Fall 1-27-2008

Some combinational optimization problems on radio network communication and machine scheduling

Xin Wang
New Jersey Institute of Technology

ABSTRACT

SOME COMBINATORIAL OPTIMIZATION PROBLEMS ON RADIO NETWORK COMMUNICATION AND MACHINE SCHEDULING

by
Xin Wang

The combinatorial optimization problems coming from two areas are studied in this dissertation: network communication and machine scheduling.

In the network communication area, the complexity of *distributed broadcasting* and *distributed gossiping* is studied in the setting of random networks. Two different models are considered: one is *random geometric networks*, the main model used to study properties of sensor and ad-hoc networks, where n points are randomly placed in a unit square and two points are connected by an edge if they are at most a certain fixed distance r from each other. The other model is the so-called *line-of-sight networks*, a new network model introduced recently by Frieze et al. (SODA'07). The nodes in this model are randomly placed (with probability p) on an $n \times n$ grid and a node can communicate with all the nodes that are in at most a certain fixed distance r and which are in the same row or column. It can be shown that in many scenarios of both models, the random structure of these networks makes it possible to perform distributed gossiping in asymptotically optimal time $O(D)$, where D is the diameter of the network. The simulation results show that most algorithms especially the randomized algorithm works very fast in practice.

In the scheduling area, the first problem is online scheduling a set of equal processing time tasks with precedence constraints so as to minimize the makespan. It can be shown that *Hu's* algorithm yields an asymptotic competitive ratio of $3/2$ for intree precedence constraints and an asymptotic competitive ratio of 1 for outtree precedences, and *Coffman-Graham* algorithm yields an asymptotic competitive ratio of 1 for arbitrary precedence constraints and two machines.

The second scheduling problem is the integrated production and delivery scheduling with disjoint windows. In this problem, each job is associated with a time window, and a profit. A job must be finished within its time window to get the profit. The objective is to

pick a set of jobs and schedule them to get the maximum total profit. For a single machine and unit profit, an optimal algorithm is proposed. For a single machine and arbitrary profit, a *fully polynomial time approximation scheme(FPTAS)* is proposed. These algorithms can be extended to multiple machines with approximation ratio less than $e/(e-1)$.

The third scheduling problem studied in this dissertation is the preemptive scheduling algorithms with nested and inclusive processing set restrictions. The objective is to minimize the makespan of the schedule. It can be shown that there is no optimal online algorithm even for the case of inclusive processing set. Then a linear time optimal algorithm is given for the case of nested processing set, where all jobs are available for processing at time $t = 0$. A more complicated algorithm with running time $O(n \log n)$ is given that produces not only optimal but also maximal schedules. When jobs have different release times, an optimal algorithm is given for the nested case and a faster optimal algorithm is given for the inclusive processing set case.

SOME COMBINATORIAL OPTIMIZATION PROBLEMS ON RADIO NETWORK COMMUNICATION AND MACHINE SCHEDULING

by
Xin Wang

A Dissertation
Submitted to the Faculty of
New Jersey Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of
Doctor of Philosophy in Computer Science

Department of Computer Science

January 2008

APPROVAL PAGE

SOME COMBINATORIAL OPTIMIZATION PROBLEMS ON RADIO NETWORK COMMUNICATION AND MACHINE SCHEDULING

Xin Wang

Dr. Artur Czumaj, Dissertation Co-Advisor — Date
Professor of Computer Science, University of Warwick, U.K.

Dr. Joseph Leung, Dissertation Co-Advisor — Date
Distinguished Professor of Computer Science, NJIT

Dr. James M. Calvin, Committee Member — Date
Associate Professor of Computer Science, NJIT

Dr. Alexandros Gerbessiotis, Committee Member — Date
Associate Professor of Computer Science, NJIT

Dr. Marvin K. Nakayama, Committee Member — Date
Associate Professor of Computer Science, NJIT

Dr. Jian Yang, Committee Member — Date
Associate Professor of Industrial and Manufacturing Engineering, NJIT

BIOGRAPHICAL SKETCH

Author: Xin Wang

Degree: Doctor of Philosophy

Date: January 2008

Undergraduate and Graduate Education:

- Doctor of Philosophy in Computer Science,
  New Jersey Institute of Technology, Newark, NJ, 2007
- Master of Computer Science,
  Hefei University of Technology, Hefei, Anhui, China, 2000
- Bachelor of Computer Science,
  Hefei University of Technology, Hefei, Anhui, China, 1997

Major: Computer Science

Presentations and Publications:

CZUMAJ, A., AND WANG, X. 2007. Communication problems in random line-of-sight ad-hoc radio networks. In Proceedings of *the 4th Symposium on Stochastic Algorithms, Foundations, and Applications.*

CZUMAJ, A., AND WANG, X. 2007. Fast message dissemination in random geometric ad-hoc radio networks. To appear in *the 18th International Symposium on Algorithms and Computation.*

HUO, Y., LEUNG, J. AND WANG, X. 2007. Online scheduling of equal-processing-time task systems. Submitted to *Theoretic Computer Science.*

HUO, Y., LEUNG, J. AND WANG, X. 2007. Integrated production and delivery scheduling with disjoint windows. Submitted to *ACM Transactions on Algorithms.*

HUO, Y., LEUNG, J. AND WANG, X. 2007. Scheduling Algorithms with Nested and Inclusive Processing Set Restrictions. Submitted to *Discrete Optimization.*

This dissertation is dedicated to my wife and parents. Their support, encouragement, and constant love have sustained me throughout my life.

ACKNOWLEDGMENT

I have been very lucky to have two great advisors during my graduate study in NJIT – Dr. Artur Czumaj and Dr. Joseph Leung. Without their support, patience and encouragement, this dissertation would not exist.

I am deeply indebted to Dr. Artur Czumaj, who is not only an advisor, but also a mentor and a friend. I am grateful to him for teaching me much about research and scholarship, for giving me invaluable advice on presentations and writings among many other things, for many enjoyable and encouraging discussions with him.

I sincerely thank Dr. Joseph Leung for bringing my attention to the field of computational complexity and scheduling theory in the first place. I am grateful for his generous support during my study. I thank him for spending a great deal of valuable time giving me technical and editorial advice for my research.

My thanks also go to the members of my dissertation committee, Dr. James Calvin, Dr. Alexandros Gerbessiotis, Dr. Marvin Nakayama, and Dr. Jian Yang, for reading previous drafts of this dissertation and providing many valuable comments that improved the contents of this dissertation.

I am also grateful to my colleagues, Hairong Zhao, Ozgur Ozkan, Ankur Gupta, Xinfa Hu, for numerous interesting and good-spirited discussions about research.

The friendship of Gang Fu, Xiaofeng Wang is much appreciated. They have given me not only advice on research in general, but also valuable suggestions about life, living and job hunting, etc.

Last, I would like to thank my wife, Yumei Huo, for her understanding and love during the past few years. Her support and encouragement were in the end what made this dissertation possible. I give my deepest gratitude to my parents for their endless love and support which provided the foundation for this work. I also thank my dearest sister for her love and for taking care of my parents during my absence.

TABLE OF CONTENTS

TABLE OF CONTENTS
(Continued)

TABLE OF CONTENTS
(Continued)

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

In this dissertation, combinatorial problems coming from two areas are studied: machine scheduling and ad-hoc radio network communication.

1.1 Communication in Ad-hoc Radio Networks

Ad-hoc radio network [Bar-Yehuda et al. 1992; Clementi et al. 2003; Chrobak et al. 2002; Czumaj and Rytter 2006; Dessmark and Pelc 2007; Elsässer and Gasieniec 2006; Gasieniec et al. 2005; Kowalski and Pelc 2007] is a classic communication model. The communication in the network is synchronous. All nodes have access to a global clock and transmit in discrete time steps called *rounds*. In such networks the nodes communicate by sending messages through the *edges* of the network. Here the edge is the logical one-hop connection between a pair of nodes, it can be wire, microwave, or any type of medium. In each round each node can either transmit the message to all its neighbors at once or can receive the message from one of its neighbors (be in the listening mode). A node x will receive a message from its neighbor y in a given round if and only if it does not transmit (is in the listening mode) and there is no *collision*. Collision is the main issue of the radio network. If more than one neighbor of node x transmits simultaneously in a given round, then a collision occurs and no message is received by the node x. Based on the fact that whether the collision can be distinguished by x from the situation when none of x's neighbors is transmitting, the radio networks are divided into two types: *radio network with collision detection* and *radio network without collision detection*. Throughout this dissertation, only the network without collision detection is considered.

Radio network can be also classified by the knowledge of topology. A network is called *centralized* if the topology of the connections is known in advance by any node in the network. The network is called *non-centralized* or *distributed* otherwise. Since there is usually no centralized access point in the practical ad-hoc network, a distributed model

is more desirable. Furthermore, suppose n is the number of nodes in the network, then the length of the message sending in one round is assumed to be polynomial of n, and thus, each node can combine multiple messages into one.

In this dissertation, two fundamental communication problems are studied: *Broadcasting* and *Gossiping*. In the broadcasting problem, a distinguished source node has a message that must be sent to all other nodes in the network. In the gossiping problem, the goal is to disseminate the messages in a network so that each node will receive messages from all other nodes. An algorithm *completes gossiping* in T rounds if at the end of round T each node receives messages from all other nodes. Solving gossiping implies also solving broadcasting, and therefore all upper bounds for gossiping yield also identical bounds for broadcasting.

Broadcasting and gossiping have been extensively studied in the ad-hoc radio networks model of communication (but *not for random geometric networks*). In the *centralized scenario*, when each node knows the entire network, Kowalski and Pelc [Kowalski and Pelc 2007] presented a centralized deterministic broadcasting algorithm that runs in $\mathcal{O}(D + \log^2 n)$ time and Gąsieniec et al. [Gasieniec et al. 2005] designed a deterministic gossiping algorithm that runs in $\mathcal{O}(D + \delta \log n)$ time, where D is the diameter and δ is the maximum degree of the network. Since in random geometric networks $\delta = \Theta(n r^2)$ and $D = \Theta(1/r)$ with high probability, this yields an optimal $\mathcal{O}(D)$-time centralized deterministic algorithms for both broadcasting and gossiping for all $r \leq \mathcal{O}\left(\frac{1}{(n \log n)^{1/3}}\right)$.

There has been also a very extensive research in the *non-centralized (distributed) setting in ad-hoc radio networks* which is presented here only very briefly. In the model of *unknown topology networks*, if the randomized algorithms are considered, then it is known that broadcasting can be performed in $\mathcal{O}(D \log(n/D) + \log^2 n)$ time [Czumaj and Rytter 2006; Kowalski and Pelc 2005], and this bound is asymptotically optimal [Alon et al. 1991; Kushilevitz and Mansour 1998]. The fastest randomized algorithm for gossiping in directed networks runs in $\mathcal{O}(n \log^2 n)$ time [Czumaj and Rytter 2006] and the fastest deterministic algorithm runs in $\mathcal{O}(n^{4/3} \log^4 n)$ time [Gasieniec et al. 2004]. For undirected networks,

both broadcasting and gossiping have deterministic $\mathcal{O}(n)$-time algorithms [Bar-Yehuda et al. 1992; Chlebus et al. 2002], and it is known that these bounds are asymptotically tight [Bar-Yehuda et al. 1992; Kowalski and Pelc 2002]. In a relaxed model, in which each node knows also IDs of all its neighbors, $o(n)$-time deterministic broadcasting is possible for undirected networks of the diameter $o(\log \log n)$ [Kowalski and Pelc 2002]. Still, *no $o(n)$-time deterministic gossiping algorithm is known.* For more about the complexity of deterministic distributed algorithms for broadcasting and gossiping in ad-hoc radio networks, see, e.g., [Clementi et al. 2003; Czumaj and Rytter 2006; Gasieniec et al. 2004; Kowalski and Pelc 2005; Kowalski and Pelc 2007] and the references therein.

Dessmark and Pelc [Dessmark and Pelc 2007] consider broadcasting in ad-hoc radio networks in a model of geometric (not random) networks. They consider scenarios in which all nodes either *know their own locations in the plane*, or the labels of the nodes within some distance from them. The nodes use disks of possibly different sizes to define their neighbors. Dessmark and Pelc [Dessmark and Pelc 2007] show that *broadcasting* can be performed in $\mathcal{O}(D)$ time.

Recently, the complexity of broadcasting in ad-hoc radio networks has been investigated in a *(non-geometric)* model of *random networks* by Elsässer and Gąsieniec [Elsässer and Gasieniec 2006], and Chlebus et al. [Chlebus et al. 2006]. In [Elsässer and Gasieniec 2006], the classical model of random graphs $G_{n,p}$ is considered. If the input is a random graph from $G_{n,p}$ (with $p \geq c \log n/n$ for a constant c), then Elsässer and Gąsieniec give a deterministic centralized broadcasting algorithm that runs in time $\mathcal{O}(\log(pn) + \log n/\log(pn))$, and a randomized distributed broadcasting $\mathcal{O}(\log n)$-time algorithm. Related results for gossiping are shown by Chlebus et al. [Chlebus et al. 2006], who consider the framework of average complexity.

1.1.1 Distributed Broadcasting and Gossiping Algorithms in Random Geometric Graph

The first problem studied is the distributed gossiping in the random geometric graph. Random geometric networks is motivated by *mobile ad hoc networks* and *sensor networks*.

A geometric network $\mathcal{N} = (V, E)$ is a graph with node set V corresponding to the set of transmitter-receiver stations placed on a plane $\mathbb{R}^2$. The edges E of the network $\mathcal{N}$ connect specific pairs of nodes. If there is a communication link between two nodes in $\mathcal{N}$, an edge between these two nodes exists in $\mathcal{N}$. The *unit disc graph* model is considered in which for a given parameter r (called the *range* or the *radius*) there is a communication link between two nodes $p, q \in V$ if and only if the distance between p and q (which is denoted by $\mathsf{dist}(p, q)$) is smaller than or equal to r.

The geometric network model $\mathcal{N} = (V, E)$ of unit disc graphs is studied, in which the n nodes in V are placed *independently and uniformly at random* (i.u.r.) [1] in the unit square $[0, 1]^2$. The model of random geometric networks described above has been extensively studied in the literature (see [Goel et al. 2004; Gupta and Kumar 1998; Penrose 2003] and the references therein), where recent interest is largely motivated by its applications in sensor networks.

Radius r plays a critical role in random geometric networks. It is known (see, e.g., [Gupta and Kumar 1998; Penrose 1997]) that when $r < (1 - o(1)) \cdot \sqrt{\ln n/(\pi\, n)}$, then the network is disconnected with high probability[2], and therefore gossiping is meaningless in that case. Therefore, throughout the entire dissertation, it will always be assumed that $r \geq c \cdot \sqrt{\log n/n}$ for some sufficiently large constant c. This ensures that the network is connected with high probability and therefore gossiping is feasible. With this assumption, some further assumptions are made about the structure of the input network. And so, it is well known (and easy to prove, see also [Penrose 2003]) that such a random geometric network has *diameter* $D = \Theta(1/r)$ and the minimum and maximum degree is $\Theta(n\, r^2) = \Theta(n/D^2)$, where all these claims hold *with high probability*, that is, with probability at least $1 - 1/n^{\Omega(1)}$. Therefore, from now on, the remaining part of this section is *implicitly* condition on these events.

[1] Another classical model assumes that the points are having Poisson distribution in $[0, 1]^2$. All the algorithms presented in this dissertation will work in the Poisson model as well.

[2] In particular, Penrose [Penrose 1997] proved that if $\mathfrak{r}$ is the random variable denoting the minimum radius for which $\mathcal{N}$ is connected, then $\lim_{n\to\infty} \mathbf{Pr}[n\, \pi\, \mathfrak{r}^2 - \ln n \leq \alpha] = e^{-e^{-\alpha}}$.

In general, the nodes do not know their positions nor do they know the positions of their neighbors, and each node only knows its ID (assumed to be a unique number in $\{1, 2, \ldots, n^\lambda\}$ for an arbitrary constant λ), its initial message, and the number of nodes n in $\mathcal{N}$ (the last assumption can be slightly relaxed). This model of *unknown topology network* has been extensively studied in the literature, see, e.g., [Bar-Yehuda et al. 1992; Clementi et al. 2003; Chrobak et al. 2002; Czumaj and Rytter 2006; Kowalski and Pelc 2007].

In many applications, one can assume that the nodes of the network have some additional devices that allow them to obtain some basic (geometric) information about the network. The most powerful model assumes that each node has a low-power Global Position System (GPS) device, which gives the node its exact location in the system [Giordano and Stojmenovic 2003]. Since GPS devices are relatively expensive, GPS is often not available. In such situation, a range-aware model is considered, the model extensively studied in the context of localization problem for sensor networks [Capkun et al. 2002; Doherty et al. 2001; Li et al. 2004]. In this model, the distance between neighboring nodes is either known or can be estimated by received signal strength (RSS) readings with some errors. Another scenario , in which each node can be aware of the direction of the incoming signals [Nasipuri et al. 2002] is also considered.

In this dissertation a thorough study of basic communication primitives in random geometric ad-hoc radio networks is presented. Even though initially both the broadcasting and the gossiping problems are considered, all gossiping algorithms proposed in this dissertation match the running time of the broadcasting algorithms, and therefore only the gossiping problem is presented. It can be shown that in many scenarios, the random structure of geometric ad-hoc radio networks make it possible to perform distributed gossiping in asymptotically optimal time $\mathcal{O}(D)$. (In general setting, no $o(n)$-time distributed deterministic gossiping algorithms are known.)

While the algorithms presented in this dissertation can be described to work with all values of $r \geq c\sqrt{\log n/n}$, they are especially efficient for small values of r, and in particular, in order to obtain asymptotically optimal running times, the randomized algorithm requires that $r \leq \mathcal{O}\left(\frac{1}{(n \log n)^{1/3}}\right)$ and the deterministic algorithms, depending

on the exact model, need either $r \leq \mathcal{O}\left(\frac{1}{n^{2/5}\log^{1/5} n}\right)$ or $r \leq \mathcal{O}\left(\frac{1}{n^{7/16}\log^{5/16} n}\right)$. Even though this does not cover the whole spectrum of values for r, it is believed that it covers the most interesting case when the radius is relatively small and hence the underlying graph is not too dense (given that, say, typical sensor networks aim at being sparse). Moreover, these algorithms are especially efficient in the most basic (and, most interesting) scenario when $r = \Theta(\sqrt{\log n/n})$.

1.1.2 Distributed Broadcasting and Gossiping Algorithms in Random Line-of-Sight Graph

The second problem studied is the distributed gossiping in the *random Line-of-Sight networks*, a model of networks introduced recently by Frieze et al. [Frieze et al. 2007]. The model of line-of-sight networks has been motivated by wireless networking applications in complex environments with obstacles. It considers scenarios of wireless networks in which the underlying environment has obstacles and the communication can only take place between objects that are close in space and are in the line of sight (are visible) to one another. In such scenarios, the classical random graph models [Bollobas 2001] and random geometric network models [Penrose 2003] seem to be not well suited, since they do not capture the main properties of environments with obstacles and of line-of-sight constraints. Therefore, Frieze et al. [Frieze et al. 2007] proposed a new random network model that incorporates two key parameters in such scenarios: range limitations and line-of-sight restrictions. In the model of Frieze et al. [Frieze et al. 2007], one places points randomly on a 2-dimensional grid and a node can see (can communicate with) all the nodes that are in at most a certain fixed distance and which are in the same row or column. One motivation is to consider urban areas, where the rows and columns correspond to "streets" and "avenues" among a regularly spaced array of obstructions.

Frieze et al. [Frieze et al. 2007] concentrated their study on basic structural properties of the line-of-sight networks like the connectivity, k-connectivity, etc. In this dissertation, the study of fundamental communication properties of the random line-of-sight networks in the scenario of *ad-hoc radio communication networks* is initiated.

Again, the main focus is on two classical communication problems: *broadcasting* and *gossiping*.

1.2 Machine Scheduling Problems

Scheduling problems are motivated by allocation of limited resources over time. The goal is to find an optimal allocation where optimality is defined by some problem specific objective. The resources are usually represented as machines. According to R. Graham et. al, scheduling problems can be described by three types of characteristics: machine environment, properties of jobs, and the objective to be optimized. The possible machine environments are: *Single machine(*1*)*, *Identical machines in parallel(P_m)*, and *Unrelated machines in parallel(R_m)* and so on. The jobs can have properties such as *processing time*, *release time*, and the *soft due date* or *hard due date*, etc. Sometimes, there are *precedence constraints* between jobs(A job cannot start until all of its ancestor jobs have been finished). The precedence constraints can be represented by a *directed acyclic graph(DAG)* G. A task i is said to be an *immediate predecessor* of another task j if there is a directed arc (i, j) in G; j is said to be an *immediate successor* of i. G is an *intree* if each vertex, except the root, has exactly one immediate successor; G is an *outtree* if each vertex, except the root, has exactly one immediate predecessor. A *chain* is an intree that is also an outtree; i.e., each task has at most one immediate predecessor and at most one immediate successor. In the literature, research in scheduling theory has mostly concentrated on these four classes of precedence constraints: *prec* (for arbitrary precedence constraints), *intree*, *outtree*, and *chains*. The objective can be the *maximum finishing time (makespan)* of jobs, the *average finishing time (mean flow time)* of jobs, *maximum lateness*, and so on.

1.2.1 Online Scheduling of Equal-Processing-Time Task System

The first scheduling problem studied is online scheduling of equal- processing-time task system. In an online problem, data is supplied to the algorithm incrementally, one piece at a time. The online solution must also produce the output incrementally, and the decisions about the output are made with incomplete knowledge about the entire input. So it is not

surprising that an on-line algorithm often can not produce an optimal solution. Motivated by these observations, Sleator and Tarjan proposed the idea of *competitive analysis*. In competitive analysis, one compares the performance of the online algorithm against the performance of the optimal offline algorithm on every input and consider the worst-case ratio.

Recently, Huo and Leung [Huo and Leung 2005] consider an online scheduling model where tasks, along with their precedence constraints, are released at different times, and the scheduler has to make scheduling decisions without knowledge of future releases. In other words, the scheduler has to schedule tasks in an online fashion. Huo and Leung [Huo and Leung 2005] obtain optimal online algorithms for the following cases:

- $P2|p_j = 1,\ prec_i\ released\ at\ r_i|C_{max}$. The algorithm is an adaptation of Coffman-Graham algorithm.
- $P|p_j = 1,\ outtree_i\ released\ at\ r_i|C_{max}$. The algorithm is an adaptation of Hu's algorithm.
- $P2|pmtn,\ prec_i\ released\ at\ r_i|C_{max}$. The algorithm is an adaptation of Muntz-Coffman algorithm.
- $P|pmtn,\ outtree_i\ released\ at\ r_i|C_{max}$. The algorithm is an adaptation of Muntz-Coffman algorithm.

Using an adversary argument, they ([Huo and Leung 2005]) show that it is impossible to have optimal online algorithms for the following cases:

- $P3|p_j = 1,\ intree_i\ released\ at\ r_i|C_{max}$.
- $P2|p_j = p,\ chain_i\ released\ at\ r_i|C_{max}$.
- $P3|p_j = 1,\ intree_i\ released\ at\ r_i|C_{max}$.

In this dissertation, those cases where optimal online algorithm is impossible to have are considered, approximation algorithms for them are proposed. It is known that any list scheduling algorithm for $P|prec_i\ released\ at\ time\ r_i|C_{max}$ will have a competitive

ratio no larger than $2 - 1/m$ ([Sgall 1998]). In order to have better competitive ratios, it appears that one has to restrict the precedence constraints and the processing times. In this regard, it is shown that an online version of *Hu's* algorithm gives an asymptotic competitive ratio of 1.5 for $P|p_j = p,\ intree_i\ released\ at\ r_i|C_{max}$. For the problem $P|p_j = p,\ outtree_i\ released\ at\ r_i|C_{max}$, it is shown that the online version of *Hu's* algorithm has an asymptotic competitive ratio of 1. Finally, it is shown that for the problem $P2|p_j = p,\ prec_i\ released\ at\ r_i|C_{max}$, an online version of Coffman-Graham algorithm yields an asymptotic competitive ratio of 1.

1.2.2 Integrated Production and Delivery Scheduling with Disjoint Windows

The second scheduling problem is a integrated production and delivery problem, with disjoint time windows. Consider a company that produces perishable goods. The company relies on a third party to deliver goods, which picks up delivery products at regular times (for example, at 10:00am every morning). Because the goods are perishable, it is infeasible to produce the goods far in advance of the delivery time. Thus, at each delivery time, there is a time window that the goods can be produced and delivered at that delivery time. Consider a planning horizon T. A set of jobs is given with each job specifying its delivery time, processing time and profit. The company can earn the profit of the job if the job is produced and delivered at its specified delivery time. From the company point of view, the company is interested in picking a subset of jobs to produce and deliver so as to maximize the total profit. The jobs that are not picked will be discarded without any penalty.

Formally, there is a planning horizon $T = \{d_1, d_2, \cdots, d_z\}$, consisting of z delivery times. For each delivery time d_j, there is a time instant $w_j < d_j$ such that a job must be completed in the time window $[w_j, d_j]$ if it were to be delivered at time d_j. The time window $[w_j, d_j]$ is denoted by W_j. The time windows are assumed to be disjoint. Thus, $w_1 < d_1 < w_2 < d_2 < \cdots < w_z < d_z$. Let $d_0 = 0$ be a dummy delivery time. Preceding each time window W_j is the *leading interval* $L_j = (d_{j-1}, w_j)$. A time window together with its leading interval is called a *time frame*, and it is denoted by $F_j = L_j \cup W_j$. Figure 1.1 depicts the definitions of time window, leading interval and time frame. Let W be the

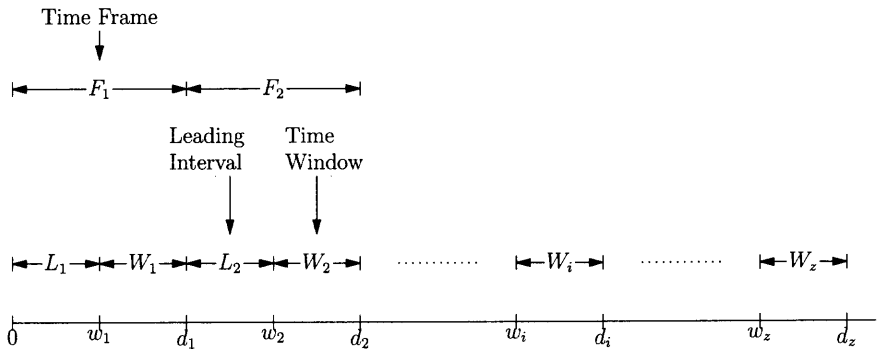

Figure 1.1 Illustrating the definitions of time window, leading interval, and time frame.

length of a time window and L be the length of a leading interval, with the assumption that all time windows have the same length W, and all leading intervals have the same length L.

Within the planning horizon, there is a set of jobs $J = \{J_1, J_2, \cdots, J_n\}$. Associated with each job J_i is a processing time p_i, a delivery time $d_i \in T$ and a profit pf_i. The job J_i is supposed to be delivered at the delivery time d_i, its processing time is p_i, and the company can earn a profit pf_i if the job can be produced and delivered at the delivery time d_i. From the company point of view, the company are interested in picking a subset of jobs to produce and deliver so as to maximize the total profit. The jobs that are not produced and delivered will be discarded without any penalty. All job information are known in advance, preemption is not allowed, and there is no vehicle limitation at any delivery date.

In the past, production scheduling have focused on the issue of how machines are allocated to jobs in the production process so as to obtain an optimal or near-optimal schedule with respect to some objective functions. In the last two decades, integrated production and delivery scheduling problems have received considerable interest. However, most of the research for this model is done at the strategic and tactical levels (see [Bilgen and Ozkarahan 2004; Chen 2004; Chen 2006; Erenguc et al. 1999; Goetschalckx et al. 2002; Sarmiento and Nagi 1999; Thomas and Griffin 1996] for examples), while very little is known at the operational scheduling level. Chen [Chen 2006] classified the model at the operational scheduling level into four classes: (1) Models with

individual and immediate delivery; (2) Models with batch delivery to a single customer; (3) Models with batch delivery to multiple customers; and (4) Models with fixed delivery departure date. In the models with individual and immediate delivery, Garcia and Lozano [Garcia and Lozano 2005] is the only dissertation that studies a model with delivery time windows. They gave a tabu-search solution procedure for the problem and their objective function is the maximum number of jobs that can be processed. In the models with individual and immediate delivery, problems with fixed delivery date are also studied in [Garcia et al. 2004]. In the models with fixed delivery departure date, no time window constraint is considered and the focus is on the delivery cost.

Bar-Noy et al. [Bar-Noy et al. 2001] considered a more general version of the scheduling problem studied in this chapter: n jobs are to be scheduled nonpreemptively on m machines. Associated with each job is a release time, a deadline, a weight (or profit), and a processing time on each of the machines. The goal is to find a nonpreemptive schedule that maximizes the weight (or profit) of jobs that meet their respective deadlines. This problem is known to be strongly NP-hard, even for a single machine. They obtained the following results [Bar-Noy et al. 2001]:

- For identical job weights and unrelated machines: there is a greedy 2-approximation algorithm.

- For identical job weights and m identical machines: the same greedy algorithm achieves a tight $\frac{(1+1/m)^m}{(1+1/m)^m-1}$ approximation factor.

- For arbitrary job weights and a single machine: an LP formulation achieves a 2-approximation for polynomially bounded integral input and a 3-approximation for arbitrary input. For unrelated machines, the factors are 3 and 4, respectively.

- For arbitrary jobs weights and m identical machines: the LP-based algorithm applied repeatedly achieves a $\frac{(1+1/m)^m}{(1+1/m)^m-1}$ approximation factor for polynomially bounded integral input and a $\frac{(1+1/2m)^m}{(1+1/2m)^m-1}$ approximation factor for arbitrary input.

- For arbitrary job weights and unrelated machines: there is a combinatorial $(3+2\sqrt{2})$-approximation algorithm.

In this dissertation three kinds of profits are considered : (1) arbitrary profit, (2) equal profit, and (3) profit proportional to its processing time. In the first case, a pseudo-polynomial time algorithm is given to find an optimal schedule on a single machine. Based on the pseudo-polynomial time algorithm, a fully polynomial time approximation scheme (FPTAS) is developed with running time $O(\frac{n^3}{\epsilon})$. In the second case, an $O(n \log n)$-time algorithm is given to find an optimal schedule on a single machine. An $\frac{7}{5}$-approximation algorithm for a single time frame on parallel and identical machines is also given; this problem is similar to the bin packing problem studied by Coffman et al. [Coffman et al. 1978]. In the third case, one can get a FPTAS with an improved running time, $O(\frac{n^2}{\epsilon})$ versus $O(\frac{n^3}{\epsilon})$. All these algorithms can be extended to parallel and identical machines with a degradation of performance ratios.

From the complexity point of view, the problem studied here is ordinary NP-hard for a single machine, but strongly NP-hard for parallel and identical machines. The more general problem studied by Bar-Noy et al. [Bar-Noy et al. 2001] is strongly NP-hard even for a single machine. By the theory of strong NP-completeness, there is no FPTAS for the problem studied by Bar-Noy et al. [Bar-Noy et al. 2001], unless P = NP. Thus, the most one can hope for the general problem is a polynomial time approximation scheme (PTAS). This shows that the problem is easier than the general problem for a single machine.

1.2.3 Preemptive Scheduling Algorithms with Nested and Inclusive Processing Set Restrictions

The problem of preemptively scheduling n independent jobs on m parallel machines is considered, where the machines differ in their functionality but not in their processing speeds. Jobs have a restricted set of machines to which they may be assigned, called its *processing set*. Such problems are called scheduling problems having *job assignment restrictions*. Specifically, two special cases are considered: (1) when the processing sets do not partially overlap and are said to be *nested*, and (2) when the processing sets are not only nested but include one another, and are called *inclusive processing set*. Clearly, inclusive

processing set is a special case of nested processing set. The objective is to minimize the makespan of the schedule.

Scheduling problems with job assignment restrictions occur quite often in practice. Glass and Mills [Glass and Mills 2006] describe an application of nested processing set in the drying stage in a flour mill in the United Kingdom. Hwang et al. [Hwang et al. 2004] give a scenario occurring in the service industry in which a service provider has customers categorized as platinum, gold, silver, and regular members, where "higher-level customers" receive better services. One method of providing such differentiated service is to label servers and customers with prespecified grade of service (GoS) levels and allow a customer to be served only by a server with GoS level less than or equal to that of the customer. Glass and Kellerer [Glass and Kellerer 2007] describe a situation of assigning jobs to computers with memory capacity. Each job has a memory requirement and each computer has a memory capacity. A job can only be assigned to a computer with enough memory capacity. Ou et al. [Ou et al. 2007] consider the process of loading and unloading cargoes of a vessel, where there are multiple nonidentical loading/unloading cranes operating in parallel. The cranes have the same operating speed but different weight capacity limits. Each piece of cargo can be handled by any crane with a weight capacity limit no less than the weight of the cargo.

Both the case where all jobs are released at the beginning and the case where jobs are released at different times are considered. It is shown that there are efficient optimal algorithms for nested (and hence inclusive) processing set when all jobs are released at the beginning. When jobs are released at different times, it is shown that there is a maximum flow approach to solve the nested processing set case. For the case of inclusive processing set, there is a more efficient algorithm to find an optimal schedule. All algorithms operate in an offline fashion; i.e., all data about the jobs are known in advance. Online scheduling algorithms are also considered; i.e., the algorithm have to schedule jobs without future knowledge of job arrivals. It is shown that there does not exist optimal online scheduling algorithms, even for the case of inclusive processing set. (An online algorithm is optimal if it produces a schedule as good as an optimal offline algorithm.) This is in contrast with

the identical and parallel machine case, where there is an optimal online algorithm for the problem $P \mid pmtn, r_j \mid C_{\max}$ due to Hong and Leung [Hong and Leung 1992].

The Model Suppose the machines are labeled from 1 to m. A set of machine intervals $MI = \{MI_1, \ldots, MI_\lambda\}$ is given, where $MI_i = \{M_{i_1}, M_{i_2}, \ldots, M_{i_x}\}$ is a set of machines with consecutive labels. One can assume that for any two different machine intervals, either they are disjoint or one includes the other [3]. Each job J_j, $1 \leq j \leq n$, can be represented by a pair (p_j, S_j), where p_j is the processing time of the job and S_j is a machine interval such that $S_j \in MI$. Job J_j can only be scheduled on the machines in S_j. If the jobs have different release times, then each job J_j will be represented by the triple (p_j, S_j, r_j), where r_j is the release time of the job. The objective is to minimize the makespan $C_{\max}$.

For each machine interval MI_i, define $J(MI_i)$ to be the set of jobs whose machine interval is exactly MI_i. That is,

$$J(MI_i) = \{J_j | S_j = MI_i\}.$$

The average load of the machine interval MI_i, denoted by $\eta(MI_i)$, is defined as

$$\eta(MI_i) = \frac{\sum_{S_j \subseteq MI_i} p_j}{|MI_i|},$$

where $|MI_i|$ is the number of machines in the machine interval MI_i.

Define $\sigma(MI_i)$ as

$$\sigma(MI_i) = \max\left\{\max_{MI_{i'} \subseteq MI_i}\{\eta(MI_{i'})\}, \max_{S_j \subseteq MI_i}\{p_j\}\right\}.$$

Clearly, $\sigma(MI_i)$ is a lower bound on the makespan of all the jobs in the machine interval MI_i (assuming the jobs are all released at time $t = 0$).

Example 1: Suppose $m = 10$ and there are three machine intervals:

$$MI_1 = \{M_1, \ldots, M_{10}\}, MI_2 = \{M_4, \ldots, M_6\} and MI_3 = \{M_7, \ldots, M_{10}\}.$$

[3]Under this assumption, it is easy to prove that the total number of intervals is at most $2m - 1$.

Six jobs defined for MI_1: $J_1 = (6, MI_1)$, $J_2 = (3, MI_1)$, $J_3 = (5, MI_1)$, $J_4 = (4, MI_1)$, $J_5 = (1, MI_1)$ and $J_6 = (2, MI_1)$.

Four jobs defined for MI_2: $J_7 = (6, MI_2)$, $J_8 = (1, MI_2)$, $J_9 = (2, MI_2)$ and $J_{10} = (1, MI_2)$.

Finally, five jobs defined for MI_3: $J_{11} = (5, MI_3)$, $J_{12} = (3, MI_3)$, $J_{13} = (3, MI_3)$, $J_{14} = (3, MI_3)$ and $J_{15} = (3, MI_3)$.

Then, $J(MI_1) = \{J_1, \ldots, J_6\}$, $J(MI_2) = \{J_7, \ldots, J_{10}\}$ and $J(MI_3) = \{J_{11}, \ldots, J_{15}\}$.

Background and Related Work The problem studied in this dissertation is a natural generalization of the identical and parallel machine case (i.e., $P \mid pmtn \mid C_{\max}$), and it is a special case of the unrelated machine case (i.e., $R \mid pmtn \mid C_{\max}$). For the problem $P \mid pmtn \mid C_{\max}$, McNaughton [McNaughton 1959] has given a linear time algorithm to find an optimal schedule. An optimal online algorithm has been given for the problem $P \mid pmtn, r_j \mid C_{\max}$ by Hong and Leung [Hong and Leung 1992]. For the problem $R \mid pmtn \mid C_{\max}$, Lawler and Labetoulle [Lawler and Labetoulle 1978] have given a linear programming formulation to find an optimal schedule. Their algorithm can be generalized to solve (offline) the problem $R \mid pmtn, r_j \mid C_{\max}$ as well.

