ABSTRACT

ALGORITHMS AND COMPLEXITY ANALYSES FOR SOME COMBINATORIAL OPTIMIZATION PROBLEMS

by
Hairong Zhao

The main focus of this dissertation is on classical combinatorial optimization problems in two important areas: scheduling and network design.

In the area of scheduling, the main interest is in problems in the master-slave model. In this model, each machine is either a master machine or a slave machine. Each job is associated with a preprocessing task, a slave task and a postprocessing task that must be executed in this order. Each slave task has a dedicated slave machine. All the preprocessing and postprocessing tasks share a single master machine or the same set of master machines. A job may also have an arbitrary release time before which the preprocessing task is not available to be processed. The main objective in this dissertation is to minimize the total completion time or the makespan. Both the complexity and algorithmic issues of these problems are considered. It is shown that the problem of minimizing the total completion time is strongly NP-hard even under severe constraints. Various efficient algorithms are designed to minimize the total completion time under various scenarios.

In the area of network design, the survivable network design problems are studied first. The input for this problem is an undirected graph $G = (V, E)$, a non-negative cost for each edge, and a nonnegative connectivity requirement r_{uv} for every (unordered) pair of vertices u, v. The goal is to find a minimum-cost subgraph in which each pair of vertices u, v is joined by at least r_{uv} edge (vertex)-disjoint paths. A Polynomial Time Approximation Scheme (PTAS) is designed for the problem when the graph is Euclidean and the connectivity requirement of any point is at most 2. PTASs or Quasi-PTASs are also designed for 2-edge-connectivity problem and biconnectivity problem and their variations in unweighted or weighted planar graphs.

Next, the problem of constructing geometric fault-tolerant spanners with low cost and bounded maximum degree is considered. The first result shows that there is a greedy algorithm which constructs fault-tolerant spanners having asymptotically optimal bounds for both the maximum degree and the total cost at the same time. Then an efficient algorithm is developed which finds fault-tolerant spanners with asymptotically optimal bound for the maximum degree and almost optimal bound for the total cost.

ALGORITHMS AND COMPLEXITY ANALYSES FOR SOME COMBINATORIAL OPTIMIZATION PROBLEMS

by
Hairong Zhao

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 Sciences

Department of Computer Science

May 2005

APPROVAL PAGE

ALGORITHMS AND COMPLEXITY ANALYSES FOR SOME COMBINATORIAL OPTIMIZATION PROBLEMS

Hairong Zhao

Dr. Joseph Leung, Dissertation Co-Advisor — Date
Distinguished Professor of Computer Science, New Jersey Institute of Technology

Dr. Artur Czumaj, Dissertation Co-Advisor — Date
Associate Professor of Computer Science, New Jersey Institute of Technology

Dr. Teunis J. Ott, Committee Member — Date
Professor of Computer Science, New Jersey Institute of Technology

Dr. Wojciech Rytter, Committee Member — Date
Professor of Computer Science, New Jersey Institute of Technology

Dr. Clifford Stein, Committee Member — Date
Professor of Industrial Engineering and Operations Research, Columbia University

BIOGRAPHICAL SKETCH

Author: Hairong Zhao

Degree: Doctor of Philosophy

Date: May 2005

Undergraduate and Graduate Education:

- Doctor of Philosophy in Computer Science,
  New Jersey Institute of Technology, Newark, NJ, 2005
- Master of Computer Science,
  Beijing University of Posts & Telecommunications, Beijing, China, 1997
- Bachelor of Computer Science,
  Taiyuan University of Technology, Shanxi, China, 1994

Major: Computer Science

Presentations and Publications:

J. Y-T. Leung and H. Zhao, "Minimizing Mean Flowtime and Makespan on Master-Slave Systems," *Journal of Parallel and Distributed Computing,* accepted for publication.

J. Y-T. Leung and H. Zhao, "Minimizing Mean Flowtime on Master-Slave Machines," *Proceedings of the 2004 International Conference on Parallel and Distributed Processing Techniques and Applications*, vol. 2, pp. 939-945, 2004.

A. Czumaj, M. Grigni, P. A. Sissokho, and H. Zhao "Approximation Schemes for Minimum 2-Edge-Connected and Biconnected Subgraphs in Planar Graphs," *Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 489 - 498, 2004.

A. Czumaj and H. Zhao "Fault-Tolerant Geometric Spanners," *Discrete and Computational Geometry*, Vol. 32, pp. 207-230, 2004.

A. Czumaj and H. Zhao "Fault-Tolerant Geometric Spanners," *Proceedings of the 19th ACM Symposium on Computational Geometry*, pp. 1-10, 2003.

J. Y-T. Leung and H. Zhao "Real-Time Scheduling Analysis," *Final report to Federal Aviation Administration*, 2003.

A. Czumaj, A. Lingas, and H. Zhao " Polynomial-Time Approximation Schemes for the Euclidean Survivable Network Design Problem, " *Proc. of the 29th International Colloquium on Automata, Languages and Programming (ICALP'02)* , pp. 973-984, 2002.

A. Berger, A. Czumaj, M. Grigni and H. Zhao. "Approximate Minimum 2-Connected Subgraphs in Weighted Planar Graphs," *submitted.*

A. Czumaj, W. Rytter, X. Wang and H. Zhao, "A Linear-Time Algorithm for 3-Path Coloring of 2-Regular Digraphs," *submitted.*

A. Berger, M. Grigni and Hairong Zhao, "A Well-Connected Separator for Planar Graphs," *submitted.*

Y. Huo, J. Y-T. Leung and H. Zhao, "Complexity of Two Dual Criteria Scheduling Problems," *submitted.*

Y. Huo, J. Y-T. Leung and H. Zhao, "Bi-criteria Scheduling Problems: Number of Tardy Jobs and Maximum Weighted Tardiness," *submitted.*

J. Y-T. Leung and H. Zhao, "Minimizing Total Completion Time in Master-Slave Systems," *submitted.*

H. Zhao, "Survivable Network Design and Fault Tolerant Spanners," invited talk at *Los Alamos National Laboratory*, March, 2005.

H. Zhao, "Minimizing Mean Flowtime and Makespan on Master-Slave Systems," invited talk at *INFORMS Annual Meeting*, October 2004.

H. Zhao, "Fault Tolerant Spanners and Their Applications," *DIMACS/CS Light Seminar: Theoretical Computer Science*, DIMACS Center, Rutgers Universe, March, 2004.

H. Zhao, "Approximation Schemes for Minimum 2-Edge-Connected and Biconnected Subgraphs in Planar Graphs," presentation at *SODA 2004*, New Orleans, January, 2004.

H. Zhao, "Fault-Tolerant Geometric Spanners," *DIMACS Workshop on Computational Geometry*, DIMACS Center, Rutgers University, November, 2002.

This dissertation is dedicated to my 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 – Joseph Leung and Artur Czumaj. Without their support, patience and encouragement, this dissertation would not exist.

I sincerely thank 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. I am deeply indebted to 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.

My thanks also go to the members of my dissertation committee, Cliff Stein, Teunis Ott and Wojciech Rytter, for reading previous drafts of this dissertation and providing many valuable comments that improved the contents of this dissertation. I must also thank my coauthors, Artur Czumaj, Joseph Leung, Michelangelo Grigni, Wojciech Rytter, Andre Berger, Andrzej Lingas, Xin Wang, Yumei Huo and Papa Sissokho. It has been such a wonderful experience to work with each of them. Although I have not even had a chance to meet some of them, each has taught me a great deal about research and about writing research.

I am also grateful to my colleagues, Haibing Li, Yumei Huo and Xin Wang for numerous interesting and good-spirited discussions about research. The friendship of Jingxuan Liu, Binghu Zhang, Hong Zhao, Yayi Hu, Sen Zhang, Min Zhang, Chang Liu, Thoa Hoang, 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. I must give my thanks to my best friends in my life, Qingrui Ping, Li Gao, Xiangping Wei. They are more than

friends, they are part of my family. Though they are far away from me, their support is always with me. Their sincere care for me and my family is a treasure of my life. I thank Maryann McCoul for her trust in me and for her great advice when I needed it the most.

Last, I would like to thank my husband, Wenxin Mao, for his understanding and love during the past few years. His 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 brother and sister for their love and for taking care of my parents during my absence.

This work is supported in part by NSF Grant DMI-0300156 and by FAA Grant 01-C-AW-NJIT.

TABLE OF CONTENTS

TABLE OF CONTENTS
(Continued)

TABLE OF CONTENTS
(Continued)

Chapter Page

TABLE OF CONTENTS
(Continued)

Chapter **Page**

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

A combinatorial optimization problem is concerned with selecting, from among a finite set of possible solutions, the one that maximizes or minimizes a certain function, the so-called objective function. Such problems are of great importance because a large number of practical problems in various fields can be formulated as combinatorial optimization problems: for example, inventory control, the scheduling of lines in flexible manufacturing facilities, planning communication in traffic networks, finding the shortest or most reliable paths in traffic or communication networks, etc. Extensive surveys of related applications of combinatorial optimization are given in [49], [60].

