1, 2$. Machine 1 is the slow machine and Machine 2 is the fast machine. Denote by C_A and C^* the values of the solutions of an algorithm A and optimal algorithm, respectively. Our objective is to minimize the makespan.

8.3.1 Lower bounds

Lemma 8.3.1 For any semi on-line algorithm A, no deterministic algorithm can have competitive ratio lower than

$$R = \begin{cases} 1 + 1/s, & if \quad s \geq 1 + \sqrt{3}; \\ (1 + \sqrt{3})/2, & if \quad 2 \leq s < 1 + \sqrt{3}; \\ 1/2 + \sqrt{1/4 + 1/s}, & if \quad 1.46557 < s < 2; \\ s, & if \quad 1 < s \leq 1.46557. \end{cases}$$

Proof. We prove the lemma with the following cases.

Case a. $s \geq 1 + \sqrt{3}$. We start with the job J_1 of processing time 1. If J_1 is scheduled on the fast machine M_2, then the last job of size s arrives. Clearly, $C_A = \min\{s^2, 1 + s\} = 1 + s$ and $C^* = s$. If J_2 is scheduled on the slow machine M_1, the last job is of size 1. We have $C^* = 2$ and $C_A = s$. Since $s \geq 1 + \sqrt{3}$, we get $s^2 - 2s - 2 \geq 0$, or equivalently, $1 + 1/s \leq s/2$. Therefore, $R_A \geq 1 + 1/s$.

Case b. $1 + \sqrt{3} > s \geq 2$. Let $x = \frac{s^2 - s + \sqrt{3}s}{2 + 2s - s^2}$. In this case, $2 + 2s - s^2 > 0$ and $2x > s$. Again, we start with first job of size 1. Note that if the first job goes to the fast machine M_2, then the last job of size s arrives, which implies that $R_A \geq 1 + 1/s \geq (1 + \sqrt{3})/2$. Now assume that it goes to the slow machine M_1. Then next job of size x comes. If it goes to the slow machine too, then the last job is of size x. Clearly $C_A = s + sx$ while $C^* = 2x$. If it goes to the fast machine, then the last job is of size $xs + s$. We have $C^* = xs + s$ and $C_A = xs + s + x$, from the definition of x. Thus, $R_A = 1 + \frac{x}{s(x+1)}$. It is easy to see that $1 + \frac{x}{s(x+1)} = \frac{1+\sqrt{3}}{2}$.

Case c. $\beta \approx 1.46557 < s < 2$, where $\beta \approx 1.46557$ is a solution of the inequality of $s^3 - s^2 - 1 > 0$. If the first job goes to the fast machine, then the last job has size of s. Thus, in this case we always have $R_A \geq \min\{s, 1 + 1/s\}$. Consider now the case that the first job is assigned on the slow machine M_1. Let $x = (s + \sqrt{s^2 + 4s})/2$. If the second job of size x is assigned to the slow machine, then the last job is of size x. Since $xs > 1 + x$, we have $C^* = xs$. $C_A = s + xs$, thus, $R_A \geq 1 + 1/s$. If the second job goes to the fast machine, the last job of size $s + xs$ comes. Clearly, $C^* = s + xs$ and $C_A = x + s + xs$. Since $s + s(x + xs) > x + s + xs$ and $x > 1/(s-1)$, we only need to show $s^3 - s - 1 > 0$. It holds clearly, since $s^3 > s^2 + 1 > s + 1$. Thus, $R_A = \frac{x+s+xs}{s+xs} = \frac{\sqrt{s^2+4s}+s}{2s} = 1/2 + \sqrt{1/4 + 1/s} < s$.

Case d. $1 < s \leq \beta \approx 1.46577$, where β is a solution of inequality $s^3 \leq s^2 + 1$. If the first job of size one is assigned to the fast machine, the next (also the last) job is of size s. We have $R_A = \min\{s, 1 + 1/s\}$. Assume that the first job goes to the slow machine. The second job has size of x, where $x = \frac{\sqrt{5s^2+2s+1}+2s^2-s-1}{2(2+s-s^2)}$. From $s^3 \leq s^2 + 1$, we get that $xs \leq 1 + x$. It is clear that $2 + s - s^2 > 0$ when $s < 2$. If the second job is scheduled on the slow machine, the last job with size of x arrives. $C_A = s(1 + x)$ and $C^* = \max\{xs, 1 + x\} = 1 + x$. It implies that $R_A \geq s$. If the second job is assigned on the fast machine, the last job with size $xs + s$ appears. $C^* = xs + s$. $C_A = \min\{x + xs + s, s + s(s + xs)\} = x + s + xs$. Then $R_A = 1 + \frac{x}{s(1+x)}$. Since $x \leq 1/(s - 1)$ and $s^3 \leq s^2 + 1$, we have $R_A \geq s$. □

8.3.2 Upper bounds

In this subsection, we will present algorithms for this semi on-line problem, which depend on the value of s. We first consider a naive algorithm which only uses the fast machine.

Theorem 8.3.2 The competitive ratio is at most $1 + 1/s$.

Proof. By scheduling all the jobs to the fast machine, $C_A = \sum_{i=1}^{n} p_i$, where p_i is the processing time of job J_i. On the other hand, $(1 + 1/s)C^* \geq \sum_{i=1}^{n} p_i = C_A$. Then the theorem follows. □

For $s < 2$, we present an improved algorithm below. Let $\alpha(s) = \frac{-(s+1)+\sqrt{5s^2+2s+1}}{2s}$. Then $s\alpha(s) < 1$ as $s < 2$. Let $\beta(s) = \frac{\alpha(s)}{1-s\alpha(s)}$.

Algorithm $A_\alpha(s)$:

Step 0. Let $j = 0$.

Step 1. If the next job J_{j+1} is the *last* job, assign it to Machine 2 and stop. Otherwise, go to Step 2.

Step 2. If $M_2(j) + p_{j+1} > \alpha(s)(M_1(j) + sp_{j+1})$, assign J_{j+1} to Machine 1; otherwise assign it to Machine 2. Let $j = j + 1$, go to Step 1.

Observation 8.3.3 In the schedule produced by algorithm $A_\alpha(s)$, $M_2(j) \leq \beta(s)M_1(j)$ holds for $0 \leq j \leq n-1$, where n is the number of jobs.

Proof. Since $M_1(0) = M_2(0) = 0$, the property holds in the beginning of the sequence. Therefore we can assume that $j \geq 1$. As long as Machine 2 does not receive jobs, the property is true. Let J_t be the last job assigned to Machine 2 at a given moment (after the arrival of j jobs). From the algorithm, we have $M_2(t-1) + p_t \leq \alpha(s)(M_1(t-1) + sp_t)$, since J_t was assigned to Machine 2. Note that $p_t \leq M_2(t) = M_2(j)$, and the load of machine 1 is non-decreasing in time. Thus

$$M_2(j) = M_2(t) = M_2(t-1) + p_t \leq \alpha(s)(M_1(t-1) + sp_t) \leq \alpha(s)(M_1(j) + sM_2(j)). \tag{8.2}$$

Since $s\alpha(s) < 1$, so we have $M_2(j) \leq \frac{\alpha(s)}{1-s\alpha(s)}M_1(j)$. □

Obviously, Observation 8.3.3 still holds for $M_1(j)$ and $M_2(j)$ when $s < 2$.

Theorem 8.3.4 The competitive ratio of algorithm $A_\alpha(s)$ is at most $R(s) = \frac{s-1+\sqrt{5s^2+2s+1}}{2s} = 1 + \alpha$.

Proof. Let y be the processing time of the last job J_n. Then the cost of the algorithm $C_{A_\alpha(s)}$ is $\max\{M_1(n) = M_1(n-1), M_2(n) = M_2(n-1) + y\}$. Consider the following two cases.