For nonpreemptive scheduling, it is well known that $P \mid\mid C_{\max}$ is strongly NP-hard; see Garey and Johnson [Garey and Johnson 1979]. Hochbaum and Shmoys [Hochbaum and Shmoys 1987] have given a PTAS (polynomial time approximation scheme) for this problem. Since the problem is strongly NP-hard, this is probably the best one can hope for. Because of the importance of the problem, there have been numerous research conducted to tackle this difficult problem in the last few decades; see the survey paper by Chen et al. [Chen et al. 1998]. For the problem $R \mid\mid C_{\max}$, Lenstra et al. [Lenstra et al. 1990] have given a polynomial-time approximation algorithm with a worst-case bound of 2. It is still an open question whether this bound can be improved.

Since $P \mid\mid C_{\max}$ is strongly NP-hard, finding an optimal nonpreemptive schedule for the nested and inclusive processing sets are strongly NP-hard as well. Therefore,

people have concentrated their efforts to polynomial-time approximation algorithms. In this regard, Glass and Kellerer [Glass and Kellerer 2007] have given a list scheduling algorithm for the nested processing set case. They show that the algorithm has a worst-case bound of $2 - 1/m$. For the inclusive processing set case, the first approximation algorithm is the Modified Largest Processing Time First algorithm given by Hwang et al. [Hwang et al. 2004]. They show that the algorithm obeys a bound of $\frac{5}{4}$ for $m = 2$ and $2 - \frac{1}{m-1}$ for $m \geq 3$. Subsequently, Glass and Kellerer [Glass and Kellerer 2007] give a $\frac{3}{2}$-approximation algorithm for this problem. More recently, Ou et al. [Ou et al. 2007] give an improved algorithm with a worst-case bound of $\frac{4}{3} + \epsilon$, where ϵ is any given positive number. They also give a PTAS for this problem.

In this dissertation a linear time algorithm is given to find an optimal schedule for the nested (and hence inclusive) processing set case, when all jobs are available for processing at the beginning. A more complicated algorithm is given that produces not only optimal but also maximal schedules. A schedule is said to be *maximal* if at any time instant t, the total amount of processing done by all the machines in the time interval $[0, t]$ is greater than or equal to that in any other feasible schedule. An algorithm is maximal if it produces maximal schedules. It is easy to see that a maximal algorithm must be an optimal algorithm, but not conversely. Maximal schedules are nice in the sense that if the machines can break down in the future at unpredictable times, then the maximal schedule would have processed as much work as possible before the machine down time. They are also useful for online scheduling; see Hong and Leung [Hong and Leung 1992] and Huo and Leung [Huo and Leung 2005]. The optimal online algorithm of Hong and Leung [Hong and Leung 1992] is a maximal algorithm.

The situation where jobs have different release times is also considered. A network flow approach is given to solve the nested processing set case. For the inclusive processing set case, a more efficient algorithm is given to find an optimal schedule.

The online scheduling algorithms are also considered. Unfortunately, it can be shown that there does not exist an optimal online scheduling algorithm, even for the inclusive processing set case.

1.3 Organization

This dissertation is organized as follows. The broadcasting and gossiping algorithm on random geometric ad-hoc radio network will be presented in Chapter 2. The broadcasting and gossiping algorithm on random line-of-sight network will be presented in Chapter 3. In Chapter 4, online algorithms for UET task system are discussed. In Chapter 5, the results for integrated production and delivery scheduling with disjoint windows are presented. In Chapter 6, the results for preemptive scheduling algorithms with nested and inclusive processing set restrictions are presented. Finally, some concluding remarks and future work are given in Chapter 7.

CHAPTER 2

COMMUNICATION PROBLEMS IN RANDOM GEOMETRIC RADIO AD-HOC NETWORKS

2.1 Preliminaries

For any node v, define $N(v)$ to be the set of nodes that are reachable from v in one hop, $N(v) = \{u \in V : \mathsf{dist}(v, u) \leq r\}$, where $\mathsf{dist}(v, u)$ is the Euclidean distance between v and u. Any node in $N(v)$ is called a *neighbor* of v, and the set $N(v)$ is called the *neighboring set* of v. For any $X \subseteq V$, let $N(X) = \bigcup_{x \in X} N(x)$.

Define the *kth neighborhood* of a node v, $N^k(v)$, recursively as follows: $N^0(v) = \{v\}$ and $N^k(v) = N(N^{k-1}(v))$ for $k \geq 1$. The *strict kth neighborhood* of v, denoted by $SN^k(v)$, is defined as $SN^k(v) = N^k(v) \setminus N^{k-1}(v)$.

Let δ be the *maximum degree* in $\mathcal{N}$ and D be the *diameter* of $\mathcal{N}$, $D = \min_{v \in V}\{k : N^k(v) = V\}$.

$\mathfrak{B}(q, R)$ is used to denote the ball with center at q and with radius R. When the context is clear, $\mathfrak{B}(q, R)$ is also used to denote the set of nodes from $\mathcal{N}$ within the ball, e.g., $\mathfrak{B}(q, R) = \{p \in \mathcal{N} : \mathsf{dist}(p, q) \leq R\}$.

Strongly-selective families. Let k and m be two arbitrary positive integers with $k \leq m$. Following [Clementi et al. 2003], a family $\mathcal{F}$ of subsets of $\{1, \ldots, m\}$ is called (m, k)-*strongly-selective* if for every subset $X \subseteq \{1, \ldots, m\}$ with $|X| \leq k$, for every $x \in X$ there exists a set $F \in \mathcal{F}$ such that $X \cap F = \{x\}$. It is known (see, e.g., [Clementi et al. 2003; Bonis et al. 2003]) that for every k and m, there exists a (m, k)-strongly-selective family of size $\mathcal{O}(k^2 \log m)$.

Strongly-selective families are known to have direct applications in the design of efficient broadcasting and gossiping algorithms (cf. [Clementi et al. 2003; Bonis et al. 2003]). In the current setting, the following lemma can be derived.(For a proof,which

follows easily from previous works on strongly-selective families, cf. [Clementi et al. 2003; Bonis et al. 2003], the proof is rewritten here for reason of completeness).

Lemma 2.1 *In random geometric networks, for any integer k, in* ***(deterministic)*** *time* $\mathcal{O}(k \cdot n^2 \cdot r^4 \cdot \log n)$ *(Notice that this is also equal to* $\mathcal{O}(k \cdot n^2 \cdot D^{-4} \cdot \log n)$*) all nodes can send their messages to all nodes in their kth neighborhood. The algorithm may fail with probability at most* $1/n^2$ *(where the probability is with respect to the random choice of a geometric network).*

Proof: The arguments follow nowadays standard approach of applying selective families to broadcasting and gossiping in radio ad-hoc networks, see, e.g., [Clementi et al. 2003]. Consider an arbitrary network with n nodes and with maximum degree δ. One can assume (as defined in the Introduction) that all IDs (labels of the nodes) are distinct integers in $\{1, 2, \ldots, n^\lambda\}$, for a certain positive constant $\lambda \geq 1$. Let $\mathcal{F} = \{F_1, F_2, \ldots\}$ be an $(n^\lambda, \delta + 1)$-strongly-selective family of size $\mathcal{O}(\delta^2 \lambda \log n) = \mathcal{O}(\delta^2 \log n)$; the existence of such family follows from the discussion above. Then, consider a protocol in which in step t only the nodes whose IDs are in the set F_t transmit. By the strong selectivity property, for every node u and for every neighbor v of u, there is at least one time step when u does not transmit and v is the only neighbor of u that transmits in that step. Therefore, with this scheme every node will receive a message from all its neighbors after $\mathcal{O}(\delta^2 \log n)$ steps. Hence, on can repeat this procedure to ensure that after $\mathcal{O}(k\,\delta^2 \log n)$ steps, every node will receive a message from its entire kth neighborhood. Now, by applying this procedure to the random geometric networks, in which the maximum degree is $\Theta(n\,r^2)$ with high probability, the proof of Lemma 2.1 can be obtained. □

Observe that for constant k, $\mathcal{O}(k \cdot n^2 \cdot r^4 \cdot \log n) \leq \mathcal{O}(D)$ if $r \leq \mathcal{O}\left(\frac{1}{n^{2/5} \log^{1/5} n}\right)$. Therefore this bound for r is necessary to use this approach (Lemma 2.1) in any algorithm running in time $\mathcal{O}(D)$.

2.2 Randomized Gossiping in Optimal $\mathcal{O}(D)$ Time

Now, a simple randomized algorithm is presented for broadcasting and gossiping problem in random geometric networks whose running time is asymptotically optimal. The algorithm can be seen as an extension of the classical broadcasting algorithm in networks due to Bar-Yehuda et al. [Bar-Yehuda et al. 1992] (see also [Czumaj and Rytter 2006]), which when applied to random geometric networks gives asymptotically optimal running time for a more complex task of gossiping. (It can be seen as ALOHA protocol, see, e.g., [Chlebus 2001], but the intuition come from [Bar-Yehuda et al. 1992; Czumaj and Rytter 2006].)

Gossiping-Fast-R
repeat
  in round, each node independently does:
    the node transmits with probability $\frac{1}{n\,r^2}$

Theorem 2.2 *If the algorithm Gossiping-Fast-R is run in a random geometric network, then the algorithm completes gossiping after $\mathcal{O}(n\,r^2\,\log n + D)$ rounds with probability at least $1 - 1/n$. In particular, if $r \leq \mathcal{O}\left(\frac{1}{(n\log n)^{1/3}}\right)$, then the number of rounds is $\mathcal{O}(D)$.*

Before the proof of Theorem 2.2, some basic notations are introduced first. Suppose the unit square is divided into $16/r^2$ *blocks* (disjoint squares), each block with the side length of $r/4$. For a block B, B is also used to denote the set of nodes in block B; in this case, $|B|$ is the number of nodes in block B.

Now, the following claim can be proven.

Lemma 2.3 *For every block B with probability at least $1 - 1/n^4$: (i) $\frac{n\,r^2}{32} \leq |B| \leq n\,r^2$, and (ii) $|N(B)| \leq 20\,n\,r^2$.*

Proof: The proof uses Chernoff bound to estimate the number of points in a given area. This is done by considering a given area A in the unit square and observing that $|A|$ is a binomial random variable with n trials and the success probability *area*(A). Then, one just has to analyze the concentration bound for binomial random variables. Next, only

(ii) will be proven to illustrate details. The proof of (i) is similar to the proof of (ii). For any block B, if a node $v \in N(B)$, then v is in a group of 9×9 blocks centered at B. The expected number of nodes in such a group is $\mu = 81\,n\,(r/4)^2$. By assumption, the number of nodes in a subarea has uniform distribution. Therefore, by Chernoff bound $\mathbf{Pr}[|N(B)| \geq 20\,n\,r^2] \leq \mathbf{Pr}[|N(B)| \geq 3.9\mu] \leq 1/n^5$, where the fact that $r \geq c\sqrt{\ln n/n}$ for a sufficiently large constant c, and that n is sufficiently large is used. By the union bound, $\mathbf{Pr}[\exists \text{block } B : |N(B)| \leq 20\,n\,r^2] \geq 1 - 1/n^4$. □

Notice that the constants involved in Lemma 2.3 have not been optimized.

A *gossiping within a block* is the task of exchanging the messages between all the nodes in the block. Gossiping within a block B is completed if every node $v \in B$ receives a message from every other $u \in B$.

Lemma 2.4 *Gossiping within every block completes in* $\mathcal{O}(n\,r^2 \log n)$ *steps with probability at least* $1 - \frac{1}{n^2}$.

Proof: Conditioning on the bounds for $|B|$ and $|N(B)|$ from Lemma 2.3 to hold for all blocks B, one must prove the lemma holds with probability at least $1 - \frac{1}{2\,n^2}$. Fix a block B and pick two nodes $v, u \in B$. In any single round, node v transmits with probability $\frac{1}{n\,r^2}$. If v transmits, then u receives a message from v only if u does not transmit and no other node from $N(u)$ transmits in that step. Since $|N(u)| \leq 20\,n\,r^2$, in any single round, the probability that no node in $N(u) \setminus \{v\}$ transmits is at least $(1 - \frac{1}{n\,r^2})^{|N(u)\setminus\{v\}|} \geq (1 - \frac{1}{n\,r^2})^{20\,n\,r^2} \geq e^{-40}$. Hence, in any single step, u will receive the message from v with probability at least $\frac{1}{e^{40}\,n\,r^2}$. Since in every step, every process makes an independent random choice, this implies that after τ steps, u receives the message from v with probability at least $1 - (1 - \frac{1}{e^{40}\,n\,r^2})^{\tau}$. By the union bound, v succeeds in τ steps in sending the message to every node in B with probability at least $1 - |B| \cdot (1 - \frac{1}{e^{40}\,n\,r^2})^{\tau}$. Thus, by the union bound, the probability that gossiping within block B will be completed after τ steps is at least $1 - |B|^2 \cdot (1 - \frac{1}{e^{40}\,n\,r^2})^{\tau}$. Hence, the probability that the gossiping within every single block will be completed after τ steps is greater than or equal to $1 - \sum_{\text{block } B} |B|^2 \cdot (1 - \frac{1}{e^{40}\,n\,r^2})^{\tau} \geq 1 - \frac{16}{r^2} \cdot (n\,r^2)^2 \cdot (1 - \frac{1}{e^{40}\,n\,r^2})^{\tau} \geq 1 - (n\,r)^2 \cdot (1 - \frac{1}{e^{40}\,n\,r^2})^{\tau}$, where the fact that there are $16/r^2$

blocks and $|B| \leq n\,r^2$ for every block is used. By choosing an appropriate large value of $\tau = \mathcal{O}(n\,r^2 \log n)$, this probability will be greater than $1 - \frac{1}{2n^2}$, as needed. □

At any time step t, let $M_t(v)$ be the set of messages currently held by node v. For any block B, let $M_t(B)$ denote the set of common messages that are currently held by all nodes of B, that is, $M_t(B) = \bigcap_{v \in B} M_t(v)$.

Lemma 2.5 *Let B and B' be two adjacent blocks and suppose that the gossiping within block B has been completed. Then, for any t, $M_t(B) \cup M_t(B') \subseteq M_{t+1}(B')$ with constant probability.*

Proof: By Lemma 2.3, $|N(B')| \leq 20\,n\,r^2$ and $|B| \geq n\,r^2/32$ with high probability. Therefore, conditioned on these two inequalities, with probability $p \geq |B| \cdot \frac{1}{n\,r^2} \cdot (1 - \frac{1}{n\,r^2})^{|N(B')|} \geq n\,r^2/32 \cdot \frac{1}{n\,r^2} \cdot (1 - \frac{1}{n\,r^2})^{20\,n\,r^2}$, among all nodes in $N(B')$, there is exactly one node in B that transmits at a given time step. For n big enough, p is greater than some positive constant c'. □

Now, Theorem 2.2 is ready to be proven. First focus on two blocks B and B'. By Lemma 2.4, gossiping within every block will be completed after the first $\mathcal{O}(n\,r^2 \log n)$ steps with high probability.

For fixed blocks B and B', there is always a sequence of blocks $B = B_1$, B_2, ..., $B_k = B'$, such that B_i and B_{i+1} are adjacent for any $1 \leq i \leq k-1$, and that $k \leq 8/r$. By Lemma 2.5, after each step, B_i will send its message $M_t(B_i)$ to B_{i+1} with probability at least c', where c' is a positive constant promised by Lemma 2.5. Consider a sequence of blocks $B = B_1, B_2, \ldots, B_k = B'$, such that B_i and B_{i+1} are adjacent for any $1 \leq i \leq k-1$, and that $k \leq 8/r$. The goal is to transmit a message from block B_1 to block B_2, then to block B_3, and so on, until the message reaches B_k. Now, for each i, let X_i be the random variable denoting the number of steps needed to successfully deliver the message from B_{i-1} to B_i, given that B_{i-1} already received the message from B_1. The goal is to estimate $\sum_{i=2}^{k} X_i$.

Since each pair of blocks B_i and B_{i+1} is adjacent, by Lemma 2.5, after each step, B_i will successfully send its message to B_{i+1} with probability at least c', where c' is a positive constant. Therefore, each X_i is an independent geometric random variable with success probability at least c'. This in turn, implies that $\sum_{i=2}^{k} X_i$ is stochastically dominated by the number of steps in the process of tossing coins at random until obtaining $k-1$ heads, where the probability of each head is c'. Since it is easy to show (for example, using Chernoff bounds) that if one tosses $c^*\ ((k-1)\,c' + \log n))$ coins (for a suitably large positive constant c^*) then at least $k-1$ heads will be obtained with probability at least $1 - 1/n^4$. Theorem 2.2 follows.

Another way of proving Theorem 2.2 is to consider $k-1$ independent random variables $Y_2, Y_3, \ldots, Y_k$, each having geometric distribution with success probability exactly c'. Clearly, X_i is stochastically dominated by Y_i and therefore to get an upper bound for $\sum_{i=2}^{k} X_i$ it is enough to consider the sum $\sum_{i=2}^{k} Y_i$ (notice that $\sum_{i=2}^{k} Y_i$ is a random variable with negative binomial distribution). Observe that $\mathbf{E}\big[\sum_{i=2}^{k} Y_i\big] = (k-1)/c'$. Next, the well known concentration bounds (easily proven by Chernoff bounds) for the sum of independent geometric random variables (or about speed of random walks on integers) is used to obtain high probability bound for this sum. In particular, use Lemma 3.7 from [Czumaj and Rytter 2006],

$$\mathbf{Pr}\big[\sum_{i=2}^{k} Y_i \leq 2\,(k-1)/c' + 8\,\ln(c'\,n^4)\,c'\big] \;\geq\; 1 - n^{-4}\ ,$$

So after $\mathcal{O}(k/c' + \log n) = \mathcal{O}(D + \log n)$ steps, all the messages from B will be successfully transmitted to B' with probability at least $1 - 1/n^4$. By applying the union bound on all pairs of blocks, one can conclude that gossiping is completed with probability at least $1 - 1/n^2$. □

2.3 Deterministic Distributed Algorithms

The problem of designing a deterministic gossiping algorithm in *unknown topology networks*, in which the nodes do not know anything about the topology of the network

except for their own IDs, is more complicated. An optimal deterministic distributed algorithm is still unknown even in the most basic and interesting case when $r = \Theta(\sqrt{\log n/n})$. Still, one can rather easily obtain an efficient (but not optimal) deterministic algorithms for gossiping in unknown topology networks using the approach of selective families (cf. [Clementi et al. 2003; Chrobak et al. 2002; Bonis et al. 2003]). As shown in [Clementi et al. 2003], when n and δ are known, a direct use of strongly-selective families yields an $\mathcal{O}(D\,\delta^2\,\log n)$-time gossiping algorithm for general graphs. In the current setting, this yields the running time of $\mathcal{O}(D\,r^4\,n^2\,\log n)$. Still, this bound can be improved by exploiting some basic geometric properties of the underlying networks and by using the approach from [Chrobak et al. 2002].

Lemma 2.6 *There is a deterministic algorithm that in any random geometric network completes gossiping in $\mathcal{O}(n^2\,r^4\,\log n + D\,r^2\,n\,\log n)$ rounds. The algorithm may fail with probability at most $1/n^2$. If $r \leq \mathcal{O}\left(\frac{1}{n^{2/5}\,\log^{1/5}}\right)$ then the running time is $\mathcal{O}(n\,r\,\log n)$, which is $o(n)$. If $r \leq \mathcal{O}(n^{-1/3})$ and $r = \Theta(\vartheta \cdot \sqrt{\log n/n})$ then the running time is $\mathcal{O}(\vartheta^2\,D\,\log^2 n)$.*

Proof: The definition of selectors from [Chrobak et al. 2002] to facilitate the analysis can be extended as follows: A family of sets $\mathcal{F} \subseteq 2^{\{1,2,\ldots,m\}}$ is called an (m,k)-*selector* if for any two disjoint subsets X and Y of $\{1,2,\ldots,m\}$, if $k/20 \leq |X| \leq 20\,k$ and $k/20 \leq |Y| \leq 20\,k$, then there is a set $Z \in \mathcal{F}$ such that $|Z \cap X| = 1$ and $Z \cap Y = \emptyset$. By using similar probabilistic arguments as those in [Chrobak et al. 2002], one can prove the existence of an (m,k)-selector of size $\mathcal{O}(k\,\log m)$.

Recall that all IDs (labels of the nodes) are distinct integers in $\{1,2,\ldots,n^\lambda\}$, for a certain positive constant λ. The definition of blocks from Section 2.2 is used again. Because of Lemma 2.1, local gossiping in each block can be done deterministically in $\mathcal{O}(n^2\,r^4\,\log n)$ time (by using strongly-selective family). Then, the algorithm runs in D phases. In each phase, the nodes transmit according to an $(n^\lambda, n\,r^2)$-selector $\mathcal{F}$ of size $\mathcal{O}(n\,r^2\,\log n)$, where the fact that all nodes have degrees between $n\,r^2/20$ and $20\,n\,r^2$, with high probability is used. By the property of the selector, in each phase, for any two

adjacent blocks B_i and B_{i+1}, there is at least one step in which exactly one node of B_i is sending and all nodes in $B_{i+1} \cup N(B_{i+1}) \setminus B_i$ keep silence; hence in this step all common messages in B_i are successfully transmitted to B_{i+1}. Hence the gossiping will be done after D phases. The total running time is $\mathcal{O}(n^2 r^4 \log n + D\,|\mathcal{F}|) = \mathcal{O}(n\,r^2 \log n\,(n\,r^2 + D))$.

□

2.4 Deterministic Distributed Algorithm: Knowing Locations Helps

The gossiping problem in random geometric networks is considered in the model, where each node knows its geometric position in the unit square. In such model, Dessmark and Pelc [Dessmark and Pelc 2007] give a deterministic algorithm for broadcasting that (in the current setting) runs in $\mathcal{O}(D)$ time. One can prove a similar result for gossiping by extending the preprocessing phase from [Dessmark and Pelc 2007] and use an appropriate strongly-selective family to collect information about the neighbors of each point.

Theorem 2.7 *If every input node knows its location* $[0,1]^2$*, then the algorithm below will complete gossiping in a random geometric network in deterministic time* $\mathcal{O}(n^2 r^4 \log n + 1/r) = \mathcal{O}(n^2 \log n/D^4 + D)$*. In particular, if* $r \leq \mathcal{O}\left(\frac{1}{n^{2/5} \log^{1/5}}\right)$ *then the running time is* $\mathcal{O}(D)$*. The algorithm may fail with probability at most* $1/n^2$*.*

```
Gossiping-Known-Locations-D
use appropriate strongly-selective family to do local gossiping from Lemma 2.1 (with k = 1)
        to ensure that each node knows messages and positions of its neighbors
repeat        { a round }
    for i = 1 to 81 do:
        for each area in parallel do:
            pick the block with label i
            the node with the minimum ID in the block with number i transmits
```

Proof: Partition the unit square into blocks, as discussed in Section 2.2, and then partition it further into "areas," each area comprising of 9×9 blocks. Then, for each area, label the blocks in the area as $1, 2, \ldots 81$ (following the same numbering scheme in each area; for example, see Figure 2.1.)

1	2	3	4	5	6	7	8	9	1	2	3	4	5	6	7	8	9
10	11	12	13	14	15	16	17	18	10	11	12	13	14	15	16	17	18
19	20	21	22	23	24	25	26	27	19	20	21	22	23	24	25	26	27
28	29	30	31	32	33	34	35	36	28	29	30	31	32	33	34	35	36
37	38	39	40	41	42	43	44	45	37	38	39	40	41	42	43	44	45
46	47	48	49	50	51	52	53	54	46	47	48	49	50	51	52	53	54
55	56	57	58	59	60	61	62	63	55	56	57	58	59	60	61	62	63
64	65	66	67	68	69	70	71	72	64	65	66	67	68	69	70	71	72
73	74	75	76	77	78	79	80	81	73	74	75	76	77	78	79	80	81
1	2	3	4	5	6	7	8	9	1	2	3	4	5	6	7	8	9
10	11	12	13	14	15	16	17	18	10	11	12	13	14	15	16	17	18
19	20	21	22	23	24	25	26	27	19	20	21	22	23	24	25	26	27
28	29	30	31	32	33	34	35	36	28	29	30	31	32	33	34	35	36
37	38	39	40	41	42	43	44	45	37	38	39	40	41	42	43	44	45
46	47	48	49	50	51	52	53	54	46	47	48	49	50	51	52	53	54
55	56	57	58	59	60	61	62	63	55	56	57	58	59	60	61	62	63
64	65	66	67	68	69	70	71	72	64	65	66	67	68	69	70	71	72
73	74	75	76	77	78	79	80	81	73	74	75	76	77	78	79	80	81

Figure 2.1 Areas and blocks used in the proof of Theorem 2.7.

After using the algorithm from Lemma 2.1, every node knows which block it belongs to, and every node knows all other nodes in its block, including their messages and positions. Therefore, in every block, all the nodes from that block can select a single representative who will be the only node transmitting (for the entire block) in all the following time slots.

By the definition of the "area," when a block (i.e., its representative) sends a message, the message will never reach the outside of the area. Hence, after each sending, the sending block, say B, will successfully send its message $M(B)$ to all its adjacent blocks.

For any two nodes in $v, u \in \mathcal{N}$, if $v \in B$, $u \in B'$, then there is a sequence of blocks $B = B_1, B_2, \ldots, B_k = B'$, such that B_i and B_{i+1} are adjacent and $k \leq 8/r$. Thus, after at most $\lceil 8/r \rceil$ rounds, the message from v will be successfully transmitted to u. The same argument holds for any pair of nodes, implying that after $\lceil 8/r \rceil$ rounds the gossiping will be completed.

Since the running time of the preprocessing step is $\mathcal{O}(n^2 r^4 \log n)$ by Lemma 2.1, the total running time of the algorithm Gossiping-Known-Locations-D is $\mathcal{O}(n^2 r^4 \log n + 81 \cdot 8/r) = \mathcal{O}(n^2 r^4 \log n + 1/r)$. □

2.5 Deterministic Distributed Algorithm: Knowing Distances Helps

In this section, the assumption is that $c\sqrt{\log n/n} \leq r \leq \mathcal{O}\left(\frac{1}{n^{7/16}\log^{5/16} n}\right)$, what is required to efficiently use Lemma 2.1. In the previous section, it has been shown that the gossiping in random geometric networks can be done optimally in time $\mathcal{O}(D)$ if each node knows its geometric position in the unit square. The model can be relaxed and the same complexity is achievable also if the nodes do not know their geometric positions, but only can detect the distance to each node from which a message is received.

2.5.1 Building a Local Map

The key property of the model of this section that will be explored in the optimal gossiping algorithm is that by checking the inter-point distances, one can create a "map" with relative locations of the points. Indeed, if for three points u, v, w, their inter-points distances is known, then if u is chosen to be the origin (that is, has location $(0, 0)$), one can give relative locations of the other two points v and w. (The relative location is not unique because there are two possible locations, but by symmetry, any of these two positions will suffice for the analysis.) It will be shown later that with such a map, the gossiping task can be performed optimally.

Lemma 2.8 *After $\mathcal{O}(D)$ communication steps, all nodes $u \in \mathcal{N}$ can learn* $\mathsf{dist}(u, v)$ *for any node $v \in N^{\tau}(u)$, where $\tau = \lceil 1/(n^2 r^5 \log n)\rceil$. (This algorithm may fail with probability at most $1 - 1/n^3$.)*

Proof: The following simple and well-known geometric lemma is essential for the proof of the lemma.

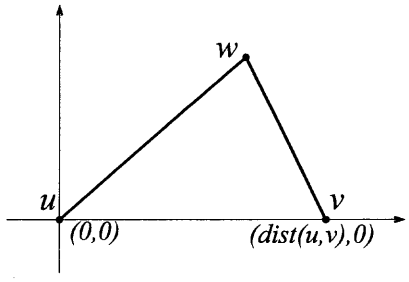

Figure 2.2 Figure depicting the first step in creating a local map of a node, as discussed in Section 2.5.1 and Lemma 2.8. The triangle shows the locations of three points u, v, w and the local map of u. The location of w is set to $\left(\frac{\mathsf{dist}(u,w)^2-\mathsf{dist}(w,v)^2+\mathsf{dist}(u,v)^2}{2\,\mathsf{dist}(u,v)}, \sqrt{\mathsf{dist}(u,w)^2 - \left(\frac{\mathsf{dist}(u,w)^2-\mathsf{dist}(w,v)^2+\mathsf{dist}(u,v)^2}{2\,\mathsf{dist}(u,v)}\right)^2}\right)$.

Fact 1 *Let $\alpha, \beta, \gamma, \delta$ be four points on the plane. If locations of α, β, γ and the distances between all pairs of the points are known, then the location of δ is uniquely determined and can be computed in constant time.*

Now, one can prove Lemma 2.8. First, use the construction described in Lemma 2.1 with $k = 1$. By Lemma 2.1, each node learns the distances to all its neighbors. Next, apply Lemma 2.1 with $k = \tau$, to ensure that each node u receives all the information about the nodes in $N^\tau(u)$. In particular, for each node $u \in \mathcal{N}$, for each node $v \in N^\tau(u)$, for each $w \in N(v)$, u knows the distance $\mathsf{dist}(v, w)$.

Now, with this information at hand, each node $u \in \mathcal{N}$ builds its relative coordinates as follows. u assumes that its coordinates are $(0, 0)$. Then, u takes an arbitrary node $v \in N(u)$ and assigns coordinate $(\mathsf{dist}(u, v), 0)$ to v. Next, u takes another arbitrary node $w \in N(u)$, $w \neq v$, and assigns to it coordinate as shown in the Figure 2.2. Once the coordinate for w is set, then one can assign coordinates to all other nodes in $N^\tau(u)$ in a unique way. First all the nodes in $N(u)$ are taken and the coordinates are assigned to all of them by Fact 1 . Next, the same process is done with the nodes in $N^2(u)$, then with the nodes in $N^3(u)$, and so on, until the (relative) coordinates to all nodes in $N^\tau(u)$ are assigned. (Notice that this construction requires that one can order the points in $N^\tau(u)$ as $p_0, p_1, \ldots$ such that $p_0 = u$, $p_1 = v$, $p_2 = w$, and for every $i > 2$ there are three points $p_{j_1}, p_{j_2}, p_{j_3}$, $j_1, j_2, j_3 < i$, such that $p_i \in N(p_{j_1}) \cap N(p_{j_2}) \cap N(p_{j_3})$. The existence of

such an order is a trivial property of random graphs and it holds with probability at least $1 - 1/n^3$.)

Observe that in this algorithm, the only communication is needed to compute the distances to the neighbors and then to compute sets $N^\tau(u)$. The computation of local coordinate system is done locally and no communication cost is involved. Therefore, by Lemma 2.1, the running time is $\mathcal{O}(\tau \cdot n^2 r^4 \log n)$, which is $\mathcal{O}(D)$ by the choice of $\tau = \lceil 1/n^2 r^5 \log n \rceil$, the assumption in Section 2.5 that $r \leq \mathcal{O}\left(\frac{1}{n^{2/5} \log^{1/5} n}\right)$, and the fact that $D = \Theta(1/r)$ with high probability. This yields the lemma.

□

This lemma implies not only that $u \in \mathcal{N}$ can learn $\mathsf{dist}(u, v)$ for any node $v \in N^\tau(u)$, but also that it can set up its own local map of the nodes in $N^\tau(u)$. From now on, $\tau = \lceil 1/(n^2 r^5 \log n) \rceil$ is assumed.

2.5.2 Boundary and Corner Nodes

In the algorithm two special types of nodes are considered: *boundary nodes* and *corner nodes*. Intuitively, a corner node is close to a corner of $[0, 1]^2$ and a boundary node is close to the boundary of the unit square. With the help of the local map, every node can itself determine if it is a corner node or a boundary node. If a node u observes that there is a sector with angle $\pi/2$ that is centered at u so that every neighbor of u in that sector is at a distance at most $r/\sqrt{2}$, then u marks itself as a *boundary node* (see also Figure 2.3). It is easy to see that with high probability, a node is a boundary node only if its distance to the boundary of $[0, 1]^2$ is less than r, and also every node which is at a distance at most $r/2$ from the boundary is a boundary node. Similarly, a node u marks itself as a *corner node* if there is a line going through u for which all neighbors of u that are on one side of the line have a distance at most $r/2$ from u (see also Figure 2.3). It is easy to see that with high probability, every corner node is at a distance at most r from a corner of $[0, 1]^2$ and every node that is at a distance at most $r/4$ from a corner of $[0, 1]^2$ is a corner node.

Figure 2.3 A figure describing the definitions from Section 2.5.2. If u is at a distance at most $r/2$ to the boundary then there is a sector with angle $\pi/2$ with no point at a distance more than $r/\sqrt{2}$ from u. If u is at a distance at most $r/4$ to a corner then there is a line going through u so that all points at one side of the line are at a distance at most $r/2$.

From now on, it will be assumed that every boundary node is at a distance at most r from the boundary of $[0,1]^2$ and every corner node is at a distance at most r from a corner of $[0,1]^2$. Since these claims hold with high probability, the assumption is conditional and holds with high probability.

Next, select one *corner representative node* for each corner of $[0,1]^2$. By Lemma 2.1, all nodes can learn its neighborhood in $\mathcal{O}(n^2 r^4 \log n) \leq \mathcal{O}(D)$ time. Thus, in $\mathcal{O}(D)$ time each corner node can select itself as a corner representative node if it has the smallest ID among all corner nodes in its neighborhood.

2.5.3 Transmitting along Boundary Nodes

Now, it can be shown that gossiping among boundary nodes can be performed in optimal $\mathcal{O}(D)$ time. Assume that each node already knows if it is a boundary node and if it is a corner representative node. Furthermore, assume that each node $u \in \mathcal{N}$ knows its own local map of the nodes in $N^{\tau}(u)$ (and the messages from these nodes), as discussed in Section 2.5.1.

The process of gossiping among the boundary nodes is initialized by the four corner representative nodes. Each corner representative node u checks its map of the nodes in $N^{\tau}(u)$ and selects two farthest boundary nodes, one for each boundary. Then, it sends a message to these two nodes with the aim of transmitting its message to the two neighboring corner representative nodes (see also Figure 2.4).

The process of sending messages to the corners can then be described to work in *phases*. In each phase, there are up to eight pairs of nodes ϖ_i^j and ϖ_{i+1}^j such that node ϖ_i^j

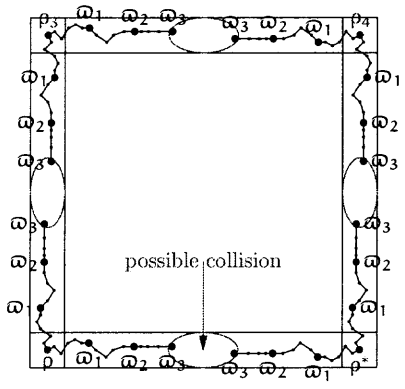

Figure 2.4 Transmitting along boundaries.

wants to transmit a message to node ϖ_{i+1}^j, with both ϖ_i^j and ϖ_{i+1}^j being boundary nodes and $\varpi_{i+1}^j \in N^\tau(\varpi_i^j)$. At the beginning of the phase, the node ϖ_i^j checks its local map and finds a path P_{ij} from ϖ_i^j to ϖ_{i+1}^j of length at most τ. Then, it transmits to its neighbors and request that only the first node on P_{ij} will transmit the message to ϖ_{i+1}^j. Then, the first node on P_{ij} will transmit to its neighbors and will request that only the second neighbor on P_{ij} will transmit, and so on, until the node ϖ_{i+1}^j will receive the message. Once ϖ_{i+1}^j received a message, it sends back an acknowledgement to ϖ_i^j that the message has been delivered. The algorithm for sending an acknowledgement is a reverse of the algorithm for transmitting a message from ϖ_i^j to ϖ_{i+1}^j.

The last step of each phase is to establish the next nodes ϖ_{i+2}^j. If ϖ_i^j sent a message to ϖ_{i+1}^j then ϖ_{i+1}^j checks its map and selects as ϖ_{i+2}^j a node in $\varpi_{i+2}^j \in N^\tau(\varpi_{i+1}^j)$ that is farthest from ϖ_i^j. As an exception, if one of the corner representative nodes is in $N^\tau(\varpi_{i+1}^j) \setminus \{\varpi_i^j\}$, then this corner representative node is selected as ϖ_{i+2}^j and then the process stops, i.e., ϖ_{i+3}^j will not be selected.

Obviously, if there are no transmission conflicts between the eight pairs ϖ_i^j and ϖ_{i+1}^j, then each phase can be performed in 2τ communication steps (including sending the acknowledgement.) The only way of having a transmission conflict is that two pairs ϖ_i^j and ϖ_{i+1}^j, and $\varpi_i^{j'}$ and $\varpi_{i+1}^{j'}$, are transmitting along the same boundary and that in this phase $N^\tau(\varpi_i^j) \cap N^\tau(\varpi_i^{j'}) \neq \emptyset$. If this happen, then the nodes ϖ_i^j and $\varpi_i^{j'}$ may not obtain

an acknowledgement. In this case, both ϖ_i^j and $\varpi_i^{j'}$ repeat the process of transmitting their messages to ϖ_{i+1}^j and $\varpi_{i+1}^{j'}$, respectively, using the selector approach from Lemma 2.1 that ensures that the phase will be completed in $\mathcal{O}(\tau \cdot n^2 r^4 \log n) = \mathcal{O}(D)$ communication steps.

Let ϱ_1 and ϱ_2 be two adjacent corner representative nodes, and $(\varrho_2, \varpi_1^j, \varpi_2^j, \varpi_3^j, \ldots \varrho_1)$ be a sequence of nodes initialized by ϱ_2 in the process described before. It is easy to see that:

(i) ϱ_1 receives all the messages of $\varrho_2, \varpi_1^j, \varpi_2^j, \varpi_3^j, \ldots$, ***(ii)*** ϱ_2 sends its message to all nodes in $\varpi_1^j, \varpi_2^j, \varpi_3^j, \ldots \varrho_1$, and ***(iii)*** for any boundary node v, there is a ϖ_i^j such that $v \in N^\tau(\varpi_i^j)$ (which holds because of the way of picking ϖ_i^j).