Because of its importance, combinatorial optimization has attracted a great deal of research effort and has experienced a particularly fast development during the last few decades. Given any specific optimization problem of interest, the first question that arises is whether there exists an efficient (polynomial-time) algorithm. While some problems in this area are relatively well understood and have known efficient algorithms, many others are intractable, typically NP-hard, e.g. scheduling problems, partitioning problems. So under the widely believed assumption that $P \neq NP$, there is no hope of getting polynomial time algorithms for solving these problems.

However, very large instances of these problems frequently arise in practice. Thus, one is forced to look for algorithms that run in polynomial time and hopefully return a near-optimal solution. Such algorithms are called *approximation algorithms*. An approximation algorithm is called an α-approximation algorithm for a problem Π, if for any instance I of Π, it always returns a solution with value at most (at least for maximization problems) α times the optimal. The value α is called the *approximation ratio* or the *performance ratio* of the algorithm. Of course, one hopes that α is as close to 1 as possible. However, it turns out

that different NP-hard problems exhibit different approximability properties. Some problems, e.g. the knapsack problem, allow a *polynomial-time approximation scheme (PTAS)*; i.e., a family of algorithms $\{\mathcal{A}_\varepsilon\}$ such that, for each fixed $\varepsilon > 0$, $\{\mathcal{A}_\varepsilon\}$ runs in time polynomial in the size of the input and produces a $(1+\varepsilon)$-approximation. On the other hand, some other problems have intrinsic limitations to approximability. For example, there is no PTAS for the general traveling salesman problem unless $\mathrm{P} = \mathrm{NP}$.

This dissertation focuses on classical combinatorial optimization problems in two important areas: *scheduling* and *network design*. Not surprisingly, most of these problems have been or will be shown in this dissertation to be NP-hard. Thus, various constant approximation algorithms or approximation schemes are designed throughout the dissertation.

1.1 Machine Scheduling Problems

Scheduling is an intensively studied class of discrete optimization problems. Scheduling problems are motivated by the allocation of limited resources to jobs over time, subject to some constraints. It is a decision-making process with the goal of optimizing one or more objectives. The resources and jobs can take on many different forms. The resource can be machines in a workshop, runways at an airport, processing units in a computing environment. The jobs can be operations in a workshop, takeoffs and landings of air planes or computer programs. Standard scheduling requirements include: a job cannot be processed by two or more machines at a time, or a machine cannot process two or more jobs at the same time. Depending on the type of scheduling system, specific constraints should be satisfied. For example, jobs may have different release times and deadlines, different jobs may have different priorities, a job may not be allowed to preempt other jobs, etc. The objective can also take on many different forms; e.g., minimizing the makespan (the maximum completion time among all jobs) or the total completion time or the maximum response time,

or maximizing the number of on time jobs. For an extensive introduction into the theory of scheduling, see, e.g., [7], [22], [76], [96].

Although scheduling problems may concern different types of resources, many of them can be modeled as scheduling jobs on machines. A *schedule* specifies, for each time instant, the set of jobs executing at that instant, and the machines on which they are executing.

Depending on the properties of the jobs, the number and type of machines, and the optimization goal, there are various problems under different models. In the simplest model, there is a single machine and n jobs, each of which is ready at time 0 and must be executed without interruption. In a complex model, there are different types of machines; each job has several tasks which have to be executed on different machines and may be in certain order. These models are known as shop models. In the open shop model, there is no restriction on the order of the tasks. In the flow shop model, each job has exactly one task that needs to be processed in each machine, and the tasks of each job must follow the same order. In the job shop model, each job has its own ordering of the tasks, and several tasks can visit the same machine.

Scheduling problems in master-slave model. Scheduling problems in the master-slave model was recently introduced by Sahni [104]. In this model there are n jobs and m machines. Each job is associated with a preprocessing task, a slave task and a postprocessing task that must be executed in this order. Each machine is either a master machine or a slave machine. While the preprocessing and postprocessing tasks are scheduled on the master machine, each slave task is scheduled on a dedicated slave machine.

The master-slave model is closely related to the two-machine flow shop model with transfer lags. In this flow shop model, each job j has two operations: the first operation is scheduled on the upstream machine and the second operation is scheduled on the downstream machine. The interval or time lag between the finish time of the first operation and

the start time of the second operation must be exactly or at least l_j. If the l_j's are large enough such that all of the first operations finish before the start of any second operation, then the flow shop problem is equivalent to the problem of scheduling on a single machine with time lags and two tasks per job, subject to the constraint that all of the first operations are scheduled first. The latter problem is identical to the single-master master-slave scheduling model.

The master-slave model finds many applications in parallel computer scheduling and industrial settings such as semiconductor testing, machine scheduling, transportation maintenance, etc.; see [104], [106], [105], [115]. For example, suppose there is a main thread running on one processor whose function is to prepare data then fork and initiate new child threads that do the computations on different processors. After the computation of a child thread, the main thread collects the computation results and performs some processing on the results. Here, each child thread can be seen as a job with three tasks: the thread initiation and data preparation is the preprocessing task, the computation is the slave task and the postprocessing of the results from the computation is the postprocessing task.

The main objective in this dissertation is to minimize the total completion time or makespan under various scenarios. First, it is shown that many of the problems are NP-hard in the strong sense. Then some special cases are considered. It is assumed that (1) there is a single master, (2) all jobs have the same release time 0, same preprocessing task length a and same postprocessing task length c; i.e. the jobs are different from each other only by their slave tasks, (3) no preemption is allowed. Optimal or approximation algorithms are developed for some problems in this case. Finally, more general cases are considered. A job can have an arbitrary release time and arbitrary processing time. There can be one or more masters. The problem can be online or offline. Efficient approximation algorithms are developed to minimize the total completion time in various settings. These are the first general results for the total completion time problem in the master-slave model.

Furthermore, these algorithms are shown to generate schedules with small makespan as well.

1.2 Network Design Problems

The problem of *network design* is concerned with connecting a collection of sites into a "good" network that satisfies some desired properties. Problems of this type arise in applications in VLSI design, telecommunication, clustering, robotics, graph theory, and distributed systems. In all these areas, it is often important to construct high quality networks. From the topology point of view, typical quality measures of networks include the *survivability* (resistance to failures) of the network, its *stretch factor* (dilation), minimum and maximum *degree*, and its *diameter*. The goal is to minimize the *cost* of the network that satisfy certain required properties. This dissertation focuses on the survivability and stretch factor of the network.

Survivability. In some applications such as communication network design, VLSI design, networks must be able to withstand the failure/deletion of one or several links or nodes. This requirement leads to survivable network design problems.

Networks and their quality can be modeled by graphs. The sites correspond to vertices (points), and the connections can be represented by edges. In the **survivable network design problem**, the input is an undirected graph $G = (V, E)$, a non-negative cost for each edge, and a nonnegative connectivity requirement r_{uv} for every (unordered) pair of vertices u, v. The goal is to find a minimum-cost subgraph in which each pair of vertices u, v is joined by at least r_{uv} disjoint paths between v and u. In the *vertex-connected* version of the problem the paths must be internally vertex-disjoint and in the *edge-connected* version of the problem the paths must be edge-disjoint.

In many applications of this problem, often regarded as the most interesting ones [41, 53], the connectivity requirement function is specified with the help of a one-argument

function which assigns to each vertex v its connectivity type $r_v \in \mathbb{N}$. Then, for any pair of vertices $v, u \in V$, the connectivity requirement $r_{u,v}$ is simply given as $\min\{r_u, r_v\}$. Notice that, in particular, this includes the *Steiner tree problem* [97], in which $r_v \in \{0, 1\}$ for any vertex $v \in V$. It also includes the most widely applied variant of the survivability problem in which $r_v \in \{0, 1, 2\}$ for any vertex $v \in V$ (see, e.g., [53, 91, 113]). If the connectivity requirements are uniform, i.e., $r_{uv} = k$ $(k > 1)$ for every pair of vertices u and v, then this is the classical k-connectivity problem.

All these problems mentioned above are well known NP-hard graph problems. Furthermore, these problems have been shown to be MaxSNP-hard for general graphs [23, 35]. This implies that there is no hope for a PTAS in general (unless P=NP), but a PTAS could still exist for special cases. Indeed, based on the framework of Arora [3], a PTAS was found [25, 26] for the problem of finding a minimum-cost k-vertex (or k-edge) connected spanning subgraph in complete Euclidean graphs in bounded dimension.

This dissertation concentrates on efficient construction of good approximations for the above problems. The aim is to develop PTASs for some special class of graphs, specifically, the geometric graphs and planar graphs. Following the literature, this dissertation adopts the standard simplification of the connectivity requirements function. That is, each vertex v has a connectivity type r_v and the connectivity requirement $r_{u,v}$ is simply $\min\{r_u, r_v\}$.