Case 1. $C_{A_\alpha(s)} = M_1(n-1)$. Let x be the processing time of the last job assigned to Machine 1, and t its index. From the algorithm, we have $M_2(n-1)+x \geq$

$M_2(t-1)+x > \alpha(s)M_1(t-1) = M_1(n-1)$ (otherwise this job would have been scheduled on Machine 2). Since J_n is a job with the longest processing time, $y \geq x$. We get $\alpha(s)M_1(n-1) < M_2(n-1)+y$. Then

$$\begin{aligned} C^* &\geq \frac{M_1(n-1)/s + M_2(n-1) + y}{1+1/s} \\ &= \frac{M_1(n-1) + s(M_2(n-1)+y)}{s+1} \\ &\geq \frac{1+s\alpha(s)}{s+1}(M_1(n-1)) \end{aligned}$$

Then $R_{A_\alpha(s)} = \frac{C_{A_\alpha(s)}}{C^*} \leq \frac{s+1}{1+s\alpha(s)} = \frac{s-1+\sqrt{5s^2+2s+1}}{2s}$.

Case 1. $C_{A_\alpha(s)} = M_2(n-1)+y$. If $y \geq M_1(n-1)+sM_2(n-1)$, then

$$\begin{aligned} R_{A_\alpha(s)} &\leq \frac{M_2(n-1)+y}{y} \\ &= 1 + M_2(n-1)/y \\ &\leq 1 + \frac{M_2(n-1)}{M_1(n-1)+sM_2(n-1)} \\ &= 1 + \frac{M_2(n-1)}{sM_2(n-1) + \frac{1-s\alpha(s)}{\alpha(s)}M_2(n-1)} \\ &= 1+\alpha(s). \end{aligned}$$

Now we consider the case that $y < M_1(n-1)+sM_2(n-1)$. Then

$$\begin{aligned} R_{A_\alpha(s)} &\leq \frac{M_2(n-1)+y}{M_2(n-1)+y+M_1(n-1)/s}(1+1/s) \\ &= 1+1/s - \frac{(1+1/s)M_1(n-1)/s}{M_1(n-1)/s + M_2(n-1) + y} \\ &\leq 1+1/s - \frac{(1+1/s)M_1(n-1)/s}{s(1/s+1)M_2(n-1) + (1/s+1)M_1(n-1)} \\ &= 1+1/s - \frac{M_1(n-1)/s}{sM_2(n-1)+M_1(n-1)} \\ &\leq 1+1/s - \frac{M_1(n-1)/s}{(s\frac{\alpha(s)}{1-s\alpha(s)}+1)M_1(n-1)} \\ &= 1+1/s - \frac{1-s\alpha(s)}{s} \\ &= 1+\alpha(s). \end{aligned}$$

□

It is easy to see that $R(s) < 1 + 1/s$ when $s \leq 2$. Thus the algorithm $A_\alpha(s)$ improves the upper bound. It is worthy to note that when $s \geq 1 + \sqrt{3}$, we have presented a best possible algorithm. Algorithm $A_\alpha(s)$ is also optimal for identical machines, i.e. when $s = 1$.

8.4 Maximizing the minimum completion time on two uniform machines

In this section we deal with the objective maximizing the minimum completion time on two uniform machines.

We denote jobs and their processing times by J_t and p_t $(t = 0, 1, 2, \ldots)$, respectively. Let $M_i(t)$ denote the workload of Machine i immediately after the job J_t is assigned, for $t = 0, 1, 2, \ldots$, and $i = 1, 2$. Denote by $1/s$ $(s \geq 1)$ the speed of the slow machine, without loss of generality, the speed of fast machine is 1. Machine 1 is the slow machine and Machine 2 is the fast machine. Denote by C_A and C^* the values of the solutions of an algorithm A and optimal algorithm, respectively. Our objective is to maximize the minimum completion time. Let $R(s) = \frac{3s+1+\sqrt{5s^2+2s+1}}{2s+2}$. We show that nearly for every value of s, $R(s)$ is the exact competitive ratio for the speed ratio s.

8.4.1 Upper bound

To define and analyze the algorithm, we use two auxiliary functions of s, $\alpha(s)$ and $\beta(s)$. Let $\alpha(s) = \frac{\sqrt{5s^2+2s+1}-(s+1)}{2s^2}$, and let $\beta(s) = \frac{\alpha(s)}{1-s\alpha(s)}$. Note that $\alpha(s)$ is the positive solution of $s^2\alpha(s)^2 + (s+1)\alpha(s) - 1 = 0$, which is equivalent to $(1+s\alpha(s))/(\alpha(s)(s+1)) = (1)/(1-s\alpha(s))$. The above choice of $\alpha(s)$ give the value $R(s)$ to those two expressions. Using simple algebra, it is possible to show that $0 < \alpha(s) < 1/(s+1)$ and $0 < \beta(s) < 1$. We are now ready to present the algorithm.

Algorithm $TM(s)$

Step 0. Let $j = 0$.

Step 1. If the next job J_{j+1} is the *last* job, assign it to Machine 2 and stop. Otherwise, go to Step 2.

Step 2. If $M_2(j) + p_{j+1} > \alpha(s)(M_1(j) + sp_{j+1})$, assign J_{j+1} to Machine 1; otherwise assign it to Machine 2. Let $j = j + 1$, go to Step 1.

Note that Observation 8.3.3 is still valid for algorithm $TM(s)$.

Theorem 8.4.1 The competitive ratio of algorithm $TM(s)$ is at most $R(s) = \frac{3s+1+\sqrt{5s^2+2s+1}}{2s+2}$.

Proof. Let y be the processing time of the last job J_n. Then the cost of the algorithm C_{TM} is $\min\{M_1(n) = M_1(n-1), M_2(n) = M_2(n-1) + y\}$. Consider the following two cases.

Case 1. $C_{TM} = M_2(n-1)+y$. Since $M_1(n) \geq M_2(n)$, machine 1 is non-empty. Let x be the processing time of the last job assigned to Machine 1, and t its index. From the algorithm, we have $M_2(n-1)+x \geq M_2(t-1)+x > \alpha(s)M_1(t-1) = M_1(n-1)$ (otherwise this job would have been scheduled on Machine 2). Since J_n is a job with the longest processing time, $y \geq x$. We get $M_1(n-1) < \frac{1}{\alpha(s)}(M_2(n-1)+y)$, then

$$\begin{aligned} C^* &\leq \frac{M_1(n-1)/s + M_2(n-1) + y}{1 + 1/s} \\ &= \frac{M_1(n-1) + s(M_2(n-1) + y)}{s+1} \\ &\leq \frac{1/\alpha(s) + s}{s+1}(M_2(n-1) + y) \\ &= \frac{3s + 1 + \sqrt{5s^2 + 2s + 1}}{2s+2} C_{TM} \end{aligned}$$

Case 2. $C_{TM} = M_1(n-1)$. If $y \geq M_1(n-1) + sM_2(n-1)$,

$$\begin{aligned} C^* = M_1(n-1) + sM_2(n-1) &\leq \left(1 + s\frac{\alpha(s)}{1 - s\alpha(s)}\right) M_1(n-1) = \\ \frac{1}{1 - s\alpha(s)} C_{TM} &= \frac{3s + 1 + \sqrt{5s^2 + 2s + 1}}{2s+2} C_{TM}. \end{aligned}$$

Now assume that $y < M_1(n-1) + sM_2(n-1)$.

$$\begin{aligned} C^* &\leq \frac{M_1(n-1)/s + M_2(n-1) + y}{1 + 1/s} \\ &\leq \frac{M_1(n-1)/s + M_2(n-1) + M_1(n-1) + sM_2(n-1)}{1 + 1/s} \\ &= M_1(n-1) + sM_2(n-1) \\ &\leq \frac{1}{1 - s\alpha(s)} M_1(n-1) \\ &= \frac{3s + 1 + \sqrt{5s^2 + 2s + 1}}{2s + 2} C_{TM}. \end{aligned}$$

□

8.4.2 Lower bound

In this subsection, we give instances to show a lower bound which matches the upper bound we have given in the previous subsection for almost every value of s.

Lemma 8.4.2 For any on-line algorithm A, the competitive ratio $R_A \geq \frac{3s+1+\sqrt{5s^2+2s+1}}{2s+2}$, when $s > (9 + 5\sqrt{5})/2$ or $1 \leq s \leq 2 + 2\sqrt{2}$. Otherwise, the competitive ratio is at least 2.