Therefore, each corner representative node will receive all messages from the boundary nodes of its incident boundaries. If this process is repeated again, then each corner representative node will receive the messages of *all* boundary nodes. If this process is repeated once again, then all ϖ_i^j nodes will receive the messages from all boundary nodes. If now the approach from Lemma 2.1 is applied, then each boundary node will receive a message from at least one ϖ_i^j, and hence it will receive messages from all boundary nodes.

By the comments above, if there is no conflict in a phase, then the phase is completed in 2τ communication steps, but if there is a conflict, then the number of communication steps in the phase is $\mathcal{O}(\tau\, n^2 r^4 \log n)$. If a corner representative node originates a transmission that should reach another corner representative node, then there will be at most a constant number of phases in which there will be a conflict. Therefore, the total running time for this algorithm is $\mathcal{O}(\tau \cdot D/\tau) + \mathcal{O}(\tau\, n^2 r^4 \log n) = \mathcal{O}(D)$.

Lemma 2.9 *The algorithm above completes gossiping among all boundary nodes in $\mathcal{O}(D)$ time.*

2.5.4 Gossiping via Transmitting along Almost Parallel Lines

Now, the result from Section 2.5.3 is extended to perform gossiping for the entire network $\mathcal{N}$. Assume that the algorithm from Section 2.5.3 has been completed and that each boundary node knows all four corner representative nodes and knows their local maps of all boundary nodes.

Let ϱ be the corner representative node with the smallest ID. Let ϱ^* be the corner representative node that shares the boundary with ϱ (there are two such nodes) and that has the smaller ID. Let ϱ select $\mathcal{O}(D/\tau)$ boundary nodes $\varsigma_1, \varsigma_2, \ldots$ such that

(i) $\varsigma_{i+1} \in N^{\lfloor \tau/4 \rfloor}(\varsigma_i)$ for every i, and **(ii)** $\varsigma_j \notin N^{\lfloor \tau/8 \rfloor}(\varsigma_i)$ for every $i, j, i \neq j$.

It is easy to see that such a sequence exists that ϱ is able to determine the sequence because after the gossiping process from Section 2.5.3, ϱ knows all boundary nodes and their τ neighbors. Next, ϱ informs all boundary nodes about its choice using the process from the previous section.

Now an algorithm in which all the nodes ς_i will originate a procedure ***Straight-line transmission*** aiming at disseminating the information contained by these nodes along a line orthogonal to the boundary shared by ϱ and ϱ^* will be presented.

There are a few problems with this approach that need to be addressed. First of all, the boundary of the unit square is unknown and instead, the goal will be to consider lines orthogonal to the line $\mathcal{L}$ going through ϱ and ϱ^*. The location of $\mathcal{L}$ can be determined from the local map known to all the boundary nodes. Notice that since the angle between the boundary of $[0,1]^2$ and $\mathcal{L}$ is at most $\mathcal{O}(r)$, $\mathcal{L}$ is a good approximation of the boundary of $[0,1]^2$. Next, observe that one is not be able to do any transmissions along any single line because the network $\mathcal{N}$ does not contain three collinear nodes with high probability. Therefore, the process will need to proceed along an approximate line.

The following lemma is the key point to quantify the angle between the perfect line to transmit along and the line along which to actually transmit.

Lemma 2.10 *Let* $\tau = \lceil 1/(n^2 r^5 \log n) \rceil$ *and* $r \leq \mathcal{O}\left(\frac{1}{n^{7/16} \log^{5/16} n}\right)$. *Let* u *be a node in* $\mathcal{N}$ *and let* ℓ_u *be any ray (half-line) starting at* u. *If all points* $q \in \ell_u$ *with* $\mathsf{dist}(u,q) \leq \tau \cdot r$ *are*

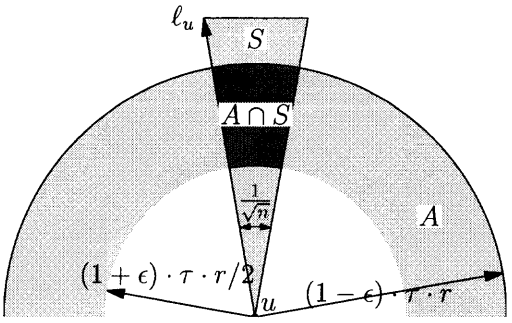

Figure 2.5 Construction used in the proof of Lemma 2.10.

contained in $[0,1]^2$ *then with high probability there is a node* $w \in N^{\lfloor \tau/4 \rfloor}(u) \setminus N^{\lceil \tau/8 \rceil}(u)$ *such that* $|\measuredangle(\ell_u uw)| \leq n^{-1/2}$. *The angle between* ℓ_u *and the line going through* u *and* w *is at most* $n^{-1/2}$.

Proof: By Lemma 2.17 (an auxiliary lemma in Section 2.8), for every node $z \in SN^{\tau}(u)$ (e.g., with z being in the strict τth neighborhood of u), one has $(1-o(1))\cdot\tau\cdot r \leq \mathsf{dist}(u,z) \leq \tau\cdot r$. One can proceed as depicted in Figure 2.5. Consider the area A which is the difference of two disks centered at u, one of radius $(1-\epsilon)\cdot\tau\cdot r/4$ and another of radius $(1+\epsilon)\cdot\tau\cdot r/8$, that is, $A = \mathfrak{B}(u,(1-\epsilon)\cdot\tau\cdot r/4) \setminus \mathfrak{B}(u,(1+\epsilon)\cdot\tau\cdot r/8)$. Here, it is assumed that ϵ is a small positive constant such that all points in A belong to $N^{\lfloor \tau/4 \rfloor}(u) \setminus N^{\lceil \tau/8 \rceil}(u)$. (The existence of ϵ follows from Lemma 2.17.)

Consider the sector S of angle $n^{-1/2}$ with the center at u and with ℓ_u being its bisector. The expected number of points in A is greater than $n/\mathrm{polylog}(n)$. Therefore, the expected number of points in $A \cap S$ is $\omega(n^{1/3})$. Hence, with high probability there is a node in $A \cap S$, that yields the lemma. □

Straight-line Transmission Now, one can use Lemma 2.10 to design a scheme that allows a point to transmit a message along an approximate line. The procedure Straight-line transmission(s,μ,ℓ) aims at transmitting a message μ from node s along (approximately) line ℓ_s, $s \in \ell_s$, so that all nodes that are close to ℓ_s will receive the message μ.

In Straight-line transmission(s, μ, ℓ_s), the node s initiates sending its message μ along the line ℓ_s, $u \in \ell_s$. The transmission process is performed in *phases*; each phase consists of sending a message from a node ϖ_i to another node ϖ_{i+1} such that ℓ_s is approximately equal to the line going through ϖ_i and ϖ_{i+1}, and $\varpi_{i+1} \in N^{\lfloor \tau/4 \rfloor}(\varpi_i) \setminus N^{\lfloor \tau/8 \rfloor}(\varpi_i)$. The nodes ϖ_i are determined recursively. Initially, $\varpi_0 = s$ and ϖ_1 is the node $q \in N^{\lfloor \tau/4 \rfloor}(s) \setminus N^{\lfloor \tau/8 \rfloor}(s)$ for which $|\measuredangle(\ell_s s q)|$ is minimized. If $i \geq 1$ and ϖ_i is determined, then

(i) if $N^{\tau}(\varpi_i) \setminus \{\varpi_{i-1}\}$ contains a boundary node then ϖ_{i+1} is undefined and the process is stopped; ***(ii)*** otherwise, ϖ_{i+1} is selected to be the node $u \in N^{\lfloor \tau/4 \rfloor}(\varpi_i) \setminus N^{\lfloor \tau/8 \rfloor}(\varpi_i)$ for which $|\measuredangle(\varpi_{i-1}\varpi_i\varpi_{i+1}) - \pi|$ is minimized.

Since ϖ_i knows the locations of all nodes in $N^{\tau}(\varpi_i)$, ϖ_i is able to select the node ϖ_{i+1} using its local map.

Observe that by Lemma 2.10, for every node ϖ_i, $i \geq 1$, $|\measuredangle(\varpi_{i-1}\varpi_i\varpi_{i+1}) - \pi| \leq \mathcal{O}(n^{-1/2})$, with high probability. Next, since $\mathsf{dist}(\varpi_i\varpi_{i+1}) = \Theta(\tau \cdot r) = \Theta(\frac{1}{n^2 r^4 \log n})$, it can be concluded that the last representative ϖ_i will have index $\mathcal{O}(\frac{1}{n^2 r^4 \log n})$. Therefore, for every i, $i \geq 2$, $|\measuredangle(\ell_s u \varpi_i)| \leq \mathcal{O}(\frac{1}{n^2 r^4 \log n} \cdot n^{-1/2}) \leq n^{-1/3}$ with high probability.

Before the running time of *Straight-line transmission* (s, μ, ℓ_s) is analyzed, the usage of this procedure is discussed first. Straight-line transmission(s, μ, ℓ_s) will be called with s being the nodes ϱ, ϱ^*, and $\varsigma_1, \varsigma_2, \ldots$, as defined at the beginning of Section 2.5.4, and with line ℓ_s being the line going through s that is orthogonal to the line $\mathcal{L}$ (which is the line going through ϱ and ϱ^*). (See also Figure 2.6).

Observe that since by Lemma 2.17, the distance between any of the points ϱ, ϱ^*, and $\varsigma_1, \varsigma_2, \ldots$ is at least $\Omega(\tau \cdot r) = \Omega(\frac{1}{n^2 r^4 \log n})$, so are the distance between the lines ℓ_s. On the other hand, as it is argued above, every procedure Straight-line transmission(s, μ, ℓ_s) is sending messages only among the nodes that are at distance at most $n^{-1/3}$ from the line ℓ_s, where this claim holds with high probability. Therefore, in particular, the communication in the calls to Straight-line transmission(s, μ, ℓ_s) will be done without any interference between the calls, with high probability.

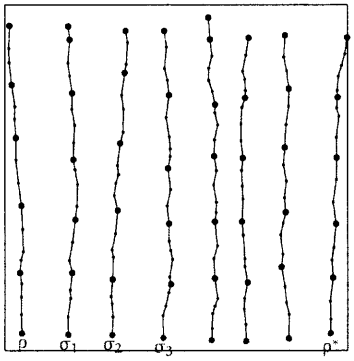

Figure 2.6 ϱ, ς_i and ϱ^* do transmitting-along-a-line.

Now one can argue that each phase can be done in time $\mathcal{O}(\tau)$. First, node s checks its local map, chooses node ϖ_1, and selects a path from s to ϖ_1 with $\mathcal{O}(\tau)$ hops. Next, s sends a message containing the message μ and the information about nodes on the selected path from s to ϖ_1. The first node on this path, say p_1, is in $N(s)$ and therefore it receives the message in the first round. Then, in the second round only the node p_1 transmits and the second node on the path, say p_2, will receive the message. Proceeding in this way, ϖ_1 will receive the message in $\mathcal{O}(\tau)$ steps. This establishes the running time of $\mathcal{O}(\tau)$ for the first phase; every other phase can be implemented in identical way. Since there are $\mathcal{O}(D/\tau)$ phases, this yields the following.

Lemma 2.11 *All calls to Straight-line transmission(s, μ, ℓ_s) with s being ϱ, ϱ^*, and $\varsigma_1, \varsigma_2, \ldots$ can be completed in $\mathcal{O}(D)$ communication steps, with high probability.*

Observe that while running the procedures Straight-line transmission, each node that is transmitting can include in its message also all the knowledge it contains at a given moment. Therefore, in particular, each last node ϖ_k will receive all the messages collected on its path from s.

Next, observe that for every node q in the network $\mathcal{N}$ either q has been selected as one of the nodes ϖ_i in one of the calls to Straight-line transmission or one of the nodes in $N^\tau(q)$ has. Indeed, since the distance between the adjacent lines ℓ_s is at most $\lfloor \tau/4 \rfloor \cdot r$, for each point $q \in \mathcal{N}$ there is a line ℓ_s with $\mathsf{dist}(q, \ell_s) \leq \lfloor \tau/4 \rfloor \cdot r/2$. Therefore, by the

analysis of the procedure Straight-line transmission, there will be at least one node ϖ_i for Straight-line transmission(s, μ, ℓ_s) with dist(q, ϖ_i) $\leq \tau \cdot r/2$. And by Lemma 2.17, this implies that $\varpi_i \in N^\tau(q)$ with high probability.

Because of this, if all nodes $u \in \mathcal{N}$ know the messages from all nodes in $N^\tau(u)$, then after completing the calls to Straight-line transmission, for each node $u \in \mathcal{N}$ there will be at least one boundary node that received the message of u.

2.5.5 Gossiping Algorithm

The complete gossiping algorithm will be presented in this section. First run the algorithms presented in Lemma 2.9, that perform gossiping among the boundary nodes. Afterwards, all boundary nodes know all the messages from other boundary nodes and also know the local maps with all boundary nodes. Next, run Straight-line transmission(s, μ, ℓ_s) with s being ϱ, ϱ^*, and $\varsigma_1, \varsigma_2, \ldots$, as discussed at the beginning of section 2.5.4. Then, as argued at the end of Section 2.5.4, the boundary nodes will have the messages from all the nodes in the network. Therefore, if one does gossiping among the boundary nodes (cf. Section 2.5.3) once again, then all the boundary nodes will have the messages from all the nodes in $\mathcal{N}$. Next, run again Straight-line transmission(s, μ, ℓ_s) with s being ϱ, ϱ^*, and $\varsigma_1, \varsigma_2, \ldots$. Then, all the nodes ϖ_i will obtain the messages from all nodes in $\mathcal{N}$. Finally, since each $q \in \mathcal{N}$ has in its τ-neighborhood a node ϖ_i, on can apply Lemma 2.1 to ensure that all nodes in $\mathcal{N}$ will receive the messages from all other nodes in $\mathcal{N}$. Since the total number of communication steps in the algorithm above is $\mathcal{O}(D)$.

Theorem 2.12 *Let* $c\sqrt{\log n/n} \leq r \leq \mathcal{O}\left(\frac{1}{n^{2/5}\log^{1/5} n}\right)$. *If during the communication between a pair of nodes the node receiving the message is able to determine the distance to the other node, then gossiping in a random geometric network can be completed in deterministic time* $\mathcal{O}(D)$. *The algorithm may fail with probability at most* $1/n^2$.

2.6 Deterministic Distributed Algorithm: Knowing Angles Helps

One can modify the algorithm from Theorem 2.12 to work in the scenario in which a node cannot determine the distance between its neighboring node but instead, it is able to

determine the relative direction where the neighbor is located: if a node received messages from any two of its neighbors, then it is able to determine the angle between them.

The algorithm for this model is essentially the same as that described in Section 2.5 with two differences. First of all, now the local map of a node does not have the exact distances but it may be re-scaled. That is, using the same approach as presented in Section 2.5.1, each node can build its local map where all the angles in the map are the actual angles between the points, but only the distances may be re-scaled. Secondly, another approach is needed to determine if a node is a boundary node or it is a corner node. This can be done by comparing the density of the neighborhoods of the nodes and details will be described in the follows.

Let $u \in \mathcal{N}$ be an arbitrary point and consider its neighborhood in $\mathcal{N}$, which is exactly the set $N(u) = \mathcal{N} \cap \mathfrak{B}(u,r)$. The goal is to establish the size of $\mathcal{N} \cap \mathfrak{B}(u,r)$ in the case when $\mathfrak{B}(u,r) \subseteq [0,1]^2$ and $\mathfrak{B}(u,r) \not\subseteq [0,1]^2$, where the former case corresponds to the situation when u should be a boundary point or a corner point.

First the case when $\mathfrak{B}(u,r) \subseteq [0,1]^2$ is considered. Then, $|\mathcal{N} \cap \mathfrak{B}(u,r)|$ is the binomial random variable with parameters $n-1$ and πr^2, i.e., $\mathbb{B}(n-1, \pi r^2)$. Now, let $\mu = \mathbf{E}[|\mathcal{N} \cap \mathfrak{B}(u,r)|] = (n-1)\pi r^2$, one can use Chernoff bound to obtain that $\mathbf{Pr}\big[|\mathcal{N} \cap \mathfrak{B}(u,r)| \leq 0.9\mu\big] \leq e^{-\mu/200}$. Next, observe that $r \geq c\sqrt{\log n/n}$, so $e^{-\mu/200} \leq e^{\frac{-(n-1)\pi c^2 \log n/n}{200}} \leq e^{-c^2 \log n/64} \leq n^{-c^2/64}$, where the last inequalities hold for large enough n.

Next, the case when u is at a distance at most $r/2$ from the boundary is considered. Then, since the area of $\mathfrak{B}(u,r) \cap [0,1]^2$ is at most $\frac{2}{3}\pi r^2 + \frac{1}{2} r^2$ (seel also Figure 2.7), then $\mu^* = \mathbf{E}[|\mathcal{N} \cap \mathfrak{B}(u,r)|] \leq (\frac{2}{3}\pi + \frac{1}{2})(n-1) r^2 < \frac{5}{6} \cdot \mu$, where $\mu = (n-1)\pi r^2$. Next, use Chernoff bound to obtain that

$$\begin{aligned}
\mathbf{Pr}\big[|\mathcal{N} \cap \mathfrak{B}(u,r)| \geq 0.9\mu\big] &\leq \mathbf{Pr}\big[|\mathcal{N} \cap \mathfrak{B}(u,r)| \geq (1+\tfrac{1}{15})\mu^*\big] \\
&\leq e^{-\frac{\mu^*}{30^2}} \leq e^{-\frac{\mu}{1800}} < n^{-\frac{c^2}{700}} ,
\end{aligned}$$

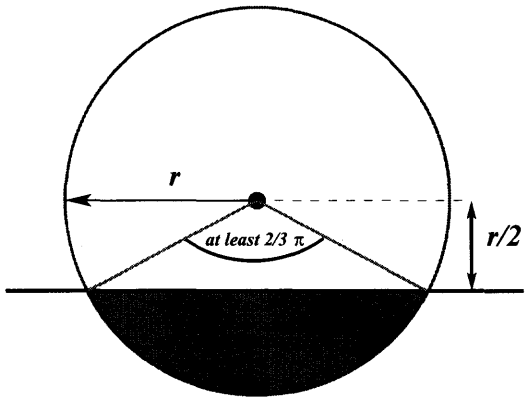

Figure 2.7 Ball $\mathfrak{B}(u, r)$ for a point at a distance at most $r/2$ from the boundary.

where the last inequality uses the fact that $r \geq c\sqrt{\log n/n}$ and assumes that n is sufficiently large.

Therefore, by the analysis, if c is sufficiently large, say $c > 50$, and a node u is defined to be a *boundary node* if $N(u) < 0.9\mu$, then with high probability every node that is at a distance at most $r/2$ from the boundary will become a boundary node and every node that is at a distance larger than r from the boundary is not a boundary node. (See also Figure 2.7.)

Corner nodes can be considered in the similar way. One can show that if c is a sufficiently large constant then if a node u is defined to be a *corner node* when $N(u) < 0.4\mu$, then with high probability every node that is at a distance at most $r/10$ from one of the corners will become a corner node and every node that is at a distance larger than r from the corners is not a corner node. (The constants are not optimized, which could be easily improved.)

With these new definitions of boundary nodes and corner nodes, one can easily convert the algorithm from Theorem 2.12 to obtain Theorem 2.13.

Theorem 2.13 *Let* $c\sqrt{\log n/n} \leq r \leq \mathcal{O}\left(\frac{1}{n^{2/5}\log^{1/5} n}\right)$. *If during the communication between a pair of nodes the node receiving the message is able to determine the relative direction from which the message arrives, then gossiping in a random geometric network*

can be completed in deterministic time $\mathcal{O}(D)$. The algorithm may fail with probability at most $1/n^2$.

2.7 Conclusions

In this chapter the first thorough study of basic communication aspects in random geometric ad-hoc radio networks is presented. It has been shown that in many scenarios, the random structure of these networks (which often may model realistic scenarios from sensor networks) makes it possible to perform communication between the nodes in the network in asymptotically optimal time $\mathcal{O}(D)$, where D is the diameter of the network and thus a trivial lower bound for any communication. This is in contrast to arbitrary ad-hoc radio networks, where deterministic bounds of $o(n)$ are unattainable.

The study shows also that while there is a relatively simple optimal randomized gossiping algorithm and a deterministic one when the nodes have knowledge about their locations in the plane, the other scenarios are more complicated. In particular, it is still unclear that if an $\mathcal{O}(D)$-time gossiping algorithm is possible in the unknown topology model. But the algorithms from Sections 2.5 and 2.6 demonstrate that if the nodes can obtain even a very limited geometric information about the neighbors, then some nice algorithms can coupe with this task.

2.8 An Auxiliary Lemma (Lemma 2.17)

For any point q in $\mathbb{R}^2$, let $q^{(x)}$ denote its x-coordinate and $q^{(y)}$ denote its y-coordinate. For any point q, a *disc of* q is the disc of radius r with the center at q. The *right half-disc* of q is the set of all points in the disc of q whose x-coordinate is greater than or equal to $q^{(x)}$. Similarly, the *left half-disc* of q is the set of all points in the disc of q whose x-coordinate is smaller than or equal to $q^{(y)}$.

Let p be any point in $\mathbb{R}^2$ and assume that it is in the origin. Define a sequence of points $X_0, X_1, \ldots$ as follows:

- $X_0 = p$, and X_i is the neighbor of X_{i-1}.

- if $X_i^{(x)} < 0$ then X_{i+1} is the point in the right half-disc of X_i with maximum y-coordinate (ties broken arbitrarily), and

- if $X_i^{(x)} \geq 0$ then X_{i+1} is the point in the left half-disc of X_i with maximum y-coordinate (ties broken arbitrarily).

Furthermore, assume that whenever one considers index i, the ball of radius r and the center at X_i ($\mathfrak{B}(X_i, r)$) is entirely contained in the unit square $[0,1]^2$. The following claims can be easily obtained.

Claim 2.14 *For every i, point X_i is in the ith neighborhood of p.*

Claim 2.15 *For every i, $|X_i^{(x)}| \leq r$.*

The following lemma is key for the analysis.

Claim 2.16 *For every i, $\mathbf{E}\left[X_{i+1}^{(y)} - X_i^{(y)} | X_i^{(y)}\right] = r \cdot \left(1 - \frac{1}{\Theta(n^{2/3} r^{4/3})}\right)$.*

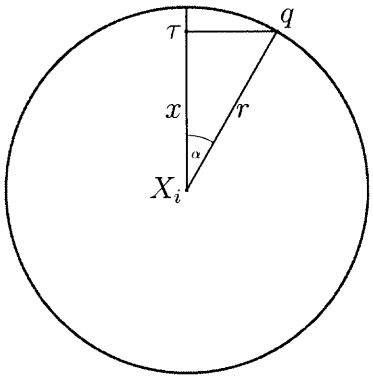

Figure 2.8 Description for the proof of Lemma 2.16.

Proof: Fix X_i and assume, without loss of generality, that $X_i^{(x)} < 0$. Consider a sector $\mathbb{S}$ of the disc of X_i of angle α as in Figure 2.8, with one boundary of the sector parallel to the y-axis. Only the case when $\alpha = o(1)$ needs to be considered. If the other boundary of the sector crosses the disc boundary at point q, then let τ be the projection of q into the y-coordinate. Let $x = \mathsf{dist}(X_i, \tau)$. Notice that $x = r \cdot \cos\alpha$ and $\mathsf{dist}(\tau, q) = r \cdot \sin\alpha$. Using Taylor's expansions, since $\alpha = o(1)$, $x = r \cdot (1 - (\frac{1}{2} + o(1)) \cdot \alpha^2)$ and $\mathsf{dist}(\tau, q) =$

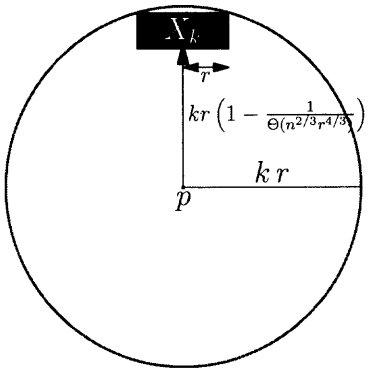

Figure 2.9 Final location of point X_k.

$r \cdot (1 + o(1)) \cdot \alpha$. Let $\mathbb{C} = \mathbb{S} - \triangle(X_i, \tau, q)$; here $\triangle(X_i, \tau, q)$ denotes the triangle on points X_i, τ, q.

Observe that $X_{i+1}^{(y)} - X_i^{(y)}$ is equal to the maximum value of x in Figure 2.8 for which $\mathbb{C}$ is empty. Therefore, in order to estimate the value of $\mathbf{E}\big[X_{i+1}^{(y)} - X_i^{(y)} | X_i^{(y)}\big]$ it is enough to prove that if the angle α is chosen so that $x = r\,(1 - 1/\Theta(\ln^{2/3} n))$ then $\mathbb{C}$ will contain at least one point with a (positive) constant probability, and if x is much bigger then $\mathbb{C}$ contains no point.

Since the probability that $\mathbb{C}$ contains a point depends on the area of $\mathbb{C}$, the goal is to estimate the area of $\mathbb{C}$. Clearly, $\frac{1}{2} \cdot \mathsf{dist}(\tau, q) \cdot (r - x) \leq \mathsf{area}(\mathbb{C}) \leq \mathsf{dist}(\tau, q) \cdot (r - x)$. Hence,

$$\begin{aligned}
\mathsf{area}(\mathbb{C}) &\leq \mathsf{dist}(\tau, q) \cdot (r - x) \\
&= (r \cdot (1 + o(1)) \cdot \alpha) \cdot \big(r \cdot (\tfrac{1}{2} + o(1)) \cdot \alpha^2\big) \;=\; r^2 \cdot \alpha^3 \cdot (\tfrac{1}{2} + o(1)) \;,
\end{aligned}$$

and similarly,

$$\mathsf{area}(\mathbb{C}) \;\geq\; \tfrac{1}{2} \cdot \mathsf{dist}(\tau, q) \cdot (r - x) \;=\; r^2 \cdot \alpha^3 \cdot (\tfrac{1}{4} + o(1)) \;.$$

Therefore, $\mathsf{area}(\mathbb{C}) = \Theta(r^2 \cdot \alpha^3)$. For $\alpha = \Theta(\frac{1}{n^{1/3} r^{2/3}})$, $\mathsf{area}(\mathbb{C}) \geq 1/n$, in which case, one will expect to see at least one point in $\mathbb{C}$. Therefore, the expected value of

$X_{i+1}^{(y)} - X_i^{(y)}$ is equal to some x for which $\alpha = \Theta(\frac{1}{n^{1/3}r^{2/3}})$. Since $x = r \cdot (1 - (\frac{1}{2} + o(1)) \cdot \alpha^2)$, if $\alpha = \Theta(\frac{1}{n^{1/3}r^{2/3}})$ then $x = r \cdot (1 - \Theta(\alpha^2)) = r \cdot \left(1 - \frac{1}{\Theta(n^{2/3}r^{4/3})}\right)$. This implies that $\mathbf{E}\left[X_{i+1}^{(y)} - X_i^{(y)} | X_i^{(y)}\right] = r \cdot (1 - 1/\Theta(n^{2/3}r^{4/3}))$.

□

Now, it is not difficult to extend the result of Lemma 2.16 to the following.

Lemma 2.17 $X_k^{(y)} = k \cdot r \cdot \left(1 - \frac{1}{\Theta(n^{2/3}r^{4/3})}\right)$ *with high probability. Therefore, in particular, if a half-line $\mathcal{L}$ is drawn starting at p at an arbitrary direction, then with high probability, for every integer k, either there is a point $q_k \in N^k(p)$ with* $\mathsf{dist}(p, q_k) = k \cdot r \cdot \left(1 - \frac{1}{\Theta(n^{2/3}r^{4/3})}\right)$ *which is at a distance at most r from $\mathcal{L}$, or there is a point* $\mathsf{out} \in \mathcal{L}$ *with* $\mathsf{dist}(p, \mathsf{out}) = k \cdot r \cdot \left(1 - \frac{1}{\Theta(n^{2/3}r^{4/3})}\right)$, *that is not contained in the unit square.*

2.9 Some Simulation Results of Randomized Algorithm (Section 2.2) and Deterministic Algorithm (Section 2.3)

It has been proven theoretically that when r is not much greater than critical range, both randomized gossiping algorithm and the deterministic algorithm finishes gossiping in $O(D)$ rounds. But one may have noticed that the constant hidden behind the big O notation could be quite large. The simulation results reveal it is not the case. From Table 2.9 and Table 2.9, one can see that when r is near critical range, the randomized algorithm works very fast, with a small constant factor (about 25). The deterministic algorithm runs slightly faster since the location information is known (the constant factor is about 17). As mentioned before, deterministic algorithm needs extra GPS devices and more resources (memory, computation, bandwidth). In most cases, the randomized algorithm is the best choise.

Table 2.1 Randomized Gossiping Algorithm. $r = 2 \times$ critical range. NR is the Number of Rounds

$n =$	1000	2000	3000	4000	5000	6000	7000	8000	9000	10000
NR	510	640	698	790	897	993	1018	1101	1176	1265
NR/D	25.4	24.4	22.4	22.4	23.1	23.5	22.7	22.0	23.5	23.8

Table 2.2 Deterministic Gossiping Algorithm. $r = 2 \times$ critical range. NR is the Number of Rounds

$n =$	1000	2000	3000	4000	5000	6000	7000	8000	9000	10000
NR	361	462	489	571	608	683	704	779	835	932
NR/D	19.0	17.6	15.8	16.2	15.6	16.2	15.6	16.3	16.6	17.5

CHAPTER 3

COMMUNICATION PROBLEMS IN RANDOM LINE-OF-SIGHT AD-HOC RADIO NETWORKS

3.1 Clarify of The Model: New Definition of Collisions

In a classic radio network, a node x gets the message from node y if and only if: ***(i)*** x does not transmit (is in the listening mode) and ***(ii)*** y is the only neighbor of x that is transmitting in that round.In the case that constraint (*ii*) is violated, a *collision* occurs, in which case no message is received by x. In particular, it is assumed that x is unable to detect if a collision happened or none of its neighbors did transmit.

In the classical radio network model a node cannot receive any message if more than one of its neighbors transmits because the radio signals from these nodes will interfere. Although this definition is often used to model radio ad hoc networks, this definition of the collision here is too narrow to fit the framework of ad hoc networks in the line-of-sight model. Since in a radio network messages are sent out via microwave signals, signals can interfere each other if they overlap in space (rather than just by saying that when two or more neighbors transmit). For example, in Figure 3.1, a sending node z can interfere the node x from receiving the message of y, even though z is not the neighbor of x.

To cope with this phenomenon, one more constraint is added to ensure that node x receives the message from y: (*iii*) no neighbor of y is transmitting *and* there is no node z that is transmitting and that lies on a grid-line perpendicular to the segment $\overline{xy}$ and is at a distance at most r from the segment $\overline{xy}$, see, e.g., Figure 3.1. If either condition (*ii*) or (*iii*) is violated, then a collision occurs. It is easy to see that any gossiping algorithm that works in the new model will certainly work in the traditional model, but not vice versa.

3.2 Properties of Random Line-of-Sight Networks

Two parameters, r and p, play a critical role in the analysis of properties of the random line-of-sight networks. Frieze et al. [Frieze et al. 2007] proved that when $r = o(n)$ and

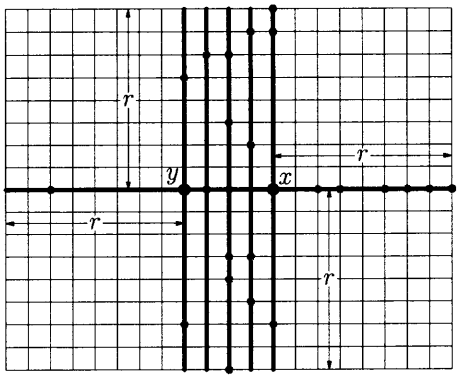

Figure 3.1 The nodes on bold line could cause collision when y sends a message to x.

$r = \omega(\log n)$, if $r \cdot p = o(\log n)$, the network is disconnected with high probability, and therefore no full information exchange (including broadcasting and gossiping processes) can be performed in that case. Therefore it is assumed that $r = o(n)$, $r = \omega(\log n)$, and $r \cdot p \geq c \cdot \log n$ for some sufficiently large constant c. This will ensure that the network is connected with high probability and therefore internode communication is feasible.

Assuming that $r \cdot p > c \cdot \log n$ for some sufficiently large constant c, some further assumptions about the structure of the input network can also been made. It is easy to prove that such a random line-of-sight network has minimum and maximum degree $\Theta(r\,p)$, and has *diameter* $D = \Theta(n/r)$, where these bounds hold with high probability. Besides these easy properties, some nontrivial properties are presented as follows.

For any $r \times r$ square in the grid T, the graph induced by the nodes in this sub-grid is called a *r-graph*.

Lemma 3.1 *If $r\,p > c \log n$ for an appropriate constant c, then with high probability:*

(1) all r-graphs are connected, and

(2) the diameter of any r-graph is $\alpha = \Theta(\log r / \log(p\,r))$.

Proof: This has been proven in [Frieze et al. 2007] (the second bound is proven in the full version of [Frieze et al. 2007]). □

For simplicity of presentation, throughout the section the term α will be used as in the lemma above.

All the claims in Lemma 3.1 hold *with high probability*, that is, with probability at least $1 - 1/n^3$. Therefore, from now on, these events are implicitly conditioned on.

3.3 Preliminaries

Let V be the set of nodes in the grid. For any node x, define $N(x)$ to be the set of nodes that are reachable from x in one hop, $N(x) = \{y \in V : \mathsf{dist}(v,u) \leq r \textit{ and } x, y \textit{ are on the same straight line}\}$, where $\mathsf{dist}(v,u)$ is the distance between v and u. Any node in $N(v)$ is called a *neighbor* of v, and the set $N(v)$ is called the *neighborhood* of v. For any $X \subseteq V$, let $N(X) = \bigcup_{x \in X} N(x)$. Define the *kth neighborhood* of a node v, $N^k(v)$, recursively as follows: $N^0(v) = v$ and $N^k(v) = N(N^{k-1}(v))$ for $k \geq 1$. Let Δ be the *maximum degree* and D be the *diameter* of the radio network. $\Delta = \max_{v \in V}\{|N(v)|\}$ and $D = \min_{v \in V}\{k : N^k(v) = V\}$. As mentioned earlier, $\Delta = \Theta(r\,p)$ and $D = \Theta(n/r)$, with high probability.

Definition 3.2 (Collision sets) *Let $x, y \in T$ with $x \in N(y)$. Define the* collision set for the communication from y to x, *denoted by $C(y,x)$, to be the set of nodes that can interfere x from receiving a message from y. The set $C(y,x)$ contains all nodes $z \in T$ that satisfy one of the following:*

1. *$z \in N(x) \cup N(y)$, or*
2. *there is a grid point q such that* **(i)** *q lies on the segment connecting x and y,* **(ii)** *grid line $\overline{zq}$ is orthogonal to the grid line $\overline{xy}$, and* **(iii)** $\mathsf{dist}(z,q) \leq r$.

It is easy to see that $C(y,x) = \mathcal{O}(r^2\,p)$ with high probability.

By using the *strongly selective family* on line-of-sight networks (and for the new notion of collision sets), one can get a similar lemma as Lemma 2.1. Since the proof of it shares the same idea as the proof of the Lemma 2.1, the proof is omitted here.

Lemma 3.3 *In random line-of-sight networks, for any integer k, in* ***(deterministic)*** *time $\mathcal{O}(k\,r^4\,p^2\,\log n)$ all nodes can send their messages to all nodes in their kth neighborhood.*

The algorithm may fail with probability at most $1/n^2$ *(where the probability is with respect to the random choice of the nodes in the line-of-sight network).*

Observe that in Lemma 3.3, if $k = \mathcal{O}(n/(r^5 p^2 \log n))$ then the running time is $\mathcal{O}(D)$. In particular, if k is a constant and $r = \mathcal{O}\left(\frac{n^{1/5}}{p^{2/5} \log^{1/5} n}\right)$ then the running time is $\mathcal{O}(D)$. Therefore, in order to use this lemma as a subroutine in an $\mathcal{O}(D)$-time algorithm, one requires that $r = \mathcal{O}\left(\frac{n^{1/5}}{p^{2/5} \log^{1/5} n}\right)$.

The following is an immediate corollary of Lemma 3.3 obtained by setting $k = D$ (which corresponds to the bound from [Clementi et al. 2003] in current setting):

Corollary 3.4 *Distributed gossiping in random line-of-sight networks can be performed in* ***deterministic*** *time* $\mathcal{O}(D\, r^4 p^2 \log n) = \mathcal{O}(n\, r^3 p^2 \log n)$. *The algorithm may fail with probability at most* $1/n^2$.

Since $r\,p = \Omega(\log n)$, the running time of this algorithm is $\Omega(n \log^4 n)$ in the best case, and thus it is *superlinear*. The goal of this chapter is to develop algorithms that are faster, optimally, those that achieve the running time of the order of D, the diameter of the network, which is a trivial asymptotic lower bound for broadcasting and gossiping.

3.4 Deterministic Algorithm with Position Information

The gossiping problem in random line-of-sight networks is considered in the model, where each node knows its own geometric position in the grid. In such model, Dessmark and Pelc [Dessmark and Pelc 2007] give a deterministic distributed broadcasting algorithm that runs in $\mathcal{O}(D)$ time. It can be applied to solve the broadcasting problem in the model of this section, with the same running time. One can prove a similar result for gossiping by extending the preprocessing phase from [Dessmark and Pelc 2007] and use an appropriate strongly-selective family to collect information about the neighbors of each node.

Theorem 3.5 *If every input node knows its location in the* $n \times n$ *grid, then the algorithm Gossiping-Known-Locations-D below will complete gossiping in a random line-of-sight network in deterministic time* $\mathcal{O}(\alpha\, r^4 p^2 \log n + n/r)$. *The algorithm may fail with probability at most* $1/n^2$.