In the geometric version of the survivable network design problem, the input is a complete Euclidean graph. The vertices are points in $\mathbb{R}^d$ and the cost of each link is equal to the Euclidean distance between its endpoints (which is a good approximation in many applications, since often the "installation" and the "service" cost is roughly proportional to the length of the link [91]). The first polynomial-time approximation schemes (PTAS) for basic variants of the survivable network design problem in Euclidean graphs are presented. First a PTAS is described for the Steiner tree problem, which is the survivable network design problem with $r_v \in \{0, 1\}$ for any vertex v. Then, the PTAS is extended to the widely

applied case where $r_v \in \{0, 1, 2\}$ for any vertex v. Finally, it is shown that the techniques yield also a PTAS for the multigraph variant of the problem where the edge-connectivity requirements satisfy $r_v \in \{0, 1, \ldots, k\}$ and $k = \mathcal{O}(1)$.

Next the 2-edge-connectivity and biconnectivity problems for planar graphs are considered: given a planar graph, find the minimum cost spanning subgraph that is 2-edge connected and biconnected, respectively. For unweighted planar graphs, approximation schemes are designed for both the minimum 2-edge-connectivity problem and the minimum biconnectivity problem, both running in polynomial time. For weighted planar graphs, Quasi-polynomial Time Approximation Schemes are designed for the 2-edge connectivity problem and biconnectivity problem. Some other variations are also considered.

Stretch factor. Let G be a weighted graph and H be a spanning subgraph of G. The *stretch factor* of H is the smallest positive t such that for any pair of vertices u and v, $d_H(u, v) \leq t d_G(u, v)$, where $d_H(u, v)$ and $d_G(u, v)$ are the weights of the shortest path distance between the vertices u and v in H and G, respectively. The graph H is called a *t-spanner* of G. If G is the complete Euclidean graph, then $d_G(u, v)$ is simply the Euclidean distance $|uv|$ of u and v. The graph H is simply called a t-spanner for V.

Traditionally, the main measure of quality of spanners are the *number of edges*, *maximum degree*, and *total cost*. In this context, in any Euclidean space $\mathbb{R}^d$ with constant d, for every positive constant ε, one can construct in $\mathcal{O}(n \log n)$ time a $(1 + \varepsilon)$-spanner in which every vertex has constant degree and whose total cost is in the order of the cost of the MST for the input point set [5, 55]; all these bounds are asymptotically optimal. (See also [1, 6, 15, 29, 30, 31, 81] for other related results on spanners.) For an arbitrarily weighted graph G, Althöfer et al. [1] designed a simple greedy algorithm that computes a t-spanner H of G for any $t > 1$. In the case of *planar* graphs, it is shown in [1] that this spanner has weight $w(H) \leq (1 + 2/(t - 1))\mathrm{MST}(G)$, where $\mathrm{MST}(G)$ is the weight of a minimum spanning tree in G.

Spanners are important structures that provide a sparse or economic representation of a given graph. They were introduced by Peleg and Schäffer [93] in the context of distributed computing and later, by Chew [20] in the context of computational geometry. Spanners have many applications in robotics, graph theory, network topology design, distributed systems; the recent $\mathcal{O}(n \log n)$-time PTAS for Euclidean TSP [101] is heavily based on the use of spanners, and so is the recent PTAS for Euclidean biconnectivity [26]. Spanners are also very extensively used in recent advances on topological issues in ad-hoc networks (see, e.g., [42, 54, 100] and the reference therein). Survey expositions [15, 34, 87, 92, 110] contain an extensive description on spanners and their applications.

Spanners are also heavily used by the approximation schemes in this dissertation. For the geometric version of the survivable network design problem, the PTASs work on the geometric spanners of the input vertices (or the subset of the input vertices). For the 2-edge connectivity problem and biconnectivity problem in weighted planar graphs, the approximation schemes also depend in a crucial way on the new construction of *light spanners* for weighted planar graphs.

Fault-tolerant spanners are natural extensions of spanners to graphs resistant to edge and vertex removal. They were introduced by Levcopoulos et al. [83]. Such graphs contain *short paths* between each pair of vertices *even after removing a vertex or an edge.*

In [83] and [86], several algorithms have been proposed to construct geometric fault-tolerant spanners with low cost and bounded maximum degree. But none of them could achieve the optimal bounds in maximum degree and total cost at the same time. The main open problem left is whether there exist fault-tolerant spanners having good bounds for both the maximum degree and the total cost.

This dissertation gives the first construction of vertex and edge fault-tolerant spanners having optimal bounds for both maximum degree and total cost at the same time. It is shown that there is a greedy algorithm that for any $t > 1$ and any non-negative integer k, constructs a k-fault-tolerant t-spanner in which every vertex is of degree $\mathcal{O}(k)$ and whose

total cost is $\mathcal{O}(k^2)$ times the cost of minimum spanning tree; these bounds are asymptotically optimal. An efficient algorithm is designed to find fault-tolerant spanners with asymptotically optimal bound for the maximum degree and almost optimal bound for the total cost based on a new, sufficient condition for a graph to be a k-fault-tolerant spanner.

1.3 Outline

The dissertation contains two parts. Part I is dedicated to scheduling problems in master-slave systems. The new results are joint work with J. Leung which appear in [79, 80]. These results are presented in three chapters. In Chapter 2, the master slave model and some of its applications are first introduced. Then the problems going to be studied are defined. Finally the new complexity results are presented. The complexity results show that many makespan and total completion time problems, with or without constraints, are NP-hard in the strong sense. Thus the following two chapters concentrate on approximation algorithms.

Chapter 3 considers special cases of the problems in master slave model, which assume that (1) there is a single master, (2) all jobs have the same release time 0, same preprocessing task length a and same postprocessing task length c; i.e. the jobs are different from each other only by their slave tasks, (3) no preemption is allowed. First it is proved that if there are canonical and order preserving constraints, then in $O(n \log n)$ time one can find an optimal schedule that minimizes the total completion time, when $a_i = a$ and $c_i = c$ for all $1 \leq i \leq n$ and $a \leq c$. After that, approximation algorithms are developed for the canonical total completion time problem and the no-wait-in makespan problem, respectively.

Chapter 4 considers the general cases of total completion time problem. Efficient approximation algorithms are developed to minimize the total completion time in various settings. These are the first general results for the total completion time problem in the

master-slave model. Furthermore, these algorithms are shown to generate schedules with small makespan as well.

The second part of this dissertation is dedicated to network design problems. In Chapter 5, the PTASs for the geometric version of the survivability problems are presented. This include the first PTAS for the Steiner tree problem, the $\{0, 1, 2\}$-connectivity problem and the multigraph variant $\{0, 1, \ldots, k\}$-edge connectivity problem. The results of this chapter have been published in [27] and they are joint work with A. Czumaj and A. Lingas.

Chapters 6 and 7 consider the 2-connectivity problem and its variations in planar graphs. In Chapter 6, the PTASs for the 2-edge-connectivity and biconnectivity problem in unweighted planar graphs are described. Chapter 7 discusses the weighted planar graphs. First a PTAS is presented for the problem of finding minimum-weight 2-edge-connected spanning subgraphs where duplicate edges are allowed. Then a new greedy spanner construction for edge-weighted planar graphs are given. This construction augments any connected subgraph A of a weighted planar graph G to a $(1+\varepsilon)$-spanner of G with total weight bounded by $\text{weight}(A)/\varepsilon$. Based on this spanner, quasi-polynomial time approximation schemes are derived for the problems of finding the minimum-weight 2-edge-connected or biconnected spanning subgraph in planar graphs. Approximation schemes are also designed for the minimum-weight 1-2-connectivity problem, which is the variant of the survivable network design problem where vertices have non-uniform (1 or 2) connectivity constraints. Chapter 6 contains joint work with A. Czumaj, M. Grigni, P. Sissokho, and appears in [24]. Chapter 7 contains joint work with A. Berger, A. Czumaj and M. Grigni in [9].

Chapter 8 presents two new results about vertex and edge fault-tolerant spanners in Euclidean spaces. First it is shown that a greedy algorithm that for any $t > 1$ and any non-negative integer k, constructs a k-fault-tolerant t-spanner in which every vertex is of degree $\mathcal{O}(k)$ and whose total cost is $\mathcal{O}(k^2)$ times the cost of minimum spanning tree; these bounds are asymptotically optimal. The next contribution is an efficient algorithm for constructing

good fault-tolerant spanners. A new, sufficient condition for a graph to be a k-fault-tolerant spanner is developed. Using this condition, one can design an efficient algorithm that finds fault-tolerant spanners with asymptotically optimal bound for the maximum degree and almost optimal bound for the total cost.

Finally, Chapter 9 summarizes the contributions of the dissertation. Some possible extensions and future research directions are also remarked.

PART I

SCHEDULING PROBLEMS IN MASTER-SLAVE MODEL

CHAPTER 2

COMPLEXITY OF SCHEDULING PROBLEMS IN MASTER-SLAVE MODEL

2.1 Master-slave Model

The master-slave model was recently introduced by Sahni [104]. In this model, each job has to be processed sequentially in three stages. In the first stage, the preprocessing task runs on a master machine; in the second stage, the slave task runs on a dedicated slave machine; and in the last stage, the postprocessing task again runs on a master machine, possibly different from the master machine in the first stage. The preprocessing, slave and postprocessing tasks and task times of job i are denoted by a_i, b_i and c_i, respectively. It is assumed that $a_i > 0$, $b_i > 0$ and $c_i > 0$.