Proof. To prove the lemma we consider three cases $s = 1$,$1 < s < 2$ and $s \geq 2$. The first case can be combined into the second, we give it separately to avoid confusion when referring to "the fast machine" and "the slow machine". Since the sequence must notify which job is last, we specify it where necessary. If not specified, it means the job is not last.

Case a: $s = 1$. We need to show a lower bound of $1 + \sqrt{2}/2$. The first two jobs have processing time 1.

sub-case a.1 They are assigned to different machines. Then the *last* job which has processing time 2 arrives. $C^* = 2$, $C_A = 1$. Thus $R_A \geq 2$.

sub-case a.2 They are assigned to the same machine, without loss of generality, assume it is M_1. The next job with processing time $x = \sqrt{2}$ arrives.

sub-case a.2.1 It is assigned to M_1, then the *last* job has processing time x. We know that $C^* = x + 1$ and $C_A = x$. Thus $R_A \geq \frac{x+1}{x} = 1 + \sqrt{2}/2$.

sub-case a.2.2 It is assigned to M_2, we add a *last* job of processing time $2+x$. Clearly $C^* = 2+x$ and $C_A \leq 2$. Which implies that $R_A \geq \frac{2+x}{2} = 1+\sqrt{2}/2$.

Case b. $1 < s \leq 2$. We start with a job of size $p_0 = 1$. We use a similar pattern for both cases (each cases relates to the machine where the first job is assigned). We use a parameter $0 < z < 1$ (the value of z depends on the machine which received the first job). For $k = 1, \cdots, K+1$, we give the jobs $p_k = z(1+z)^{k-1}$. Clearly, the sum of all jobs up to job j, is $(1+z)^j$. After this sequence we give another *last* job of size $P_{K+2} = P_{K+1}$. The total sum of jobs is $S_K = (2z+1)(z+1)^K$. Note that the last job is indeed the largest since starting the third job, the sequence is non-decreasing. We would like to show that if K is large enough, then the optimal schedule is almost balanced. We start with the following claim.

Claim 8.4.3 Given the set $U = \{1, z, z(1+z), \cdots, z(1+z)^j\} = \{p_0, \cdots, p_{j+1}\}$. Given a number y such that $y \in [0, (1+z)^{j+1}]$ (i.e. y is positive and smaller or equal than the sum of all elements in U), then we can choose a subset of W, $W \subseteq U$ such that $|y - \sum_{i \in W} p_i| \leq 1$.

Proof. By induction. If $j = 0$, then $y \in [0, \cdots, 1+z]$ and the possible sums are $\{0, z, 1, 1+z\}$. The large gap between two consecutive elements is $\max\{z, 1-z\} \leq 1$, therefore it is possible to choose the required subset. Given $j > 0$, we choose the subset in the following way. If $y \geq z(1+z)^j$, we choose $z(1+z)^j$ and set $y' = y - z(1+z)^j$, we get $0 \leq y' \leq (1+z)^j$. Otherwise we do not choose $z(1+z)^j$ and set $y' = y$. In both cases we can continue using the inductive hypothesis for $j-1$. Note that in the second case $y = y' < z(1+z)^j \leq (1+z)^j$ since $z \leq 1$. □

Next we show how to use this claim to get a convenient off-line schedule.

Claim 8.4.4 Given some $\varepsilon > 0$, it is always possible to choose K such that we can always assign the jobs $p_1, \ldots, p_{K+2}$ such that the following is satisfied. $S_K/(1+1/s) \leq C^*(1+\varepsilon)$. I.e. the schedule is almost flat.

Proof. In a flat schedule, the loads both machines would be $S_K/(1+1/s)$. We always assign the last job to the fast machine. The amount that the fast machine

still needs to achieve a flat schedule is

$$\begin{aligned}\frac{S_K}{(1+\frac{1}{s})} - P_{K+2} &= \frac{s(2z+1)(z+1)^K}{s+1} - z(z+1)^K \\ &= (z+1)^K\left(\frac{s(2z+1)}{s+1} - z\right) \\ &= \frac{(z+1)^K(zs+s-z)}{s+1}.\end{aligned}$$

This value is positive since $z \leq 1 < s$. Also since $S_K/(1+1/s) - P_{K+2} \leq (z+1)^{K+1}$, the difference satisfies that conditions of the previous claim, and there exists a subset of jobs we can add to the fast machine getting load of at least $S_K/(1+1/s) - 1$ and at most $S_K/(1+1/s) + 1$. This means that the load of the slow machine which receives all other jobs is at least $S_K/(1+1/s) - s$ and at most $S_K/(1+1/s) + s$. We get $C^* \geq S_K/(1+1/s) - s$. Since S_K can be arbitrarily large, we choose K such that $S_K \geq (s+1)(1+1/\varepsilon)$. □

We can proceed with the proof now. If the first job is assigned to the fast machine we use $z(s) = R(s)/s - 1$, where $R(s) = \frac{3s+1+\sqrt{5s^2+2s+1}}{2s+2}$. Since $s < R(s) < s+1$ for the case $s < 2$. We get that $sz(s) = R(s) - s$, so $0 < z(s) \leq sz(s) < 1$.

Choose ε and let K be an integer satisfying claim 8.4.4. We give the jobs. If some job p_{j+1} is assigned to the slow machine it has the load $sz(1+z)^j$, and the load in the fast machine is $(1+z)^j$. Thus we stop the sequence and give the last job of size $s(1+z)^{j+1}$ (Clearly this is the largest job, since it is larger than the sum of all previous jobs). Since $C^* = s(1+z)^{j+1}$, and $C_A = (1+z)^j$ (using $sz < 1$), which gives the competitive ratio $s(1+z) = R$.

Upon arrival of job $K+2$ we have $C^* \geq S_K/(1+1/s)/(1+\varepsilon)$. We get a load of $sz(1+z)^K < (1+z)^K$ on the slow machine, and $(z+1)^{K+1}$ on the fast. So, the ratio $R_A = \frac{C^*}{C_A} \geq \frac{2z+1}{z(s+1)}/(1+\varepsilon)$.

We get $R_A = \frac{2R/s-1}{(s+1)(R/s-1)} = \frac{2R-s}{(s+1)(R-s)}$. In the following, we will show that $R_A \geq R$. This is equivalent to showing $2R - s \geq (s+1)(R-s)R = R^2(s+1) - s(s+1)R$. From the definition of R. We get $2R - s \geq (3s+1)R - s - s(s+1)R$, this gives $2R - s \geq R(-s^2+2s+1) - s$ or $R(s^2-2s+1) \geq 0$ which clearly holds.

If the first job is assigned to the slow machine we use $z(s) = R(s) - 1$, Since $s < R(s) < 2$ for the case $s < 2$. We get that $0 < z(s) < 1 < s$.

Choose ε and let K be an integer satisfying claim 8.4.4. We give the jobs. If some job p_{j+1} is assigned to the fast machine it has the load $z(1+z)^j$, and the load of the slow machine is $s(1+z)^j$. Thus we stop the sequence and give a last job of size $s(1+z)^{j+1}$ Since $C^* = s(1+z)^{j+1}$, and $C_A = s(1+z)^j$ (using $z < s$), which give the competitive ratio $1 + z(s) = R(s)$.

After the arrival of job $K+2$ we have $C^* \geq S_K/(1+1/s)/(1+\varepsilon)$. We get a load of $z(1+z)^K$ on the fast machine, and $s(z+1)^{K+1}$ on the slow one. So, the ratio $R_A = \frac{C^*}{C_A} \geq \frac{s(2z+1)}{z(s+1)}/(1+\varepsilon)$.

We get $R_A = \frac{s(2R-1)}{(s+1)(R-1)} = R(s)$ due to the definition of $R(s)$ as the solution of $(s+1)R^2 - (3s+1)R + s = 0$.