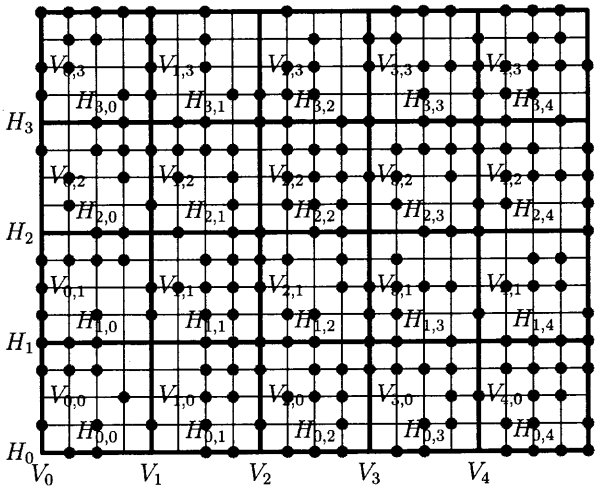

Figure 3.2 Horizontal and vertical segments.

Observe that for $r = \mathcal{O}\left(\frac{n^{1/5}}{\alpha^{1/5}\, p^{2/5}\, \log^{1/5} n}\right)$ (and hence if $r = \mathcal{O}(n^{1/5}/\log^{2/5} n)$) the algorithm achieves the asymptotically optimal running time of $\mathcal{O}(D)$.

Here are some notations. First label the horizontal lines in the grid as H_1, H_2, ..., H_n from bottom to top; label the vertical lines in the grid as $V_1, V_2, \ldots, V_n$ from left to right. The crossing point of H_i and V_j is denoted as (H_i, V_j). For each H_i, H_i is further divide into *segments* of length of $r/2$, except for the last segment which is of length at most $r/2$, and label them as: $H_{i,1}, H_{i,2}, \ldots, H_{i,\lceil 2n/r \rceil}$ from left to right. $V_{j,1}, \ldots, v_{j,\lceil 2n/r \rceil}$ is defined in a similar way, from bottom to top. See Figure 3.2.

```
Gossiping-Known-Locations-D
Preprocessing: do local gossiping using Lemma 3.3 (with k = α)
        to ensure that each node knows messages and positions of its neighbors
    for j = 1 to ⌈2n/r⌉ do:
        for c = 1 to 4 do:
            for each i with i mod 4 + 1 ≡ c in parallel do:
                the node with the minimum ID in the H_{i,j} transmits
    for j = ⌈2n/r⌉ downto 1 do:
        for c = 1 to 4 do:
            for each i with i mod 4 + 1 ≡ c in parallel do:
                the node with the minimum ID in the H_{i,j} transmits
    for j = 1 to ⌈2n/r⌉ do:
        for c = 1 to 4 do:
            for each i with i mod 4 + 1 ≡ c in parallel do:
                the node with the minimum ID in the V_{i,j} transmits
    for j = ⌈2n/r⌉ downto 1 do:
        for c = 1 to 4 do:
            for each i with i mod 4 + 1 ≡ c in parallel do:
                the node with the minimum ID in the V_{i,j} transmits
Postprocessing: do local gossiping using Lemma 3.3 (with k = α)
```

It is easy to see that for $rp \geq c \log n$ with a sufficiently large c, with high probability, there is a node in each segment. For any node x, define $M_t(x)$ as the messages known by x at step t. For any segment S, define $M_t(S)$ as the common messages known by all nodes in segment S at step t.

Proof: After the preprocessing in the algorithm Gossiping-Known-Locations-D, by Lemma 3.3, every node knows which segment it belongs to, and every node knows all other nodes in its segment, including their messages and positions. Therefore, in every segment, all the nodes from that block can select a single representative who will be the only node transmitting (for the entire segment) in all the following time slots.

Call all nodes that are scheduled to send by the algorithm *representative nodes*. By the definition of the segment, in any time slot, when a segment (its representative) sends a message, the nearest sending segment is at a distance of $2r$ from it. So, after each sending, the sending segment, say $H_{i,j}$, will successfully send its message $M_t(H_{i,j})$ to segments $H_{i,j-1}$ and $H_{i,j+1}$ if there are such segments. The statement is also true for any segment $V_{i,j}$. For any two nodes v and u in the grid, the algorithm will send the message of u to v successfully: There are two representative nodes that are within the $r \times r$ square centered

at u and v respectively, with high probability. After preprocessing, the message of u will be sent to its representative node. Then, after $\mathcal{O}(n/r)$ steps, the message of u will be sent to the representative node of v. After the postprocessing, the message of u will eventually reach v.

By Lemma 3.3, the running time of the preprocessing step is $\mathcal{O}(\alpha\, r^4\, p^2\, \log n)$, and the running time of the postprocessing phase is also $\mathcal{O}(\alpha\, r^4\, p^2\, \log n)$. Therefore, the total running time of the algorithm Gossiping-Known-Locations-D is $\mathcal{O}(\alpha\, r^4\, p^2\, \log n + n/r)$. □

3.5 Broadcasting and Deterministic Gossiping with a Leader

A more natural model is considered in this section in which each node knows the values of n, r, and p, knows its own ID (which is a unique integer bounded by a polynomial of n), but it is not aware of any other information about the network. In particular, the node does not know its own location. This is the main model for the study of principles of communication in random line-of-sight networks.

It is helpful to start with a slightly relaxed model: there is a special node (*leader*) ℓ in the network, such that ℓ knows that she is the leader, and all other nodes in the network know that they are not the leader. It can be shown that in this model distributed gossiping can be done *deterministically* in time $\mathcal{O}(\alpha\, r^4\, p^2\, \log n + n/r)$. This will immediately imply a deterministic distributed broadcasting algorithm with asymptotically the same running time.

The same preprocessing as that used in algorithm Gossiping-Known-Locations-D from Section 3.4 is executed first. This takes $\mathcal{O}(\alpha\, r^4\, p^2\, \log n)$ steps. After the preprocessing, each node knows its second neighborhood with high probability.

3.5.1 Gossiping along a Grid Line

First, gossiping among the nodes belonging to the same grid line can be performed in optimal $\mathcal{O}(1/r)$ time if there is a leader on the grid line.

The following two lemmas that estimate the size of the common neighborhood in random line-of-sight networks can be easily derived from Chernoff bounds.

Lemma 3.6 *For any pair of nodes u and v belonging to the same grid line, with high probability:*

(i) if $\mathsf{dist}(u,v) \leq r/2$ *then* $|N(u) \cap N(v)| \geq 1.3\,r\,p$*, and*

(ii) if $\mathsf{dist}(u,v) \geq r$ *then* $|N(u) \cap N(v)| \leq 1.2\,r\,p$.

Proof: Let $B(N,P)$ be the binomial random variable with N trials and the success probability P, that is, for every integer s, $0 \leq s \leq N$, $\mathbf{Pr}[B(N,P) = s] = \binom{N}{s} P^s (1-P)^{N-s}$. Two classical Chernoff bounds are used: for $0 < \epsilon < 1$: $\mathbf{Pr}[B(N,P) \geq (1+\epsilon) \cdot N\,P] \leq e^{-\epsilon^2 N P/4}$ and $\mathbf{Pr}[B(N,P) \leq (1-\epsilon) \cdot N\,P] \leq e^{-\epsilon^2 N P/2}$.

(i) If $\mathsf{dist}(u,v) \leq r/2$ then it is easy to see that $\mathbf{Pr}[|N(u) \cap N(v)| \leq 1.3\,r\,p] \leq \mathbf{Pr}[B(1.5\,r,p) \leq 1.3\,r\,p]$. Therefore, by Chernoff bound, $\mathbf{Pr}[|N(u) \cap N(v)| \leq 1.3\,r\,p] \leq \mathbf{Pr}[B(1.5\,r,p) \leq 1.3\,r\,p] \leq e^{-r\,p/75}$, that is smaller than or equal to $1/n^3$, for $r\,p \geq 225 \ln n$.

(ii) If $\mathsf{dist}(u,v) \geq r$ then, similarly, $\mathbf{Pr}[|N(u) \cap N(v)| \geq 1.2\,r\,p] \leq \mathbf{Pr}[B(r,p) \geq 1.2\,r\,p] \leq e^{-r\,p/100} \leq 1/n^3$, for $r\,p \geq 300 \ln n$.

□

Lemma 3.7 *For any node u, if the distance between u and the nearest boundary is greater than or equal to r, then, in each of the four directions, with high probability, there is a neighboring node v of u such that* $1.2\,r\,p \leq |N(u) \cap N(v)| \leq 1.3\,r\,p$.

Proof: Fix a node u and one direction. It is easy to see that with high probability, there is a node v such that $0.74\,r \leq \mathsf{dist}(u,v) \leq 0.76\,r$. $1.24\,r\,p \leq \mathbf{E}[|N(u) \cap N(v)|] \leq 1.26\,r\,p$. Similarly as in the proof of Lemma 3.6, one can show that the following holds with high probability: $1.2\,r\,p \leq |N(u) \cap N(v)| \leq 1.3\,r\,p$. □

The process of gossiping among the nodes on a grid-line is initialized by one specific node (call it a *launching node*). The launching node u checks its second neighborhood $N^2(u)$, and selects one *representative node*, say v, such that $1.2\,r\,p \leq |N(u) \cap N(v)| \leq 1.3\,r\,p$. Then, u sends a message to v with the aims: *(i)* u transmits

its message to v and *(ii)* u informs v that it is picked as representative node. Because of Lemmas 3.6 and 3.7, $r/2 \leq \mathsf{dist}(u, v) \leq r$.

The process of gossiping along a grid line works in steps. At the beginning of each step, a node ϖ_t receives a message from node ϖ_{t-1}, and ϖ_t is informed that it is the representative node. Then ϖ_t will pick a representative node ϖ_{t+1} for the next step, and then send a message to ϖ_{t+1} to inform about it. ϖ_t picks ϖ_{t+1} by checking $N^2(\varpi_t)$, and selecting as ϖ_{t+1} any node fulfilling:

1. $1.2\,r\,p \leq |N(\varpi_t) \cap N(\varpi_{t+1})| \leq 1.3\,r\,p$,

2. $|N(\varpi_{t-1}) \cap N(\varpi_{t+1})| \leq 1.3\,r\,p$.

Because of Lemmas 3.6 and 3.7, it is easy to see that $r/2 \leq \mathsf{dist}(\varpi_{t+1}, \varpi_t) \leq r$. Moreover, ϖ_{t+1} and ϖ_{t-1} are at different sides of ϖ_t, for otherwise $\mathsf{dist}(\varpi_{t+1}, \varpi_{t-1}) \leq r/2$ and then the second constraint would be violated with high probability. If in one step ϖ_t is unable to find the ϖ_{t+1} as defined above, then the distance between ϖ_t and its nearest boundary is less than r. In that case ϖ_t simply stops the process and makes itself as the *last representative node.*

This process is run for $2\,n/r$ steps and the steps spent in this process is called *Phase 1*. Then the last representative node, say ϖ_{t+1}, picks ϖ_t as next representative node and initialize the process again, for another $2\,n/r$ steps. These steps define *Phase 2*. Then, the representative nodes in Phase 2 send their messages in reverse order and with that the gossiping along straight-line will be done. These steps form *Phase 3*. The total running time is $\mathcal{O}(n/r)$.

3.5.2 Broadcasting and Gossiping with the Leader in the Whole Grid

The gossiping algorithm in the model with a distinguished leader ℓ will be presented in this section. First, the leader will pick an arbitrary direction and do gossiping along the corresponding grid line. As a by product of this algorithm, a set of representative nodes will be chosen and the minimum distance between any pair of them is greater than or equal to $r/2$. Next, each of the representative nodes treat itself as the pseudo-leader, and

do gossiping along a grid line in parallel, in an orthogonal direction to that first chosen by the leader. There are two issues to be solved. First, since the distance between these pseudo leaders could be as small as $r/2$, one can interleave the transmissions in adjacent pseudo-leaders, which yields a constant-factor slow-down. Second, by checking its second neighborhood, a pseudo-leader can indeed find a node in orthogonal direction that was first chosen by the leader. (ϖ_i can pick a node y from its neighbors such that $1.2\,r\,p \leq |N(\varpi_t) \cap N(y)| \leq 1.3\,r\,p$ and $|N(\varpi_{t-1}) \cap N(y)| \equiv 1$.)

Next, the whole process is repeated (starting from the first gossiping along a grid line initialized by the leader) once again. It is easy to see that gossiping is done among all representative nodes. For any pair of nodes u and v, there are two representative nodes that are within the $r \times r$ squares centered at u and v respectively, with high probability. After preprocessing, u will send its message to its representative nodes. The message of u then will be sent to the representative node of v in the following steps. Next the postprocessing defined in the algorithm of the Section 3.4 will be run, and the message of u will be sent to v. The running time is $\mathcal{O}(\alpha\, r^4\, p^2\, \log n + n/r)$.

Theorem 3.8 *If there is a leader in the random line-of-sight network, then gossiping can be completed in* ***deterministic*** *time* $\mathcal{O}(\alpha\, r^4\, p^2\, \log n + n/r)$. *The algorithm may fail with probability at most* $1/n^2$.

Since in the broadcasting problem, there always is a given source node which can be used as a leader, Theorem 3.8 immediately implies that broadcasting can also be done in the same time.

Theorem 3.9 (Deterministic broadcasting) Distributed broadcasting in random line-of-sight networks can be performed in deterministic time $\mathcal{O}(\alpha\, r^4\, p^2\, \log n + n/r)$.

3.6 Fast Distributed Randomized Gossiping

The model of random line-of-sight networks in which each node knows the values of n, r, and p, knows its own ID, but it is not aware of any other information about the network is studied in this section. As one has seen in the previous section, if the nodes in the network

can elect a leader then gossiping could be done within the time bounds stated in Theorem 3.8. The problem of leader election is difficult in current setting, but as it will be shown in this section, randomized leader election can be solved efficiently.

At the beginning, each node independently and uniformly at random selects itself as the *leader* with probability $\frac{\log n}{n^2 p}$. By simple probabilistic arguments, one can prove that exactly $\Theta(\log n)$ leaders are chosen with high probability. Then a distributed *minimum finding* algorithm can be executed to eliminate all of them but one, and the one chosen will have the lowest ID.

Each of these leaders will pick four representative nodes along four directions, respectively, and execute the process of *Phase 1* as described in Section 3.5.1. It can be called *fast transmission*. The fast transmission can be interleaved with the preprocessing of the algorithm in Section 3.4, that is called *slow transmission*. In the odd steps, every node follows the schedule of fast transmission, and in the even steps, every node follow the schedule of slow transmission. Since there is more than one leader, it is possible that transmission *collisions* can occur. These collisions are able to be detected, because after ϖ_t picks ϖ_{t+1} as the next representative node and informs it, in the next step, ϖ_t is expected to receive an acknowledgement of the successful transmission from ϖ_{t+1}. If the acknowledgement is not received, ϖ_t knows that a collision happened.

When a node, say u, detects a collision, in the following $\mathcal{O}(\alpha\, r^4 p^2 \log n)$ steps u will send nothing (stay in the listening mode) in odd steps, and run slow transmission as before in even steps. By the property of strongly selective family, u will eventually receive the messages of other representative nodes that are transmitting for their own leaders during this period. Then u compares the ID of its own leader with all other leader's ID that it just received. If (at least) one of those ID is smaller than the ID of its leader, u will send an "eliminating" message back to its leader, reversely along the path through which the leader sent the message to it. The leader eliminates itself after getting this message. If the ID of u's leader is the smallest one, u resumes the fast transmission. Altogether, a node will encounter at most $\mathcal{O}(\log n)$ collisions when it transmits toward any boundary.

[1] Therefore the slow down caused by collisions is small and the total running time is $\mathcal{O}(r^4 p^2 \alpha \log^2 n + n/r)$.

Now the leader is either eliminated or successfully transmits its ID along four directions. Thus, for any pair of surviving leaders, there is a pair of representative nodes in one $r \times r$ square. By running the postprocessing from Section 3.4, the two representative nodes will exchange information about their leaders. Again, the representative node that holds the larger ID will transmit an "eliminating" message to its leader. The running time is $\mathcal{O}(r^4 p^2 \alpha \log^2 n + n/r)$. After this procedure, all but one leader with the smallest ID will survive.

Summarize the result above:

Theorem 3.10 *In the random line-of-sight network, there is a distributed randomized algorithm that finds a single node which is approved by all the nodes in the network as a leader, and which completes the task in time $\mathcal{O}(r^4 p^2 \alpha \log^2 n + n/r)$, with high probability.*

Finally, Theorem 3.10 with the result from the previous section can be combined with Theorem 3.8, to obtain the following result.

Theorem 3.11 (Randomized gossiping) In the random line-of-sight network, distributed gossiping can be done in randomized time $\mathcal{O}(r^4 p^2 \alpha \log^2 n + n/r)$, with high probability.

3.7 Conclusions

Three efficient algorithms are presented for broadcasting and gossiping in the model of random sight-of-line networks. If $r = \mathcal{O}(n^{1/5}/\log^{3/5} n)$, then all algorithms perform $\mathcal{O}(D)$ steps to complete the task, which is clearly asymptotically optimal. While it is certainly very interesting to extend the optimality of these bounds to larger values of r and this is an interesting open problems of extending the results in this chapter to larger values of r or providing lower bounds for the running times for larger values of r, it is believed

[1] It seems that if two nodes with distance less than or equal to r send along the same direction, there are a lot of collisions. But in this case, both nodes will notice the collision at once, so after waiting for $\mathcal{O}(\alpha r^4 p^2 \log n)$ steps, the one with small ID will stop transmitting.

that the case $r = \mathcal{O}(n^{1/5}/\log^{3/5} n)$ covers the most interesting cases, when the graph is relatively sparse and each node is able to communicate only with the nodes that are not a large distance apart. Therefore, the most interesting specific open problems left in this dissertation is to extend the result from Section 3.6 to obtain a distributed *deterministic* algorithm for gossiping. Even more interesting is a more general question: what are the important aspects of random sight-of-line networks to perform fast communication in these networks. The work in this dissertation is only the very first step in that direction.

CHAPTER 4

ONLINE SCHEDULING OF EQUAL-PROCESSING-TIME TASK SYSTEMS

4.1 The 3/2 Bound

In this chapter it will be shown that Hu's algorithm yields a competitive ratio of 3/2 for the problem $P \mid p_j = 1, intree_i\ released\ at\ time\ r_i \mid C_{\max}$; i.e., $C_{\max}(S) \leq (3/2) * C_{\max}(S^*)$, where S is the schedule produced by the online version of Hu's algorithm and S^* is the optimal (offline) schedule.

It is well known that Hu's algorithm is optimal for $P \mid p_j = 1, intree \mid C_{\max}$. It works by first assigning a label to each task which corresponds to the priority of the task; tasks with higher labels have higher priority. Once the labels are assigned, the tasks are scheduled as follows. Whenever a machine becomes free for assignment, assign that task all of whose predecessors have already been executed and which has the largest label among those tasks not yet assigned. In Hu's algorithm, the label of a task is a function of the level of the task.

Definition 4.1 *The level of a task i with no immediate successor is its processing time p_i. The level of a task with immediate successor(s) is its processing time plus the maximum level of its immediate successor(s).*

Hu's labeling algorithm assigns higher labels to tasks at higher levels; ties can be broken in an arbitrary manner.

For any instance of $P \mid p_j = 1, intree \mid C_{\max}$, let S_{Hu} denote the schedule produced by Hu's algorithm. Suppose that in S_{Hu}, the first t ($t \geq 1$) columns are all full columns (i.e., time units during which all machines have been assigned), but the $(t+1)^{st}$ column is a partial column (i.e., time unit during which some machines(s) are idle). Then it is easy to see that all the columns after the $(t+1)^{st}$ column are also partial columns. For any task in the time interval $[t, t+1]$, if it has no predecessor in the first t columns, then it can be moved backward to the time interval $[0, 1]$ and the moving distance of this task is t, which

is the largest possible. Any task in the time interval $[t+i+1, t+i+2]$ $(i \geq 0)$ has at least one predecessor in the time interval $[t+i, t+i+1]$ and it must be scheduled after its predecessor. So it can be moved backward by at most t time units as well.

The improvement that can be made to S_{Hu} in terms of the number of tasks that can finish at any time t' $(t' > t)$ can be bounded. Clearly, one can move the tasks scheduled in the $(t+1)^{st}$ column and thereafter to earlier columns and then correspondingly move some tasks scheduled in the first t columns out to accommodate the tasks that were moved in. The best place to put the tasks that were moved out of the first t columns will be the idle machines in the time interval $[t, t+1]$ and thereafter. The net effect of the move is that the tasks in the time interval $[2t, 2t+1]$ and thereafter are moved to the time interval $[t, t+1]$ and thereafter. For any time point t' $(t' > t)$, the net effect of the move is that the tasks in the time interval $[t+t', t+t'+1]$ and thereafter are moved to the time interval $[t', t'+1]$ and thereafter. In S_{Hu}, one may assume that the first machine contains the largest number of tasks. Since Hu's algorithm is optimal for $P \mid p_j = 1, intree \mid C_{\max}$, the makespan of S_{Hu} cannot be reduced. So one may assume that the tasks on the first machine cannot be moved. Let S'_{Hu} be the schedule after the move; see Figure 1 for illustration. (Note that S'_{Hu} may not be a feasible schedule since some of the precedence constraints may be violated.) With respect to S'_{Hu}, the following property can be obtained:

Property 4.2 *Compared with any other feasible schedule, S'_{Hu} has the largest number of tasks finish at time t' $(t' > t)$.*

The online algorithm is described below, which will be called Algorithm A.

Algorithm A

Whenever new tasks arrive

$t \leftarrow$ the current time

$U \leftarrow$ the set of tasks active (i.e., not finished) at time t

Reschedule the Tasks in U by Hu's Algorithm

Figure 4.2 illustrates Algorithm A. Shown in Figure 4.2 are two intrees released at time $t = 0$ and at time $t = 3$, respectively. Figure 4.2(b) shows the schedule produced by

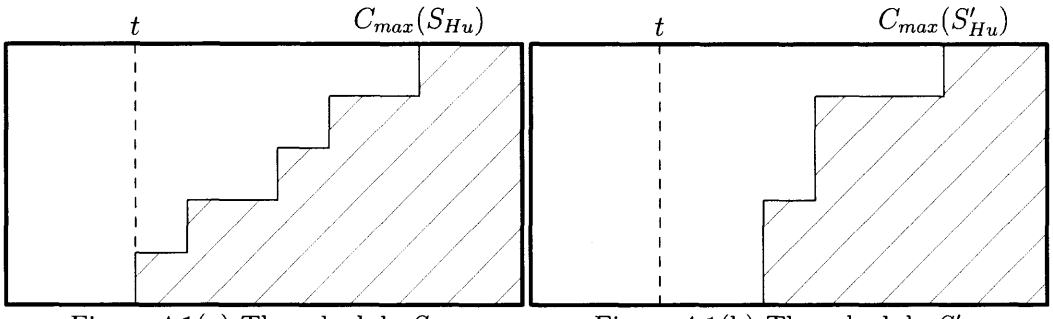

Figure 4.1(a) The schedule S_{Hu} Figure 4.1(b) The schedule S'_{Hu}

Figure 4.1 Bounding the improvement to s_{hu}.

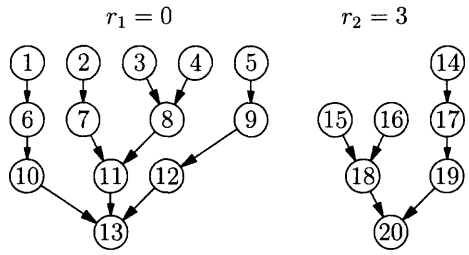

Figure 4.2(a): Two trees released at time 0 and 3

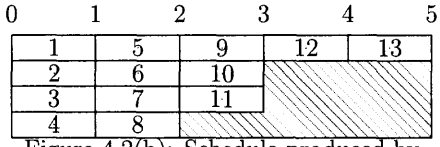

Figure 4.2(b): Schedule produced by algorithm A at time 0

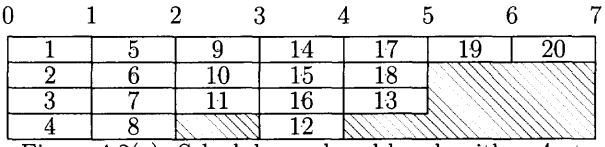

Figure 4.2(c): Schedule produced by algorithm A at time 3

Figure 4.2 An example illustrating algorithm A.

Algorithm A at time $t = 0$, and Figure 4.2(c) shows the schedule produced by Algorithm A at time $t = 3$.

For any instance of $P \mid p_j = 1, intree_i \ released\ at\ time\ r_i \mid C_{\max}$, let S denote the schedule produced by Algorithm A, and let S^* denote the optimal (offline) schedule. Without loss of generality, one may assume that in S, the number of tasks on the jth machine is greater than or equal to the number of tasks on the $(j+1)^{st}$ machine. If not, one can relabel the machines. Assume that the first release time is r_1 and the last release time is r_k. Without loss of generality, one may assume that $r_1 = 0$. The following two cases will be considered.

1. All the machines are full in the time interval $[0, r_k]$.

2. There are some idle machines in the time interval $[0, r_k]$.

These two cases are considered separately in Sections 4.1.1 and 4.1.2, respectively.

4.1.1 Case 1

Let t be the last time instant such that there is a chain, say CH, of tasks scheduled in the time interval $[t, C_{\max}(S)]$, and T_t is the task scheduled in the time interval $[t, t+1]$ of which no predecessor is scheduled in the time interval $[t-1, t]$. Assume chain CH belongs to $intree_j$, its release time is r_j, and it has c tasks (i.e., $c = C_{\max}(S) - t$). Furthermore, assume the level of T_t is h.

Two cases are considered. The first case is when $t = r_j$ while the second case is when $t > r_j$. In the former case, it is easy to see that $C_{\max}(S)$ cannot be reduced. Thus, S is an optimal schedule. In the second case, if all the tasks scheduled in the time interval $[0, t]$ have levels at least h, then $C_{\max}(S)$ cannot be reduced and hence S is again an optimal schedule. However, if some tasks scheduled in the time interval $[0, t]$ have levels less than h, then it is possible to move these tasks to later time, and execute the chain CH earlier. Consequently, $C_{\max}(S)$ can be reduced. In the following the case where some tasks scheduled in the time interval $[0, t]$ have levels less than h is considered. See Figure 4.3 for illustration.

From time 0, let t_{i1} be the first time instant such that there is at least one task with level less than h executing in the time interval $[t_{i1}, t_{i1} + 1]$.

From time t_{i1} on, let t_{j1} be the first time instant such that all the tasks executing in the time interval $[t_{j1}, t_{j1} + 1]$ have levels at least h; see Figure 3 for illustration. Let k_1 be the number of tasks with levels at least h executing in the time interval $[t_{j1} - 1, t_{j1}]$. So $1 \leq k_1 \leq m-1$. Let Γ_1 be all the tasks released prior to t_{j1}, but have not yet been executed. Let Θ_1 be all the tasks in Γ_1 that have levels at least h, and let $\Psi_1 \subseteq \Theta_1$ be all the tasks that are ready for execution at time t_{j1} (i.e., all of whose predecessors have been executed by time t_{j1}). Since in an intree any task has at most one successor, the number of tasks in

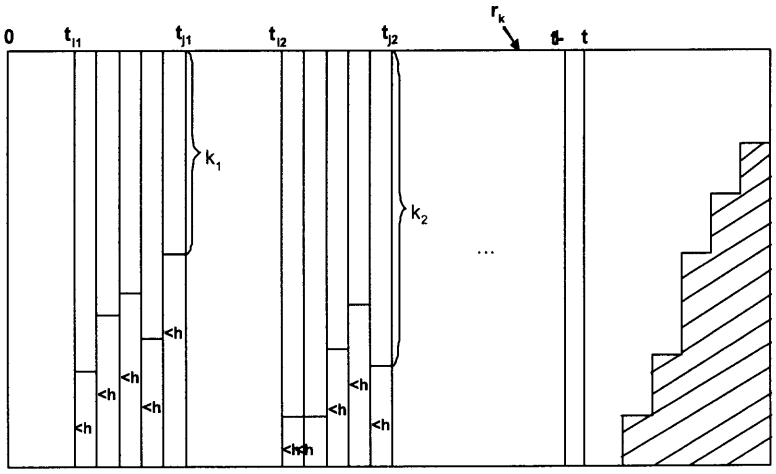

Figure 4.3 Some tasks scheduled in $[0, t]$ with levels $< h$.

Ψ_1 is at most k_1. Therefore, at most $k_1 * t_{i1}$ tasks in Θ_1 can be scheduled before t_{j1} and the $k_1 * t_{i1}$ tasks executed prior to t_{i1} can replace the tasks with level less than h executing in the time interval $[t_{i1}, t_{j1}]$; see Figure 1 for illustration. The tasks that are replaced will be scheduled at a later time.

From time t_{j1} on, let t_{i2} be the first time instant such that there is at least one task with level less than h executing in the time interval $[t_{i2}, t_{i2} + 1]$.

From time t_{i2} on, let t_{j2} be the first time instant such that all the tasks executing in the time interval $[t_{j2}, t_{j2} + 1]$ have levels at least h. Let k_2 be the number of tasks with levels at least h executing in the time interval $[t_{j2} - 1, t_{j2}]$. So $1 \leq k_2 \leq m - 1$. Let Γ_2 be all the tasks released prior to t_{j2}, but have not yet been executed. In Γ_2, if there are tasks that belong to Γ_1 and that can be scheduled prior to t_{j1}, remove these tasks from Γ_2. Let Θ_2 be all the tasks in Γ_2 that have levels at least h, and let $\Psi_2 \subseteq \Theta_2$ be all the tasks that are ready for execution at time t_{j2}. Since in an intree any task has at most one successor, the number of tasks in Ψ_2 is at most k_2. Therefore, at most $k_2 * (t_{i2} - t_{j1})$ tasks in Θ_2 can be scheduled before t_{j2} and the $k_2 * (t_{i2} - t_{j1})$ tasks executed prior to t_{i2} can replace the tasks with level less than h executing in the time interval $[t_{i2}, t_{j2}]$. The tasks that are replaced will be scheduled at a later time.

Continue to find $[t_{i3}, t_{j3}]$, $\cdots$, $[t_{il}, t_{jl}]$ until time t. Let k_3, $\cdots$, k_l be defined as before. Assume before t, there are a total of a columns that contain only tasks with level at least h and there are a total of b columns that contain tasks with level less than h. The number of tasks with level less than h that can be replaced is at most

$$a * \max\{k_1, k_2, \cdots, k_l\} \leq a * (m-1) < a * m.$$

If R is the number of tasks with level less than h that can be replaced, then the chain C can be moved earlier by at most R/m time units, and the tasks that have been replaced can be moved to the idle machines in the time interval $[t - R/m, t - R/m - 1]$ and thereafter. Consequently, a better schedule than S with a reduced $C_{\max}$ can be obtained.

Now, $C_{\max}(S) = a + b + c$. If $a > b$, then there are at most

$$b * (m-1) < b * m$$

tasks with level less than h, and hence the chain CH can be moved earlier by at most b time units. Therefore,

$$C_{\max}(S^*) \geq a + \max\{b, c\}$$

and

$$C_{\max}(S) - C_{\max}(S^*) \leq a + b + c - (a + \max\{b, c\}) = b + c - \max\{b, c\} = \min\{b, c\}$$

$$\leq 0.5(a + \min\{b, c\}) \leq 0.5(a + \max\{b, c\}) \leq 0.5 * C_{\max}(S^*).$$

Hence,

$$C_{\max}(S) \leq (3/2) * C_{\max}(S^*).$$

On the other hand, if $a \leq b$, then at most $a * m$ tasks with level less than h can be replaced and hence the chain CH can be moved earlier by at most a time units. Therefore,

$$C_{\max}(S^*) \geq b + \max\{a, c\}$$

and

$$C_{\max}(S) - C_{\max}(S^*) \leq a + b + c - (b + \max\{a, c\}) = a + c - \max\{a, c\} = \min\{a, c\}$$

$$\leq 0.5(b + \min\{a, c\}) \leq 0.5(b + \max\{a, c\}) \leq 0.5 * C_{\max}(S^*).$$

Hence,

$$C_{\max}(S) \leq (3/2) * C_{\max}(S^*).$$

From the above discussions, the following lemma can be obtained.

Lemma 4.3 *If there is no idle machine in the time interval* $[0, r_k]$*, then* $C_{\max}(S) \leq (3/2) * C_{\max}(S^*)$.

4.1.2 Case 2

The basic idea in proving the worst-case bound is to convert the schedule S into a new schedule S', which has a smaller $C_{\max}$ than S^* but may violate some precedence constraints. If one can show that $C_{\max}(S)/C_{\max}(S') \leq 3/2$, then $C_{\max}(S)/C_{\max}(S^*) \leq 3/2$ can be immediately obtained. The conversion is done by a sequence of conversions, eventually arriving at the schedule S'.

Starting from time r_1, let r_{i_1} be the first release time such that there are some partial columns or there is a level 1 task scheduled before r_{i_1} in S. Let $\hat{S}_1$ be the schedule produced by Algorithm A for the intrees released from the first $i_1 - 1$ release times only; see Figure 4.4 for an illustration. The improvement that can be made to $\hat{S}_1$ in terms of the number of tasks that can finish by r_{i_1} can be bounded.

One can first examine the improvement that can be made to the $C_{\max}$ of $\hat{S}_1$. In $\hat{S}_1$, the last time instant t_1 such that there is a chain, say CH_1, of tasks executing in the time interval $[t_1, C_{\max}(\hat{S}_1)]$ is located. Let T_{t1} be the head of the chain executing in the time interval $[t_1, t_1 + 1]$ of which no predecessor is executing in the time interval $[t_1 - 1, t_1]$, and let the level of T_{t1} be h_1. Let a_{i_1} be the total number of columns before t_1 during which all the tasks executing have levels greater than or equal to h_1, b_{i_1} be the total number of

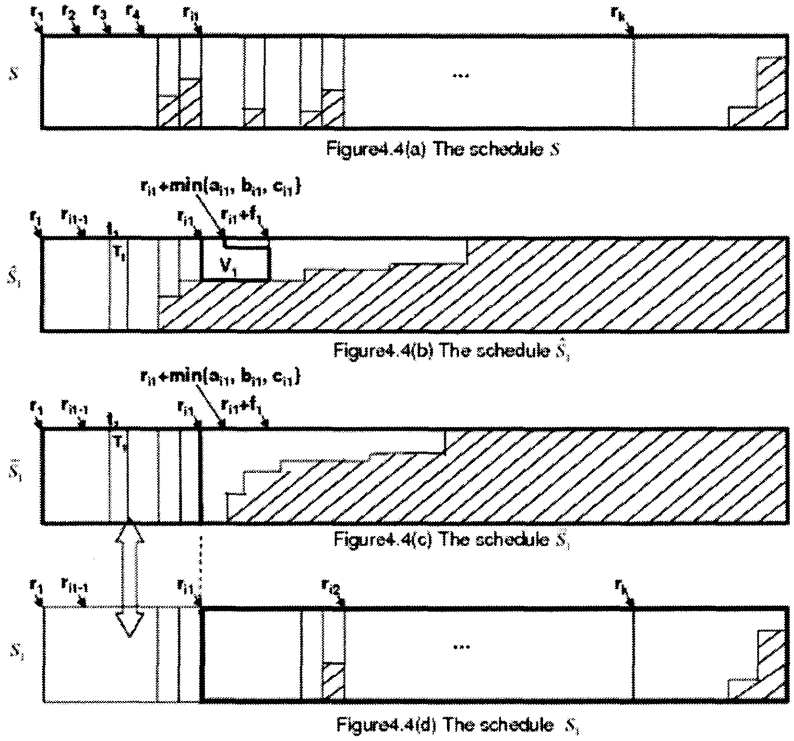

Figure 4.4 Illustrating the conversion process.

columns before t_1 during which some tasks executing have levels less than h_1, and c_{i_1} be the length of the chain CH_1 (i.e., $c_{i_1} = C_{\max}(\hat{S}_1) - t_1$). From Case 1 in Section 4.1.1, the $C_{\max}$ of $\hat{S}_1$ can be reduced by at most $\min\{a_{i_1}, b_{i_1}, c_{i_1}\}$.

Now check the improvement that can be made in terms of the number of tasks that can finish by r_{i_1}. In the time interval $[r_1, r_{i_1}]$, let F_1 be the set of full columns in which there is no level 1 task executing, and P_1 be the set of remaining columns. Let $f_1 = \mid F_1 \mid$. One can assert that $\min\{a_{i_1}, b_{i_1}, c_{i_1}\} \leq f_1/2$. For if $h_1 \leq 1$, then $b_{i_1} = 0$ and hence $\min\{a_{i_1}, b_{i_1}, c_{i_1}\} = 0 \leq f_1/2$. On the other hand, if $h_1 \geq 2$, then $a_{i_1} + b_{i_1} \leq f_1$ and hence $\min\{a_{i_1}, b_{i_1}, c_{i_1}\} \leq f_1/2$. For any task T executing in the time interval $[r_{i_1}, r_{i_1} + 1]$, some predecessor of T must be executing in some column of P_1. Since in an intree any task has at most one successor, the tasks executing in the time interval $[r_{i_1}, r_{i_1} + f_1]$ can possibly finish by time r_{i_1}; see Figure 1 for illustration. In $\hat{S}_1$, one may assume that the first machine contains the largest number of tasks, compared with any other machines. Since the $C_{\max}$ of $\hat{S}_1$ can be reduced by at most $\min\{a_{i_1}, b_{i_1}, c_{i_1}\}$, one may assume that the tasks executing

in the time interval $[r_{i_1}, r_{i_1} + \min\{a_{i_1}, b_{i_1}, c_{i_1}\}]$ on the first machine can finish by r_{i_1}. Let V_1 be the set of tasks executing in the time interval $[r_{i_1}, r_{i_1} + \min\{a_{i_1}, b_{i_1}, c_{i_1}\}]$ on the first machine plus all the tasks executing in the time interval $[r_{i_1}, r_{i_1} + f_1]$ on the other $m-1$ machines.