A job may have a release time $r_i \geq 0$, i.e., a_i cannot start until r_i. Without loss of generality, one can assume that $\min r_j = 0$. Unless stated otherwise, all jobs are assumed to have the same release time. There are two cases when arbitrary release time is present. The first case deals with offline problems, i.e., the release times and processing times of all jobs are known in advance. The second case deals with online problems, i.e., no information of a job i is given until it arrives at r_i, and when it arrives, all parameters about job i is given. The quadruple (r_i, a_i, b_i, c_i) is used to denote job i. For simplicity, if $r_i = 0$, one can use the triplet (a_i, b_i, c_i) to represent job i.

Each machine is either a master machine or a slave machine. The master machines are used to run preprocessing and/or postprocessing tasks, and the slave machines are used to run slave tasks, one slave machine for each slave task. In a single-master system, there is a single master to execute all preprocessing tasks (a tasks) and postprocessing tasks (c tasks). In a multi-master system, there are more than one master, each of which is capable of processing both a tasks and c tasks. Finally, in some systems, there are distinct

preprocessing masters (preprocessors) and postprocessing masters (postprocessors), which are dedicated to process a tasks and c tasks, respectively.

The master-slave model is closely related to the flow shop model. The system which has a single preprocessor and a single postprocessor can be seen as a two-machine flow shop with transfer lags. In this flow shop model, each job j has two operations: the first operation is scheduled on the upstream machine and the second operation is scheduled on the downstream machine. The interval or time lag between the finish time of the first operation and the start time of the second operation must be exactly or at least l_j. If the l_j's are large enough such that all of the first operations finish before the start of any second operation, then the flow shop problem is equivalent to the problem of scheduling on a single machine with time lags and two tasks per job, subject to the constraint that all of the first operations are scheduled first. The latter problem is identical to the single-master master-slave scheduling model.

When there are more than one preprocessing and postprocessing masters, the master-slave model can be seen as a two-stage hybrid flow shop with transfer lags. In this sense, the single master case can be regarded as a three-stage hybrid flow shop where the first and the last stage has a single machine and the second stage has n machines. Hybrid flow shop is often found in electronic manufacturing environment such as IC packaging and make-to-stock wafer manufacturing. In recent years, hybrid flow shop has received significant attention, see [12], [75], [78] and [112].

2.2 Applications of Master-slave Model

The master-slave model finds many applications in parallel computer scheduling and industrial settings such as semiconductor testing, machine scheduling, transportation maintenance, etc. Some of them are listed in the following. For more applications, see [104], [106], [105] and [115].

Several applications of the master-slave model are found in parallel computer scheduling. A common parallel programming paradigm involves the use of a main computational thread whose function is to prepare data then fork and initiate new child threads that do the computations on different processors. After the computation of a child thread, the main thread collects the computation results and performs some processing on the results. Here, each child thread can be seen as a job with three tasks: the thread initiation and data preparation is the preprocessing task, the computation is the slave task and the postprocessing of the results from the computation is the postprocessing task.

The master-slave paradigm also has applications in certain semiconductor testing operations. In the case of burn-in operations, chips are subject to thermal stress for an extended period of time. The whole process for each chip consists of three phases. First, an initial burn-in operation is accomplished by maintaining the oven at a constant temperature while powering up the chip. The burn-in times for each chip are specified by the customer and thus fixed a priori. Then, in the second phase, the chip cools off for a specified amount of time that depends on the length and intensity of the initial burn-in period. In the last phase, the chip is subject to a final burn-in operation. In this application the burn-in oven corresponds to the master machine, the two burn-in tasks correspond to preprocessing and postprocessing and the cooling period corresponds to the slave task. Since the burn-in operations are near the end of the production process, scheduling is critical in determining on-time delivery and output performance for the entire company.

Industrial applications of the master-slave paradigm include the case of consolidators that receive orders to manufacture quantities of various items. The actual manufacturing is done by a collection of slave agencies. In this example, the consolidator is the master machine and the slave agencies are the slave machines. The consolidator needs to assemble the raw material needed for each task, load the trucks that will deliver this material to the slave machines, and perform an inspection before the consignment leaves. All of these work belong to preprocessing task. The slave machines need to wait for the arrival of

the raw material, inspect the received goods, perform the manufacture, load the goods onto the trucks for delivery, perform an inspection as the trucks are leaving. These activities together with the delay involved in getting the trucks to their destination (i.e., the consolidator) represent the slave work. When the finished goods arrive at the consolidator, they are inspected and inventoried. This represents the postprocessing.

It is easy to see that all of the above examples generalize to multi-master systems or distinct preprocessing and postprocessing master systems.

2.3 Scheduling Problems in Master-slave Model: Definitions and Notations

Given a set of jobs in the master-slave system and a schedule S of the jobs, two jobs i and j are said to *overlap* in S if the master machine is working on the preprocessing/postprocessing task of job i while a slave machine is working on the slave task of job j. Note that there may be several jobs overlapping with a given job i.

The completion (or finish) time of job i in a schedule S is the time when the postprocessing task c_i finishes. The completion time of i in S is denoted by $C_i(S)$. If S is clear from the context, C_i, instead of $C_i(S)$, is used. The *makespan* of S is the earliest time when all the tasks have been completed. The makespan of S is denoted by $C_{max}(S)$, or C_{max} if S is clear from the context. The *total completion time* of S, denoted by $C(S)$, is the sum of the completion times of all n jobs, i.e., $C(S) = \sum_{j=1}^{n} C_j(S)$.

Makespan and total completion time are two common objectives to minimize. The problems of finding a schedule that minimizes the makespan and total completion time are referred to as the *makespan* (C_{max}) *problem* and *total completion time* ($\sum C_j$) *problem*, respectively. A schedule that minimizes C_{max} or $\sum C_j$ is usually denoted by S^*. Throughout this dissertation, C^*_{max} and C^* are used to denote the minimum makespan and the minimum total completion time, respectively.

A *non-preemptive* schedule is one that schedules each task without interruption. Note that in such a schedule, it is still possible that there is an interval between the finish

time of a_i and the start time of b_i, or the finish time of b_i and the start time of c_i. However, without loss of generality, one can always assume that b_i is scheduled immediately as soon as a_i completes. In a preemptive schedule, a job running on one machine may be interrupted for some time, and later resumed on possibly a different machine. Both non-preemptive and preemptive schedules have some applications. In the consolidators example, non-preemptive schedules are more realistic than preemptive schedules. On the other hand, in the parallel computer scheduling example, preemptive schedules are as realistic as non-preemptive schedules.

A non-preemptive schedule S is *order preserving* if for any two jobs i and j such that a_i completes before a_j, c_i must also complete before c_j. A *no-wait-in* schedule is one such that each slave task must be scheduled immediately after the corresponding preprocessing task finishes and each postprocessing task must be scheduled immediately after the corresponding slave task finishes. In other words, once a job starts, it will not stop until it finishes. It is easy to see that a no-wait-in schedule must be non-preemptive.

A *canonical* schedule on the single master system is one such that all the preprocessing tasks complete before any postprocessing tasks can start (Note that the definition of canonical schedule is slightly different from the one given in [104]). In the multi-master system, a canonical schedule is one that is canonical on each master. Both canonical and non-canonical schedules have some applications. In the consolidators example, canonical schedules make sense while non-canonical schedules do not. On the other hand, in the parallel computer scheduling example, non-canonical schedules make sense while canonical schedules do not.

It is easy to see that if all jobs have the same release time, one can always arrange a schedule to be canonical without increasing the makespan. Thus, in order to minimize the makespan in this case, one only needs to focus on canonical schedules. However, this is not true if one wants to minimize $\sum C_j$. In fact, the ratio of the total completion time of the best canonical schedule versus that of the best non-canonical schedule can be arbitrarily

large. Consider the example: $(n-1)$ identical jobs $(1, \varepsilon, 1)$ and one job $(n^2, \varepsilon, 1)$, where ε is an arbitrary small positive number. The optimal canonical schedule has total completion time $O(n^3)$, while the optimal non-canonical schedule has total completion time $O(n^2)$.

2.4 Previous Work

So far the main research efforts to the master-slave model are for makespan minimization, assuming all jobs have the same release time. As noted before, it is sufficient to focus on canonical schedules for the makespan objective in this case. The general makespan problem without constraints has been shown to be NP-hard by Kern and Nawijn [69]. Sahni [104] showed that the no-wait-in makespan problem is NP-hard in the ordinary sense, even when there is order preserving constraint. He also gave an $O(n \log n)$-time algorithm that solves the order preserving makespan problem.

Sahni and Vairaktarakis [106] proposed several constant approximation algorithms for the makespan problem in the single-master and multi-master systems. For the general problem without any constraints, they gave a $\frac{3}{2}$-approximation algorithms for the single master system and a 2-approximation algorithms for the multi-master systems.