Case c. $2 < s \leq 2+2\sqrt{2}$ and $s > (9+5\sqrt{5})/2$. We consider the following instance: given the sequence of jobs $\{J_1, J_2, J_3, J_4\}$ with size of $\{1, z, x(1+z), x(1+z)\}$, respectively. Where $x = R(s) - 1$, again let $R(s) = \frac{3s+1+\sqrt{5s^2+2s+1}}{2s+2}$. It is worthy to note that $R(s)$ is a non-decreasing function of s. $2 \leq R(s) \leq (3+\sqrt{5})/2$ in the case that $s \geq 2$. For simplicity, we use R instead of $R(s)$ without cause any confuse. The value of z depends on s,

$$z = \begin{cases} \frac{2x+1}{s-2x}, & if \quad 2 < s \leq 2+2\sqrt{2}; \\ \frac{s+1}{2R-1} - 1, & if \quad s > (9+5\sqrt{5})/2. \end{cases}$$

In the following we will prove that $z \geq x$ in this case. Firstly, we consider $2 < s \leq 2+2\sqrt{2}$. $z = \frac{2x+1}{s-2x} = \frac{2R-1}{s+2-2R}$. $s+2-2R > 0$, since $s^3 \geq 3s+2$ if $s \geq 2$. In order to show $z \geq R-1 = x$, we only need to show that $2R-1 \geq s(R-1)-2(R-1)^2$, which is equivalent to $s \leq 2(R-1) + \frac{1}{R-1} + 2$. It holds clearly, since $2R + \frac{1}{R-1} \geq 2+2\sqrt{2}$.

Now we consider $s > (9+5\sqrt{5})/2$. From the definition of R, $(s+1)R^2 - (3s+1)R + s = 0$, we have $z = \frac{s+1}{2R-1} - 1 = \frac{s}{R(R-1)} - 1$. In order to show $z \geq x = R-1$, we only need to prove that $s \geq 2R^2 - R - 1$. This clearly holds, since $R \leq (3+\sqrt{5})/2$, then $2R^2 - R - 1 \leq (9+5\sqrt{5})/2$.

We can proceed the proof of this case now. We start with job J_1 of size 1.

Case c.1. J_1 goes to M_2. Then the next job is claimed to be the last job of size s. We have $C^* = s$ and $C_A = 1$, thus $R_A \geq \frac{C^*}{C_A} = s > R$, when $s < 2$.

Case c.2. J_1 goes to M_1. The next job J_2 of size z comes.

Case c.2.1 J_2 goes to M_2. Then the next job of size $s(1+z)$ is claimed to be the last job. We have $C^* = s(1+z)$, $C_A = \max\{s, z\}$. Thus $R_A \geq \max\{1+z, s+s/z\} \geq R$, from $z \geq x = R-1$.

Case c.2.2 J_2 goes to M_1. Then next job J_3 of size $x(1+z)$ comes.

Case c.2.2.1 J_3 goes to M_2. Then the last job of size $s(1+z)+sx(1+z)$ comes. Clearly, $C^* = s(1+z)+sx(1+z)$ and $C_A = s(1+z)$, since $s \geq R > x$. Thus $R_A \geq 1+x = R$.

Case c.2.2.2 J_3 goes to M_1. The last job of size $x(1+z)$ comes. Clearly, $C_A = x(1+z)$. Now we consider the optimal value C^*. In the case $2 < s \leq 2+2\sqrt{2}$, assign job J_2 of size z to slow machine, and all the other jobs to fast machine. Then $C^* = \min\{sz, 1+2x(1+z)\}$, from the definition of $z = \frac{2x+1}{s-2x}$, we have $sz = 1+sx(1+z)$. Then $R_A = \frac{sz}{x(1+z)} = \frac{s(2x+1)}{x(s+1)}$. Note that $s(2x+1) = (s-1)x+s+x(s+1)$. From the definition of x, $(s+1)x^2-(s-1)x-s = 0$, $s(2x+1) = (s+1)x^2+x(s+1) = (1+x)x(s+1)$. Then $R_A = \frac{s(2x+1)}{x(s+1)} = 1+x = R$. Now we consider the optimal value C^* in the case $s > (9+5\sqrt{5})/2$. Just assign the first job of size 1 to slow machine and all the other jobs to the fast machine. Therefore, $C^* = \min\{s, 2x(1+z)+z\}$. From the definition of z, we have $s = 2x(1+z)+z$. Then, $R_A = \frac{s}{x(1+z)} = \frac{s}{(R-1)(1+s/(R(R-1))-1)} = R$.

Case d. $2+2\sqrt{2} < s \leq (9+5\sqrt{5})/2$. In this case we only give an instance to show that $R_A \geq 2$. We start with a job of size 1. If it goes to M_2, then the last job of size s comes. $R_A \geq s \geq R(s)$. Assume the first job is assigned to M_1, the next job is also of size 1. If the second job is assigned to M_1, then the last job of size 1 appears. Thus $C_A = 1$ and $C^* = \min\{2, s\} = 2$, $R_A \geq 2$. If the second job is assigned to M_2, then the last job is of size $2s$. Clearly, $C_A = s$ and $C^* = 2s$. Then $R_A \geq 2$. □

For $s > (9+5\sqrt{5})/2$ or $1 \leq s \leq 2+2\sqrt{2}$, the algorithm $TM(s)$ is a best possible on-line algorithm.

Chapter 9

Conclusions

In this chapter, we summarize in Section 9.1 the results presented in this thesis. Some open problems and future research are given in Section 9.2.

9.1 Summary of our results

In this thesis we have studied on-line scheduling of parallel jobs on specific network topologies. In Chapter 3, the UET job systems with dependencies on 2-dimensional mesh were considered to minimize the makespan. The dependencies between jobs are unknown to the on-line scheduler. Thus, the jobs appear in an order depending on what strategies the scheduler used to assign the current available jobs. We showed a lower bound of 3.859 and presented a 5.25-competitive algorithm. It has significantly improved the previous known lower bound of 3.25 and the previous known upper bound of 46/7. We also considered the rotated 2-dimensional mesh, in which the rotation of 90^0 of jobs is feasible. A lower bound 3.535 was proved and an on-line algorithm with competitive ratio of at most 4.25 was derived (see Table 9.1).

Table 9.1: Bounds for $P|$ *on-line-prec, 2-d mesh*$, p_j = 1|C_{\max}$.

rotation	*lower bound*	*upper bound*
No	3.859	5.25
Yes	3.535	4.25

In Chapter 4, we dealt with the problem of on-line parallel jobs scheduling on PRAM. Jobs arrive over list, i.e., jobs appear one by one in an arbitrary order. The scheduler must schedule the incoming job upon its arrival without any knowledge about future jobs. Once a decision has been made (a job is scheduled) it is not allowed to change later. We showed that the *waiting strategy* is necessary. The proposed algorithm *DW* (Dynamic Waiting) classifies the jobs and decides the waiting time (if needed) based on the current schedule. Then we proved that the competitive ratio of the algorithm *DW* is exactly 7. Then we turned to some special cases. Firstly, we studied the case that jobs arrive in non-increasing order of processing times. The algorithm *Greedy* with competitive ratio of at most 2 was presented. Next, we analyzed the algorithm *Greedy* for the case that jobs arrive in non-increasing order of job size. It was proven that the competitive ratio is in between 2.75 and 2.5. Moreover, we considered the problem where the longest processing time of jobs is known in advance. An on-line algorithm with competitive ratio of 4 was presented. Finally, we considered the problem allowing preemption. The processing of a job can be interrupted in any time and can resume later. However, a preemptive schedule for the incoming job must be made before the next job appears. We presented a semi greedy algorithm with competitive ratio of $2 - 1/m$. The algorithm works also for the line network topology, and the bound remains true.

In Chapter 5, we studied the malleable parallel jobs scheduling problem, in which, the processing time of a job is a function of the number of machines allotted to it. The number of machines to execute the parallel job is not fixed but must be determined before execution, and it does not change until the completion of the job. We explored the monotonic variant of malleable parallel jobs scheduling on a 2-dimensional mesh. For the case when the number of machines is sufficiently large, an asymptotic fully polynomial time approximation scheme (AFPTAS) was provided.

In Chapter 6, we focused on the objective maximizing the throughput. The problem of scheduling n independent parallel jobs on hypercubes were investigated. A parallel job is required to be scheduled on a subcube of machines. All jobs are available at the beginning. Each job is associated with a due date. The general

problem where the job processing times are arbitrary is obviously NP-hard. Then we provided an optimal polynomial algorithm for the unit processing time job system. We also considered scheduling parallel jobs on two identical machines, where each job has a release time and a due date. An optimal polynomial algorithm for the unit processing time job system was also devised.