All the precedence constraints related to the tasks in V_1 are removed and the tasks are moved to the idle machines before r_{i_1}. The movement is stopped as soon as all the machines before r_{i_1} are full. Let V_1' be the remaining tasks in V_1. Now a new schedule $\bar{S}_1$ from $\hat{S}_1$ is obtained. By the discussions before, $\bar{S}_1$ has the largest number of tasks finish by r_{i_1}, compared with any other feasible schedule for the intrees released from the first $i_1 - 1$ release times.

Now S can be converted into a new schedule S_1 as follows. In S_1, the schedule in the time interval $[r_1, r_{i_1}]$ is identical to that of the schedule $\bar{S}_1$. The remainder of the schedule is obtained as follows. Let TS_1 be the task system consisting of all the tasks in V_1', all the tasks that have not yet been scheduled by time r_{i_1} from the intrees released from the first $i_1 - 1$ release times, and all the intrees released at or after r_{i_1} up to and including r_k. Call Algorithm A to schedule the tasks in TS_1, starting at time r_{i_1}. The schedule obtained by Algorithm A will be the remainder of S_1.

S_1 can be converted into a new schedule S_2 by the same procedure as above. Starting from r_{i_1}, let r_{i_2} be the first release time such that there are some partial columns or there is some level 1 task scheduled before r_{i_2} in S_1. Let $\hat{S}_2$ be the schedule produced by Algorithm A for the tasks in V_1', all the tasks that have not been scheduled by r_{i_1} from the intrees released from the first $i_1 - 1$ release times, and all the intrees released at or after r_{i_1} but before r_{i_2}. The schedule $\hat{S}_2$ starts at time r_{i_1}. The improvement that can be made to $\hat{S}_2$ in terms of the number of tasks that can finish by r_{i_2} can be bounded.

In $\hat{S}_2$, let t_2 be the last time instant such that there is a chain, say CH_2, of tasks executing in the time interval $[t_2, C_{\max}(\hat{S}_2)]$. Let T_{t2} be the head of the chain executing in the time interval $[t_2, t_2 + 1]$ of which no predecessor is executing in the time interval $[t_2 - 1, t_2]$, and let the level of T_{t2} be h_2. Let a_{i_2} be the total number of columns before t_2 during which all the tasks executing have levels greater than or equal to h_2, b_{i_2} be the total

number of columns before t_2 during which some tasks executing have levels less than h_2, and c_{i_2} be the length of the chain CH_2 (i.e., $c_{i_2} = C_{\max}(\hat{S}_2) - t_2$). From Case 1 in Section 2.1, the $C_{\max}$ of $\hat{S}_2$ can be reduced by at most $\min\{a_{i_2}, b_{i_2}, c_{i_2}\}$.

Consider the improvement that can be made in terms of the number of tasks that can finish by r_{i_2}. In the time interval $[r_{i_1}, r_{i_2}]$, let F_2 be the set of full columns in which there is no level 1 task executing, and P_2 be the set of remaining columns. Let $f_2 =\mid F_2 \mid$. By the same reasoning as above, it can be shown that $\min\{a_{i_2}, b_{i_2}, c_{i_2}\} \leq f_2/2$. For any task T with level greater than 1 executing in the time interval $[r_{i_2}, r_{i_2} + 1]$, some predecessor of T must be executing in some column of P_2. Since in an intree any task has at most one successor, the tasks executing in the time interval $[r_{i_2}, r_{i_2} + f_2]$ can possibly finish by time r_{i_2}; see Figure 1 for illustration. In $\hat{S}_2$, one may assume that the first machine contains the largest number of tasks, compared with any other machines. Since the $C_{\max}$ of $\hat{S}_2$ can be reduced by at most $\min\{a_{i_2}, b_{i_2}, c_{i_2}\}$, one may assume that the tasks executing in the time interval $[r_{i_2}, r_{i_2} + \min\{a_{i_2}, b_{i_2}, c_{i_2}\}]$ on the first machine can finish by time r_{i_2}. Let V_2 be the set of tasks executing in the time interval $[r_{i_2}, r_{i_2} + \min\{a_{i_2}, b_{i_2}, c_{i_2}\}]$ on the first machine plus all the tasks executing in the time interval $[r_{i_2}, r_{i_2} + f_2]$ on the other $m - 1$ machines, but not in V_1.

All the precedence constraints related to the tasks in V_2 are removed and these tasks are moved to the idle machines before r_{i_2}. The movement is stopped as soon as all the machines before r_{i_2} are full. Let V_2' be the remaining tasks in V_2. Now a new schedule $\bar{S}_2$ from $\hat{S}_2$ is obtained. By the discussions before, $\bar{S}_2$ has the largest number of tasks finish by r_{i_2}, compared with any other feasible schedule for the intrees released between r_{i_1} and r_{i_2} plus the remaining tasks from the intrees released from the first $i_1 - 1$ release times.

Now S_1 can be converted into a new schedule S_2 as follows. In S_2, the schedule in the time interval $[r_1, r_{i_1}]$ is identical to that of the schedule S_1. The schedule in the time interval $[r_{i_1}, r_{i_2}]$ is identical to that of the schedule $\bar{S}_2$. The remainder of the schedule is obtained as follows. Let TS_2 be the task system consisting of all the tasks in V_2', all the tasks that have not yet been scheduled by time r_{i_2} from the intrees released prior to r_{i_2}, and all the intrees released at or after r_{i_2} up to and including r_k. Call Algorithm A to schedule

the tasks in TS_2, starting at time r_{i_2}. The schedule obtained by Algorithm A will be the remainder of S_2.

One can repeat the above process until the schedule S_y is obtained such that in S_y, there is no partial column and there is no level 1 task scheduled before r_k. Let $\hat{S}_{y+1}$ be the portion of the schedule S_y in the time interval $[r_{i_y}, C_{\max}(S_y)]$. Clearly, $\hat{S}_{y+1}$ belongs to Case 1 in Section 2.1. In $\hat{S}_{y+1}$, let t_{y+1} be the last time instant such that there is a chain, say CH_{y+1}, of tasks executing in the time interval $[t_{y+1}, C_{\max}(\hat{S}_{y+1})]$. Let $T_{t(y+1)}$ be the head of the chain executing in the time interval $[t_{y+1}, t_{y+1}+1]$ of which no predecessor is executing in the time interval $[t_{y+1}-1, t_{y+1}]$, and let the level of $T_{t(y+1)}$ be h_{y+1}. Let $a_{i_{y+1}}$ be the total number of columns before t_{y+1} during which all the tasks executing have levels greater than or equal to h_{y+1}, $b_{i_{y+1}}$ be the total number of columns before t_{y+1} during which some tasks executing have levels less than h_{y+1}, and $c_{i_{y+1}}$ be the length of the chain CH_{y+1} (i.e., $c_{i_{y+1}} = C_{\max}(\hat{S}_{y+1}) - t_{y+1}$). From Case 1 in Section 2.1, the $C_{\max}$ of $\hat{S}_{y+1}$ can be reduced by at most $\min\{a_{i_{y+1}}, b_{i_{y+1}}, c_{i_{y+1}}\}$.

From r_{i_y} on, let t' be the first time instant such that in the time interval $[t', t'+1]$ there is an idle machine or there is a level 1 task scheduled. Let $f_{y+1} = t' - r_{i_y}$. Using the same reasoning as above, it can be shown that $\min\{a_{i_{y+1}}, b_{i_{y+1}}, c_{i_{y+1}}\} \leq f_{y+1}/2$.

Let V_{y+1} be the set of tasks executing in the time interval $[t', t'+\min\{a_{i_{y+1}}, b_{i_{y+1}}, c_{i_{y+1}}\}]$ on any machine. Move the tasks in V_{y+1} to the idle machines in the time interval $[t', t'+1]$ and thereafter. Finally, move the tasks from time $t'+\min\{a_{i_{y+1}}, b_{i_{y+1}}, c_{i_{y+1}}\}$ until time $C_{\max}(\hat{S}_{y+1})$ backwards by $\min\{a_{i_{y+1}}, b_{i_{y+1}}, c_{i_{y+1}}\}$ time units. The new schedule will be called $\bar{S}_{y+1}$.

Now S_y can be converted into a new schedule S' as follows. In S', the schedule in the time interval $[r_1, r_{i_y}]$ is identical to that of S_y, while the schedule in the time interval $[r_{i_y}, C_{\max}(\bar{S}_{y+1})]$ is identical to that of $\bar{S}_{y+1}$. In the next lemma, it will be shown that S' has a smaller $C_{\max}$ than S^*.

Lemma 4.4 *S' has a smaller $C_{\max}$ than S^*.*

Proof: By converting the schedule S into the schedule S_1, a new task system TS_1 is obtained. TS_1 contains the tasks from the intrees released in the first $i_1 - 1$ release times that have not yet been executed by r_{i_1} in S_1, plus all the intrees released at or after r_{i_1}. Let OPT_1 be the optimal schedule for TS_1. The claim is that $r_{i_1} + C_{\max}(OPT_1) \leq C_{\max}(S^*)$.

From S^*, all the tasks from the intrees released in the first $i_1 - 1$ release times are deleted, and the portion of the schedule in the time interval $[r_1, r_{i_1}]$ is replaced by S_1. Let the resulting schedule be $\tilde{S}$. Now fill the idle machines after time r_{i_1} in $\tilde{S}$ by the tasks from the intrees released in the first $i_1 - 1$ release times (that have not yet been executed by the time r_{i_1} in S_1). Since S_1 has the largest number of tasks finished at time r_{i_1}, the total idle time in $\tilde{S}$ is larger than or equal to the number of tasks from the intrees released in the first $i_1 - 1$ release times. Since all other tasks with levels greater than 1 and their successors in TS_1, with the exception of the tasks in V_1', cannot finish before r_{i_1}, these tasks in TS_1 can be filled into the idle machines as they are scheduled in S^*, and the remaining tasks in the idle machines in S^* can be filled in an arbitrary manner. Thus, $r_{i_1} + C_{\max}(OPT_1) \leq C_{\max}(S^*)$.

Similarly, after converting the schedule S_1 into the schedule S_2, a new task system TS_2 is obtained. Let OPT_2 be the optimal schedule for TS_2. Then $r_{i_2} + C_{\max}(OPT_2) \leq r_{i_1} + C_{\max}(OPT_1) \leq C_{\max}(S^*)$. Finally, $C_{\max}(S') \leq r_{i_y} + C_{\max}(OPT_y) \leq C_{\max}(S^*)$. □

Now $C_{\max}$ of S' is shown to be at least twice the improvement made to it; i.e., $C_{\max}(S') \geq 2(C_{\max}(S) - C_{\max}(S'))$. First, one can prove a lemma which is instrumental to the proof of $C_{\max}(S') \geq 2(C_{\max}(S) - C_{\max}(S'))$.

Suppose there are two intrees, $intree_1$ and $intree_2$, both released at time 0, and $intree_3$, $intree_4$, $\cdots$, $intree_x$ are released at times r_3, r_4, $\cdots$, r_x, respectively. Let $\dot{S}$ be the schedule produced by Hu's algorithm for $intree_1$, and assume that at least one machine is idle in the first time interval $[0, 1]$ in $\dot{S}$. Let $\tilde{S}$ be the schedule produced by Algorithm A for all x intrees.

Lemma 4.5 *If y tasks out of $intree_1$ are removed with the following constraints: (1) if a task is removed, then all its predecessors are also removed, (2) for any longest path in $intree_1$, at most c tasks on the longest path are among the y tasks removed (because of (1),*

these c tasks must be at the highest c levels), then the $C_{\max}$ of $\tilde{S}$ can be reduced by at most $\max\{y/m, c\}$.

Proof: Let t be the last time instant such that there is a chain executing in the time interval $[t, C_{\max}(\tilde{S})]$. Let T_t be the head of the chain executing in the time interval $[t, t+1]$ of which no predecessor is executing in the time interval $[t-1, t]$. Assume task T_t belongs to the intree that is released at time r_i. There are two cases to consider.

Case I: $r_i = t$.

There are two subcases to consider.

Subcase (i): $r_i > 0$.

Since $r_i > 0$, T_t does not belong to $intree_1$, and since $r_i = t$, T_t is on the longest path of the intree released at time r_i. Clearly, the $C_{\max}$ of $\tilde{S}$ cannot be reduced.

Subcase (ii): $r_i = 0$.

If T_t is from $intree_2$, then the $C_{\max}$ of $\tilde{S}$ cannot be reduced. If T_t is from $intree_1$, then it is easy to see that the chain executing in the time interval $[t, C_{\max}(\tilde{S})]$ is one of the longest paths in $intree_1$. Since at most c tasks are removed from this path, the $C_{\max}$ of $\tilde{S}$ can be reduced by at most $c \leq \max\{y/m, c\}$.

Case II: $r_i < t$.

It is easy to see that all the tasks executing in the time interval $[r_i, t]$ have levels higher than or equal to that of task T_t. Furthermore, there must be some tasks not from $intree_1$ executing in the time interval $[r_i, t]$. Again, there are two subcases to consider.

Subcase (i): $r_i > 0$.

Let U_t be the set of tasks that are released before r_i, but not finished by r_i. One has y tasks removed from $intree_1$. Since in an intree every task has at most one successor, at most y tasks from U_t can finish by r_i. So, at most y tasks from the time interval $[r_i, t]$ of $\tilde{S}$ can be moved to an earlier time than r_i, which implies that the $C_{\max}$ of $\tilde{S}$ can be reduced by at most $y/m \leq \max\{y/m, c\}$.

Subcase (ii): $r_i = 0$.

If task T_t is from $intree_2$, then after y tasks are removed from $intree_1$, at most y tasks will be removed in the time interval $[0, t]$. Therefore, the $C_{\max}$ of $\tilde{S}$ can be reduced by at most $y/m \leq \max\{y/m, c\}$.

If task T_t is from $intree_1$, then since there is at least one task not belonging to $intree_1$ in the time interval $[0, t]$, after y tasks of $intree_1$ are removed, the $C_{\max}$ of $\tilde{S}$ can be reduced by at most $y/m \leq \max\{y/m, c\}$. □

The above lemma will be used in the following way. In converting the schedule from S into S_1, let the unfinished tasks at r_{i_1} from the intrees released from the first $i_1 - 1$ release times be $intree_1$, let the intree released at r_{i_1} be $intree_2$, and let the intrees released at subsequent release times be $intree_3$, $intree_4$, $\cdots$, $intree_x$. V_1 has been defined to be the set of tasks belonging to $intree_1$ that can be moved to earlier times to fill up the idle machines. With such a move, how much can one reduce the $C_{\max}$ of the schedule of $intree_1$, $intree_2$, $\cdots$, $intree_x$? Lemma 4.5 provides an answer to this question.

Lemma 4.6 $C_{\max}(S) - C_{\max}(S') \leq C_{\max}(S')/2$.

Proof: In the process of converting S into S', S_1, S_2, $\cdots$, S_y are obtained. Also, the sets V_1, V_2, $\cdots$, V_y and V_{y+1} are obtained, and all the precedence constraints related to the tasks in these sets have been removed. $\mid V_i \mid$ is used to denote the number of tasks in V_i, $1 \leq i \leq y + 1$.

Let $\tilde{TS}_1$ be the task system consisting of all the unfinished tasks at r_{i_1} from the intrees released from the first $i_1 - 1$ release times in S, plus the intrees released at or after r_{i_1} but before r_{i_2}. Let $\tilde{TS}_2$ be the task system consisting of all the unfinished tasks at r_{i_2} from the intrees released from the first $i_2 - 1$ release times in S_1, plus the intrees released at or after r_{i_2} but before r_{i_3}. Similarly, let $\tilde{TS}_y$ be the task system consisting of all the unfinished tasks at r_{i_y} from the intrees released from the first $i_y - 1$ release times in S_{y-1}, plus the intrees released at or after r_{i_y}. Let $\tilde{S}_1$ be the schedule produced by Algorithm A for the task system $\tilde{TS}_1$, $\tilde{S}_2$ be the schedule produced by Algorithm A for the task system $\tilde{TS}_2$, $\cdots$, $\tilde{S}_y$ be the schedule produced by Algorithm A for the task system $\tilde{TS}_y$.

Assuming all the tasks out of V_1 are taken from $\tilde{S}_1$, from Lemma 4.5 it is clear that the makespan of $\tilde{S}_1$ can be reduced by at most

$$\max\{\min\{a_{i_1}, b_{i_1}, c_{i_1}\}, \frac{\mid V_1 \mid}{m}\}.$$

Now, in $\tilde{TS}_2$, at most $\max\{\min\{a_{i_1}, b_{i_1}, c_{i_1}\}, \frac{|V_1|}{m}\}$ tasks from any longest path in the unfinished intrees released before r_{i_2} can be completed before r_{i_2}, and hence can be taken out of $\tilde{S}_2$. If there are tasks of V_1 in $\tilde{S}_2$, these tasks will be taken out. Since in an intree, any task has at most one successor, when tasks of V_1 is taken out of $\tilde{S}_1$, their successor tasks may finish before r_{i_2}. Then all the tasks of V_2 are taken out of $\tilde{S}_2$. Totally $\mid V_1 \mid + \mid V_2 \mid$ tasks are taken out of $\tilde{S}_2$. By Lemma 4.5, the makespan of $\tilde{S}_2$ can be reduced by at most

$$\max\{\min\{a_{i_2}, b_{i_2}, c_{i_2}\} + \max\{\min\{a_{i_1}, b_{i_1}, c_{i_1}\}, \frac{\mid V_1 \mid}{m}\}, \frac{\mid V_1 \mid + \mid V_2 \mid}{m}\}$$

$$= \max\{\min\{a_{i_2}, b_{i_2}, c_{i_2}\} + \min\{a_{i_1}, b_{i_1}, c_{i_1}\}, \min\{a_{i_2}, b_{i_2}, c_{i_2}\} + \frac{\mid V_1 \mid}{m}, \frac{\mid V_1 \mid + \mid V_2 \mid}{m}\}.$$

Similarly, the makespan of $\tilde{S}_y$ can be reduced by at most

$$\max_{x=0}^{y}(\sum_{k=1}^{x} \frac{\mid V_k \mid}{m} + \sum_{k=x+1}^{y} \min\{a_{i_k}, b_{i_k}, c_{i_k}\}).$$

From S_y to S', the makespan is reduced by at most

$$\min\{a_{i_{y+1}}, b_{i_{y+1}}, c_{i_{y+1}}\}.$$

So, from S to S', the makespan is reduced by at most

$$\max_{x=0}^{y}(\sum_{k=1}^{x} \frac{\mid V_k \mid}{m} + \sum_{k=x+1}^{y} \min\{a_{i_k}, b_{i_k}, c_{i_k}\}) + \min\{a_{i_{y+1}}, b_{i_{y+1}}, c_{i_{y+1}}\}. \tag{4.1}$$

It is easy to see that $\mid V_i \mid \leq f_i \cdot m$ for each $1 \leq i \leq y$. Furthermore, $\min\{a_{i_1}, b_{i_1}, c_{i_1}\} \leq f_1/2$, $\min\{a_{i_2}, b_{i_2}, c_{i_2}\} \leq f_2/2$, $\cdots$, and $\min\{a_{i_y}, b_{i_y}, c_{i_y}\} \leq f_y/2$. Finally, $\min\{a_{i_{y+1}}, b_{i_{y+1}}, c_{i_{y+1}}\} \leq f_{y+1}/2$.

From Equation 4.1, let the maximum occurs at j. Then Equation 4.1 can be written as

$$\sum_{k=1}^{j} \frac{\mid V_k \mid}{m} + \sum_{k=j+1}^{y} \min\{a_{i_k}, b_{i_k}, c_{i_k}\} + \min\{a_{i_{y+1}}, b_{i_{y+1}}, c_{i_{y+1}}\} \leq \sum_{k=1}^{j} \frac{\mid V_k \mid}{m} + \sum_{k=j+1}^{y+1} f_k/2. \tag{4.2}$$

The tasks in V_i are scheduled in S', but not in the time intervals of F_i. So

$$\sum_{k=1}^{j} \frac{\mid V_k \mid}{m} \leq C_{\max}(S') - \sum_{k=1}^{y+1} f_k. \tag{4.3}$$

Furthermore,

$$\sum_{k=1}^{j} \frac{\mid V_k \mid}{m} \leq \sum_{k=1}^{j} f_k. \tag{4.4}$$

Adding Equation 4.3 and Equation 4.4 together,

$$2(\sum_{k=1}^{j} \frac{\mid V_k \mid}{m}) \leq C_{\max}(S') - \sum_{k=j+1}^{y+1} f_k. \tag{4.5}$$

Substituting Equation 4.5 into Equation 4.2,

$$\sum_{k=1}^{j} \frac{\mid V_k \mid}{m} + \sum_{k=j+1}^{y} \min\{a_{i_k}, b_{i_k}, c_{i_k}\}) + \min\{a_{i_{y+1}}, b_{i_{y+1}}, c_{i_{y+1}}\} \leq C_{\max}(S')/2, \tag{4.6}$$

which means that

$$C_{\max}(S) - C_{\max}(S') \leq C_{\max}(S')/2.$$

□

The following theorem follows immediately from Lemmas 4.3 and 4.6,

Theorem 4.7 *For any instance of* $P \mid p_j = 1, intree_i$ *released at time* $r_i \mid C_{\max}$, *let* S *denote the schedule produced by Algorithm A and let* S^* *denote the optimal schedule. Then* $C_{\max}(S)/C_{max}(S^*) \leq 3/2$.

4.2 Equal-Processing-Time Tasks

In this section, the case where the processing time of each task is p units long is considered. First, the relationship between $\alpha \mid p_j = 1, online \mid C_{\max}$ and $\alpha \mid p_j = p, online \mid C_{\max}$ is studied. That is, both problems are online problems. In the first case the processing time of each task is one unit, while in the second case the processing time of each task is p units.

Suppose there is an online algorithm X that has a competitive ratio λ for the online problem $\alpha \mid p_j = 1, online \mid C_{\max}$. Consider a restricted version of $\alpha \mid p_j = p, online \mid C_{\max}$ in which tasks are only released at integral multiples of p units; i.e. at time kp for some integer k. Now, if algorithm X is directly applied to this restricted problem, it is easy to see that it will also have a competitive ratio λ.

Now, if jobs are released at any time units, the following algorithm, Algorithm Delay-X, will be used to schedule the tasks.

Algorithm Delay-X

1. Suppose r is the release time of a set of new jobs. If $kp < r < (k+1)p$ for some integer k, then delay the jobs from consideration until the time instant $(k+1)p$.

2. Suppose r is the release time of a set of new jobs. If $r = kp$ for some integer k, then use Algorithm X to schedule all the active tasks (i.e., tasks that have been released but have not yet finished).

The next theorem relates the competitive ratio of Algorithm Delay-X with the competitive ratio of Algorithm X.

Theorem 4.8 *Suppose Algorithm X is an online algorithm for the problem* $\alpha \mid p_j = 1, online \mid C_{\max}$, *with a competitive ratio of* λ. *For any instance of* $\alpha \mid p_j = p, online \mid$

$C_{\max}$, let S denote the schedule obtained by Algorithm Delay-X and S^* denote an optimal schedule. Then $C_{\max}(S) \leq \lambda(C_{\max}(S^*) + p)$.

Proof: Let TS be the original task system and TS' be the task system obtained from TS by setting the release time of any job between kp and $(k+1)p$ to be $(k+1)p$. Let S denote the schedule produced by Algorithm Delay-X for TS and let S^* denote an optimal schedule. By shifting right the schedule S^* by p time units, one can get a feasible schedule for TS'. So, $C_{\max}(S^*) + p \geq \frac{C_{\max}(S)}{\lambda}$, or equivalently, $C_{\max}(S) \leq \lambda(C_{\max}(S^*) + p)$. □

From Theorems 4.7 and 4.8, the following theorem is immediately obtained.

Theorem 4.9 , *For any instance of $P \mid p_j = p, intree_i$ released at $r_i \mid C_{\max}$, let S denote the schedule obtained by Algorithm Delay-A and let S^* denote an optimal schedule. Then $C_{\max}(S) \leq (3/2)(C_{\max}(S^*) + p)$.*

Huo and Leung [Huo and Leung 2005] have shown that Algorithm A is optimal for $P \mid p_j = 1$, *outtree$_i$ released at time $r_i \mid C_{\max}$*. Using this result and together with Theorem 2, the following result will be obtained.

Theorem 4.10 *For any instance of $P \mid p_j = p, outtree_i$ released at time $r_i \mid C_{\max}$, let S denote the schedule obtained by Algorithm Delay-A and let S^* denote an optimal schedule. Then $C_{\max}(S) \leq C_{\max}(S^*) + p$.*

Huo and Leung [Huo and Leung 2005] have also shown that an online version of Coffman-Graham algorithm is optimal for $P2 \mid p_j = 1, prec_i$ *released at time* $r_i \mid C_{\max}$. The Coffman-Graham algorithm is optimal for $P2 \mid p_j = 1, prec \mid C_{\max}$. It works by first assigning a label to each task which corresponds to the priority of the task; tasks with higher labels have higher priority. Once the labels are assigned, tasks are scheduled as follows: whenever a machine becomes free for assignment, assign that task all of whose predecessors have already been executed and which has the largest label among those tasks not yet assigned.

To describe the labeling algorithm, a linear order on decreasing sequences of positive integers is defined as follows.

Definition 4.11 *Let* $N = (n_1, n_2, \ldots, n_t)$ *and* $N' = (n'_1, n'_2, \ldots, n'_{t'})$ *be two decreasing sequences of positive integers.* $N < N'$ *if either*

1. *For some* i, $1 \leq i \leq t$, *either* $n_j = n'_j$ *for all* j *satisfying* $1 \leq j \leq i-1$ *and* $n_i < n'_i$, *or*

2. $t < t'$ *and* $n_j = n'_j$ *for all* j *satisfying* $1 \leq j \leq t$.

Example: $(7, 5, 2, 1) < (7, 5, 4)$ and $(10, 7, 6) < (10, 7, 6, 3, 1)$.

Let n denote the number of tasks in *prec*. The labeling algorithm assigns to each task i an integer label $\alpha(i) \in \{1, 2, \ldots, n\}$. The mapping α is defined as follows. Let $IS(i)$ denote the set of immediate successors of task i and let $N(i)$ denote the decreasing sequence of integers formed by ordering the set $\{\alpha(j) \mid j \in IS(i)\}$.

1. An arbitrary task i with $IS(i) = \emptyset$ is chosen and $\alpha(i)$ is defined to be 1.

2. Suppose for some $k \leq n$ that the integers $1, 2, \ldots, k-1$ have been assigned. From the set of tasks for which α has been defined on all elements of their immediate successors, choose the task j such that $N(j) \leq N(i)$ for all such tasks i. Define $\alpha(j)$ to be k.

3. Repeat the assignment in 2 until all tasks of *prec* have been assigned some integer.

The online algorithm utilizes the Coffman-Graham algorithm to schedule tasks. Whenever new tasks arrive, the new tasks along with the unexecuted portion of the unfinished tasks will be rescheduled by the Coffman-Graham algorithm again.

Algorithm B

Whenever new tasks arrive, do

$t \leftarrow$ the current time

$U \leftarrow$ the set of tasks active (i.e., not finished) at time t

Call the Coffman-Graham algorithm to reschedule the tasks in U

By the result of Huo and Leung [Huo and Leung 2005] and Thereom 2, the following theorem can be obtained immediately .

Theorem 4.12 *For any instance of* $P2 \mid p_j = p, prec_i$ *released at time* $r_i \mid C_{\max}$, *let* S *denote the schedule obtained by Algorithm Delay-B and let* S^* *denote an optimal schedule. Then* $C_{\max}(S) \leq C_{\max}(S^*) + p$.

4.3 Conclusions

In this chapter, it has been shown that Algorithm Delay-A has an asymptotic competitive ratio of 3/2 for $P \mid p_j = p, intree_i$ *released at time* $r_i \mid C_{\max}$, and an asymptotic competitive ratio of 1 for $P \mid p_j = p, outtree_i$ *released at time* $r_i \mid C_{\max}$. Furthermore, it is shown that Algorithm Delay-B has an asymptotic competitive ratio of 1 for $P2 \mid p_j = p, prec_i$ *released at time* $r_i \mid C_{\max}$.

For the problem $P \mid p_j = p, prec_i$ *released at time* $r_i \mid C_{\max}$, the asymptotic competitive ratio of Algorithm Delay-B has not been determined. On the other hand, it is known that the ratio is at least $2 - \frac{2}{m}$, since Lam and Sethi [Lam and Sethi 1977] have shown that Coffman-Graham algorithm is a $(2 - \frac{2}{m})$-approximation algorithm for $P \mid p_j = 1, prec \mid C_{\max}$. For future research, it will be interesting to determine this value. Furthermore, it will be interesting to see if there are other classes of precedence constraints and processing times that yield an asymptotic competitive ratio less than 2.

In this chapter, the approximation algorithms for preemptive scheduling have not been studied. Since for $P \mid pmtn, intree \mid C_{\max}$, Muntz-Coffman algorithm gives an optimal solution, one can consider an online version of Muntz-Coffman algorithm for $P \mid pmtn, intree_i$ *released at time* $r_i \mid C_{\max}$. What is the competitive ratio of this algorithm?

CHAPTER 5

INTEGRATED PRODUCTION AND DELIVERY SCHEDULING WITH DISJOINT WINDOWS

5.1 Arbitrary Profit

In this section the arbitrary profit case is considered. In Section 5.1.1, a pseudo-polynomial time algorithm is given for a single machine; the algorithm has a running time $O(nV)$, where $V = \sum_{i=1}^{n} pf_i$. In Section 5.1.2, a FPTAS for a single machine is given; the FPTAS has running time $O(\frac{n^3}{\epsilon})$. Finally, in Section 5.1.3, the algorithm is extended to parallel and identical machines.

5.1.1 Pseudo-polynomial Time Algorithm

In this section a dynamic programming algorithm is presented for a single machine. The running time of the algorithm is $O(nV)$, where $V = \sum_{i=1}^{n} pf_i$ is the total profit of all the jobs.

For each $1 \leq t \leq z$, let J^t denote the set of jobs whose delivery time is d_t. Relabel all the jobs in J such that jobs in earlier time frame get smaller labels than jobs in later time frame, and for the jobs in the same time frame, jobs with longer processing time get smaller labels than jobs with shorter processing time. That is, for any pair of jobs $J_i \in J^t$ and $J_{i+1} \in J^{t'}$, either $t < t'$, or $t = t'$ and $p_i \geq p_{i+1}$.

Lemma 5.1 *For any feasible schedule with total profit P and finishing time t, there exists a feasible schedule $S = \{J_{i_1}, J_{i_2}, \cdots, J_{i_k}\}$ with the same profit P and the same finishing time t and $i_1 < i_2 < \cdots < i_k$.*

Proof: First, for any pair of jobs from any feasible schedule, if they are finished in different time windows, then the job that finished in the earlier time window must have smaller label than the job that finished in the later time window. Therefore, only the order of jobs from the same time window needs to be considered. Suppose in the feasible schedule,

$J_{i_1}, J_{i_2}, \cdots, J_{i_x}$ is the set of jobs that finished in the same time window, say $[w_t, d_t]$, and J_{i_1} is the first job that starts executing at time s. Rearrange these jobs after the time instant s in descending order of their processing times. It is easy to see that after rearranging: (1) all jobs are finished at or before d_t, and (2) all jobs are finished at or after w_t. The reason is that the processing time of the first job is greater than or equal to the processing time of J_{i_1}, and $s + p_{i_1} \geq w_t$. After rearranging, the finishing time of the first job is greater than or equal to w_t. Therefore, the finishing time of all other jobs after rearranging is also greater than or equal to w_t. □

Define a table $T(i, j)$, where $0 \leq i \leq n$ and $0 \leq j \leq V$. $T(i, j)$ contains the minimum finishing time for scheduling the first i jobs such that a total profit of exactly j can be obtained. If there is no feasible schedule, let $T(i, j)$ contain ∞. Here is the rule to compute $T(i, j)$.

1. For $i = 0$, $T(0, 0) = 0$ and $T(0, j) = \infty$ for $j > 0$.

2. For $i \geq 1$, let $J_i \in J^t$. Then,

$$T(i,j) = \min \begin{cases} T(i-1,j) & \\ T(i-1,j-pf_i)+p_i & \text{if } j \geq pf_i \text{ and} \\ & w_t \leq T(i-1,j-pf_i)+p_i \leq d_t \\ w_t & \text{if } j \geq pf_i \text{ and } T(i-1,j-pf_i)+p_i < w_t \\ \infty & \text{if } j \geq pf_i \text{ and } T(i-1,j-pf_i)+p_i > d_t \end{cases}$$

After filling in the whole table, one can check the last row (row n) from right to left until the first entry $T(n, j)$ such that $T(n, j) < \infty$ is found; j is the total profit of the optimal schedule. The running time of the algorithm is simply the size of the table which is $O(nV)$, where $V = \sum_{i=1}^{n} pf_i$. It can be shown that the table is computed correctly.

Theorem 5.2 *The above algorithm correctly computes the optimal schedule.*

Proof: The theorem can be proven by induction on the number of rows in the table. The basis case, row 0, is filled correctly because one can only get zero profit from an empty job set.

Assume that rows $0, 1, \ldots, i-1$ are computed correctly, it will be shown that row i is also computed correctly. For row i, there are two cases to consider: (1) job J_i is not scheduled, and (2) job J_i is scheduled. In the former case, the minimum finishing time to obtain a profit exactly j is $T(i-1, j)$, by the induction hypothesis. In the latter case, it can always be assumed that J_i is the last job to be scheduled, by Lemma 5.1. In this case, one wants to find a schedule with total profit exactly j and job J_i will be finished as early as possible in this schedule. There are several sub-cases to consider: (1) If $j \geq pf_i$ and $w_t \leq T(i-1, j-pf_i) + p_i \leq d_t$, then the minimum finishing time is $T(i-1, j-pf_i) + p_i$. This is because, by the induction hypothesis, $T(i-1, j-pf_i)$ is the minimum finishing time to obtain a profit exactly $j - pf_i$ from the first $i-1$ jobs, so by scheduling job J_i immediately after, job J_i will finish at time $T(i-1, j-pf_i) + p_i$. (2) If $j \geq pf_i$ and $T(i-1, j-pf_i) + p_i < w_t$, then the minimum finishing time is w_t. This is because job J_i does not finish at or beyond w_t. Therefore, job J_i have to be shifted right so that it finishes at exactly w_t. (3) If $j \geq pf_i$ and $T(i-1, j-pf_i) + p_i > d_t$, then job J_i finishes beyond its delivery time. Hence, $T(i, j) = \infty$. □

5.1.2 Fully Polynomial Time Approximation Scheme

The above dynamic programming algorithm is a pseudo-polynomial time algorithm. It is efficient only when the total profit is not too large. Using the method from [11], a FPTAS for one machine can be obtained.

The algorithm works as follows. Let K be a parameter to be determined later.

- Create a new set of job instance by replacing each job J_i with a job J_i' such that $pf_i' = \lfloor \frac{pf_i}{K} \rfloor$, and keep all other parameters unchanged.

- Run the dynamic programming algorithm to obtain an optimal solution for the new job instance.

- Translate the solution for the new job instance back to the solution for the original job instance.

It is clear that the running time of this algorithm is $O(\frac{nV}{K})$. Let PF_{opt} be the total profit of the optimal schedule of the original job instance and $PF_{opt'}$ be the total profit of the optimal schedule of the new job instance. Clearly, $PF_{opt'}$ can be obtained by the dynamic programming algorithm for the new job instance. For each job J_i, since $pf_i' = \lfloor \frac{pf_i}{K} \rfloor$, $pf_i - K \cdot pf_i' \leq K$. It follows that $PF_{opt} - K \cdot PF_{opt'} \leq Kn$. Let PF_{alg} be the total profit of the algorithm. It is clear that $PF_{alg} \geq K \cdot PF_{opt'}$. By setting $Kn = \frac{\epsilon \cdot v_{\max}}{(1+\epsilon)}$, where $v_{\max} = \max_{i=1}^{n}\{PF_i\}$, the following equation can be obtained:

$$\begin{aligned} PF_{opt} - PF_{alg} &\leq PF_{opt} - K \cdot PF_{opt'} \\ &\leq Kn \\ &\leq \frac{\epsilon \cdot v_{\max}}{(1+\epsilon)} \\ &\leq \frac{\epsilon \cdot PF_{opt}}{(1+\epsilon)}. \end{aligned}$$

It follows that $\frac{PF_{opt}}{PF_{alg}} \leq (1+\epsilon)$. The running time is

$$O(\frac{nV}{K}) = O(\frac{n^2 \cdot v_{\max}}{K}) = O(\frac{n^2 \cdot v_{\max}}{\epsilon \cdot v_{\max}/n(1+\epsilon)}) = O(n^3(1+\frac{1}{\epsilon})) = O(\frac{n^3}{\epsilon}).$$

Theorem 5.3 *There is a FPTAS for arbitrary profits on a single machine with running time* $O(\frac{n^3}{\epsilon})$.

5.1.3 Arbitrary Number of Machines

The same technique as in [Bar-Noy et al. 2001] can be used to extend the algorithm for a single machine to arbitrary number of machines. Suppose there is an *Algorithm A* with approximation ratio β. *Algorithm A* can be used repeatedly to schedule jobs, one machine after another, until all m machines are scheduled. The following lemma can be proved.

Lemma 5.4 *For any β-approximation algorithm on one machine, it can be extended to m machines with approximation ratio at most* $\frac{1}{1-e^{-1/\beta}}$.