Further algorithms were given by Vairaktarakis [115] when there are m_1 preprocessors and m_2 postprocessors. Let $m = \max\{m_1, m_2\}$. He gave approximation algorithms with a worst-case bound of $2 - \frac{1}{m}$ for the makespan problems with no constraint, or with the constraints of order preserving.

Flow shop is a classical model that has been studied for a long time. Let m be the number of stages. For the makespan problem, Johnson [65] developed an $O(n \log n)$ time optimal algorithm when $m = 2$. The problem becomes NP-hard when $m \geq 3$. In this case, Hall [58] presented a $(1+\varepsilon)$-approximation algorithm for any fixed positive ε. For the total completion time problem, it is NP-hard in the strong sense even if $m = 2$ and preemption is allowed [33]. Gonzalez and Sahni [46] developed an approximation algorithm for this problem in the m-stage flow shop model. The approximation ratio of their algorithms is

m. Let p_i be the total processing time of all operations of job i. The algorithm schedules the jobs in nondecreasing order of p_i at each stage. By a careful analysis, Hoogeveen et al. [61] showed that this algorithm has approximation ratio $2\beta/(\alpha+\beta)$, where α denotes the minimal processing time of all tasks and β denotes the maximal processing time of all tasks. If the jobs have different weights, Schulz [108] obtained an approximation algorithm with performance guarantee of $2m$ (or $2m+1$ in case of arbitrary release time) for total weighted completion time based on linear programming.

When there are more than one machine in either or both stages, the model is called a flexible or hybrid flow shop. Both makespan and total completion time minimization problems are NP-hard, even if preemption is allowed; see [57] and [33]. Lee and Vairaktarakis [78] developed heuristics for makespan minimization with approximation ratio of $2 - 1/\max\{m_1, m_2\}$, where m_1 and m_2 are the number of machines in stages 1 and 2, respectively. Based on linear programming, Schulz [108] obtained an approximation algorithm with performance guarantee of $3m$ (or $3m+1$ in case of arbitrary release time) for the total weighted completion time, where m is the number of stages. Thus, if $m = 2$, it is a 6-approximation in the case of identical release times and a 7-approximation in the case of arbitrary release times.

For the two-stage flow shop with transfer lags model, some research has been done, most of which is about makespan minimization. Dell'Amico [2] proved that the makespan problem is NP-hard, even if preemption is allowed and each stage has only one machine. Later, Yu, Hoogeveen and Lenstra [121] showed that the problem is NP-hard even if all tasks have unit length. This is in contrast to the fact that the problem is solvable in polynomial time when there is no transfer lags. By the above discussion, this model is the same as the master-slave model when the preprocessing and postprocessing masters are distinct. Thus, the heuristics given in [115] for the master-slave model also work here. Little is known about the total completion time minimization problem.

2.5 New Results: Complexity of Scheduling Problems in Master-slave Model

First, some previous complexity results for the makespan problem is strengthened. Then some new results for the total completion time problem are developed, based on the result from [121]. It is shown that many problems are strongly NP-hard, even with some constraints. The main results can be summarized as follows:

- The makespan problem is strongly NP-hard, even if $a_i = c_i = 1$ for $1 \leq i \leq n$ and only no-wait-in schedules are considered.
- The order preserving and no-wait-in makespan problem is NP-hard in the strong sense, even if $a_i = c_i$ for all $1 \leq i \leq n$.
- The total completion time problem is NP-hard in the strong sense, even if (i) all the preprocessing and postprocessing tasks have unit time, (ii) only canonical, or no-wait-in, or canonical and no-wait-in schedules are considered.
- The order preserving and no-wait-in total completion time problem is NP-hard in the strong sense, even if $a_i = c_i$ for all $1 \leq i \leq n$.

It is sufficient to prove the above results in the simple case: a single master and all jobs have the same release time. Following is a theorem about C_{max} problem in a two-machine flow shop with delays which was recently proved by Yu et al. [121].

Theorem 2.5.1 (See Theorem 21 and Corollary 22 [121]) *The flow shop problem* $F2|l_j, p_{ij} = 1|C_{max}$ *is strongly NP-hard, even if exact delays are required.*

By the discussion in Section 2.1, the above theorem immediately implies the following theorem.

Theorem 2.5.2 *The makespan problem in master slave system with* $a_i = c_i = 1$ *for all* $1 \leq i \leq n$ *is strongly NP-hard, even if there is no-wait-in constraint.*

Proof : The outline of the proof of this theorem is given below, as it is relevant to the discussion of the total completion time problem later. The reduction is almost the same as in the proof of Theorem 21 in [121]. But for each job i, instead of having a lag l_i between its two tasks, it now has a slave task b_i which must start after the preprocessing task and finish before the postprocessing task. The preprocessing and postprocessing tasks are performed on the same master machine, instead of two machines. To ensure the same argument goes through, let $b_i = l_i + Y$ for each job i, where $Y = n - (mB + 2)$ and n is the number of jobs in the instance of the two-machine flow shop with delays. The bound for the makespan is also increased by Y. The proof of Theorem 21 in [121] used a reduction from the 3-partition problem, which is known to be strongly NP-complete; see [43].

A 3-partition problem instance has *input* as a set of non-negative integers $X = \{x_1, x_2, \ldots, x_{3m}\}$ and a non-negative integer B such that $\sum_{i=1}^{3m} x_i = mB$ and $B/4 < x_i < B/2$ for all $1 \leq i \leq 3m$. The problem is to decide whether X can be partitioned into m disjoint subsets X_1, ..., X_m such that for all $1 \leq j \leq m$, $\sum_{x_i \in X_j} x_i = B$? In the following X_j is used to denote a partition subset, where $1 \leq j \leq m$.

An instance of the makespan problem can be constructed as follows. There are $n = m^2B + mB$ jobs. For job i, $1 \leq i \leq 3m$, referred to as a P-job in [121], $b_i = x_i + Y$ where $Y = n - (mB + 2)$; for job i, $3m + 1 \leq i \leq mB$, referred to as a Z-job in [121], $b_i = Y$, and for job i, $mB + 1 \leq i \leq m^2B + mB$, referred to as an L-job in [121], $b_i = (m + 1)B + 1 + Y$. For all job i, $1 \leq i \leq n$, let $a_i = c_i = 1$. Let the bound for the makespan be $mB + n + 2 + Y = 2n$.

Using the same argument as in [121], one can show that if there is a canonical schedule S with makespan less than or equal to $2n$, then S must have the following properties: (1) The makespan of S is exactly $2n$, (2) S is a no-wait-in schedule and the finish times of the jobs are $n + 1, \ldots, 2n$. See Figure 2.1 for an illustration of the schedule on the master machine. Also, it can be shown that there is a solution to the 3-partition problem if and only if there is a schedule S with makespan exactly $2n$. □

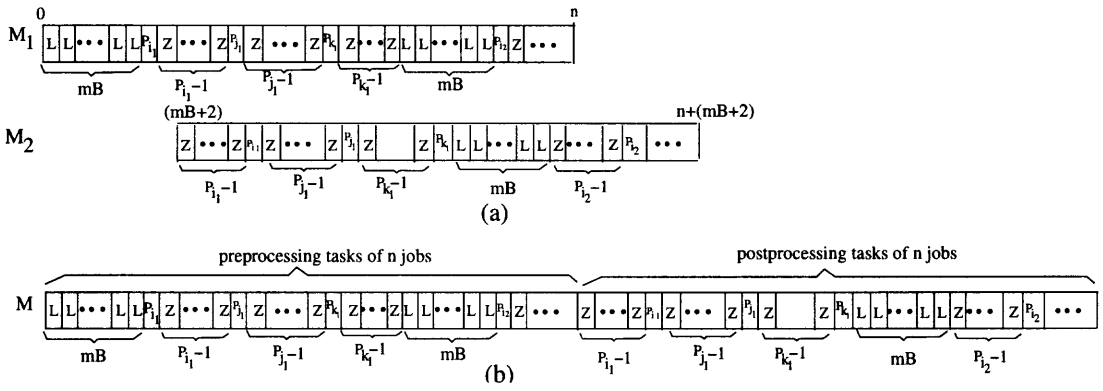


Figure 2.1 (a) An illustration of the schedule that minimizes the makespan for the problem instance in the two-machine flow shop model with lags reduced from 3-partition in [121], (b) An illustration of the schedule on the master machine that minimizes the makespan problem instance reduced from 3-partition in Theorem 2.5.2.

The above result can be used to show that the total completion time problem is also strongly NP-hard.

Theorem 2.5.3 *The total completion time problem with* $a_i = c_i = 1$ *for all* $1 \leq i \leq n$ *is strongly NP-hard, even if (1) only canonical schedules are considered, or (2) only no-wait-in schedules are considered, or (3) canonical and no-waited-in schedules are considered.*

Proof : The reduction is still almost the same as above, the only difference is that now one asks the question: is there a schedule of the n tasks with total completion time at most $n^2 + \frac{n(n+1)}{2}$?

It is sufficient to prove that there is a schedule with total completion time less than or equal to $n^2 + \frac{n(n+1)}{2}$ if and only if there is a schedule with makespan less than or equal to $2n$.