We then addressed some extended models on classical scheduling, where one job can be processed by exactly one machine. In Chapter 7, we explored the scheduling problem on parallel machines with extendable working times. The on-line algorithm H_x, designed by Speranza and Tuza [122], was carefully analyzed for $m = 2, 3, 4$. The best choices of parameter x were determined and the competitive ratios were shown. It implied that H_x is a best possible on-line algorithm for two machines. A new algorithm A_α was proved to have a competitive ratio of $7/6$, which is better than H_x for three machines. Unfortunately, algorithm A_α does not work for $m > 3$. For $m = 3$ and 4, we presented a lower bound $(17 + \sqrt{577})/36$ and $9/8$, respectively (see Table 9.2).

We paid much attention to the more general case that the machines have different regular working times. Under the assumption (7.1), we analyzed the LS algorithm more carefully and proved the competitive ratios for each m and each collection $\mathcal{B}$ of regular working times. The ratios differ for an even number of machines and an odd number of machines. It also showed that the competitive ratio of LS for the identical machines is exactly $5/4$ when m is even and $5/4 - 1/(4m^2)$ when m is odd (See Table 9.3). We then presented an improved on-line algorithm for $m = 2$ and $m = 3$. We also discussed the problem without the assumption (7.1). A lower bound $6/5$ for the overall competitive ratio is presented. We proved the competitive ratio of algorithm LS for the two-machine case and presented an improved on-line algorithm. Finally, we studied a variant for parallel jobs, in which jobs may require more than one machine simultaneously. We considered the off-line version of this problem and showed an upper bound of 2.

In Chapter 8, we studied the classical scheduling problem where some partial information is known beforehand. Jobs arrive one by one, as classical on-line scheduling. However, it is known that the last job has the longest processing time (but

Table 9.2: Bounds for extendable scheduling on small number of machines

m	*tight bound of* H_x	*lower bound*	*upper bound of* A_α
2	7/6	7/6	7/6
3	11/9	$(17+\sqrt{577})/36 \approx 41/36$	7/6
4	19/16	9/8	

Table 9.3: Bounds of list scheduling for unequal extendable scheduling

m	*competitive ratio*	*overall competitive ratio*
m is even	$1+\frac{mb_{min}}{4\sum_{j=1}^{m} b_j}$	5/4
m is odd	$1+\frac{(m^2-1)b_{min}}{4m\sum_{j=1}^{m} b_j}$	$5/4-\frac{1}{4m^2}$

the exact value is not available before it arrives). The on-line scheduler will also be informed when the last job arrives. We presented optimal on-line algorithms for $m = 2$ and 3 whose competitive ratios are $\sqrt{2}$ and 3/2, respectively. Then we considered two uniform machines, where the machines have different speeds. On-line algorithms were developed depending on the speed ratio s, which are optimal for many s. Finally, we explored the problem to maximize the minimum completion time on two uniform machines. The competitive ratios are best possible excluding one interval of s. Table 9.4 and Table 9.5 list the results, where $R(s) = (3s+1+\sqrt{5s^2+2s+1})/(2s+2)$, $R'(s) = (s-1+\sqrt{5s^2+2s+10})/(2s)$.

Table 9.4: Bounds for two uniform machines, Min-max

speed s	*Lower bound*	*Upper bound*
$s \geq 1+\sqrt{3}$	$1+1/s$	$1+1/s$
$1+\sqrt{3} > s \geq 2$	$(1+\sqrt{3})/2$	$1+1/s$
$2 > s \geq 1.46557$	$1/2+\sqrt{1/4+1/s}$	$R'(s)$
$1.46557 > s > 1$	s	$R'(s)$
$s = 1$	$\sqrt{2}$	$\sqrt{2}$

Table 9.5: Bounds for two uniform machines, Max-min

speed s	*Lower bound*	*Upper bound*
$s > (9+5\sqrt{5})/2$	$R(s)$	$R(s)$
$(9+5\sqrt{5})/2 \geq s > (2+2\sqrt{2})$	2	$R(s)$
$1 \leq s \leq (2+2\sqrt{2})$	$R(s)$	$R(s)$

9.2 Open problems and future research

Not surprisingly, there are plenty of open problems arising from our research, in particular in dealing with on-line scheduling of parallel jobs under various network topologies.

It is significant to explore on-line schedule of parallel jobs with dependencies on 2-dimensional mesh while the processing times of jobs are arbitrary. It is still open whether or not this problem allows a constant competitive ratio.

There is still a big gap between the lower bound and the upper bound of the on-line parallel jobs scheduling problem where jobs arrive over list. It is a challenge to improve them and extend the problem to other network topologies, such as line, 2-dimensional mesh, hypercube and so on. The open field in on-line parallel jobs scheduling is to investigate the cases that partial information about the input is known. We are concerned with more realistic situations where some partial information on the uncertainties would be available. The problem is how to effectively utilize the partial, but valuable, information to achieve a better schedule. It is also interesting to improve the bounds in the case that preemption is allowed.

There are also many open questions in the extendable scheduling problem. Although we have proved that the algorithm $A_2(\alpha)$ is optimal in terms of the overall competitive ratio, it is still open to find a best possible on-line algorithm for two machines in terms of the competitive ratio. It is interesting to design an algorithm with a competitive ratio better than algorithm LS for any number of machines. The problem without the assumption (7.1) becomes more difficult. We only proved the competitive ratio of LS for the two-machine case. Any improvement on the lower bounds is also of interest.

Bibliography

[1] M. Ahuja, Y. Zhu, An $O(n \log n)$ feasibility algorithm for preemptive scheduling of n independent jobs on a hypercube, *Information Processing Letters* **35** (1990), 7-11.

[2] S. Albers, Better bounds for online scheduling, *SIAM Journal on Computing*, **29** (1999), 459-473.

[3] S. Albers, Online algorithms: A survey, *Mathematical Programming*, **97** (2003), 3-26.

[4] N. Alon, Y. Azar, G.J. Woeginger, T. Yadid, Approximation schemes for scheduling on parallel machines, *Journal of Scheduling* **1** (1998), 55-66.

[5] A. Avizienis, G. C. Gilley, F. P. Mathur, D. A. Rennels, J. A. Rohr, D. K. Rubin, The star (self-testing and repairing) computer: An investigate of the theory and practice of fault-tolerant computer design, *IEEE Transactions on Computers* **20** (1971), 1312-1321.

[6] Y. Azar, O. Regev, On-line bin-stretching, *Theoretical Computer Science* **268** (2001), 17-41.

[7] B. S. Baker, D. J. Brown, H. P. Katseff, A 5/4 algorithm for two-dimensional packing, *Journal of algorithms* **2** (1981), 348-368.

[8] B. S. Baker, E. Coffman, A Two-dimensional Bin-packing Model of Preemptive, FIFO Storage Allocation, *Journal of algorithms* **3** (1982), 303-316.

[9] B. S. Baker, E. Coffman, R. Rivest, Orthogonal packings in two dimensions, *SIAM Journal on Computing* **9** (1980), 846-855.

[10] B. S. Baker, J. S. Schwarz, Shelf algorithm for two-dimensional packing problems, *SIAM J. Comput.* **12/3** (1983), 508-525.

[11] P. Baptiste, Polynomial time algorithms for minimizing the weighted number of late jobs on a single machine with equal processing times, *Journal of Scheduling* **2** (1999), 245-252.

[12] P. Baptiste, A Note on Scheduling Multiprocessor Tasks with Identical Processing times, *Computers and Operations Research* **30/13** (2003), 2071-2078.

[13] P. Baptiste, P. Brucker, S. Knust and V. G. Timkovsky, Fourteen Notes on Equal-Processing-Time Scheduling. *IBM research report*, 2002.

[14] A. Bar-Noy, R. Canetti, S. Kutten, Y. Mansour, B. Schieber, Bandwidth Allocation with Preemption, *SIAM Journal on Computing* **28/5** (1999), 1806-1828.