Proof: Let J be the entire set of jobs, S^* be the optimal schedule for J on m machines and PF_{opt} be its total profit. *Algorithm A* is repeatedly applied on the m machines, one machine after another, until all m machines are scheduled. Let PF_A be the total profit obtained by this algorithm. Let S_i be the set of jobs scheduled on machine i and $A_i = \sum_{J_k \in S_i} pf_k$. Let U be the set of jobs scheduled on the first $k-1$ machines, i.e., $U = \cup_{i=1}^{k-1} S_i$. Consider the scheduling of the set of jobs $J \setminus U$ on m totally unoccupied machines. Let S_1^* be the optimal schedule on one machine for the set of jobs $J \setminus U$, and let $PF_{opt'}$ be the total profit of S_1^*. The following claim can be proven.

Claim 5.5 $PF_{opt'} \geq \frac{PF_{opt} - \sum_{i=1}^{k-1} A_i}{m}$.

Proof: For any job $J_i \in U$, if J_i is scheduled in S^*, by deleting it from S^*. A new schedule $\hat{S}$ is obtained. Suppose that jobs $J_{i_1}, J_{i_2}, \cdots, J_{i_r}$ are deleted from S^*. Then the total profit of $\hat{S}$ will be $PF_{opt} - \sum_{j=1}^{r} pf_{i_j} \geq PF_{opt} - \sum_{i=1}^{k-1} A_i$. Clearly, $\hat{S}$ is a feasible schedule for the set of jobs $J \setminus U$. And in this feasible schedule, there must be one machine containing jobs whose total profit is at least $\frac{PF_{opt} - \sum_{i=1}^{k-1} A_i}{m}$. Therefore, $PF_{opt'} \geq \frac{PF_{opt} - \sum_{i=1}^{k-1} A_i}{m}$. □

Now, if *Algorithm A* is used to schedule the set of jobs $J \setminus U$ on one machine, the same schedule as S_k will be gotten. Since *Algorithm A* is a β-approximation algorithm, $A_k \geq \frac{PF_{opt'}}{\beta}$, and hence $A_k \geq \frac{1}{\beta m}(PF_{opt} - \sum_{i=1}^{k-1} A_i)$.

Adding $\sum_{i=1}^{k-1} A_i$ to both sides, one gets $\sum_{i=1}^{k} A_i \geq \frac{1}{\beta m} PF_{opt} + (1 - \frac{1}{\beta m}) \sum_{i=1}^{k-1} A_i$. Letting $f(k) = \sum_{i=1}^{k} A_i$,

$$\begin{aligned}
f(k) &\geq \frac{1}{\beta m} PF_{opt} + (1 - \frac{1}{\beta m}) f(k-1), \text{ or} \\
f(k) - PF_{opt} &\geq (1 - \frac{1}{\beta m})(f(k-1) - PF_{opt}), \text{ or} \\
f(m) - PF_{opt} &\geq (1 - \frac{1}{\beta m})^m \cdot (f(0) - PF_{opt}), \text{ or} \\
f(m) &\geq PF_{opt} - (1 - \frac{1}{\beta m})^m \cdot PF_{opt} \geq (1 - e^{-1/\beta}) PF_{opt}.
\end{aligned}$$

Therefore, $\frac{PF_{opt}}{PF_A} = \frac{PF_{opt}}{f(m)} \leq \frac{1}{1 - e^{-1/\beta}}$. □

Theorem 5.6 *There is an $\frac{e}{e-1}$-approximation algorithm for m machines with running time $O(mnV)$. Moreover, for any $\epsilon > 0$, there is an approximation algorithm with approximation ratio at most $\frac{1}{1-e^{-1/\beta}} = \frac{1}{1-e^{-1/(1+\epsilon)}}$ and running time $O(\frac{mn^3}{\epsilon})$.*

Proof: The pseudo-polynomial time algorithm for a single machine is optimal; so, $\beta = 1$. By extending it to m machines, a $\frac{e}{e-1}$-approximation algorithm is obtained whose running time is $O(mnV)$.

By Lemma 5.4, the FPTAS can be extended to m machines to obtain an approximation algorithm with approximation ratio at most $\frac{1}{1-e^{-1/\beta}} = \frac{1}{1-e^{-1/(1+\epsilon)}}$ and running time $O(\frac{mn^3}{\epsilon})$. □

5.2 Equal Profit

In this section, the model where the profit is constant for all jobs is considered. Maximizing the total profit is equivalent to maximizing the total number of jobs scheduled. For a single machine, the dynamic programming algorithm of Section 2.1 can be used to obtain an optimal schedule; the running time of the algorithm is $O(n^2)$. However, there is a more efficient $O(n \log n)$-time algorithm. In Section 5.2.1, this algorithm will be presented. The algorithm can be extended to m machines with an approximation ratio at most $\frac{e}{e-1}$ and running time $O(mn \log n)$. This result is presented in Section 5.2.2. Finally, the special case where there is a single time frame and $m \geq 2$ parallel and identical machines is considered. This problem is similar to the bin packing problem studied by Coffman et al. [Coffman et al. 1978], except that a piece can be packed beyond the threshold of a bin. For the bin packing problem, Coffman et al. [Coffman et al. 1978] proposed the *First-Fit-Increasing (FFI)* algorithm and showed that it obeys a bound of $\frac{4}{3}$. The FFI algorithm can be adapted in a natural way and shown to obey a bound of $\frac{7}{5}$. This result is presented in Section 3.3.

5.2.1 Single Machine

In this section, the $O(n \log n)$-time algorithm is presented to obtain an optimal schedule. The basic idea of the algorithm is scheduling the jobs window by window, starting with

the first window W_1. For each window W_i, the jobs are scheduled backwards from time d_i, using the Shortest-Processing-Time (SPT) rule, until no jobs can be scheduled. The jobs scheduled in this window is then left-shifted as much as possible, with the constraint that the completion time of all jobs must be within this window and no two jobs can be scheduled at the same time. The algorithm is described as follows.

Algorithm A

$k = 1$

while $k \leq z$ **do**

  $s = d_k$, $G = J^k$

  repeat

    Take the current shortest job of G, say J_i

    If the machine is idle in the time interval $[s - p_i, s]$,

    then schedule J_i in the time interval $[s - p_i, s]$.

    $s = s - p_i$.

    $G = G \setminus \{J_i\}$

  until $s < w_k$ or $G = \emptyset$

  Left-shift the jobs scheduled in the window W_k as much as possible.

  $k = k + 1$

endwhile

First sort the jobs with the same delivery time in ascending order of their processing times. The running time for sorting is $O(n \log n)$. After sorting, Algorithm A can be done in linear time. Altogether, the running time is $O(n \log n)$. Algorithm A can be shown to be optimal for equal profit case. The following lemma is given to prove the the algorithm is optimal.

Lemma 5.7 *Algorithm A has the following properties: (1) For any $1 \leq k \leq z$, the total number of jobs finished at or before d_t is maximized. (2) Compared with any other feasible schedule with the same number of jobs completed at or before d_t, the schedule produced by Algorithm A has the longest idle time left for the next time frame F_{t+1}.*

Proof: This lemma can be proven by induction on the number of time frames. For the basis case $k = 1$, there is only one time frame to consider. Let $J^1 = \{J_1, J_2, \cdots, J_{n_1}\}$ be the set of jobs with delivery time d_1, and jobs are labeled in decreasing order of their processing times, i.e., $p_1 \geq p_2 \geq \cdots \geq p_{n_1}$. Suppose Algorithm A has scheduled the jobs $J_{x_1}, \cdots, J_{n_1}$ into the time window W_1. Let S^* be any feasible schedule with the maximum number of jobs completed at or before d_1 and with the longest idle time left for the next time frame. By Lemma 5.1, one may assume that in S^* jobs are scheduled in decreasing order of their processing times. Let $J_{i_1}, J_{i_2}, \cdots, J_{i_y}$ be the jobs scheduled in S^* such that $p_{i_1} \geq p_{i_2} \geq \cdots \geq p_{i_y}$. Now compare the processing time of p_{i_y} with p_{n_1}. If $p_{i_y} > p_{n_1}$, the job J_{i_y} is replaced by the job J_{n_1} in the schedule S^*. The resulting schedule is also feasible. Similarly, compare the processing time of $p_{i_{y-1}}$ with p_{n_1-1}. If $p_{i_{y-1}} > p_{n_1-1}$, the job $J_{i_{y-1}}$ is replaced by the job J_{n_1-1} in S^*. This process is repeated until the job J_{i_1}. Since the jobs $J_{x_1}, \cdots, J_{n_1}$ are the smallest jobs in J^1, the resulting schedule must also be feasible. But the new schedule is exactly the same schedule produced by Algorithm A.

Assume that the lemma holds for $k = j$. Let S^* be the feasible schedule with the maximum number of jobs completed at or before d_{j+1} and with the longest time left for the next time frame F_{j+2}. Let S be the schedule produced by Algorithm A. Suppose that S^* have x jobs completed at or before d_j and S have y jobs completed at or before d_j. By the induction hypothesis, $x \leq y$. There are three cases.

Case 1: $x = y$. By the induction hypothesis, S has the maximum idle time left for the next time frame F_{j+1}. Using the same argument as in the basis case, one can prove that the lemma holds for $k = j + 1$.

Case 2: $x \leq y-2$. In this case, S^* has less jobs completed at or before d_j than S, so S^* may have more idle time left for F_{j+1} than S. But this idle time can be used to schedule at most one job and the job must be completed at or after w_{j+1}. So S^* can schedule at most one more job than S in the time frame F_{j+1}. That is, S has strictly more completed jobs than S^* at or before d_{j+1}. This is a contradiction.

Case 3: $x = y - 1$. In this case, S^* has less jobs completed at or before d_j than S, so S^* may have more idle time left for F_{j+1} than S. But this idle time can be used to

Table 5.1 Comparison of Performance Bounds

Number of machines	1	2	3	4	5	6	7	8	9
Algorithm in this section	1	1.33	1.42	1.46	1.49	1.50	1.51	1.52	1.53
[Bar-Noy et al. 2001]	2	1.80	1.73	1.70	1.67	1.66	1.65	1.64	1.63

schedule at most one job and this job must be completed at or after w_{j+1}. Let J_i be such a job in S^*. Then, $C_i \geq w_{j+1}$. After job J_i is scheduled in S^*, the remaining idle time in F_{j+1} is less than or equal to W. But in S, the remaining idle time in F_{j+1} is greater than or equal to W. Using the same argument as in the basis case, it can be proven that the lemma holds for $k = j + 1$.

□

From Lemma 5.7, the following result immediately follows.

Theorem 5.8 *Algorithm A is an optimal algorithm for the case of equal profit on a single machine.*

5.2.2 Arbitrary Number of Machines

For parallel and identical machines, one can use Algorithm A repeatedly to schedule jobs, one machine after another, until all m machines are scheduled. Since Algorithm A is optimal for a single machine, $\beta = 1$. By Lemma 5.4, the approximation ratio for m machines is at most $\frac{1}{1-e^{-1}} = \frac{e}{e-1}$. This algorithm yields better approximation ratios than the algorithm given in [Bar-Noy et al. 2001] for finite m. The difference is significant when m is small, as shown in Table 5.1.

Theorem 5.9 *There is an $\frac{e}{e-1}$-approximation algorithm on $m \geq 1$ parallel and identical machines for the equal profit case; the running time is $O(mn \log n)$.*

5.2.3 A Special Case of A Single Window

In this section the special case where there is a single time frame with m machines is considered. This problem is similar to the bin packing problem where the objective is to maximize the number of pieces packed [Coffman et al. 1978], except that in current problem one piece is allowed to be packed beyond the threshold of the bin. Bin packing problems where one piece can be packed beyond the threshold of the bin has received some attention in the literature; see [6, 15] for examples. In [Coffman et al. 1978] an approximation algorithm called the *First-Fit-Increasing (FFI)* rule has been proposed. The FFI rule packs pieces in ascending order of the piece size. When a piece cannot be packed into a bin, it will be packed into the next bin. In [Coffman et al. 1978] it has been shown that the FFI rule obeys a bound of $\frac{4}{3}$.

The FFI rule can be adapted to solve this problem. Jobs are scheduled in ascending order of their processing times. The jobs are scheduled backwards in time, starting at time d_1. When a job cannot complete inside the window $[w_1, d_1]$, it will be scheduled on the next machine. This process is continued until all the jobs are scheduled or all the machines are scheduled, whichever occurs first. It will be shown that this FFI rule is an $\frac{7}{5}$-approximation for a single time frame. This result is better than $\frac{e}{e-1} \approx 1.6$. A set of jobs such that $\frac{OPT}{FFI} = \frac{18}{13} \approx 1.38$ is also given, where FFI is the number of jobs scheduled by the FFI rule and OPT is the number of jobs scheduled by an optimization algorithm.

Apply the FFI rule machine by machine to the set of jobs J. Let A_i be the number of jobs scheduled on machine M_i. It is clear that for any $1 \leq i \leq m-1$, $A_i \geq A_{i+1}$.

Lemma 5.10 *Any feasible schedule can schedule at most m more jobs than the schedule produced by the FFI rule.*

Proof: If there is one idle machine, then all jobs are scheduled and hence the FFI rule is optimal. Thus, one may assume that there is no idle machine. Let M_i be the machine with the largest amount of idle time in the time window. There are two cases.

Case (1): There is no idle time in the time window of machine M_i. Since M_i has the largest amount of idle time among all machines, it is easy to see that there must be

no idle time in the time window of any other machine. Therefore, no more jobs can be scheduled, and hence the lemma holds.

Case (2): There is idle time in the time window of machine M_i. By the nature of the FFI rule, the length of the idle time in the time window on machine M_i is less than the processing time of any unscheduled jobs. Therefore the length of idle time on any machine is less than the processing time of any unscheduled jobs. If m jobs are taken from the remaining job set, and put one job onto each machine, then the time window of each machine will be more than filled up. An optimal schedule certainly can not schedule any more jobs. □

Let S be a schedule produced by the FFI rule. The machines are divided into three categories.

- *Type I:* Exactly one job is scheduled on the machine.
- *Type II:* Exactly two jobs are scheduled on the machine.
- *Type III:* More than two jobs are scheduled on the machine.

Similarly, the jobs in S are divided into three categories as well.

- *Type I:* Jobs scheduled on *Type I* machines.
- *Type II:* Jobs scheduled on *Type II* machines.
- *Type III:* Jobs scheduled on *Type III* machines.

Let R be the set of unscheduled jobs, i.e., $R = J \setminus S$.

Observation 5.11 *For any feasible schedule S', (1) there is at most one* Type I *job on any machine, (2) there are at most two* Type II *jobs on any machine, and (3) if there is one* Type I *job or two* Type II *jobs scheduled on one machine, then there is no job from R scheduled on that machine.*

Theorem 5.12 *The FFI rule is an $\frac{7}{5}$-approximation algorithm for a single time frame with $m \geq 2$ machines. Moreover, there are sets of jobs such that $\frac{OPT}{FFI} = \frac{18}{13}$, where FFI is the number of jobs scheduled by the FFI rule and OPT is the number of jobs scheduled by an optimization algorithm.*

Proof: Let S be a scheduled produced by the FFI rule. Divide the machines and the jobs in S into three categories as described above. If there is one idle machine in S, then all jobs have been scheduled and hence S is optimal. If $|S| \geq 3m$, then, by Lemma 5.10, the approximation ratio is bounded by $\frac{4}{3} < \frac{7}{5}$. Therefore, one may assume that there is no idle machine and $|S| < 3m$ from now on.

Let S^* be the set of jobs scheduled by an optimization algorithm. Let $\delta = |S^*| - |S|$. Without loss of generality, one may assume that $S \subseteq S^*$. Consider several cases depending on the schedule S.

Case 1: In S all machines are *Type I* machines. This means that all jobs scheduled in S are *Type I* jobs. By Observation 1, no job from R can be scheduled on any machine in any feasible schedule. Hence, S is an optimal schedule.

Case 2: In S all machines are *Type II* machines. This means that all jobs are *Type II* jobs. By Observation 1, no job from R can be scheduled on any machine in any feasible schedule. Hence, S is an optimal schedule.

Case 3: In S all machines are *Type I* or *Type II* machines. Let $T1$ be the number of *Type I* machines and $\Gamma 1$ be the set of *Type I* jobs. It is clear that $T1 = |\Gamma 1|$. By Observation 1, to accommodate one extra job from R, one *Type II* job have to be moved to a *Type I* machine. It implies that the number of extra jobs that can be accommodated by S^* is bounded by the number of *Type I* machines, i.e., $\delta \leq T1$. Now focus on the set of jobs $\Gamma 1 \cup R$. By Lemma 5.10 and Observation 1, from this set of jobs, S^* can schedule at most m jobs. That is, $T1 + \delta \leq m$, and hence $\delta \leq m - T1$. Thus, $\delta \leq \min\{T1, m - T1\}$. Therefore, $\frac{\delta}{|S|} \leq \frac{\min\{T1, m-T1\}}{T1+2(m-T1)} \leq \frac{1}{3}$. This implies that $\frac{|S^*|}{|S|} \leq \frac{4}{3}$.

Case 4: In S all machines are *Type I* or *Type III* machines. Let $T1$ be the number of *Type I* machines and $T3$ be the number of *Type III* machines. Let $\Gamma 1$ be the set of *Type*

I jobs and $\Gamma 3$ be the set of *Type III* jobs. Now focus on the set of jobs $\Gamma 1 \cup R$. From this job set, by Lemma 5.10 and Observation 5.11, S^* can schedule at most m jobs. That is, $T1 + \delta \leq m$, and hence $\delta \leq m - T1$. It follows that $\frac{\delta}{|S|} \leq \frac{m-T1}{|\Gamma 3|} = \frac{m-T1}{3(m-T1)} \leq \frac{1}{3}$. This implies that $\frac{|S^*|}{|S|} \leq \frac{4}{3}$.

Case 5: In S all machines are *Type II* or *Type III* machines. Let $T2$ be the number of *Type II* machines and $T3$ be the number of *Type III* machines. Consider two sub-cases.

Case 5.1: $T3 \geq T2$. By Lemma 5.10, $\delta \leq m$. Therefore $\frac{\delta}{|S|} \leq \frac{m}{2 \cdot T2 + 3 \cdot T3} \leq \frac{m}{(5/2)m} = \frac{2}{5}$. This implies that $\frac{|S^*|}{|S|} \leq \frac{7}{5}$.

Case 5.2: $T3 < T2$. Let $\Gamma 2$ be the set of *Type II* jobs and $\Gamma 3$ be the set of *Type III* jobs. Then, $J = \Gamma 2 \cup \Gamma 3 \cup R$. Consider the jobs from $\Gamma 2 \cup R$. By Observation 1, S^* can schedule at most two jobs from this set on each machine. Therefore, $\delta + |\Gamma 2| \leq 2m$. Since $|\Gamma 2| = 2 \cdot T2$ and $m = T2 + T3$, it follows $\delta \leq 2 \cdot T3$. Hence, $\frac{\delta}{|S|} \leq \frac{2 \cdot T3}{2 \cdot T2 + 3 \cdot T3} \leq \frac{2}{5}$. This implies that $\frac{|S^*|}{|S|} \leq \frac{7}{5}$.

Case 6: In S all machines are *Type I*, *Type II*, or *type III* machines. Let $T1$ be the number of *Type I* machines, $T2$ be the number of *Type II* machines, and $T3$ be the number of *Type III* machines. Let $\Gamma 1$ be the set of *Type I* jobs, $\Gamma 2$ be the set of *Type II* jobs, and $\Gamma 3$ be the set of *Type III* jobs. Then, $J = \Gamma 1 \cup \Gamma 2 \cup \Gamma 3 \cup R$. Now focus on the set of jobs $\Gamma 1 \cup R$. By Observation 1, S^* can only schedule one job from this set on each machine. Therefore, $\delta + T1 \leq m$. Since $m = T1 + T2 + T3$, $\delta \leq T2 + T3$. Next, consider the set of jobs $X = \Gamma 2 \cup R$. By Observation 1, for any machine M_i, if S^* schedules one *Type I* job on M_i, then S^* can schedule on M_i at most one more job from the job set X; on the other hand, if S^* does not schedule any *Type I* job on M_i, then S^* can schedule at most two jobs from the job set X. It is clear that in the schedule S^*, exactly $T1$ machines are used to schedule the *Type I* jobs. Therefore, $\delta + 2 \cdot T2 \leq T1 + 2 \cdot (m - T1)$. Since $m = T1 + T2 + T3$, it follows $\delta \leq 2 \cdot T3 + T1$. From the above discussion, $\frac{\delta}{|S|} \leq \frac{\min\{2 \cdot T3 + T1, T2 + T3\}}{3 \cdot T3 + 2 \cdot T2 + T1}$. If $2 \cdot T3 + T1 \leq T2 + T3$, then

$$T3 + T1 \leq T2,$$

or equivalently,

$$4 \cdot T3 + 3 \cdot T1 \leq 4 \cdot T3 + 4 \cdot T1 \leq 4 \cdot T2.$$

Adding $6 \cdot T3 + 2 \cdot T1$ to both sides of the equation, one gets

$$10 \cdot T3 + 5 \cdot T1 \leq 6 \cdot T3 + 4 \cdot T2 + 2 \cdot T1,$$

or equivalently,

$$2 \cdot T3 + T1 \leq \frac{2}{5}(3 \cdot T3 + 2 \cdot T2 + T1).$$

Therefore, $\frac{\min\{2 \cdot T3 + T1, T2 + T3\}}{3 \cdot T3 + 2 \cdot T2 + T1} = \frac{2 \cdot T3 + T1}{3 \cdot T3 + 2 \cdot T2 + T1} \leq \frac{2}{5}$. On the other hand, if $2 \cdot T3 + T1 > T2 + T3$, then

$$T2 < T3 + T1 \leq T3 + 2 \cdot T1.$$

Adding $4 \cdot T2 + 5 \cdot T3$ to both sides of the equation, one gets

$$5 \cdot T2 + 5 \cdot T3 < 6 \cdot T3 + 4 \cdot T2 + 2 \cdot T1,$$

or equivalently,

$$T2 + T3 < \frac{2}{5}(3 \cdot T3 + 2 \cdot T2 + T1).$$

Therefore, $\frac{\min\{2 \cdot T3 + T1, T2 + T3\}}{3 \cdot T3 + 2 \cdot T2 + T1} = \frac{T3 + T2}{3 \cdot T3 + 2 \cdot T2 + T1} \leq \frac{2}{5}$. In both cases:

$$\frac{\min\{2 \cdot T3 + T1, T2 + T3\}}{3 \cdot T3 + 2 \cdot T2 + T1} \leq \frac{2}{5}.$$

Hence, $\frac{|S^*|}{|S|} \leq \frac{7}{5}$.

□

Figure 5.2.3 shows a set of jobs such that $\frac{OPT}{FFI} = \frac{18}{13} \approx 1.38$, where FFI is the number of jobs scheduled by the FFI rule and OPT is the number of jobs scheduled by an optimization algorithm.

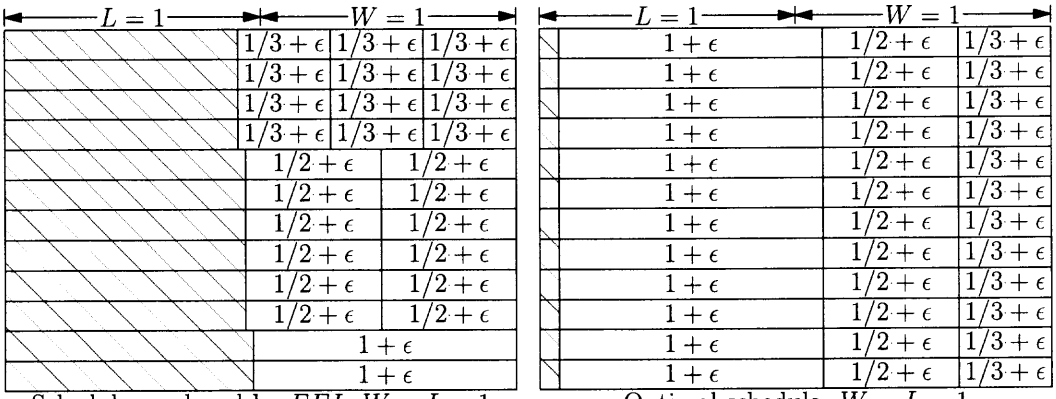

Figure 5.1 Illustrating the worst-case ratio of the FFI rule.

5.3 Profit Proportional to Processing Time

In this section, the case where the profit of a job J_i is proportional to its processing time is studied; i.e., $pf_i = \alpha \cdot p_i$ for some constant α. Since α is a constant, it can be scaled. So it is enough to consider only $\alpha = 1$. It will be shown that in this case, the running time of the FPTAS can be reduced to $O(\frac{n^2}{\epsilon})$. This compares favorably with the running time of the FPTAS for the arbitrary profit case, which is $O(\frac{n^3}{\epsilon})$.

The original problem can be changed to a slightly different problem: In each time frame, set the length of the time window to be 0 and the length of the leading interval to be the length of the entire time frame; i.e., $w_i = d_i$ for each $1 \leq i \leq z$. In this new problem, a job must be finished at the end of the time frame, and each time frame can schedule at most one job. This problem can be solved optimally in $O(n^2)$ time.

Let $G(i)$ be the maximum total profit that can be obtained by scheduling the jobs whose delivery time is d_i or earlier. $G(1), G(2), \cdots, G(z)$ will be computed, and the maximum total profit will be given by $G(z)$. The base case can be computed easily: $G(1)$ is the profit of the longest job that can be finished at time d_1. Assume that $G(1)$, $G(2)$, $\cdots$, $G(i-1)$ have been computed correctly, $G(i)$ can be computed as follows.

Algorithm B

$max = G(i-1)$

For each job $J_k \in J^i$

Suppose $d_i - p_k$ is in the time frame $F_{i'}$.

If $G(i'-1) + pf_k > max$, then $max = G(i'-1) + pf_k$.

$G(i) = max$

The correctness of the algorithm is straightforward: At most one job can be scheduled in the time frame F_i and the above procedure tries all possibilities. Moreover, for each job $J_k \in J^i$ and $d_i - p_k$ in the time frame $F_{i'}$, the maximum total profit that can be obtained is $G(i'-1) + pf_k$, since one cannot schedule any other jobs in the time frames $F_{i'}, \cdots, F_i$. The running time of the dynamic programming algorithm is $O(n^2)$, since there are at most $O(n)$ time frames and for each time frame at most $O(n)$ time is spent.

Let S be the schedule produced by Algorithm B for the modified problem. Now convert the schedule S into the schedule $\hat{S}$ for the original problem as follows. In the schedule S, change back each time window W_i into its original length W and change back each leading interval L_i into its original length L. Clearly, in S, at most one job is completed in any time window. Then the schedule S is scanned, window by window, to construct the schedule $\hat{S}$. When the first window W_1 is scanned, there are two cases to consider.

Case I: No job of J^1 is completed in this time window. There are two sub-cases to consider.

Case I(a): No job is scheduled in this time window. In this case, do nothing and scan the next time window W_2.

Case I(b): There is a job, say job J_k, scheduled in this time window. In this case, it is easy to see that job J_k is not in J^1. Assume that job J_k is completed at time d_i. The next time window W_{i+1} is scanned.

Case II: There is one job, say J_k, completed in the time window W_1. Again, there are two sub-cases to consider.

Case II(a): $p_k \geq \frac{W}{2}$. If the processing time of J_k is greater than or equal to $\frac{W}{2}$, then keep the position of J_k unchanged; i.e., the completion time of J_k will be at d_1. Scan the next time window W_2.

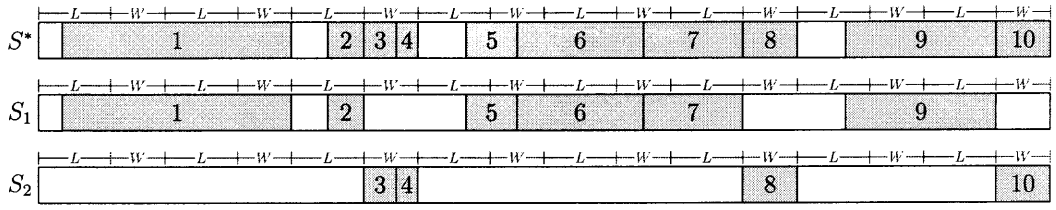

Figure 5.2 Decomposition of S^* into S_1 and S_2.

Case II(b): $p_k < \frac{W}{2}$. If the processing time of J_k is less than $\frac{W}{2}$, then move job J_k as far as possible. That is, schedule job J_k as early as possible but keep it in the time frame F_1. Schedule the jobs of $J^1 \setminus \{J_k\}$ in the remaining space of the time window W_1 by the Largest-Processing-Time (LPT) rule until no jobs can be scheduled or the total processing time of the scheduled jobs is greater than or equal to $\frac{W}{2}$. Scan the next time window W_2.

The schedule S is scanned, window by window, until the last window. Finally, the schedule $\hat{S}$ is obtained.

Let PF_S be the total profit obtained by S and $PF_{\hat{S}}$ be the total profit obtained by $\hat{S}$. Clearly, $PF_S \leq PF_{\hat{S}}$. Let S^* be an optimal schedule and PF_{S^*} be the total profit obtained by S^*. By Lemma 5.1, one may assume that within each time frame, the jobs are scheduled in descending order of their processing times. Divide S^* into two schedules, say S_1 and S_2, such that S_1 contains the longest job from each time frame and S_2 contains all the remaining jobs. Figure 5.3 shows a schedule S^* and its subdivision into S_1 and S_2.

Consider the schedule S_1 first and label all the jobs from S_1 from left to right as $J_1, J_2, \cdots, J_x$. Divide S_1 further into S_1' and S_1'' such that S_1' contains the jobs with odd labels and S_2'' contains the jobs with even labels. Figure 5.3 shows the schedule S_1 and its subdivision into S_1' and S_2''. In S_1', for each time frame, there is at most one job that completes in that time frame. Moreover, any pair of jobs in S_1' do not share a time window. S_1' is a feasible schedule for the new instance where each time window has length zero. Therefore, the total profit of S_1' is less than or equal to PF_S. Similarly, the total profit of S_1'' is less than or equal to PF_S as well. Therefore, the total profit of S_1 is less than or equal to $2 \cdot PF_S$. Since $PF_S \leq PF_{\hat{S}}$, the total profit of S_1 is less than $2 \cdot PF_{\hat{S}}$.

The following lemma can be used to compare the schedule S_2 with $\hat{S}$.

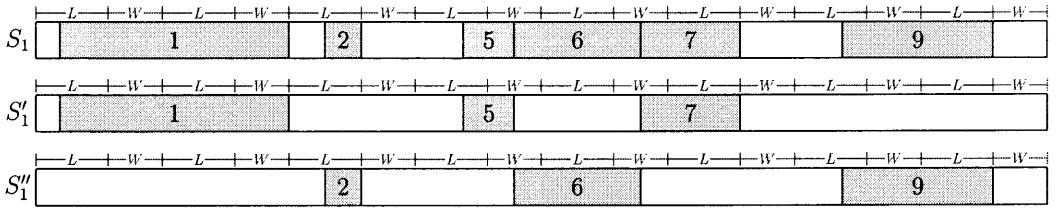

Figure 5.3 Decomposition of S_1 into S_1' and S_1''.

Lemma 5.13 *Let $PF_{\hat{S}}$ be the total profit of $\hat{S}$ and PF_{S_2} be the total profit of S_2. Then $PF_{S_2} \leq 2 \cdot PF_{\hat{S}}$.*

Proof: S_2 is obtained by deleting the longest job from each time window from the optimal schedule S^*. There is no job spanning two time windows in S_2; i.e., S_2 is scanned window by window, either the time window is empty or the time window has some small jobs whose total processing time is less than or equal to W. Now, one can compare the total profits between S_2 and $\hat{S}$, window by window. Let $PF_{S_2}(i)$ be the total profit of the jobs scheduled in S_2 in the time window W_i. Likewise, let $PF_{\hat{S}}(i)$ be the total profit of the jobs scheduled in the time window W_i. It can be shown that $\sum_{i=1}^{z} PF_{S_2}(i) \leq \sum_{i=1}^{z} 2 \cdot PF_{\hat{S}}(i)$.

Consider the window W_i. If S_2 does not schedule any jobs in W_i, then it is clear that $PF_{S_2}(i) = 0 \leq 2 \cdot PF_{\hat{S}}(i)$. Now, suppose that S_2 schedules some jobs in W_i. Let $J_{i_1}, J_{i_2}, \cdots, J_{i_x}$ be the jobs scheduled in S_2 such that $p_{i_1} \geq p_{i_2} \geq \cdots \geq p_{i_x}$. Now consider how $\hat{S}$ is constructed from S as stated above.

Case I(a): This case cannot occur. It can be proven by contradiction. If Case I(a) occurs, job J_{i_1} can always be scheduled in the time window W_i in S, and hence get a better schedule than S. But this contradicts the fact that Algorithm B is optimal for the modified problem.

Case I(b): If Case I(b) occurs, let job J_k be scheduled in $\hat{S}$ in this time window and completed at time $d_{i'}$, $i' > i$. Then, the schedule $\hat{S}$ is all full in the time interval $[d_i, d_{i'}]$. Clearly, $\sum_{j=i}^{i'} PF_{S_2}(j) \leq \sum_{j=i}^{i'} 2 \cdot PF_{\hat{S}}(j)$.

Case II(a): If Case II(a) occurs, let job J_k be scheduled in $\hat{S}$ and completed in the time window W_i. The processing time of J_k must be greater than or equal to $\frac{W}{2}$. It is clear that $PF_{S_2}(i) \leq W \leq 2 \cdot PF_{\hat{S}}(i)$.

Case II(b): If Case II(b) occurs, let job J_k be scheduled in $\hat{S}$ and completed in the time window W_i. The processing time of J_k is less than $\frac{W}{2}$. It can be shown that $p_k \geq p_{i_1}$. If not, one can replace J_k by J_{i_1} in S, increasing the total profit of S. This contradicts the fact that Algorithm B is optimal for the modified problem. Therefore,

$$\frac{W}{2} > p_k \geq p_{i_1} \geq p_{i_2} \geq \cdots \geq p_{i_x}.$$

When obtaining $\hat{S}$ from S, the job J_k is scheduled such that the completion time of job J_k is as early as possible, and the jobs of $J^i \setminus \{J_k\}$ are scheduled in the remaining space of the time window W_i by the LPT rule until no jobs can be scheduled or the total processing time of the scheduled jobs in W_i is greater than or equal to $\frac{W}{2}$. If at last the total processing time of the scheduled jobs in W_i is greater than or equal to $\frac{W}{2}$, then it must be true that $PF_{S_2}(i) \leq W \leq 2 \cdot PF_{\hat{S}}(i)$. If at last the total processing time of the scheduled jobs in W_i is less than $\frac{W}{2}$ and no jobs can be scheduled, then $J_{i_1}, J_{i_2}, \cdots, J_{i_x}$ must all have been scheduled in W_i. Hence, $PF_{S_2}(i) \leq PF_{\hat{S}}(i)$.

After scanning all the windows, one must have $\sum_{i=1}^{z} PF_{S_2}(i) \leq \sum_{i=1}^{z} 2 \cdot PF_{\hat{S}}(i)$. Therefore, $PF_{S_2} \leq 2 \cdot PF_{\hat{S}}$. □

Finally, the following result is obtained for the case of profit proportional to its processing time.

Theorem 5.14 *There is an $O(\frac{n^2}{\epsilon})$-time FPTAS for the case of profit proportional to its processing time on a single machine. The algorithm can be extended to $m \geq 2$ machines with running time $O(\frac{mn^2}{\epsilon})$ and performance bound $\frac{1}{1-e^{-1/(1+\epsilon)}}$.*

Proof: The total profit of S^* is the total profit of S_1 plus the total profit of S_2. Since the total profit of S_1 is less than or equal to twice the total profit of $\hat{S}$ and the total profit of S_2 is less than or equal to twice the total profit of $\hat{S}$, so $PF_{S^*} \leq 4 \cdot PF_{\hat{S}}$.

The FPTAS consists of first constructing a schedule S by Algorithm B for the modified problem. Then, convert S into $\hat{S}$ as described above. Let $E = PF_{\hat{S}}$. The same algorithm as in Section 2.2 can be used again. This time let $Kn = \frac{\epsilon \cdot E}{1+\epsilon}$. The analysis is the same as in Section 2.2. Let PF_{opt} be the total profit of an optimization algorithm, PF_{alg} be the total profit of the algorithm, and $PF_{opt'}$ be the total profit of the new job instance where the profit of each job J_i is replaced by $\lfloor \frac{pf_i}{K} \rfloor$. Then,

$$\begin{aligned} PF_{opt} - PF_{alg} &\leq PF_{opt} - K \cdot PF_{opt'} \\ &\leq K \cdot n \\ &= \frac{\epsilon \cdot E}{1+\epsilon} \\ &\leq \frac{\epsilon}{1+\epsilon} PF_{opt}. \end{aligned}$$

Therefore,

$$\frac{PF_{opt}}{PF_{alg}} \leq 1 + \epsilon.$$

Because it has been known that the total profit of an optimal solution is less than $4 \cdot E$ for any feasible solution, the size of the table can be reduced to $O(\frac{n \cdot E}{K})$. The running time is $O(\frac{n \cdot E}{K}) = O(\frac{n^2}{\epsilon})$.

The algorithm for a single machine can be extended to parallel and identical machines. The result follows from Lemma 5.4. □

5.4 Conclusion

In this chapter a model of production and delivery scheduling is given where each job is supposed to be completed within a specified time window and the time windows are disjoint. Three kinds of profits are considered: (1) arbitrary profit, (2) equal profit, and (3) profit proportional to its processing time. In the first case, a pseudo-polynomial time algorithm is given to find an optimal schedule for a single machine. Based on the pseudo-polynomial time algorithm, A FPTAS with running time $O(\frac{n^3}{\epsilon})$ is developed. In

the second case, an $O(n \log n)$-time algorithm is given to find an optimal schedule for a single machine. In the third case, an improved FPTAS is given with running time $O(\frac{n^2}{\epsilon})$. All algorithms can be extended to parallel and identical machines with a certain degradation of performance bounds. For the equal profit case, a $\frac{7}{5}$-approximation for the special case where there is a single time frame and $m \geq 2$ identical and parallel machines is given.