"If" part. Let S be a schedule with makespan less than or equal to $mB+n+Y-2 = 2n$. By the proof of Theorem 2.5.2, S must be a canonical and no-wait-in schedule and the completion time of the jobs are $n+1, n+2, \ldots, 2n$. Thus the total completion time is exactly $n^2 + \frac{n(n+1)}{2}$.

"Only if" part. Let S^* be a schedule for the n jobs constructed above such that $\sum C_j(S^*) \leq n^2 + \frac{n(n+1)}{2}$. For any job i, $1 \leq i \leq n$, let $C_{a_i}(S^*)$ denote the time when

the task a_i finishes in the schedule S^*. Then, $C_{a_i}(S^*) \geq 1$, and if $i \neq j$ then $C_{a_i}(S^*) \neq C_{a_j}(S^*)$. Thus $\sum_{i=1}^{n} C_{a_i}(S^*) \geq (1+2+\cdots+n) = \frac{1}{2}n(n+1)$. Since $C_i \geq C_{a_i}(S^*) + b_i + c_i = C_{a_i}(S^*) + b_i + 1$, the following inequality holds.

$$\sum_{i=1}^{n} C_i(S^*) \geq \sum_{i=1}^{n} (C_{a_i}(S^*) + b_i + 1) = \sum_{i=1}^{n} C_{a_i}(S^*) + \sum_{i=1}^{n} b_i + n \geq \frac{1}{2}n(n+1) + \sum_{i=1}^{n} b_i + n \ .$$

Note that $\sum_{i=1}^{n} b_i = nY + \sum_{i=1}^{3m} x_i + m^2B((m+1)B+1) = n^2 - n$. This means that $\sum_{i=1}^{n} C_i(S^*) \geq n^2 + \frac{n(n+1)}{2}$. By the assumption about S^*, it must be true that $\sum_{i=1}^{n} C_i(S^*) = n^2 + \frac{n(n+1)}{2}$. Thus S^* must be a canonical and no-wait-in schedule. Furthermore, the completion time of the jobs must be $n+1, n+2, \ldots, 2n$. Thus the makespan of the schedule is $2n$.

This completes the proof. □

The above theorem implies that the total completion time problem is NP-hard in the strong sense even if preemption is allowed. Observe that the optimal schedule S for the constructed instance in the proof of Theorem 2.5.2 is not order preserving, so the above results are not applicable to order preserving scheduling problems. In the following, the complexity of no-wait-in makespan and no-wait-in total completion time problem with the constraint of order preserving will be considered. Both problems will be shown to be NP-hard in the strong sense by a reduction from the 3-partition problem.

Theorem 2.5.4 *The problem of minimizing the order preserving and no-wait-in makespan is strongly NP-hard, even if* $a_i = c_i$ *for* $1 \leq i \leq n$.

Proof : To reduce an instance of 3-partition problem to an instance of the scheduling problem, one first create two types of jobs. This include (1) $3m$ *partition* jobs: $a_i = c_i = x_i$ and $b_i = 3B + 2 - x_i$, $1 \leq i \leq 3m$, and (2) $2m$ *separation* jobs: $a_i = c_i = B + 1$ and $b_i = B$, $3m + 1 \leq i \leq 5m$. The problem is to determine whether there is an order preserving and no-wait-in-schedule such that $C_{max} \leq 2m(3B + 2)$. Clearly, the reduction can be done in polynomial time.

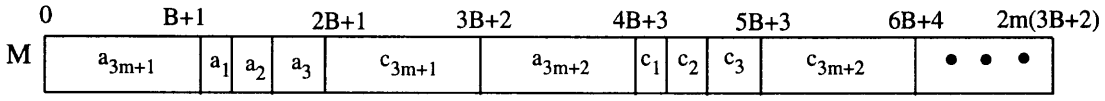


Figure 2.2 An illustration of the schedule on the master machine for the instance reduced from 3-partition in Theorem 2.5.4. Jobs 1, 2 and 3 are partition jobs, jobs $3m+1$ and $3m+2$ are separation jobs.

If there is a solution to the 3-partition problem, then one can schedule the separation jobs in any order without overlapping, and for each group of 3-partition jobs corresponding to a partition subset, schedule their preprocessing tasks fully overlapping with the slave task of one separation job and the postprocessing tasks fully overlapping with slave task of the separation job immediately following the previous one. See Figure 2.2 for an illustration of the schedule. It is clear that the schedule is order preserving and no-wait-in and $C_{max} = 2m(3B + 2)$. Now suppose the scheduling problem has a solution; i.e., there is an order preserving and no-wait-in schedule S such that $C_{max} \leq 2m(3B + 2)$. Since only no-wait-in schedules are considered and since $a_i = c_i > b_j$ for any two separation jobs i and j, a separation job can not overlap with another one. Hence, $C_{max} \geq \sum_{j=3m+1}^{5m}(a_j + b_j + c_j) = 2m(3B + 2)$. By assumption, $C_{max} \leq 2m(3B + 2)$. Therefore $C_{max} = 2m(3B + 2)$, which means that all the partition jobs must fully overlap with the separation jobs. For each partition job i, $2B < b_i < 3B + 2$. Therefore, each partition job i must overlap with exactly two adjacent separation jobs in the schedule S. Because $B/4 < a_i = c_i < B/2$, at most three preprocessing and/or postprocessing tasks of the partition jobs overlap with one separation job. Since there are $2m$ separation jobs only, there must be exactly three preprocessing or postprocessing tasks fully overlapping with each separation job. For each separation job j, $b_j = B$. Thus, the integers corresponding to the three preprocessing or postprocessing tasks that overlap with b_j have a total exactly B. Hence, the 3-partition problem has a solution. □

Theorem 2.5.5 *The problem of minimizing the order preserving and no-wait-in total completion time is strongly NP-hard, even if* $a_i = c_i$ *for* $1 \leq i \leq n$.

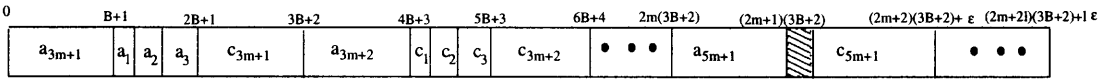


Figure 2.3 An illustration of the schedule on the master machine for the instance reduced from 3-partition in Theorem 2.5.5. Jobs 1, 2 and 3 are partition jobs, jobs $3m+1$ and $3m+2$ are medium jobs and job $5m+1$ is a large job.

Proof : As in the previous proof, the reduction is from 3-partition problem. First three types of jobs are created for the scheduling problem: (1) $3m$ *small jobs*: $a_i = c_i = x_i$ and $b_i = 3B + 2 - x_i$, $1 \leq i \leq 3m$; (2) $2m$ *medium jobs*: $a_i = c_i = B + 1$ and $b_i = B$, $3m+1 \leq i \leq 5m$; (3) l *large jobs*: $a_i = c_i = 3B+2$ and $b_i = \varepsilon$, $5m+1 \leq i \leq 5m+l$, ε is a small positive number and l is an integer greater than $36m^2+19m+24m^2/B+12m/B$.

Let

$$B_M = (3B+2) \cdot m(2m+1) \ ,$$

$$B_L = 2l\,m(3B+2) + l\,(l+1)(3B+2) + \frac{1}{2}\,l(l+1)\,\varepsilon, \text{ and}$$

$$B_S = 3m\left[3mB + 2m + \frac{7}{4}\,B + 1\right] \ .$$

Let $B^* = B_S + B_M + B_L$. The scheduling problem is: Is there an order preserving and no-wait-in schedule of these jobs such that $\sum_{j=1}^{5m+l} C_j \leq B^*$?

If the partition problem has a solution, then schedule the jobs as follows: first schedule the medium jobs in any order without overlapping; schedule any three small jobs that correspond to the three integers in the same partition subset fully overlapping with two adjacent medium jobs; finally, schedule the large jobs after the medium jobs one by one in any order without overlapping. See Figure 2.3 for an illustration of the schedule. One can easily verify that the schedule is an order preserving and no-wait-in schedule. To bound the total completion time, the total completion time of each type of jobs is calculated separately. Without loss of generality, suppose that the medium jobs are scheduled in the order of $3m+1, 3m+2, \ldots, 5m$. Since the medium jobs are scheduled one by one from time

0 without overlap, the total completion time of all medium jobs is

$$\sum_{i=3m+1}^{5m} (i-3m)\cdot(a_i+b_i+c_i) = \sum_{i=1}^{2m} i\cdot(3B+2) = (3B+2)\cdot m(2m+1) = B_M \ .$$

Similarly, the total completion time of all large jobs is

$$\begin{aligned}
&\sum_{i=5m+1}^{5m+l} (2m(3B+2)+(i-5m)\cdot(a_i+b_i+c_i)) \\
&= l\cdot 2m(3B+2) + \sum_{i=1}^{l} i(2(3B+2)+\varepsilon) \\
&= 2lm(3B+2) + l(l+1)(3B+2) + \frac{1}{2}l(l+1)\varepsilon \\
&= B_L \ .
\end{aligned}$$