[15] S. Bischof, Efficient algorithm for on-line scheduling and load distribution in parallel system, Ph. D. Thesis, Institut für Informatik, Technische Universität München, 1999.

[16] S. Bischof, E.W. Mayr, On-line scheduling of parallel jobs with runtime restrictions, *Theoretical Computer Science* **268** (2001), 67-90.

[17] E. Blayo, L. Debreu, G. Mounié, D. Trystram, Dynamic load balancing for ocean circulation with adaptive meshing. *Proceedings of the 5th European Conference on Parallel Computing (Euro-Par'99)*, LNCS **1685** (1999), 303-312.

[18] J. Blazewicz, W. Cellary, R. Slowinski, J. Weglarz, Scheduling under Resource Constraints - Deterministic Models, *Annals of Operations Research*, **7** (1986), 1-359.

[19] J.Blazewicz, P. Dell'Olmo and M. Drozdowski, Scheduling multiprocessor tasks on two parallel processors, *RAIRO - Operations Research (RO)* **36/1** (2002), 37-51.

[20] J. Blazewicz, M. Drabowski, J. Weglarz, Scheduling multiprocessor tasks to minimize schedule length, *IEEE Transactions on Computers* **35** (1986), 389-393.

[21] J. Blazewicz, J. K. Lenstra, A. H. G. Rinnooy Kan, Scheduling subject to resource constraints: Classification and complexity, *Discrete Applied Mathematics* **5** (1983) 11-24.

[22] T. Bönniger, R. Esser, D. Krekel, CM-5, KSR2, Paragon XP/S: A comparative description of massively parallel computers, *Parallel Computing* **21** (1995), 199-232.

[23] D. J. Brown, B. S. Baker, H. P. Katseff, Lower bounds for on-line two-dimensional packing algorithms, *Acta Informatica* **18/2** (1982), 207-225.

[24] http://www.mathematik.uni-osnabrueck.de/research/OR/class/ maintained by P. Brucker, S. Knust.

[25] P. Brucker, Scheduling Algorithm, Springer, Berlin, 1998, 217-218.

[26] P. Brucker, S. Knust, D. Roper and Y. Zinder, Scheduling UET task systems with concurrency on two parallel identical processors, *Mathematical Methods of Operations Research* **52/3** (2000), 369 -387.

[27] B. Chen, C.N. Potts,G.J. Woeginger, A review of machine scheduling: Complexity, algorithms and approximability. *Handbook of Combinatorial Optimization, ed. D.-Z. Du and P. Pardalos*, (Volume **3**), 1998.

[28] B. Chen, A. P. A. Vestjens, Scheduling on identical machines: How good is LPT in an on-line setting?, *Operations Research Letters* **21** (1997), 165-169.

[29] B. Chen, A. Van Vliet, G. Woeginger, An Optimal Algorithm for Preemptive On-line Scheduling, *Proceedings of the Second Annual European Symposium*, ESA 1994, LNCS **855**, 300-306.

[30] G. Chen, T. Lai, Scheduling independent jobs on partitionable hypercubes, *Journal of Parallel and Distributed Computing*, **12/1** (1991), 74-78.

[31] G. Chen, T. Lai, Preemptive scheduling of independent jobs on a hypercube, *Information Processing Letters* **28** (1988), 201-206.

[32] E. Coffman, M. Garey, D. S. Johnson, A. S. Lapaugh, Scheduling file transfers, *SIAM Journal on Computing* **14** (1985), 744-780.

[33] E. Coffman, M. Garey, D. S. Johnson, R. Tarjan, Performance bounds for level-oriented two-dimensional packing problems, *SIAM Journal on Computing* **9** (1980), 808-826.

[34] E.G. Coffman, G.S. Lueker, Approximation algorithms for extensible bin packing, *Proceedings of the twelfth Annual ACM-SIAM Symposium on Discrete Algorithms* (SODA 01), 2001, 586-588.

[35] G. Confessore, P. Dell'Olmo, S. Giordani, Complexity and approximation results for scheduling multiprocessor tasks on a ring, *Discrete Applied Mathematics* **133** (2004), 29-44.

[36] M. Cosnard, D. Trystram, Parallel Algorithms and Architectures, International Thomson Publishing, 1995.

[37] J. Csirik, G. Woeginger, Shelf algorithms for on-line strip packing, *Information Processing Letters* **63** (1997), 171-175.

[38] J. Csirik, G. Woeginger, On-line packing and covering problems, *In Online algorithms – The State of Art*, LNCS **1442** (1998), 147-195.

[39] P. Dell'Olmo, H. Kellerer, M.G. Speranza, Z. Tuza, A 13/12 approximation algorithm for bin packing with extendable bins, *Information Processing Letters* **65** (1998), 229-233.

[40] P. Dell'Olmo, M.G. Speranza, Approximation algorithms for partitioning small items in unequal bins to minimize the total size, *Discrete Applied Mathematics* **94** (1999), 181-191.

[41] P. Dharwadkar, H. J. Siegel, E. K. P. Chong, A heuristic for dynamic bandwidth allocation with preemption and degradation for prioritized requests,

Proceedings of the 21th International Conference on Distributed Computing Systems (ICDCS 01), IEEE, 2001.

[42] G. Dobson, U. Karmarkar, Simultaneous resource scheduling with batching to minimize weighted flow times, *Operations Research* **37** (1989), 592-600.

[43] M. Drozdowski, Scheduling multiprocessor tasks on hypercube, *Bulletin of the Polish Academy of Sciences, Technical Science* **42** (1994), 437-445.

[44] M. Drozdowski, On complexity of multiprocessor tasks scheduling, *Bulletin of the Polish Academy of Sciences, Technical Science* **43** (1995), 381-392.

[45] M. Drozdowski, Scheduling multiprocessor tasks – an overview, *European Journal on Operations Research* **94**, 215-230, 1996.

[46] M. Drozdowski, Scheduling parallel tasks – algorithms and complexity, *Handbook of scheduling: Algorithms, Models and Performance Analysis, ed. Joseph Y.-T. Leung*, Chapter 25, CRC Press, 2004.

[47] M. Drozdowski, P. Dell' Olmo, Scheduling multiprocessor tasks for mean flow time criterion, *Computers & Operations Research*, 27(2000), 571-585.

[48] ATM Forum White Paper, ATM service categories: The benefit to the user. ATM Forum, San Francisco. http://www.atmforum.com/

[49] J. Du, J. Leung, Complexity of scheduling parallel task systems, *SIAM Journal on Discrete Mathematics* **2** (1989), 473-487.

[50] P. Dutot, G. Mounié, D. Trystram, Scheduling parallel tasks approximation algorithms, *Handbook of scheduling: Algorithms, Models and Performance Analysis, ed. Joseph Y.-T. Leung*, Chapter 26, CRC Press, 2004.

[51] L. Epstein, Lower bounds for on-line scheduling with precedence constraints on identical machines *Information Processing Letters* **76** (2000), 149-153.

[52] L. Epstein, Two dimensional packing: the power of rotation, *Proceedings of 28th International Symposium on Mathematical Foundations of Computer Science* (MFCS 03), LNCS **2747** (2003), 398-407.

[53] L. Epstein, Bin stretching revisited, *Acta Informatica* **39** (2003), 97-117.

[54] U. Faigle, W. Kern, G. Turan, On the performance of on-line algorithms for partition problems, *Acta Cybernetica* **9** (1989), 107-119.

[55] D. G. Feitelson, Scheduling parallel jobs on clusters, In Rajkumar Buyya, editor, *High Performance Cluster Computing*, Vol.1, Chapter 21, Architectures and Systems, 519-533. Prince Hall, Upper Saddle River, NJ, 1999.

[56] D. G. Feitelson, The forgotten factor: facts; on performance evaluation and its dependence on workloads, *Proceedings of the Second International EURO-PAR Conference on Parallel Processing* (Euro-Par 02), LNCS **2400** (2002), 49-60.

[57] D. G. Feitelson, L. Rudolph, Parallel Job Scheduling: Issues and Approaches, *Proceedings of the Job Scheduling Strategies for Parallel Processing*, (JSSPP 95), LNCS **949** (1995), 1-18.