In the current model, it has been assumed that all time windows have the same length W and all leading intervals have the same length L. This assumption can in fact be relaxed to allow for variable window lengths and variable leading interval lengths. The pseudo-polynomial time algorithm and polynomial time algorithm will work under this relaxation.

CHAPTER 6

PREEMPTIVE SCHEDULING ALGORITHMS WITH NESTED AND INCLUSIVE PROCESSING SET RESTRICTIONS

6.1 Online Algorithm

An example presented in this section shows that there does not exist an optimal online algorithm, even for the inclusive processing set case. Consider the following example.

Let $m = 4$ and let there be two machine intervals $MI_1 = \{M_1, M_2, M_3\}$ and $MI_2 = \{M_1, M_2, M_3, M_4\}$. At time $t = 0$, eight jobs are released: $J_1 = J_2 = J_3 = J_4 = J_5 = J_6 = (3, MI_1)$ and $J_7 = J_8 = (5, MI_2)$.

Suppose there is an optimal online algorithm A. An adversary will observe the schedule produced by algorithm A until time $t = 6$. There are two cases to consider.

Case 1: If the first six jobs are all completed by time $t = 6$, then machines M_1, M_2 and M_3 are fully occupied by the six jobs in the time interval $[0, 6]$. In this case the makespan of the schedule produced by Algorithm A is at least 8. (Algorithm A can schedule three time units each for jobs J_7 and J_8 in the time interval $[0, 6]$, and the remaining two units of time in the time interval $[6, 8]$.) However, the optimal schedule has makespan 7. Therefore, algorithm A cannot be optimal.

Case 2: If some job(s) among the first six jobs are not completed by time $t = 6$, then the adversary releases 12 jobs with processing time one unit and these 12 jobs can only be scheduled on the machines in MI_1. It is clear that the makespan of the scheduled produced by algorithm A is strictly greater than 10. However, the optimal schedule has makespan 10.

In either case, algorithm A does not produce an optimal schedule.

6.2 A Simple Algorithm

McNaughton's rule solves the problem $P \mid pmtn \mid C_{\max}$ in linear time. It first computes $\theta = \max\left\{\max_{j=1}^{n}\{p_j\}, \sum_{j=1}^{n} p_j/m\right\}$. It then treats θ as a deadline and schedules the jobs

so that no job can complete beyond θ. Jobs are scheduled on the first machine, one after another, until a job completes beyond θ. It then cuts the job at time θ and schedule the remaining portion of the job on the second machine. This process is repeated until all jobs are scheduled. It is clear that McNaughton's rule yields a linear time implementation.

6.2.1 Extended McNaughton's Rule

Suppose there is one job among the n jobs that cannot be scheduled in a time interval, say $[t_1, t_1']$, but can be scheduled in any other time before θ. Furthermore, suppose there is enough time to schedule the job before θ. Then this version of the problem can be solved by the Extended McNaughton's Rule as follows. This job is called the *restricted job*.

The restricted job is scheduled on the first machine in the time intervals $[\tau_1, t_1]$ and $[t_1', \tau_1']$, where $0 \leq \tau_1 < \tau_1' \leq \theta$. Now the restricted job occupies the time intervals $[\tau_1, t_1]$ and $[t_1', \tau_1']$. Other unrestricted jobs are then scheduled on the first machine in the time intervals $[0, \tau_1]$, $[t_1, t_1']$ and $[\tau_1', \theta]$, until the total processing time of the jobs on the first machine exceeds θ. Then, the extra part of the job is cut and scheduled on the second machine. At this moment, the same situation happens again: one has at most $n-1$ jobs and $m-1$ machines, and there is at most one restricted job (which is part of the job cut from the first machine) that can only be scheduled in some fixed time intervals. So the problem can be solved recursively. This algorithm is called the *Extended McNaughton's Rule*. By the above discussions, if the number of time intervals is a constant, then there are at most a constant number of preemptions on each machine. So the running time of the Extended McNaughton's Rule is still $O(n)$.

6.2.2 Algorithm *Schedule-Nested-Intervals*

An optimal preemptive algorithm will be presented for the nested processing set case. The algorithm is a recursive algorithm and will be called *Schedule-Nested-Intervals*. The algorithm works from the outermost interval to the innermost one. Assume there is only one outermost interval, say MI_x; otherwise, one can work on each outermost interval separately. First, compute the value $\sigma(MI_x)$ and let $\theta = \sigma(MI_x)$. Suppose

$MI_1, MI_2, \ldots, MI_z$ are the machine intervals directly under MI_x. Then, schedule the jobs from $J(MI_x)$ on the machines $Y = MI_x - (MI_1 \cup \ldots \cup MI_z)$ by the McNaughton's rule. Let R' be the set of jobs remaining after the McNaughton's rule is applied. There are two cases to consider.

Case 1: R' is empty — This means that the jobs in $J(MI_x)$ are all scheduled on the machines in Y. One can now consider the machine intervals $MI_1, \ldots, MI_z$ separately, since they are disjoint. Jobs in $J(MI_i)$, $1 \leq i \leq z$, will be scheduled on the machines in MI_i.

Case 2: R' is not empty — For each interval MI_i, $1 \leq i \leq z$, one can compute a value $extra_i = (\theta - \eta(MI_i)) \cdot |MI_i|$. It is clear that $extra_i$ represents the amount of processing time that can be taken from R', and executes on the machines in MI_i, without any job completing after θ. One can take a subset of jobs with total processing time $extra_1$ from R', and add them to $J(MI_1)$. Similarly, take a subset of jobs with total processing time $extra_2$ from R', and add them to $J(MI_2)$. This process will be repeated until no jobs are left in R'. (It can always be done; for a proof, see the proof of Lemma 6.1.) Now, schedule the jobs of $J(MI_i)$ on the machines in MI_i recursively, $1 \leq i \leq z$. Note that it is possible to have a restricted job (the unfinished part of the job when McNaughton's rule is applied to MI_x). So it is necessary to use the Extended McNaughton's Rule, rather than the McNaughton's rule. It is easy to see that for each machine interval, $MI_1, \ldots, MI_z$, there is at most one restricted job.

Shown above is the recursive procedure Schedule-Nested-Intervals. It has five parameters. The procedure calls itself recursively to schedule jobs in the inner machine intervals. Shown below is the main procedure. Assuming that there is only one outermost machine interval, say MI_x, it computes $\sigma(MI_x)$, let θ be $\sigma(MI_x)$, and then calls procedure Schedule-Nested-Intervals.

Lemma 6.1 *The procedure* Schedule-Nested-Intervals *schedules the set of jobs* $\{J_j | S_j \subseteq MI_x\}$ *on the machines of* MI_x *with completion time at most* $\sigma(MI_x)$.

```
Recursive Procedure Schedule-Nested-Intervals(MI_x, R, θ, J_j, [a, b]);
Input: MI_x: a machine interval; R: a set of jobs coming from the outer
       machine interval; θ: a deadline; J_j: a restricted job in R; [a, b]: a
       restricted time interval;
if there is no machine interval inside MI_x then
    Schedule the jobs of R ∪ J(MI_x) by the Extended McNaughton's
    Rule; θ is the deadline;
else
    Let MI_1, ..., MI_z be the machine intervals inside MI_x;
    Schedule the jobs of R ∪ J(MI_x) on the machines in
    MI_x − (MI_1 ∪ ... ∪ MI_z) by the Extended McNaughton's rule; θ is
    the deadline;
    Let R' be the remaining jobs;
    for i = 1 to z do
        Let extra_i = (θ − η(MI_i)) · |MI_i| ;
        Whenever possible, pick a subset of jobs from R', denoted by R'_i,
        with total processing time extra_i;
        Let the job J_res_i ∈ R'_i be the restricted job (that is left from the
        machine interval MI_{i−1} for i > 1 or from MI_x for i = 1);
        R' = R' − R'_i;

    Get the restricted interval for J_res_1, say [t_1, t'_1], by checking the
    schedule on the machines in MI_x − (MI_1 ∪ ... ∪ MI_z);
    for i = 1 to z do
        Call Schedule-Nested-Intervals(MI_i, R'_i, θ, J_res_i, [t_i, t'_i]);
        Get the restricted interval for J_res_{i+1}, say [t_{i+1}, t'_{i+1}], by checking
        the scheule on the machines in MI_i;
```

```
Procedure Main;
Let MI_x be the outermost machine interval;
Compute σ(MI_x) and let θ = σ(MI_x);
Call Schedule-Nested-Intervals(MI_x, ∅, θ, ∅, ∅)
```

Proof: The correctness of the lemma can be proved by induction on the level of MI_x. If the level of MI_x is 1, the algorithm is essentially McNaughton's rule and $C_{max} = \sigma(MI_x)$.

Assuming that the lemma is true if the level of MI_x is h, the goal is to prove that it is also true if the level of MI_x is $h+1$. If the level of MI_x is $h+1$, then for each machine interval MI_i, $1 \leq i \leq z$, directly under MI_x, the total processing time of the extra jobs assigned to it is at most $(\sigma(MI_x) - \eta(MI_i)) \cdot |MI_i|$. So, $\sigma(MI_i)$ is at most $\sigma(MI_x)$. By the inductive hypothesis, one can obtain a feasible schedule with completion time at most $\sigma(MI_x)$ for each MI_i, $1 \leq i \leq z$. The schedule for the machines

in $Y = MI_x - (MI_1 \cup \ldots \cup MI_z)$ is obtained by the Extended McNaughton's Rule. So its completion time is at most $\sigma(MI_x)$ as well.

It remains to be shown that no job (or job part) left after assigning the jobs of R' to $MI_1, \ldots, MI_i$. Suppose not, then the total processing time on each machine of MI_x is exactly $\sigma(MI_x)$. So, the total processing time of the jobs of $\{J_j | S_j \subseteq MI_x\}$ is strictly greater than $\sigma(MI_x) \cdot |MI_x|$. This contradicts the definition of $\sigma(MI_x)$. □

From Lemma 6.1, one can obtain a feasible schedule with $C_{max} = \sigma(MI_x)$, by using the procedure *Schedule-Nested-Intervals*. It is optimal because $\sigma(MI_x)$ is a trivial lower bound of any feasible schedule. The running time of the Extended McNaughton's Rule is linear with respect to the number of jobs scheduled. As mentioned before, there is at most one restricted job for each machine interval. So the number of restricted jobs is $O(m)$. Only the restricted jobs will be considered in different machine intervals. Therefore, the running time of the algorithm is $O(m + n)$. [1] Since it is reasonable to assume that $n > m$, the running time is $O(n)$.

Theorem 6.2 *The procedure* Schedule-Nested-Intervals *produces an optimal schedule with running time* $O(n)$.

6.3 A Maximal Algorithm

Hong and Leung [Hong and Leung 1992] give a maximal algorithm for the problem $P \mid pmtn \mid C_{max}$; their algorithm can be extended to an online algorithm for the problem $P \mid pmtn, r_j \mid C_{\max}$ as well. It is shown in this section that their algorithm can be adapted to produce a maximal schedule for this problem as well.

6.3.1 Hong and Leung's Algorithm

One may assume that the processing time of the jobs are in nonincreasing order; i.e., $p_1 \geq p_2 \ldots \geq p_n$. Hong and Leung's algorithm works as follows.

[1] One can assume that the intervals have been organized as an interval tree so each directly nested machine interval can be retrieved in constant time.

```
Algorithm:Hong and Leung's Algorithm
i = 1;
ave = Σ_{j=1}^{n} p_j/m;
while p_i > ave do
    Schedule job J_i on machine M_i;
    i = i + 1;
    ave = Σ_{j=i}^{n} p_j/(m − i + 1);
Schedule jobs J_i, ..., J_n on machines M_i, ..., M_m by McNaughton's
rule;
```

For any job J_i, if $p_i > \sum_{j=i}^{n} p_j/(m-i+1)$, this job is called the *long* job; otherwise, it is called the *short* job. It is clear that each long job is scheduled exclusively on a single machine by Hong and Leung's algorithm, while the short jobs are scheduled by McNaughton's rule. For any machine, let the total processing time of the jobs assigned to the machine be the *length* of the machine. The machine with the shortest length is called the *shortest machine.*

Lemma 6.3 *Given a schedule S for the m machines, let $l(M_i)$ be the length of machine M_i and $J(M_i)$ be the set of jobs assigned to machine M_i. Suppose*

1. $l(M_1) = l(M_2) \ldots = l(M_k)$, $1 \leq k < m$.
2. $l(M_{k+1}) \leq l(M_{k+2}) \ldots \leq l(M_m) \leq l(M_1)$.
3. *for $k+1 \leq i \leq m$, any job in $J(M_i)$ can only be scheduled on M_i.*
4. *for $1 \leq i \leq k$, any job J_j in $J(M_i)$ can be scheduled on S_j.*

Let S' be a new schedule produced by Hong and Leung's algorithm on the same set of jobs, but without the constraint (3); i.e., any job in $J(M_i)$, $k+1 \leq i \leq m$, can be scheduled on any machines. Suppose the length of the shortest machine in S' is c, and $c \geq l(M_m)$. Then, there is a feasible schedule S'' such that

1. *there is no idle time on any machine before time instant c.*
2. *any job not finished by the time instant c is scheduled continuously from time 0 until it completes.*

Proof: Schedule S'' is obtained from S' as follows. In S', divide all jobs into two subsets G and H: G contains all jobs (or job parts) executed after time c and H contains all jobs (or job parts) executed before time c.

To obtain schedule S'', one can first reschedule the jobs in H by using the procedure Schedule-Nested-Intervals as described in Section 6.2. It is easy to see that $\theta = c$. So all jobs can be finished by time c. Since there is no idle time before c in S', there is no idle time before time c in S'' as well.

For each job of G, it will be scheduled on one machine, starting from time c. There are at most m such jobs. For each job in G, there is a part with processing time exactly c in H. So this job part must be scheduled continuously from time 0 until c. This implies that the job must be scheduled continuously from time 0 until it completes in S''. □

6.3.2 Maximal Algorithm

The algorithm works from the innermost interval to the outermost interval. For the innermost interval, jobs are scheduled by Hong and Leung's algorithm. The algorithm is presented recursively. Suppose currently machine interval MI_x is processed, under which are the machine intervals MI_1 , ..., MI_z. First, schedule the jobs in $J(MI_x)$ on the machines $Y = MI_x - (MI_1 \cup \ldots \cup MI_z)$ by Hong and Leung's algorithm. There are two cases to consider.

Case 1: The length of the shortest machine in Y is less than or equal to the length of the shortest machine in $MI_1 \cup \ldots \cup MI_z$. In this case, the interval MI_x is done, and one can proceed to schedule its outer interval.

Case 2: The length of the shortest machine in Y is greater than the length of the shortest machine in $MI_1 \cup \ldots \cup MI_z$. In this case, more work needs to be done as follows. One may assume that the machines in $MI_1 \cup \ldots \cup MI_z$ is relabeled in ascending order of their lengths; i.e., $l(M_1) \leq l(M_2) \leq \ldots \leq l(M_g)$. Now find the largest index g' such that Lemma 6.3 can be applied on the machines in $Y \cup \{M_1, M_2, \ldots, M_{g'}\}$. (Recall that to apply Lemma 6.3, one thing that must be sure is that after applying Hong and Leung's algorithm, the length of shortest machine must be greater than or equal to $l(M_{g'})$). g' can be obtained

by iteratively adding one machine from $\{M_1, \ldots, M_g\}$ until the first time a machine $g'+1$ such that Lemma 6.3 can not be applied to the set of machines $\{M_1, M_2, \ldots, M_{g'+1}\}$ is encountered. The jobs on the machines $Y \cup \{M_1, M_2, \ldots, M_{g'}\}$ is then rescheduled by the method as described in Lemma 6.3. Suppose c is the length of the shortest machine in $Y \cup \{M_1, M_2, \ldots, M_{g'}\}$. It is clear that $c \leq l(M_{g'+1}) \leq \ldots \leq l(M_g)$.

Figure 1 illustrates the maximal algorithm when applied to the set of jobs in Example 1. Figure 1(a) shows the schedule after the maximal algorithm schedules the two innermost intervals. Figure 1(b) shows the schedule after the maximal algorithm schedules the jobs of $J(MI_1)$ on the machines in Y (M_1, M_2 and M_3). Figure 1(c) shows the schedule after the maximal algorithm merges the machines in Y with M_5 and M_6. Figure 1(d) shows the schedule after the maximal algorithm merges the machines M_3, M_5 and M_6 with the machines M_8, M_9 and M_{10}. Figure 1(e) shows the final schedule.

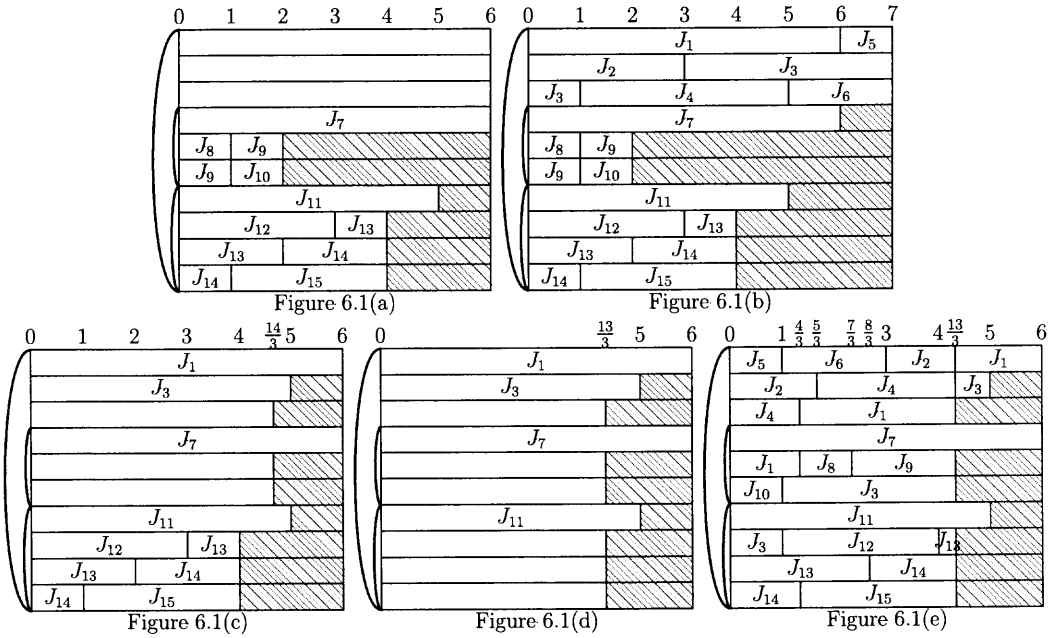

Figure 6.1 An example illustrating the maximal algorithm. Figure 6.1(a) shows the schedule of the innermost intervals. Figure 6.1(b) shows the schedule of the outermost interval on the machines in Y. Figure 6.1(c) and Figure 6.1(d) shows the iterations to find the set of machines to be merged. Figure 6.1(e) shows the final schedule.

Shown below is the procedure Maximal-Schedule-One-Interval. It schedules the jobs in one machine interval only, namely, MI_x. The main procedure will repeatedly call

this procedure for all the machine intervals, from the innermost interval to the outermost interval.

```
Algorithm:Maximal-Schedule-One-Interval(MI_x)
Input: MI_x: a machine interval
if there is no machine interval inside MI_x then
    Schedule the jobs in J(MI_x) on the machines in MI_x by Hong and
    Leung's algorithm;
else
    Let MI_1, MI_2, ..., MI_z be the machine intervals directly under
    MI_x;
    Let c_inner be the length of the shortest machine in MI_1 ∪ ... ∪ MI_z;
    Let Y = MI_x − (MI_1 ∪ ... ∪ MI_z);
    if Y = ∅ then
        Schedule the jobs in J(MI_x) on the shortest machine in
        MI_1 ∪ ... ∪ MI_z;
    else
        Schedule the jobs in J(MI_x) on the machines in Y by Hong and
        Leung's algorithm;
    Let c_outer be the length of the shortest machine in the schedule just
    produced;
    if c_outer > c_inner then
        Let M' = {M_1, ..., M_g} be the subset of machines in
        MI_1 ∪ ... ∪ MI_z with length less than l_outer, and let
        l(M_1) ≤ l(M_2) ... ≤ l(M_g);
        Let J(M_i) be the set of jobs scheduled on machine M_i;
        i = 0;
        repeat
            i = i + 1;
            Reschedule the jobs in J(MI_x) ∪ J(M_1) ... ∪ J(M_i) on the
            machines in Y ∪ M_1 ∪ ... ∪ M_i by Hong and Leung's
            algorithm, without considering any processing set constraints;
            Let c be the length of the shortest machine in the schedule just
            produced;
        until c < l(M_i) or i > g ;
        Reschedule the jobs in J(MI_x) ∪ J(M_1) ∪ ... ∪ J(M_{i-1}) on the
        machines in MI_x ∪ M_1 ∪ ... ∪ M_{i-1} by the method as decribed
        in Lemma 6.3;
```

Lemma 6.4 *Let S be the schedule produced by the algorithm* Maximal-Schedule-One-Interval(MI_x). *Then S is a maximal schedule with respect to the jobs in $\{J_j | S_j \subseteq MI_x\}$ and the machines in MI_x.*

```
Procedure Main;
Order the machine intervals from the innermost one to the outermost one,
  say MI_1, MI_2, ..., MI_k;
for i = 1 to k do
  | Call Maximal-Schedule-One-Interval(MI_i)
```

Proof: The lemma can be proven by induction on the level of MI_x. If the level of MI_x is 1, then the algorithm is the same as Hong and Leung's algorithm, and hence the schedule it produces is maximal.

Assume the lemma is true if the level of MI_x is less than or equal to h. One can prove that it is also true when the level of MI_x is $h+1$. Suppose S is the schedule produced by the algorithm and S is not maximal. Then there is a schedule S^* and a time instant t such that the amount of work done by all machines in the time interval $[0, t]$ is strictly greater than that done by S. Let c be the length of the shortest machine in the schedule S. One may assume that $t > c$. There are two cases to consider.

Case 1: Before time t, S^* executes more of the *jobs* from $J(MI_x)$ than S. In this case, by Lemma 6.3, the jobs from $J(MI_x)$ is either finished before time c or it is scheduled continuously from time 0 until it finishes. Therefore, S^* cannot execute more of the jobs from $J(MI_x)$ than S.

Case 2: Before time t, S^* executes more of the *jobs* in $J(MI_i)$, $1 \leq i \leq z$, than S. In this case, it is clear that the jobs in $J(MI_i)$ can only be scheduled on the machines in the machine interval MI_i. Denote the schedule on the machines of MI_i, before rescheduling, by $\hat{S}$. By the inductive hypothesis, $\hat{S}$ is maximal with respect to all the jobs that can be scheduled on the machines in MI_i. For any machine $M_{i'} \in MI_i$, if $M_{i'}$ is not rescheduled, then before time t, the total amount of processing done on $M_{i'}$ by S is the same as that of $\hat{S}$; if $M_{i'}$ is rescheduled, it is clear that all jobs scheduled by $\hat{S}$ on machine $M_{i'}$ are also finished by S before time c. So before time t, S^* cannot execute more of the jobs in $J(MI_i)$, $1 \leq i \leq z$, than S.

Summarize the discussion above, S is a maximal schedule. □

The frist step of this algorithm is running Hong and Leung's algorithm on the machines in Y; the running time is $O(|J(MI_x)| \log |J(MI_x)|$. Next, one needs to find the first g' machines to fulfill the requirment of Lemma 6.3. The running time of Hong and Leung's algorithm is $O(n \log n)$ for n jobs. But it is known that only the jobs in $J(MI_x)$ can be long jobs, and one only needs to compute the length of the shortest machine instead of schedule all the jobs. So the running time of each try is $O(|J(MI_x)|)$. If binary search instead of searching g' sequentially is used, the running time will be $O(|J(MI_x)| \log m)$. Finally, one needs $O(n)$ time to reschedule all the machines involved. The total running time is clearly $O\left(n + |J(MI_x)|(\log m + \log |J(MI_x)|)\right)$.

The machine intervals are scheduled from the innermost one to the outermost one. When the outermost interval is reached, one will obtain a maximal schedule. Since there are at most $O(m)$ intervals, and for each interval, the running time for MI_x is $O(n + |J(MI_x)|(\log m + \log |J(MI_x|))$, the total running time is $O(mn + n(\log m + \log n))$. It becomes $O(mn + n \log n)$ if assuming n is greater than m.

Theorem 6.5 *For nested intervals, the algorithm* Maximal-Schedule *produces a maximal schedule in time* $O(mn + n \log n)$.

6.4 Different Release Times

In this section, it is assumed that the jobs have different release times. Let $r_0 < r_1 < \ldots < r_{k-1}$ be the k distinct release times. Without loss of generality, one may assume that $r_0 = 0$. An upper bound for the optimal makespan is $U = r_{k-1} + \sum_{j=1}^{n} p_j$ and an obvious lower bound is $L = r_{k-1}$. One can conduct a binary search in the interval $[L, U]$, searching for the optimal makespan C^*. For each value C obtained in the binary search, one can test if there is a feasible schedule with makespan C.

First, divide the time span into k segments: $TS_0 = [r_0, r_1]$, $TS_1 = [r_1, r_2]$, $\ldots$, $TS_{k-1} = [r_{k-1}, C]$. For each segment TS_j, $0 \leq j \leq k-1$, use $l(TS_j)$ to denote the length of the segement. In the next subsection, a network flow approach is given to solve

the nested processing set case, followed by a more efficient algorithm for the inclusive processing set case.

6.4.1 Nested Processing Set

For the nested processing set case, one can construct a network with nodes and arcs as follows. it will be shown later that there is a feasible schedule with makespan C if and only if the solution of the max-flow is $\sum_{j=1}^{n} p_j$.

- For each machine M_i, $1 \leq i \leq m$, and each time segment TS_j, $0 \leq j \leq k-1$, a node $M_{i,j}$ is created. The node is called a *machine-segment node.*

- For each job J_j, $1 \leq j \leq n$, nodes $J_{j,t}$, $J_{j,t+1}$, ..., $J_{j,k-1}$ are created, where the release time of J_j is r_t. Call these nodes the *job-segment nodes.*

- For each job J_j, $1 \leq j \leq n$, a *job node* J_j is created. Finally, a source node S and a sink node T are added.

- For each *job node* J_j, an arc is added from S to J_j with capacity p_j.

- For each *job node* J_j and each *job-segment node* $J_{j,q}$, $t \leq q \leq k-1$, an arc is added from J_j to $J_{j,q}$ with capacity $l(TS_q)$.

- For each *job-segment node* $J_{j,q}$ and each *machine-segment node* $M_{i,q}$, if J_j can be scheduled on machine M_i, then an arc is added from $J_{j,q}$ to $M_{i,q}$ with capacity $l(TS_q)$.

- For each *machine-segment node* $M_{i,q}$, an arc is added from $M_{i,q}$ to the sink T with capacity $l(TS_q)$.

Figure 6.2 shows the picture for three jobs and two machines. There are three release times: r_0, r_1 and r_2. Job 1 is released at r_0, job 2 is released at r_1 and job 3 is released at r_2. Job 1 can be scheduled on machines M_1 and M_2, while jobs 2 and 3 can only be scheduled on M_2.

Lemma 6.6 *There is a feasible schedule with makespan C if and only if the solution of max-flow is $\sum_{j=1}^{n} p_j$.*

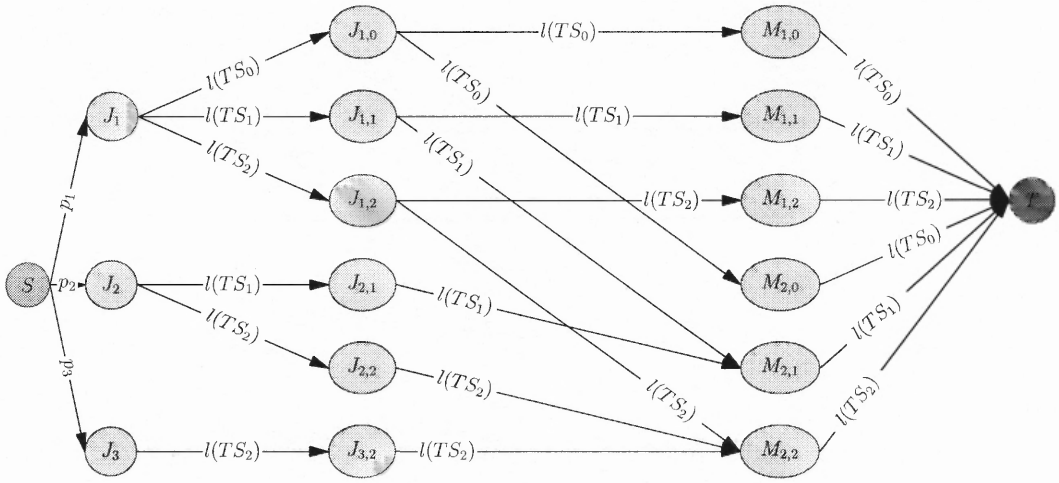

Figure 6.2 The reduction to maxflow problem.

Proof: A known fact is introduced to facilitate the proving Lemma 6.6, ; see Gonzalez and Sahni [Gonzalez and Sahni 1976].

Lemma 6.7 *Let T be a $m \times n$ matrix such that $T(i,j)$ denotes the total processing time of job j on machine i. If for any i, $\sum_{j=1}^{n} T(i,j) \leq C$ and for any j, $\sum_{i=1}^{m} T(i,j) \leq C$, then there is a feasible preemptive schedule with makespan at most C. Moreover, the schedule can be found in time $O(m^2 n^2)$.*

Given a feasible schedule with makespan at most C, it is easy to see that the max-flow of the network constructed above is $\sum_{j=1}^{n} p_j$.

Given the max-flow of the network constructed above, if the flow on the arc $(J_{j,q}, M_{i,q})$ is x, then one can schedule job J_j on machine i in the time segment TS_q for x time units. It is easy to see that for any time segment TS_q: (1) for each job J_j, the total amount of J_j scheduled in TS_q will never exceed $l(TS_q)$, and (2) for each machine-segment $M_{i,q}$, the total processing time assigned to $M_{i,q}$ will never exceed $l(TS_q)$. Because of Lemma 6.7, there is a feasible schedule in the time segment TS_q. □

There are $O((m+n)k)$ nodes and $O(mnk)$ edges in the network. One may assume that $n > m$. Using the max-flow algorithm of Goldberg and Tarjan [Goldberg and Tarjan 1988], the running time is $O(mn^2k^2 \log \frac{nk}{m})$. Thus, the overall running time for testing

feasibility is $O(mn^2k^2 \log \frac{nk}{m} \log P)$, where $P = \sum_{j=1}^{n} p_j$. To obtain the final schedule, One needs to construct the schedule segment by segment. For each time segment, One needs to solve a stochastic matrix with running time $O(m^2 n^2)$. Therefore, the total time for solving all intervals is $O(m^2 n^2 k)$.

Theorem 6.8 *For the nested processing set case, an optimal schedule can be obtained in time $O(mn^2k^2 \log \frac{nk}{m} \log P + m^2 n^2 k)$, where $P = \sum_{j=1}^{n} p_j$.*

6.4.2 Inclusive Processing Set

In this section, a more efficient algorithm is given for the inclusive processing set case. For convenience of presentation, let $r_k = C^*$. First, do some preprocessing on the jobs: for any job J_l released at time r_i, if p_l is greater than $r_{i+1} - r_i$, then cut it into two pieces: a job J_l' released at r_i with processing time $r_{i+1} - r_i$, and a job J_l'' released at time r_{i+1} with processing time $p_l - (r_{i+1} - r_i)$. If the processing time of J_l'' is still greater than $r_{i+2} - r_{i+1}$, it is cut once again into two pieces, and so on. After the preprocessing step, one may assume that for any job released at time r_i, its processing time is less than or equal to $r_{i+1} - r_i$. An array $T(l, i)$, $1 \leq l \leq n$ and $0 \leq i \leq k-1$ is used to store the processing time of job l in the time segment TS_i. Clearly, $T(l, i) = 0$ if the release time of job l is greater than r_i. When constructing the optimal schedule, $T(l, i)$ will be updated so that it reflects the amount of processing of job l done in the time segment TS_i in the optimal schedule. Once $T(l, i)$ is updated, the optimal schedule can be obtained by McNaughton's rule segment by segment, and within each segment the jobs are scheduled from the outermost machine interval to the innermost one.

Assume there are z machine intervals in the job set: $MI_1 \supset MI_2 \supset \ldots \supset MI_z$. For each machine interval MI_j, $1 \leq j \leq z-1$, let Y_j denote the machines in $MI_j - MI_{j+1}$, and let Y_z denote all the machines in MI_z. An array $IDLE(j, i)$, $1 \leq j \leq z$ and $0 \leq i \leq k-1$ is used to store the total idle times on the machines in Y_j during the time segment TS_i. From now on, assume that for every time segment, there are *exactly* z machine intervals. The symbol $J(MI_j, i)$ is used to denote the set of jobs with machine interval MI_j released

at time r_i. Clearly, $J(MI_j, i) = \emptyset$ if there is no job with machine interval MI_j released at time r_i.

In the next subsection, a test will be given to see if there is a feasible schedule with makespan C. In the following subsection, an algorithm will be described to obtain an optimal schedule once the optimal makespan C^* is determined.

Feasibility Test The algorithm works backwards in time. The time segment $TS_{k-1} = [r_{k-1}, r_k]$ can easily be determined. The jobs are scheduled by McNaughton's rule, starting with the jobs in the outermost interval and ending with the jobs in the innermost interval. If some jobs cannot be feasibly scheduled, one can declare that it is impossible to have a schedule with makespan C. Otherwise, $IDLE(j, k-1)$ is computed for all $1 \leq j \leq z$. Let $\beta_j = IDLE(j, k-1)$ for all $1 \leq j \leq z$.

Suppose it have been determined that the jobs in the time segment $[r_{i+1}, r_{i+2}]$ can be feasibly scheduled. Now consider the jobs released at time r_i. the jobs are considered from the outermost interval to the innermost interval. For each MI_j, $1 \leq j \leq z$, let $extra_j$ be the total idle times on the machines in Y_j (i.e., the total idle times in the interval $[r_i, r_{i+1}]$ plus the total idle times in the interval $[r_{i+1}, r_k]$, minus the total processing times of the jobs in $J(MI_j, i)$). If $extra_j < 0$, then some jobs in this machine interval have to be scheduled on some other machines of an inner interval.

After $extra_j$ is computed, the following procedure (Algorithm *Compute-Idle-and-Test-Feasibility*) can be used to compute the amount of processing times needed to be moved between any pair of Y_j's. The variable $move(p, q)$ is used to store the amount of processing times needed to be moved from the machines in Y_p to the machines in Y_q. $extra_j$ is also updated for $1 \leq j \leq z$. Finally, let $\beta_j = extra_j$ for $1 \leq j \leq z$.

The running time of algorithm *Compute-Idle-and-Test-Feasibility* for one time segment is $O(m + n_i)$, where n_i is the number of jobs released at r_i. The for-loop in Steps 2 to 3 takes $O(m + n_i)$ time. The for-loop in Steps 5 to 16 takes $O(m)$ time because in each iteration, either the value of p is increased, or the value of q is increased, and the algorithm will terminate when either p or q reaches z. The for-loop in Steps 17 to 18 takes

```
Algorithm:Compute-Idle-and-Test-Feasibility
Input: r_i: release time; β_1, ..., β_z computed at r_{i+1}.
for j = 1 to z do
    extra_j = |Y_j| · (r_i − r_{i+1}) + β_j −
    (total processing times of the jobs in J(MI_j, i));
q = 2;
for j = 1 to z do
    while extra_j < 0 do
        if q > z then stop and output "Not Feasible";
        if extra_q > 0 then
            move(j, q) = min(extra_q, −extra_j);
            extra_q = extra_q − move(j, q);
            extra_j = extra_j + move(j, q);
            if extra_q = 0 then q = q + 1;
        else
            q = q + 1;

for j = 1 to z do
    β_j = extra_j;
output "Feasible";
```

$O(m)$ time. Thus, the overall running time for testing k time segments is $O((m+n)k)$. Since one may assume that $n > m$, the running time becomes $O(nk)$. To find the optimal makespan, the time needed is $O(nk \log P)$, where $P = \sum_{j=1}^{n} p_j$.

The above algorithm determines if there is enough room to schedule the jobs released at r_i. But there is one issue that needs to be resolved. It is possible that a job released at r_i is the same job as the jobs released at $r_{i+1}, \ldots, r_{k-1}$, due to the preprocessing step. Therefore, the jobs released at r_i must be scheduled carefully so that there is no overlap with the same jobs in subsequent time segments. Fortunately, it can be shown that the jobs can indeed be feasibly scheduled if the jobs released at r_i pass the test of Algorithm *Compute-Idle-and-Test-Feasibility*. In the next subsection, an algorithm will be given to construct an optimal schedule.

Obtaining An Optimal Schedule As was mentioned in Section 6.4.2, to obtain an optimal schedule, $T(l,i)$ needs to be computed which stores the amount of processing of job l done in the time segment TS_i in an optimal schedule. Initially, $T(l,i)$ is the processing time of job l in the time segment TS_i. $T(l,i)$ is computed backwards in time, starting with the time segment TS_{k-1}. $T(l,k-1)$ is exactly the initial value. The jobs

are then scheduled in the time segment TS_{k-1} by McNaughton's rule, starting with the jobs in the outermost interval and ending with the jobs in the innermost interval. From the schedule, set $IDLE(j, k-1)$, $1 \leq j \leq z$ to be the total idle times on the machines in Y_j during the time segment TS_{k-1}. Let $\alpha_j = IDLE(j, k-1)$ for all $1 \leq j \leq k-1$.