Now consider the total completion time of the small jobs. Suppose that the small jobs corresponding to the partition subset X_j, $1 \le j \le m$, are j_1, j_2 and j_3; furthermore, suppose that j_1 is scheduled before j_2 which is scheduled before j_3. Suppose that these jobs overlap with two consecutive medium jobs $3m+2j-1$ and $3m+2j$. Let C_{j_i} denote the completion time of the small job j_i. Then

$$\begin{aligned}
C_{j_1} &= [2(j-1)(3B+2)+(B+1)] + a_{j_1} + b_{j_1} + c_{j_1} \\
&= [2(j-1)(3B+2)+(B+1)] + x_{j_1} + (3B+2-x_{j_1}) + x_{j_1} \\
&= [(2j-1)(3B+2)+(B+1)] + x_{j_1} \\
&< [(2j-1)(3B+2)+(B+1)] + \frac{1}{2}B \ .
\end{aligned}$$

Similarly, one can get

$$\begin{aligned}
C_{j_2} &= [2(j-1)(3B+2)+(B+1)+a_{j_1}] + a_{j_2} + b_{j_2} + c_{j_2} \\
&= [2(j-1)(3B+2)+(B+1)+x_{j_1}] + x_{j_2} + (3B+2-x_{j_2}) + x_{j_2} \\
&= [(2j-1)(3B+2)+(B+1)] + x_{j_1} + x_{j_2} \\
&< [(2j-1)(3B+2)+(B+1)] + \frac{3}{4}B
\end{aligned}$$

and

$$C_{j_3} = [2(j-1)(3B+2)+(B+1)+a_{j_1}+a_{j_2}] + a_{j_3} + b_{j_3} + c_{j_3}$$

$$= [2(j-1)(3B+2) + (B+1) + x_{j_1} + x_{j_2}] + x_{j_3} + (3B+2-x_{j_3}) + x_{j_3}$$

$$= [(2j-1)(3B+2) + (B+1)] + x_{j_1} + x_{j_2} + x_{j_3}$$

$$= [(2j-1)(3B+2) + (B+1)] + B \ .$$

Thus,

$$C_{j_1} + C_{j_2} + C_{j_3} < 3 \cdot [(2j-1)(3B+2) + (B+1)] + \frac{1}{2}B + \frac{3}{4}B + B$$
$$= 3 \cdot [(2j-1)(3B+2) + (B+1)] + \frac{9}{4}B \ .$$

Hence, the total completion time of all small jobs is

$$\sum_{j=1}^{3m} C_j = \sum_{j=1}^{m} \sum_{i=1}^{3} C_{j_i} < \sum_{j=1}^{m} \left[3 \cdot ((2j-1)(3B+2) + (B+1)) + \frac{9}{4}B\right]$$
$$= 3m^2 \cdot (3B+2) + 3m(B+1) + \frac{9}{4}mB$$
$$= 3m\left[3mB + 2m + \frac{7}{4}B + 1\right]$$
$$= B_S \ .$$

Therefore, the total completion time of all jobs is

$$\sum_{j=1}^{3m} C_j + \sum_{j=3m+1}^{5m} C_j + \sum_{j=5m+1}^{5m+1} C_j < B_S + B_M + B_L = B^* \ .$$

Now, suppose there is an order preserving and no-wait-in schedule of all these jobs such that $\sum_{j=1}^{5m+1} C_j \leq B^*$. One need to show that there is a solution to the partition problem. Let S^* be such a schedule with the smallest total completion time. Some observations about S^* are listed as follows.

First, $a_j = c_j > b_i$ for any two jobs i and j that are both medium jobs or large jobs. Therefore, i and j can not overlap with each other in S^*. For the same reason, a large job can not overlap with a medium job in S^*, nor can it overlap with a small job. Hence, overlapping can only occur between the small jobs or between the small and medium jobs. Next, because $a_i = c_i = x_i > B/4$ for any small job i, $1 \leq i \leq 3m$, there are at most three small preprocessing/postprocessing tasks that can overlap with the slave task b_j of a

medium job j. Since $2B < b_i < 3B + 2$ for any small job i, b_i can overlap with tasks of at most two medium jobs in the schedule S^*.

Finally, there are two other properties of S^*.

- Large jobs are scheduled after all medium jobs finish in S^*.

  As shown above, the large jobs can not overlap with the medium jobs and small jobs. Suppose that some medium jobs and small jobs are scheduled between two large jobs. Then one can modify the schedule by moving the first large job so that it is scheduled immediately before the second large job. Since no small or medium jobs can overlap with the large jobs, this movement will not affect the feasibility of the schedule, i.e., it is still an order preserving and no-wait-in schedule. However, the new schedule has a smaller total completion time which contradicts the assumption that S^* is optimal.

- Exactly three small jobs overlap with a medium job in S^*.

  It is clear that a small job can not be scheduled after a large job; otherwise, one can interchange them without increasing the total completion time. Similarly, a small job can not be scheduled between two medium jobs. Therefore, a small job can only be scheduled either before all medium and large jobs or fully overlapping with medium jobs.

  Suppose there is a preprocessing task a_i of a small job i which is scheduled before any medium or large jobs in S^*. Then,

$$\begin{aligned}\sum_{i=1}^{5m+l} C_i(S^*) &> \sum_{j=3m+1}^{5m+l} C_j(S^*) \\ &\geq \left\lceil \sum_{j=3m+1}^{5m} (a_i + (j-3m)\cdot(a_j + b_j + c_j)) \right\rceil \\ &\quad + \left\lceil \sum_{j=5m+1}^{5m+l} (a_i + 2m(3B+2) + (j-5m)\cdot(a_j + b_j + c_j)) \right\rceil\end{aligned}$$

$$= (2m + l)a_i + \left[\sum_{j=1}^{2m} j \cdot (3B + 2)\right]$$

$$+ \left[l2m(3B + 2) + \sum_{j=1}^{l} j \cdot (2(3B + 2) + \varepsilon)\right]$$

$$= (2m + l)a_i + M + L$$

$$> \frac{1}{4}(2m + l)B + B^* - B_S \ .$$

By assumption, $l > 36m^2 + 19m + 24m^2/B + 12m/B$. Thus, $B_S < \frac{1}{4}(2m + l)B$. Hence, $\sum_{i=1}^{5m+l} C_i(S^*) > B^*$, contradicting the assumption that $\sum_{i=1}^{5m+l} C_i(S^*) \leq B^*$.

Therefore, every small job must overlap with exactly two adjacent medium jobs in S^*. Since there are $2m$ medium jobs and $3m$ small jobs, there must be exactly three preprocessing tasks or three postprocessing tasks overlapping with a medium job.

For any medium job j, there are exactly three small jobs overlapping with it in S^*. Because $b_j = B$, the sum of the preprocessing or postprocessing tasks of the three small jobs overlapping with j is exactly B, which means that the corresponding three integers have a total exactly B. Thus, the partition problem has a solution.

□

CHAPTER 3

OPTIMAL AND APPROXIMATION ALGORITHMS: SPECIAL CASES

This chapter considers special cases of the total completion time minimization problem and no-wait-in-makespan problem. It is assumed that (1) there is a single master, (2) all jobs have the same release time 0, same preprocessing task length a and same postprocessing task length c; i.e., the jobs are different from each other only by their slave tasks, (3) no preemption is allowed. It is proved that in this case if there are canonical and order preserving constraints, then in $O(n \log n)$ time one can find an optimal schedule that minimizes the total completion time, when $a_i = a$ and $c_i = c$ for all $1 \leq i \leq n$ and $a \leq c$. After that, some approximation algorithms are developed for the canonical total completion time problem and the no-wait-in makespan problem.

3.1 Optimal Algorithms for $\sum C_j$: Canonical and Order Preserving Schedules

It has been shown in Chapter 2 that minimizing total completion time is strongly NP-hard, even under severe constraints. This section shows that there is a special case that admits a polynomial time algorithm.

Theorem 3.1.1 *The problem of minimizing the canonical and order preserving total completion time can be solved in* $O(n \log n)$ *time if* $a_i = a$ *and* $c_i = c$ *for* $1 \leq i \leq n$ *and* $a \leq c$.

Proof : The proof is by showing that a canonical schedule S^* that schedules the preprocessing tasks in nondecreasing order of the processing times of the slave tasks gives an optimal order preserving schedule.

For any job i, let π_i denote the rank of job i in S^*; i.e., a_i is the π_i-th task scheduled in S^*. Because only canonical schedules are considered, the earliest possible time to start

c_i is $r_i = \max(na, \pi_i a + b_i)$. For any two jobs i and j, if $\pi_i < \pi_j$ and $b_i \leq b_j$, then $r_i \leq r_j$. By the definition of canonical schedules, c_i will be scheduled before c_j. Thus, S^* is order preserving.