[58] D. G. Feitelson, L. Rudolph, Toward convergence in job schedulers for parallel supercomputers, *Proceedings of the Job Scheduling Strategies for Parallel Processing*, (JSSPP 96), LNCS **1162** (1996), 1-26.

[59] D. Feitelson, L. Rudolph, U. Schwiegelshohn, Parallel Job Scheduling – A Status Report, *Proceedings of the Job Scheduling Strategies for Parallel Processing*, (JSSPP 04), 2004.

[60] D. G. Feitelson, L. Rudolph, U. Schwiegelshohn, K. C. Sevcik, P. Wong, *Proceedings of the Job Scheduling Strategies for Parallel Processing*, (JSSPP 97), LNCS **1291** (1997), 1-34.

[61] A. Feldmann, M.-Y. Kao, J. Sgall, and S.-H. Teng, Optimal online scheduling of parallel jobs with dependencies, *Journal of Combinatorial Optimization* **1** (1998), 393-411.

[62] A. Feldmann, J. Sgall, and S.-H. Teng, Dynamic scheduling on parallel machines, *Theoretical Computer Science* **130** (1994), 49-72.

[63] A. V. Fishkin, G. Zhang, On maximizing the throughput of multiprocessor tasks, *Theoretical Computer Science*, **302** (2003), 319-335.

[64] R. Fleischer, M. Wahl, On-line scheduling revised, *Journal of Scheduling* **3** (2000), 343-353.

[65] M. Garey, R. L. Graham, Bounds for multiprocessor scheduling with resource constraints, *SIAM Journal on Computing* **4/2** (1975), 187-200.

[66] M. Garey, D. S. Johnson, Complexity results for multiprocessor scheduling under resource constraints, *SIAM Journal on Computing* **4** (1975), 397-411.

[67] M.R. Garey, D.S. Johnson, Computers and Intractability: A guide to the Theory of NP-completeness, W.H.Freeman, San Francisco, 1979.

[68] M. N. Garofalakis, Y. E. Ioannidis, Parallel query scheduling and optimization with time- and space-shared resources, *Proceedings of the 23rd International Conference on Very Large Data Bases* (VLDB 97), 1997, 296-305.

[69] O. Gerstel, B. Li, A. McGuire, G. N. Rouskas, K. Sivalingam, Z. Zhang (eds.), Special issue on protocols and architectures for next generations optical WDM networks, *IEEE Journal Selected Area in Communications* **8** (2000).

[70] T. F. Gonzales, D. S. Johnson, A new algorithm for preemptive scheduling of trees, *Journal of ACM* **27** (1980), 287-312.

[71] T. Gormley, N. Reingold, E. Torng, and J. Westbrook, Generating adversaries for request-answer games, *Proceedings of the 11th ACM-SIAM Symposium on Discrete Algorithms* (SODA 00), 2000, 564-565.

[72] R. L. Graham, Bounds for certain multiprocessor anomalies, *Bell System Technical Journal* **45** (1966), 1563-1581.

[73] R. L. Graham, Bounds on multiprocessing anomalies, *SIAM Journal of Applied Math*, **17** (1969), 416-429.

[74] R. L. Graham, E. L. Lawlwer, J. K. Lenstra, A. H. G. Rinnooy Kan, Optimization and approximation in deterministic scheduling: a survey, *Annals of Discrete Mathematics*, **5** (1979), 287-326.

[75] D. Gusfield, Bounds for naive multiple machine scheduling with release times and deadlines, *Journal of Algorithms* **5** (1984), 1-6.

[76] L. A. Hall, D. B. Shmoys, Approximation schemes for constrained scheduling problems, *Proceedings of the 30th Annual IEEE Symposium on Foundations of Computer Science* (FOCS 89), 1989, 134-139.

[77] Y. He, G. Zhang, Semi on-line scheduling on two identical machines, *Computing* **62** (1999), 179-187.

[78] C. P. M. van Hoesel, Preemptive scheduling on a hypercube, *Technical Report* **8963/A**, Erasmus University, Rotterdam, 1989.

[79] K. S. Hong, J. Y. T. Leung, On-line scheduling of real-time tasks, *IEEE Transaction on Computing* **41** (1992), 1326-1331.

[80] A. L. Hopkins, J. M. Lala, T. B. Smith, FTMP – a highly reliable fault-tolerant multiprocessor for aircraft, *Proceedings of IEEE* **66** (1978), 1221-1239.

[81] F. Hwang, F. A. Briggs, Computer Architecture and Parallel Processing, McGraw-Hill, 1984.

[82] R. Jain, K. Somalwar, J. Werth, J. C. Browne, Scheduling parallel I/O operations in multiple bus systems, *Journal of Parallel and Distribute Computing* **16** (1992), 352-362.

[83] K. Jansen, Scheduling malleable parallel tasks: An asymptotic fully polynomial-time approximation scheme, *Algorithmica* **39** (2004), 59-81.

[84] K. Jansen, L. Porkolab Preemptive parallel task scheduling in O(n)+poly(m) time, *Proceedings of the 11th International Symposium on Algorithms and Computation* (ISAAC 00), LNCS **1969** (2000), 398-409.

[85] K. Jansen, L. Porkolab, Linear-time approximation schemes for scheduling malleable parallel tasks, *Algorithmica* **32** (2002), 507-520.

[86] K. Jansen, G. Zhang, On Rectangle Packing: Maximizing Benefits, *Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms (SODA 04)*, 2004, 204-213.

[87] K. Jansen, G. Zhang, Maximizing the number of packed rectangles, *Proceedings of the 9th Scandinavian Workshop on Algorithm Theory (SWAT 04)*, LNCS **3111** (2004), 362-371.

[88] K. Jansen, H. Zhang, Scheduling malleable tasks with precedence constraints, *manuscript*, 2004.

[89] K. Jansen, H. Zhang, An approximation algorithm for scheduling malleable tasks under general precedence constraints, *manuscript*, 2004.

[90] B. Johannes, Scheduling parallel jobs to minimize makespan, *Technical Report* **723-2001**, TU Berlin, 2001.
http://www.math.tu-berlin.de/coga/publications/techreports/

[91] D. S. Johnson, Near-optimal bin packing algorithms. *PhD thesis*, Massachusetts Institute of Technology, Department of Mathematics, Cambridge, 1973.

[92] D.S. Johnson, A. Demers, J.D. Ullman, M.R. Garey, R.L. Graham, Worst-case performance bounds for simple one-dimensional packing algorithm, *SIAM Journal on Computing* **3** (1974), 299-325.

[93] R. M. Karp, V. Ramachandran, A survey of parallel algorithms for shared memory machines, *Technical Report UCB/CSD* **88-408**, UC. Berkeley, 1988.

[94] C. Kenyon, E. Remila, A near-optimal solution to a two-dimensional cutting stock problem, *Mathematics of Operations Research* **25** (2000), 645-656.

[95] H. Kellerer, V. Kottov, M.R. Speranza, Z. Tuza, Semi on-line algorithms for the partition problem, *Operations Research Letters* **21** (1997), 235-242.

[96] C. Kenyon, E. Remila, Approximate strip packing, *Proceedings of the 37th IEEE symposium on Foundations of Computer Science* (FOCS 96), 1996, 31-36.

[97] H. Kopetz, P. Veríssimo, Real time and dependability concepts, in Mullender, S., (ed.), *Distribute Systems* Addison-Wesley and ACM Press, Reading, MA, 1994, 411-446.

[98] Y. Kopidakis, V. Zissimopoulos, An approximation scheme for scheduling independent jobs into a subcubes of hypercube of fixed dimension, *Theoretical Computer Science* **178** (1997), 265-273.

[99] M. Krunz, W. Zhao, I. Matta, Scheduling and bandwidth allocation for the distribution of archived video in VOD systems, *Journal of Telecommunications Systems, Special issue on Multimedia* **9** (1998), 335-355.

[100] F. F. Kuo, Protocols and Techniques for Data Communication Networks, Prentice Hall, New Jersey, 1981.

[101] E. L. Lawler, Sequencing to minimizing the weighted number of tardy jobs, *RAIRO Recherche Opéra.*, **10** (1976), 27-33.