Suppose $T(l, i+1)$ has been computed for the jobs in the time segment TS_{i+1}. Now consider the jobs released at time r_i. For each machine interval MI_j, let $\alpha_j = \sum_{x=i+1}^{k-1} IDLE(j, x)$; i.e., α_j is the total idle times on the machines of Y_j after time r_{i+1}. Then algorithm *Compute-Idle-and-Test-Feasibility* is invoked to test feasibility for this time segment. Algorithm *Compute-Idle-and-Test-Feasibility* will compute $extra_j$ and $move(p, q)$, in addition to testing feasibility; see the algorithm in Section 6.4.2

After the Algorithm *Compute-Idle-and-Test-Feasibility* is called, if $extra_j \geq \alpha_j$, then there is no need to fill any job in the idle times in $IDLE(j, x)$, $i+1 \leq x \leq k-1$. The jobs in $J(MI_j, i)$ will be scheduled exclusively on the machines of Y_j in the time segment $[r_i, r_{i+1}]$. On the other hand, if $extra_j < \alpha_j$, then besides scheduling all the machines of Y_j in the time segment $[r_i, r_{i+1}]$, one needs to fill in an amount of processing time equal to $fill_j = \alpha_j - extra_j$ in the idle times in $IDLE(j, x)$, $i+1 \leq x \leq k-1$. Let $fill_j = \max\{\alpha_j - extra_j, 0\}$ for all $1 \leq j \leq z$.

If $fill_j = 0$ for each machine interval MI_j, then all the jobs released at time r_i will be scheduled in the time segment TS_i. Thus, $T(l, i)$ will be the same as the initial value. McNaughton's rule can be used to construct a schedule in this segment. From the schedule, $IDLE(j, i)$ is updated for all $1 \leq j \leq z$.

If $fill_j > 0$ for some machine interval Y_j, then it is necessary to fill some jobs released at time r_i in the idle times in $IDLE(j, x)$, $i+1 \leq x \leq k-1$. It is possible that a job released at r_i is the same job as the jobs released at $r_{i+1}, \ldots, r_{k-1}$, due to the preprocessing step. Therefore, the jobs released at r_i must be scheduled carefully so as to avoid any overlap with the same job in subsequent time segments. In the following, a method to schedule these jobs will be described. The basic idea is that first jobs with total processing times equal to $fill_j$ will be scheduled in the idle times in $IDLE(j, x)$, $i+1 \leq x \leq k-1$. After this step is finished for all the machine intervals, the remaining

jobs will be scheduled in the time segment TS_i by McNaughton's rule. Finally, $IDLE(j,i)$ is updated for all $1 \leq j \leq z$.

Suppose $fill_1 > 0$. Then the jobs of $J(MI_1, i)$ is scheduled with total processing times equal to $fill_1$ in the idle times in $IDLE(1,x)$, $i+1 \leq x \leq k-1$, on the machines of Y_1. The jobs are scheduled in the following manner. For any job $l \in J(MI_1, i)$, this job is tried to schedule in the idle times in $IDLE(1, i+1)$ if $IDLE(1, i+1) > 0$. One can schedule job l for an amount equal to $\min\{fill_1, IDLE(1, i+1), T(l,i), r_{i+2} - r_{i+1} - T(l, i+1)\}$ in the time segment TS_{i+1}. Then $fill_1$, $IDLE(1, i+1)$, $T(l,i)$ and $T(l, i+1)$ are updated. If job l is completely scheduled and $IDLE(1, i+1)$ is still greater than 0, another job of $J(MI_1, i)$ is tried to scheduled in the idle times in $IDLE(1, i+1)$ using the same method. If job l still has some processing time not scheduled, it is tried to schedule in $IDLE(1, i+2)$ and so on. It will be shown later in Lemma 6.9 that one can always schedule the jobs of $J(MI_1, i)$ with total processing times equal to $fill_1$ in the idle times in $IDLE(1,x)$, $i+1 \leq x \leq k-1$, without producing any overlap.

Now, consider the next machine interval Y_j with $fill_j > 0$ and $j > 1$. First find the maximum machine interval index p such that no processing time need to be moved from Y_a to Y_b for any $1 \leq a \leq p-1$ and $p \leq b \leq j$; i.e., $move(a,b) = 0$ for all $1 \leq a \leq p-1$ and $p \leq b \leq j$. Then, for each pair of indexes c and d such that $p \leq c < d-1 \leq j-1$ and $move(c,d) > 0$, the value $move(c,d)$ is added to each item $move(c, c+1), move(c+1, c+2), \ldots,$ and $move(d-1, d)$. Finally, set $move(c,d)$ to be 0.

Then, from $g = p$ to $j-1$, all jobs of $J(MI_g, i)$ will be tried to fill in a total amount up to at most $\min\{fill_j, move(g, g+1), move(g+1, g+2), \ldots, move(j-1, j)\}$ in the idle times in $IDLE(j,x)$, $i+1 \leq x \leq k-1$, on the machines of Y_j. Each job is filled in exactly the same manner as in the idle times in $IDLE(1,x)$, $i+1 \leq x \leq k-1$. Let ρ_g be the total processing times among all jobs of $J(MI_g, i)$ that is filled in. The sequence $move(g, g+1), move(g+1, g+2), \ldots, move(j-1, j)$ will be updated by reducing ρ_g from them, and the array $T(\cdot,\cdot)$ and $IDLE(j,x)$, $i+1 \leq x \leq k-1$ are updated. This procedure will be stopped when either $fill_j = 0$ or all jobs of $J(MI_p, i)$, $J(MI_{p+1}, i), \ldots,$

and $J(MI_{j-1}, i)$ have been tried. In the latter case, all jobs of $J(MI_j, i)$ will be tried to fill in an amount equal to $fill_j$ in the idle times in $IDLE(j, x)$, $i + 1 \leq x \leq k - 1$, on the machines of Y_j, and the array $T(\cdot, \cdot)$ and $IDLE(j, x)$ are updated. (Again, the jobs are filled in exactly the same manner as in the idle times in $IDLE(1, x)$, $i + 1 \leq x \leq k - 1$.) It will be shown later in Lemma 6.9 that one can always schedule jobs of $J(MI_j, i)$ with total processing times $fill_j$ in the idle times in $IDLE(j, x)$, $i + 1 \leq x \leq k - 1$, without producing any overlap. After all the machine intervals Y_j with $fill_j > 0$ are considered by the above procedure, every machine interval Y_j must have $fill_j = 0$. Then the remaining jobs $T(l, i)$, $1 \leq l \leq n$ are scheduled, in the time segment TS_i by McNaughton's rule. From the schedule, $IDLE(j, i)$, $1 \leq j \leq z$ is updated. The process described above is shown in the algorithm *Schedule-Inclusive-Intervals-with-Release-Time.*

The procedure is repeated for all the time segments. After all the time segments are finished, $T(l, i)$ is computed for any job l and any $0 \leq i \leq k - 1$. Then a feasible schedule can be obtained by McNaughton's rule, segment by segment.

Lemma 6.9 *Suppose the jobs released at time r_i passed the test of Algorithm* Compute-Idle-and-Test-Feasibility. *For each machine interval Y_j, one can fill in exactly $fill_j$ amount of processing time in the idle times in $IDLE(j, x)$ for $i + 1 \leq x \leq k - 1$ by algorithm Schedule-Inclusive-Intervals-with-Release-Time, without any overlap.*

Proof: First, it is easy to see that no overlap can be produced by algorithm Schedule-Inclusive-Intervals-with-Release-Time and for each machine interval Y_j, no more than $fill_j$ amount of processing time is filled in the idle times in $IDLE(j, x)$ for $i + 1 \leq x \leq k - 1$. Now one can prove that for each machine interval Y_j, one can always fill in at least $fill_j$ amount of processing time in the idle times in $IDLE(j, x)$ for $i + 1 \leq x \leq k - 1$. For any set J of jobs, let $\rho(J)$ denote the total processing times of all the jobs in J. It will be proven, by induction on j, the following claim.

Claim 6.10 *For any machine interval Y_j, $1 \leq j \leq z$, with $extra_j \leq \alpha_j$, after algorithm* Compute-Idle-and-Test-Feasibility *is called, the following equation always holds:*

$$\rho(J(MI_j, i)) + \sum_{x=1}^{j-1} move(x, j) - \sum_{x=j+1}^{z} move(j, x) = |Y_j| * (r_{i+1} - r_i) + fill_j.$$

Recall that α_j is the total idle times in $IDLE(j, x)$ for $i + 1 \leq x \leq k - 1$ before algorithm *Compute-Idle-and-Test-Feasibility* is called in this iteration, and $extra_j$ is the total idle times in $IDLE(j, x)$ for $i \leq x \leq k - 1$ after algorithm *Compute-Idle-and-Test-Feasibility* is called in this iteration. So the claim means that after algorithm *Compute-Idle-and-Test-Feasibility* is called, for any machine interval Y_j in which at least $|Y_j| * (r_{i+1} - r_i)$ of processing times can be filled, the total processing times of $J(MI_j, i)$ plus the total processing times that needs to be moved to Y_j minus the total processing times that needs to be moved out of Y_j is exactly equal to the total idle times in Y_j in the time segment TS_i plus $fill_j$.

The claim can be proven by induction on j. For the base case $j = 1$, if $fill_1 = 0$, the claim clearly holds. On the other hand, if $fill_1 > 0$, then

$$\rho(J(MI_1, i)) - \sum_{x=2}^{z} move(1, x) = |Y_1| * (r_{i+1} - r_i) + fill_1.$$

It can be shown that one can always schedule jobs of $J(MI_1, i)$ with total processing times $fill_1$ in the idle times in $IDLE(1, x)$ for $i+1 \leq x \leq k-1$ on the machines of Y_1. Suppose not. Then there must be a time segment b, $i + 1 \leq b \leq k - 1$), such that some idle time of $IDLE(1, b)$ can not be filled. Since $\rho(J(MI_1, i)) \geq |Y_1| * (r_{i+1} - r_i)$, then at least $|Y_1|$ jobs are in $J(MI_1, i)$. Since all these jobs can not be filled in $IDLE(1, b)$, they must have been scheduled in the time segment TS_b for a duration equal to $r_{b+1} - r_b$. But this means that there is no idle time on the machines of Y_1, contradicting the fact that $IDLE(1, b) > 0$.

After $fill_1$ amount of processing times is filled in the idle times in $IDLE(1, x)$, $i+1 \leq x \leq k-1$, $J(MI_1, i)$ is updated by removing the scheduled jobs or parts of the jobs, and $fill_1$ is updated to 0. So $\rho(J(MI_1, i)) - \sum_{x=2}^{z} move(1, x) = |Y_1| * (r_{i+1} - r_i) + fill_1$ still holds.

Assume that the claim holds for the first $j-1$ machine intervals. Next one can prove that it also holds for the j-th interval. If $fill_j = 0$, the claim obviously holds. If $fill_j > 0$, then $extra_j < \alpha_j$. So $\rho(J(MI_j, i)) + \sum_{x=1}^{j-1} move(x, j) - \sum_{x=j+1}^{z} move(j, x) = |Y_j| * (r_{i+1} - r_i) + fill_j$. Let p be the maximum index such that no processing time need to be moved from Y_a to Y_b for any $1 \leq a \leq p-1$ and $p \leq b \leq j$. Then for each pair of indexes c and d, $p \leq c < d \leq j$, such that $move(c, d) > 0$, it is obavious that for all $c \leq u \leq d-1$, $extra_u = 0$ and $\rho(J(MI_u, i)) + \sum_{x=1}^{u-1} move(x, u) - \sum_{x=u+1}^{z} move(u, x) = |Y_u| * (r_{i+1} - r_i) + fill_u$. So, after all such pairs of c and d, $p \leq c < d \leq j$, such that $move(c, y) > 0$ are considered, for all $p \leq u \leq j-1$, the following equations hold: $\rho(J(MI_u, i)) + \sum_{x=1}^{u-1} move(x, u) - \sum_{x=u+1}^{z} move(u, x) = |Y_u| * (r_{i+1} - r_i) + fill_u$.

For each pair of indexes c and d, $p \leq c < d-1 \leq u-1$, such that $move(c, d) > 0$, $move(c, d)$ is added to each of the items $move(c, c+1), move(c+1, c+2), \ldots, move(d-1, d)$, and set $move(c, d) = 0$. So, after all such pairs are considered, for all $p \leq u < j$, $move(u, u+1) > 0$ and $\rho(J(MI_u, i)) + move(u-1, u) - \sum_{x=u+1}^{z} move(u, x) = |Y_u| * (r_{i+1} - r_i) + fill_u$.

In the algorithm, from $g = p$ to $j-1$, all jobs of $J(MI_g, i)$ are tried to schedule up to $\min\{fill_j, move(g, g+1), move(g+1, g+2), \ldots, move(j-1, j)\}$ amount of processing time in the idle times in $IDLE(j, x)$, $i+1 \leq x \leq k-1$, on the machines of Y_j. Let ρ be the total processing time that is filled. the sequence $move(g, g+1), move(g+1, g+2), \ldots, move[j-1][j]$ is updated by reducing ρ from each of them. Furthermore, $fill_j$ is also updated in the course of scheduling the jobs. Finally, $J(MI_g, i)$ is updated by removing all the scheduled jobs or parts of the jobs. Since both $move(j-1, j)$ and $fill_j$ are reduced by ρ and other items are not changed, the following equation still holds: $\rho(J(MI_j, i)) + move(j-1, j) - \sum_{x=j+1}^{z} move(j, x) = |Y_j| * (r_{i+1} - r_i) + fill_j$. Similarly, since both $move(g, g+1)$ and $\rho(J(MI_x, i))$ are educed by ρ and other items are not changed, the following equation still holds: $\rho(J(MI_x, i)) + move(g-1, g) - \sum_{y=x+1}^{z} move(g, y) = |Y_g| * (r_{i+1} - r_i) + fill_g$. Notice that $fill_g = 0$ now. There is no change in other machine intervals, so all the equations still hold.

The procedure is stopped when either $fill_j = 0$ or all jobs of $J(MI_p, i)$, $J(MI_{p+1}, i)$, ..., and $J(MI_{j-1}, i)$ have been tried. If, after all jobs of $J(MI_p, i)$, $J(MI_{p+1}, i)$, ..., and $J(MI_{j-1}, i)$ have been tried, $fill_j$ is still greater than 0, then all jobs of $J(MI_j, i)$ are tried to schedule $fill_j$ amount of processing time in the idle times in $IDLE(j, x)$ for $i + 1 \leq x \leq k - 1$ on the machines of Y_j, and $J(MI_j, i)$ is updated by removing all the scheduled jobs or parts of the jobs. Since both $\rho(J(MI_j, i))$ and $fill_j$ are reduced by the same amount and the other items are not changed, the following equation still holds: $\rho(J(MI_j, i)) + move(j-1, j) - \sum_{x=j+1}^{z} move(j, x) = |Y_j| * (r_{i+1} - r_i) + fill_j$.

Now it can be shown that after all the jobs of $J(MI_j, i)$ have been tried, $fill_j$ must be 0. Suppose not. Let w be the last machine index such that $move(w, w+1) = 0$. That is, from $w+1$ on, each time one try to fill $\min\{fill_j, move(x, x+1), \ldots, move(j-1, j)\}$, $x > w$, of processing time in the idle times in $IDLE(j, x)$, $i+1 \leq x \leq k-1$, on the machines of Y_j, it is only possible to fill only $\delta < \min\{fill_j, move(x, x+1), \ldots, move(j-1, j)\}$ of processing time in the idle times in $IDLE(j, x)$, $i + 1 \leq x \leq k - 1$, on the machines of Y_j. That means all the jobs in $J(MI_x, i)$, $w + 1 \leq x \leq j$ are tried, to fill in the idle times in $IDLE(j, x)$, $i + 1 \leq x \leq k - 1$. Since $fill_j > 0$, there must be a time segment b, $i + 1 \leq b \leq k - 1)$, such that some idle time in $IDLE(j, b)$ can not be filled. Since for each $w + 1 \leq u \leq j$, then

$$\rho(J(MI_u, i)) + move(u-1, u) - \sum_{x=u+1}^{z} move(u, x) = |Y_u| * (r_{i+1} - r_i) + fill_u$$

and

$$move(w, w+1) = 0,$$

then,

$$\rho(J(MI_{w+1}, i)) - \sum_{x=w+2}^{z} move(w+1, x) = |Y_{w+1}| * (r_{i+1} - r_i) + fill_{w+1}$$

and

$$\rho(J(MI_{w+1},i)) + \rho(J(MI_{w+2},i)) + \ldots + \rho(J(MI_j,i)) +$$
$$move(w+1,w+2) + move(w+2,w+3) + \ldots + move(j-1,j) -$$
$$(\sum_{x=w+2}^{z} move(w+1,x) + \sum_{x=w+3}^{z} move(w+2,x) + \cdots + \sum_{x=j+1}^{z} move(j,x)) \geq$$
$$(|Y_{w+1}| + |Y_{w+2}| + \ldots + |Y_j|) * (r_{i+1} - r_i) + fill_{w+1} + \ldots + fill_j.$$

Notice that $fill_u = 0$ for all $w+1 \leq u \leq j-1$. Thus, $\rho(J(MI_{w+1},i)) + \rho(J(MI_{w+2},i)) + \ldots + \rho(J(MI_j,i)) \geq (|Y_{w+1}| + |Y_{w+2}| + \ldots + |Y_j|) * (r_{i+1} - r_i) + fill_j$.

Since each of these jobs has length at most $(r_{i+1} - r_i)$, there must be at least $|Y_{w+1}| + |Y_{w+2}| + \ldots + |Y_j|$ jobs in $J(MI_{w+1},i)$, $J(MI_{w+2},i)$, ..., and $J(MI_j,i)$. Since all these jobs can not be filled in the idle times in $IDLE(j,b)$, these jobs must have been scheduled in the time segment TS_b for a duration equal to $(r_{b+1} - r_b)$. Therefore, there should be no idle time in time segment TS_b on the machines of Y_j, contradicting the fact that $IDLE(j,b) > 0$.

So, for machine interval Y_j, the claim still holds and exactly $fill_j$ of processing time can be filled in the idle times in $IDLE(j,x)$, $i+1 \leq x \leq k-1$, on the machines of Y_j.

□

Algorithm 6.1 constructs the schedule for one time segment. Steps 1 to 6 take $O(n+m)$ times. Step 9 takes $O(m)$ time for each machine interval MI_j. Since there are $O(m)$ machine intervals, Step 9 takes a total of $O(m^2)$ time. Steps 10 to 13 take $O(m^3)$ time since there are $O(m^2)$ pairs of c and d. Steps 15 to 22 take $O(mnk)$ time. Therefore, the total time needed for one time segment is $O(mnk+m^3)$. Hence the total time for k time segment is $O(mnk^2 + m^3k)$. After the array $T(\cdot,\cdot)$ is updated, one can obtain a schedule by McNaughton's rule which takes $O(nk)$ time.

Theorem 6.11 *For the inclusive processing set case, one can obtain an optimal schedule in time* $O(nk \log P + mnk^2 + m^3k)$.

6.5 Conclusion

In this paper first an efficient algorithm for constructing a minimal length preemptive schedule for nested processing set (and hence inclusive processing set) restrictions is given. An efficient algorithm for generating maximal schedules is also given. When jobs have different release times, a network flow approach can be used to construct an optimal schedule. A more efficient algorithm is given for the inclusive processing set restriction. The network flow approach also works for arbitrary processing set restriction. For future research, it will be interesting to see if there are more efficient algorithms than the ones presented in this paper.

Online scheduling algorithm is considered as well. Since there is no optimal online algorithm, the best one can hope for is an approximation algorithm. In this regard, the maximal algorithm presented in Section 6.3 have been considered as an approximation algorithm. Whenever new jobs arrive, the unfinished portions of the jobs along with the new jobs are reschedule. What is the competitive ratio of this algorithm? It is possible to show that the competitive ratio lies between 3/2 and 2. For future research, it will be interesting to determine the exact value of the competitive ratio of this algorithm.

```
Schedule-Inclusive-Intervals-with-Release-Time Input: T(l, y) and
        IDLE(j, y) 1 ≤ l ≤ n, i + 1 ≤ y ≤ k − 1 and 1 ≤ j ≤ z
Output: T(l, i): the amount of processing of job J_l done in the time
        segment TS_i
IDLE(j, i) = |Y_j| · (r_{i+1} − r_i) for 1 ≤ j ≤ z;
α_j = Σ_{x=i+1}^{k−1} IDLE(j, x) for 1 ≤ j ≤ z;
Call algorithm Compute-Idle-and-Test-Feasibility to compute move(·, ·)
and extra_j for all 1 ≤ j ≤ z;
For each 1 ≤ j ≤ z, fill_j = max{α_j − extra_j, 0};
Let J(MI_j, i) be the set of jobs with machine interval MI_j released at r_i,
1 ≤ j ≤ z;
j = 1;
repeat
    if fill_j > 0 then
        Let p be the maximum index such that move(a, b) = 0 for all
        1 ≤ a ≤ p − 1 and p ≤ b ≤ j;
        foreach pair of c and d such that p ≤ c < d − 1 < j and
        move(c, d) > 0 do
            for x = c to d − 1 do
                move(x, x + 1) = move(x, x + 1) + move(c, d)
            move(c, d) = 0;
        g = p;
        while fill_j > 0 and g < j do
            For each job l of J(MI_g, i), fill in an amount up to min{
            move(g, g + 1), ..., move(j − 1, j), fill_j } in the idle times
            in IDLE(j, x), i + 1 ≤ x ≤ k − 1. Update fill_j, T(·, ·), and
            IDLE(j, x), i + 1 ≤ x ≤ k − 1;
            Let ρ be the total processing times filled in;
            for x = g to j − 1 do move(x, x + 1) = move(x, x + 1) − ρ;
            g = g + 1;
        if fill_j > 0 then
            For each job l of J(MI_j, i), fill in an amount up to fill_j in the
            idle times in IDLE(j, x) for i + 1 ≤ x ≤ k − 1. Update
            fill_j, T(l, i), T(l, x) and IDLE(j, x);

    j = j + 1;
until j > z;
Schedule the remaining jobs of J(MI_j, i) in the time segment TS_i by
McNaughton's rule and update IDLE(j, i) for 1 ≤ j ≤ z.
```

Figure 6.3 Fast algorithm for inclusive restrictions.

CHAPTER 7

CONCLUSIONS

This dissertation studied combinatorial optimization problems arising from two areas: radio network and scheduling. The main focus was to design efficient algorithms for some combinatorial problems.

7.1 Communication in Random Geometric Radio Networks

The first model studied in this dissertation is a uni-disk model, the main model for wireless communication. It has been shown in this dissertation that when network is sparse: (1)A simple randomized algorithm can be used to do gossiping in $O(D)$ steps, without any geometric position knowledge. (2)A deterministic algorithm can be used to do gossiping in $O(D)$ steps if every node know its own geometric position. (3)A deterministic algorithm can be used to do gossiping in $O(D)$ steps if the distance or angle between two nodes can be determined by checking the incoming signal. Since D is a trivial lower bound for doing any global communication in the network, all these algorithms are asymptoticly optimal. These results will appear in [Czumaj and Wang 2007b].

The second model studied in this dissertation is the random line-of-sight radio network. In this dissertation, it has been shown when network is sparse: (1)A deterministic algorithm can be used to do gossiping in $O(D)$ steps, if every node knows its own geometric position. (2)A deterministic algorithm can be used to do broadcasting in $O(D)$ steps without having any geometric knowledge. (3)A randomized algorithm can be used to do gossiping in $O(D)$ steps without having any geometric knowledge. Again, the running time of all these algorithms is asymptoticly optimal. These results have been published in [Czumaj and Wang 2007a].

There are several questions that have not been answered. In this dissertation, in both models, the networks are assumed to be sparse. When network becomes dense, the

running time of these algorithms is significantly increased. It will be interesting to design faster algorithms for dense networks.

7.2 Scheduling Problems

In scheduling area, the first model studied in this dissertation is the online scheduling of equal processing time task systems. It has been shown that Algorithm Delay-A has an asymptotic competitive ratio of 3/2 for $P \mid p_j = p, intree_i\ released\ at\ time\ r_i \mid C_{\max}$, and an asymptotic competitive ratio of 1 for $P \mid p_j = p, outtree_i\ released\ at\ time\ r_i \mid C_{\max}$. Furthermore, it is shown that Algorithm Delay-B has an asymptotic competitive ratio of 1 for $P2 \mid p_j = p, prec_i\ released\ at\ time\ r_i \mid C_{\max}$. All this results have been submitted for review [Huo et al. 2007b].

There are several open problems for this model. For the problem $P \mid p_j = p, prec_i\ released\ at\ time\ r_i \mid C_{\max}$, the asymptotic competitive ratio of Algorithm Delay-B has not been determined. On the other hand, it is known that the ratio is at least $2 - \frac{2}{m}$, since Lam and Sethi [Lam and Sethi 1977] have shown that Coffman-Graham algorithm is a $(2 - \frac{2}{m})$-approximation algorithm for $P \mid p_j = 1, prec \mid C_{\max}$. For future research, it will be interesting to determine this value. Furthermore, it will be interesting to see if there are other classes of precedence constraints and processing times that yield an asymptotic competitive ratio less than 2. Since for $P \mid pmtn, intree \mid C_{\max}$, Muntz-Coffman algorithm gives an optimal solution, one can consider an online version of Muntz-Coffman algorithm for $P \mid pmtn,\ intree_i\ released\ at\ time\ r_i \mid C_{\max}$. It will be interesting to know the competitive ratio of this algorithm.

The second model studied in this dissertation is a scheduling model integrated production and delivery. Each job is supposed to be completed within a specified time window and the time windows are disjoint. Three kinds of profits are considered: (1) arbitrary profit, (2) equal profit, and (3) profit proportional to its processing time. In the first case, a pseudo-polynomial time algorithm is given to find an optimal schedule for a single machine. Based on the pseudo-polynomial time algorithm, A FPTAS with running time $O(\frac{n^3}{\epsilon})$ is developed. In the second case, an $O(n \log n)$-time algorithm is given to

find an optimal schedule for a single machine. In the third case, an improved FPTAS is given with running time $O(\frac{n^2}{\epsilon})$. All algorithms can be extended to parallel and identical machines with a certain degradation of performance bounds. For the equal profit case, a $\frac{7}{5}$-approximation for the special case where there is a single time frame and $m \geq 2$ identical and parallel machines is given. All these results have been submitted for review [Huo et al. 2007a].

The model can be extended in several direction. For example, a job not scheduled incurs a small penalty; the delivery capacity is limited (at most c jobs can be delivered every time); or the time windows are not disjoint.

The third model studied in this dissertation is preemptive scheduling with nested and inclusive processing set restrictions. Four algorithms are given in this dissertation: (1)An efficient algorithm for constructing a minimal length preemptive schedule for nested processing set (and hence inclusive processing set) restrictions. (2)An efficient algorithm for generating maximal schedules. (3) A network flow approach to construct an optimal schedule when jobs have different release times. (4)A more efficient algorithm for the inclusive processing set restriction when jobs have different release times. All these results have been submitted for review [Huo et al. 2007c].

For future research, it will be interesting to see if there are more efficient algorithms than the ones presented in this paper. Online scheduling algorithms are another interesting topic. Since there is no optimal online algorithm, the best one can hope for is an approximation algorithm. Especially, can the maximal algorithm presented in Chapter 6 be adapted to produce good competitive ratio?

REFERENCES

ALON, N., BAR-NOY, A., LINIAL, N., AND PELEG, D. 1991. A lower bound for radio broadcast. *Journal of Computer and System Sciences 43,* 2, 290-298.

BAR-NOY, A., GUHA, S., NAOR, J., AND SCHIEBER, B. 2001. Approximating the throughput of multiple machines in real-time scheduling. *SIAM Journal on Computing 31,* 2, 331-352.

BAR-YEHUDA, R., GOLDREICH, O., AND ITAI, A. 1992. On the time-complexity of broadcast in multi-hop radio networks: an exponential gap between determinism and randomization. *Journal of Computer and System Sciences 45,* 1, 104-126.

BILGEN, B. AND OZKARAHAN, I. 2004. Strategic tactical and operational production-distribution models: A review. *International Journal of Technology Management 28,* 2, 151-171.

BOLLOBAS, B. 2001. *Random graphs.* Cambridge University Press, London, England.

BONIS, A. D., GASIENIEC, L., AND VACCARO, U. 2003. Generalized framework for selectors with applications in optimal group testing. In *Proceedings of the 30th International Colloquium on Automata, Languages and Programming.* 81-96.

CAPKUN, S., HAMDI, M., AND HUBAUX, J.-P. 2002. Gps-free positioning in mobile ad hoc networks. *Cluster Computing 5,* 2, 157-167.

CHEN, B., POTTS, C., AND WOEGINGER, G. 1998. *A review of machine scheduling: Complexity, algorithms and approximability, Handbook of Combinatorial Optimization.* Kluwer, Boston, Chapter 17, 21-169.

CHEN, Z.-L. 2004. *Modeling in the E-Business Era, Handbook of Quantitative Supply Chain Analysis.* Springer, Berlin, Chapter 17, 711-735.

CHEN, Z.-L. 2006. Integrated production and outbound distribution scheduling in a supply chain: Review and extensions, working paper.

CHLEBUS, B. 2001. *Randomized communication in radio networks, Handbook on Randomized Computing.* Kluwer Academic Publishers, Norwell, MA, USA, Chapter 11, 401-445.

CHLEBUS, B. S., GASIENIEC, L., GIBBONS, A., PELC, A., AND RYTTER, W. 2002. Deterministic broadcasting in ad hoc radio networks. *Distributed Computing 15,* 1, 27-38.

CHLEBUS, B. S., KOWALSKI, D. R., AND ROKICKI, M. A. 2006. Average-time complexity of gossiping in radio networks. In *Proceedings of the 13th Colloquium on Structural Information and Communication Complexity.* 253-267.

CHROBAK, M., GASIENIEC, L., AND RYTTER, W. 2002. Fast broadcasting and gossiping in radio networks. *Journal of Algorithms 43,* 2, 177-189.

CLEMENTI, A. E. F., MONTI, A., AND SILVESTRI, R. 2003. Distributed broadcast in radio networks of unknown topology. *Theoretical Computer Science 302,* 1-3, 337-364.

COFFMAN, E. G., LEUNG, J. Y.-T., AND TING, D. W. 1978. Bin packing: Maximizing the number of pieces packed. *Acta Informatica 9*, 263-271.

CZUMAJ, A. AND RYTTER, W. 2006. Broadcasting algorithms in radio networks with unknown topology. *Journal of Algorithms 60,* 2, 115-143.

CZUMAJ, A. AND WANG, X. 2007a. Communication problems in random line-of-sight ad-hoc radio networks. In *The 4th Symposium on Stochastic Algorithms, Foundations, and Applications.* 70-81.

CZUMAJ, A. AND WANG, X. 2007b. Fast message dissemination in random geometric ad-hoc radio networks. In *The 18th International Symposium on Algorithms and Computation, to appear.*

DESSMARK, A. AND PELC, A. 2007. Broadcasting in geometric radio networks. *Journal of Discrete Algorithms 5,* 1, 187-201.

DOHERTY, L., PISTER, K. S. J., AND GHAOUI, L. E. 2001. Convex optimization methods for sensor node position estimation. In *Proceedings of the 20th Annual Joint Conference of the IEEE Computer and Communications Societies.* 1655-1663.

ELSÄSSER, R. AND GASIENIEC, L. 2006. Radio communication in random graphs. *Journal of Computer and System Sciences 72,* 3, 490-506.

ERENGUC, S. S., SIMPSON, N. C., , AND VAKHARIA, A. J. 1999. Integrated production/distribution planning in supply chains: An invited review. *European Journal of Operational Research 115,* 2, 219-236.

FRIEZE, A., KLEINBERG, J. M., RAVI, R., AND DEBANY, W. 2007. Line-of-sight networks. In *Proceedings of the 18th ACM-SIAM Symposium on Discrete Algorithms.* 968-977.

GARCIA, J. M. AND LOZANO, S. 2005. Production and delivery scheduling problem with time windows. *Computers and Industrial Engineering 48,* 4, 733-742.

GARCIA, J. M., LOZANO, S., AND CANCA, D. 2004. Coordinated scheduling of production and delivery from multiple plants. *Robotics and Computer-Integrated Manufacturing. 20,* 3, 191-198.

GAREY, M. R. AND JOHNSON, D. S. 1979. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman.

GASIENIEC, L., PELEG, D., AND XIN, Q. 2005. Faster communication in known topology radio networks. In *Proceedings of the 24 Annual ACM Symposium on Principles of Distributed Computing.* 129-137.

GASIENIEC, L., RADZIK, T., AND XIN, Q. 2004. Faster deterministic gossiping in directed ad hoc aadio networks. In *Proceedings of the 9th Scandinavian Workshop on Algorithm Theory*. 397-407.

GIORDANO, S. AND STOJMENOVIC, I. 2003. *Position-based ad hoc routes in ad hoc networks, The Handbook of ad hoc Wireless Networks*. CRC Press, Inc., Boca Raton, FL, USA, Chapter 6, 1-14.

GLASS, C. A. AND KELLERER, H. 2007. Parallel machine scheduling with job assignment restrictions. *Naval Research Logistics 54*, 250-257.

GLASS, C. A. AND MILLS, H. R. 2006. Scheduling unit length jobs with parallel nested machine processing set restrictions. *Comput Oper Res 33*, 620-638.

GOEL, A., RAI, S., AND KRISHNAMACHARI, B. 2004. Sharp thresholds for monotone properties in random geometric graphs. In *Proceedings of the 36 annual ACM symposium on Theory of computing*. 580-586.

GOETSCHALCKX, M. L., VIDAL, C. J., AND DOGAN, K. 2002. Modeling and design of global logistics systems: A review of integrated strategic and tactical models and design algorithms. *European Journal of Operational Research 143,* 1, 1-18.

GOLDBERG, A. V. AND TARJAN, R. E. 1988. A new approach to the maximum flow problem. *Journal of ACM 35*, 921-940.

GONZALEZ, T. AND SAHNI, S. 1976. Open shop scheduling to minimize finish time. *Journal of ACM 23*, 665-679.

GUPTA, P. AND KUMAR, P. 1998. *Stochastic Analysis, Control, Optimization and Applications*. Birkhauser publisher, Boston, 547-566.

HOCHBAUM, D. S. AND SHMOYS, D. B. 1987. Using dual approximation algorithms for scheduling problems: practical and theoretical results. *Journal of ACM 34*, 144-162.

HONG, K. S. AND LEUNG, J. Y.-T. 1992. On-line scheduling of real-time tasks. *IEEE Transactions on Computers 41*, 1326-1331.

HUO, Y. AND LEUNG, J. Y.-T. 2005. Online scheduling of precedence constrained tasks. *SIAM Journal on Computing 34,* 3, 743-762.

HUO, Y., LEUNG, J. Y.-T., AND WANG, X. 2007a. Integrated production and delivery scheduling with disjoint windows, working paper.

HUO, Y., LEUNG, J. Y.-T., AND WANG, X. 2007b. Online scheduling of equal-processing-time task systems, working paper.

HUO, Y., LEUNG, J. Y.-T., AND WANG, X. 2007c. Preemptive scheduling algorithms with nested and inclusive processing set restrictions, working paper.

HWANG, H.-C., CHANG, S. Y., AND LEE, K. 2004. Parallel machine scheduling under a grade of service provisions. *Computers and Operations Research 31*, 2055-2061.

KOWALSKI, D. R. AND PELC, A. 2002. Deterministic broadcasting time in radio networks of unknown topology. In *Proceedings of the 43rd Symposium on Foundations of Computer Science*. 63-72.

KOWALSKI, D. R. AND PELC, A. 2005. Broadcasting in undirected ad hoc radio networks. *Distributed Computing 18,* 1, 43-57.

KOWALSKI, D. R. AND PELC, A. 2007. Optimal deterministic broadcasting in known topology radio networks. *Distributed Computing 19,* 3, 185-195.

KUSHILEVITZ, E. AND MANSOUR, Y. 1998. An $\omega(d \log(n/d))$ lower bound for broadcast in radio networks. *SIAM Journal on Computing 27,* 3, 702-712.

LAM, S. AND SETHI, R. 1977. Worst case analysis of two scheduling algorithms. *SIAM Journal on Computing 6,* 3, 518-536.

LAWLER, E. L. AND LABETOULLE, J. 1978. On preemptive scheduleing of unrelated parallel processors by linear programming. *Journal of ACM 25*, 612-619.

LENSTRA, J. K., SHMOYS, D., AND TARDOS, E. 1990. *Math. Program 46*, 259-271.

LI, X., SHI, H., AND SHANG, Y. 2004. A partial-range-aware localization algorithm for ad-hoc wireless sensor networks. In *Proceedings of the 29th Annual IEEE International Conference on Local Computer Networks*. 77-83.

MCNAUGHTON, R. 1959. Scheduling with deadlines and loss functions. *Management Science 6*, 1-12.

NASIPURI, A., LI, K., AND SAPPIDI, U. 2002. Power consumption and throughput in mobile ad hoc networks using directional antennas. In *Proceedings of the 11th IEEE International Conference on Computer Communication and Networks*. 125-137.

OU, J., LEUNG, J. Y.-T., AND LI, C.-L. 2007. Scheduling parallel machines with inclusive processing set restrictions, working paper.

PENROSE, M. D. 1997. The longest edge of the random minimal spanning tree. *Annals of Applied Probability 7,* 2, 340-361.

PENROSE, M. D. 2003. *Random Geometric Graphs*. Oxford University Press, UK.

SARMIENTO, A. M. AND NAGI, R. 1999. A review of integrated analysis of production-distribution systems. *IIE Transactions 31,* 14, 1061-1074.

SGALL, J. 1998. Online algorithms: The state of the art. In *Developments from a June 1996 seminar on Online algorithms*, A. Fiat and G. J. Woeginger, Eds. Springer-Verlag, London, UK.

THOMAS, D. J. AND GRIFFIN, P. M. 1996. Coordinated supply chain management. *European Journal of Operational Research 94,* 1, 1-15.