[102] C. Y. Lee and X.Cai, Scheduling Multiprocessor tasks without prespecified processor allocations, *Naval Research Logistics* **45** (1998), 231-242.

[103] R. Lepere, D. Trystram, G. Woeginger, Approximation scheduling for malleable tasks under precedence constraints, *Proceedings of the 9th Annual European Symposium on Algorithms (ESA 01)*, LNCS **2161** (2001), 146-157.

[104] W. Ludwig and P. Tiwari, Scheduling malleable and nonmalleable parallel tasks, *Proceedings of the 5th ACM-SIAM Symposium on Discrete Algorithms*, (SODA 94), 1994, 167-176.

[105] E. L. Lloyd, Concurrent task system, *Operations Research* **29** (1981), 189-201.

[106] R. McNaughton, Scheduling with deadlines and loss functions, *Management Science* **6** (1959), 1-12.

[107] G. Mounié, C. Rapine, D. Trystram, Efficient approximation algorithms for scheduling malleable tasks, *Proceedings of the 11th ACM Symposium on Parallel Algorithms and Architectures (SPAA 99)* 1999, 23-32.

[108] John Noga, Steve Seiden, Scheduling Two Machines with Release Times, *Proceedings of the 7th International Integer Programming and Combinatorial Optimization Conference (IPCO 99)*, LNCS **1610**, 1999, 391-399.

[109] E. Naroska, U. Schwiegelshohn, On an on-line scheduling problem for parallel jobs, *Information Processing Letters* **81** (2002), 297-304.

[110] K. Pruhs, J. Sgall, Eric Torng, Online Scheduling, *Handbook of scheduling: Algorithms, Models and Performance Analysis, ed. Joseph Y.-T. Leung*, Chapter 15, CRC Press, 2004.

[111] E. Rahm, Dynamic load balancing in parallel database systems, *Proceedings of the Second International EURO-PAR Conference on Parallel Processing* (Euro-PAR 96), LNCS **1123** (1996), 37-52.

[112] A. Ranade, How to emulate shared memory, *Proceedings of 28th Annual Symposium on Foundations of Computer Science*, (FOCS 87), 1987, 185-194.

[113] R. Rom, M. Sidi, Multiple Access Protocols: Performance and Analysis, Springer Verlag, New York (1990).

[114] J. F. Rudin III, Improved bounds for the on-line scheduling problem, Ph. D. Thesis, The University of Texas at Dallas, 2001.

[115] H.E. Salzer, The approximation of numbers as sums of reciprocals, *Amer. Math. Monthly* **54** (1947), 135-142.

[116] I. Schiermeyer, Reverse-Fit: A 2-optimal algorithm for packing rectangles, *Proceedings of the 2nd European Symposium of Algorithms (ESA 94)*, LNCS **855** (1994), 290-299.

[117] J. Sgall, On-line scheduling on parallel machines, Ph. D. Thesis, School of Computer Science, Carnegie Mellon University, Pittsburgh, PA, 1994.

[118] J. Sgall, On-line scheduling, *In Online algorithms – The State of Art*, LNCS **1442** (1998), 196-231.

[119] D. Shmoys, J. Wein, D. Williamson, Scheduling parallel machines on-line, *SIAM Journal on Computing* **24/6** (1995), 1313-1331.

[120] D. Simchi-Levi, New worst-case results for the bin packing problem, *Naval Research Logistics* **41** (1994), 79-585.

[121] D. D. Sleator, R. E. Tarjan, Amortized efficiency of list update and paging rules, *Communications of the ACM* **28/2** (1985), 202-208.

[122] M.G. Speranza, Z. Tuza, On-line approximation algorithms for scheduling tasks on identical machines with extendable working time, *Annals of Operations Research* **86** (1999), 494-506.

[123] A. Steinberg, A strip packing algorithm with absolute performance bound 2, *SIAM Journal on Computing* **26** (1997), 401-409.

[124] D. Thaker, G. N. Rouskas, Multi-destination communication in broadcast WDM networks: A survey. *Technical Report* **2000-08**, North Caroline State University, 2000.

[125] J. Turek, J. Wolf, P. Yu, Approximation algorithms for scheduling parallelizable tasks, *Proceedings of the 4th ACM Symposium on Parallel Algorithms and Architectures* (SPAA 92), 1992, 323-332.

[126] B. Veltman, B. J. Lageweg, J. K. Lenstra, Multiprocessor scheduling with communication delays, *Parallel Computing* **16** (1990), 173-182.

[127] R. E. Wagner, R. C. Alferness, A. A. M. Saleh, M. S. Goodman, MONET: Multiwavelength optical networking, *Journal of Lightware Technology* **14** (1996), 1349-1355.

[128] G.J. Woeginger, When does a dynamic programming formulation guarantee the existence of an FPTAS? *Proceedings of the tenth Annual ACM-SIAM Symposium on Discrete Algorithms* (SODA 99), 1999, 820-829.

[129] D. Ye, G. Zhang, On-line scheduling of parallel jobs, *Proceedings of the 11th Colloquium on Structural Information and Communication Complexity*, (SIROCCO 04), LNCS **3104** (2004), 279-290.

[130] D. Ye, H. Zhang, The range assignment problem in static ad-hoc networks on metric spaces, *Proceedings of the 11th Colloquium on Structural Information and Communication Complexity* (SIROCCO 04), LNCS **3104** (2004), 291-302.

[131] D. Ye, G. Zhang, On-line extensible bin packing with unequal bin sizes, *Proceedings of the First Workshop on Approximation and Online Algorithms* (WAOA 03), LNCS **2909** (2004), 235-247.

[132] D. Ye, G. Zhang, On-line scheduling of parallel jobs with dependencies on 2-dimensional meshes, *Proceedings of the 14th International Symposium on Algorithms and Computation* (ISAAC 03), LNCS **2906** (2003), 329-338.

[133] D. Ye, G. Zhang, On-line scheduling with extendable working time on a small number of machines, *Information Processing Letters* **85** (2003), 171-177.

[134] G. Zhang, A simple semi on-line algorithm for $P2//C_{\max}$, *Information Processing Letters* **61** (1997), 145-148.

[135] G. Zhang, Absolute worst-case analysis for on-line bin packing, manuscript, 2002.

[136] G. Zhang, D. Ye, A note on on-line scheduling with partial information, *Computers and Mathematics with Applications* **44** (2002), 539-543.

[137] H. Zhang, Approximation algorithm for Min-Max resource sharing and malleable tasks scheduling, *Ph. D. Thesis, University of Kiel*, Germany, 2004.

Appendix A

List of publications during the period of Ph. D. study

1. D. Ye and G. Zhang, On-line scheduling of parallel jobs, *Proceedings of the 11th Colloquium on Structural Information and Communication Complexity (SIROCCO)*, Springer LNCS **3104**, 279-290, 2004.

2. D. Ye and H. Zhang, The range assignment problem in static ad-hoc networks on metric spaces, *Proceedings of the 11th Colloquium on Structural Information and Communication Complexity (SIROCCO)*, Springer LNCS **3104**, 291-302, 2004.

3. D. Ye and G. Zhang, On-line extensible bin packing with unequal bin sizes, *Proceedings of the First Workshop on Approximation and Online Algorithms (WAOA)*, Springer LNCS **2909**, 235-247, 2003.

4. D. Ye and G. Zhang, On-line scheduling of parallel jobs with dependencies on 2-dimensional meshes, *Proceedings of the 14th International Symposium on Algorithms and Computation (ISAAC)*, Springer LNCS **2906**, 329-338, 2003.

5. D. Ye and G. Zhang, On-line scheduling with extendable working time on a small number of machines, *Information Processing Letters* **85**, 171-177, 2003.

6. G. Zhang and D. Ye, A note on on-line scheduling with partial information, *Computers and Mathematics with Applications* **44**, 539-543, 2002.

7. J. Chlebíková, D. Ye and H. Zhang, Assign ranges in general Ad-hoc networks, manuscript.

8. D. Ye and G. Zhang, An optimal algorithm for maximizing the throughput of parallel jobs on hypercubes, manuscript.