The next step is to show that S^* has the minimum total completion time among all canonical and order preserving schedules. Suppose there is another canonical and order preserving schedule S that is optimal. Suppose the jobs in S are in the order of $1, 2, \ldots, n$, and there are two jobs i and $i+1$ such that $b_i > b_{i+1}$. Let their finish times in S be C_i and C_{i+1}, respectively. By interchanging them, one can get new finish times C_i' and C_{i+1}' and all the other jobs have the same finish times as before. One can show that $C_i' \leq C_{i+1}$ and $C_{i+1}' \leq C_i$. Thus, the new schedule has total completion time no larger than before. By repeatedly interchanging jobs, one will arrive at the schedule S^*. □

Note that Sahni [104] showed that when $a_i = a$, $c_i = c$ and $a = c$, scheduling jobs in nondecreasing order of the processing times of the slave tasks also minimizes the makespan.

If $a > c$, then the canonical schedule that schedules the jobs in nondecreasing order of the processing times of the slave tasks is still order preserving but may not be optimal. The complexity of the problem of finding an optimal canonical and order preserving schedule when $a > c$ is not known at the present time. However, we will show in the next section that scheduling the jobs in nondecreasing order gives a $\frac{5}{4}$-approximation.

3.2 Approximation Algorithms for $\sum C_j$: Canonical Schedules

This section considers canonical schedules only. So a schedule in this section always means a canonical schedule. The goal is to minimize the total completion time when $a_i = a$ and $c_i = c$ for all i.

Consider n jobs all of which have preprocessing time a and postprocessing time c. Each job i has a slave task b_i. Let S be a canonical schedule of these n jobs. Let $\pi_i(S)$ denote the rank of job i in S; i.e., a_i is the $\pi_i(S)$-th preprocessing task that is processed.

Let $r_i(S) = \max(na, \pi_i(S) \cdot a + b_i)$ be the ready time of c_i; i.e., at time $r_i(S)$ the master machine will schedule c_i if it finishes the postprocessing tasks that are ready earlier than c_i. Let S^* be the canonical schedule of these n jobs with minimum total completion time $\sum_{j=1}^{n} C_j(S^*)$.

Given a subset G of the n jobs, let S_G^* denote the schedule that minimizes $\sum_{j \in G} C_j$ among all schedules of the n jobs. Let C_G^* be the sum of the completion time of the jobs from G in S_G^*.

The next lemma gives a lower bound of the total completion time of the optimal schedule S^*. This lemma is useful for later analysis.

Lemma 3.2.1

$$\sum_{i=1}^{n} C_i(S^*) \geq n^2 a + \frac{1}{2} n(n+1)c \tag{3.1}$$

$$\sum_{i=1}^{n} C_i(S^*) \geq \frac{1}{2} n(n+1)a + \sum_{i=1}^{n} b_i + nc \tag{3.2}$$

Proof : First since only canonical schedules are considered, the earliest possible time to schedule a postprocessing task is na. Hence, the best possible schedule is one that first executes a postprocessing task at na, and thereafter the remaining postprocessing tasks, one after another without any idle time. Thus,

$$\sum_{i=1}^{n} C_i(S^*) \geq \sum_{i=1}^{n} (na + ic) = n^2 a + \frac{1}{2} n(n+1)c \ .$$

For (3.2), recall that a postprocessing task can not start earlier than its ready time $r_i(S)$. Therefore,

$$\begin{aligned}
\sum_{i=1}^{n} C_i(S^*) &\geq \sum_{i=1}^{n} (r_i(S^*) + c) \\
&\geq \sum_{i=1}^{n} (\pi_i(S^*)\, a + b_i + c) \\
&= \left(\sum_{i=1}^{n} \pi_i(S^*) a \right) + \sum_{i=1}^{n} b_i + nc \\
&= \frac{1}{2} n(n+1)a + \sum_{i=1}^{n} b_i + nc \ .
\end{aligned}$$

□

The next two lemmas show that there are some special cases that can be solved by a polynomial time algorithm.

Lemma 3.2.2 *If* $\max_{1\le i\le n} b_i \le (n-1)\cdot\min(a,c)$, *then any canonical schedule is an optimal schedule.*

Proof : Let S be a canonical schedule. Suppose the jobs are scheduled in the order of $j_1, j_2, \ldots, j_n$. By definition, $\pi_{j_i}(S) = i$. By assumption, $b_{j_i} \le (n-1)a$. Thus, $r_{j_1}(S) = \max(na, a + b_{j_i}) = na$. So,

$$C_{j_1} = r_{j_1}(S) + c = na + c \ .$$

For job j_2, the following can be derived.

$$r_{j_2}(S) = \max(na, 2a + b_{j_2}) \le \max(na, (2a + (n-1)\min(a,c))) \le na + c = C_{j_1}$$

Therefore, c_{j_2} will start immediately after c_{j_1} completes at $na+c$. Similarly, one can prove that there is no idle time before all c tasks finish. Thus, $\sum_{i=1}^{n} C_i(S) = \sum_{i=1}^{n}(na + ic) = n^2 a + \frac{1}{2}n(n+1)c$. By (3.1), S is an optimal schedule. □

Lemma 3.2.3 *Suppose* $(n-1)a \le b_1 \le b_2 \le \ldots \le b_n$. *If* $a \ge c$, *or* $a < c$ *but* $b_{i+1} - b_i \ge c - a$ *for* $1 \le i \le n-1$, *then an optimal schedule can be obtained by scheduling jobs in nondecreasing order of the processing times of the slave tasks.*

Proof : Let S be a schedule that schedules jobs in nondecreasing order of the processing times of the slave tasks. Then $\pi_i(S) = i$. Because $b_i \ge (n-1)a$, $r_i(S) = \max(na, ia + b_i) = ia + b_i$.

If $a \ge c$, then $r_{i+1}(S) = (i+1)a + b_{i+1} \ge ia + b_i + c = r_i(S) + c$, which means that the master machine can schedule c_i at r_i and finish at $r_i + c$ by the time c_{i+1} becomes ready. Thus, $C_i(S) = r_i + c = ia + b_i + c$. The total completion time of S is

$$\sum_{i=1}^{n} C_i(S) = \sum_{i=1}^{n}(ia + b_i + c) = \frac{1}{2}n(n+1)a + \sum_{i=1}^{n} b_i + nc \ .$$

By (3.2), S is an optimal schedule.

If $a < c$ but $b_{i+1} - b_i \geq c - a$, then it is still the case that $r_{i+1}(S) = (i+1)a + b_{i+1} \geq r_i(S) + c$. Similarly, one can show that S is an optimal schedule. □

The above lemma does not apply if only $a < c$ holds. Consider the three jobs $J_1 = (1, 7, 3)$, $J_2 = (1, 8, 3)$ and $J_3 = (1, 20, 3)$. If the jobs are scheduled in nondecreasing order of the processing times of the slave tasks, then the total completion time is 51. But if the jobs are scheduled in the order of J_1, J_3 and J_2, one can get a smaller total completion time of 50.

The next theorem gives a bound of the worst schedule versus the best schedule.

Theorem 3.2.4 *Let S be a schedule that schedules jobs in an arbitrary order. If* $a \leq c$, *then*

$$\frac{\sum_{i=1}^{n} C_i(S)}{\sum_{i=1}^{n} C_i(S^*)} < 1 + \frac{1}{1 + \frac{2a}{c}} \; ; \tag{3.3}$$

If $a > c$, *then*

$$\frac{\sum_{i=1}^{n} C_i(S)}{\sum_{i=1}^{n} C_i(S^*)} < 1 + \frac{1}{2 + \frac{c}{a}} \; . \tag{3.4}$$

Proof : Let $C_1 < C_2 < \ldots < C_n$ be the completion times in S. Let j be the last job that finishes before the first idle time. The jobs will be divided into two subsets G_1 and G_2 as follows.

Let j and all the jobs that finish before j be in G_1. By assumption, for any $k \leq j$, $k \in G_1$ and $C_k(S) = na + kc$. By (3.1), the total completion time of the jobs in G_1 in any schedule can not be smaller than that in S. Thus, $\sum_{k \in G_1} C_k(S) = C^*_{G_1}$.

Let the remaining jobs $k > j$ be in G_2. Then, for any job k in G_2, one can derive that $r_k = \pi_k a + b_k > na$ and $r_k \leq r_{k+1}$. For job $j + 1$, $C_{j+1} = r_{j+1} + c$. For job $j + 2$, one can obtain the following.

$$C_{j+2} = \max(C_{j+1}, r_{j+2}) + c \leq \max(r_{j+2} + c, r_{j+2}) + c = r_{j+2} + 2c$$

Similarly, one can show that for $j+1 \le k \le n$,

$$C_k \le r_k + (k-j)c = \pi_k a + b_k + (k-j)c \ .$$

Thus,

$$\begin{aligned}
\sum_{k\in G_2} C_k(S) &= \sum_{k=j+1}^{n} C_k \\
&\le \sum_{k=j+1}^{n} (\pi_k a + b_k + (k-j)c) \\
&= \left(\sum_{k=j+1}^{n} (\pi_k a + b_k)\right) + \sum_{k=1}^{n-j} kc \\
&= \sum_{k=j+1}^{n} \pi_k a + \left(\sum_{k=j+1}^{n} b_k\right) + \left(\sum_{k=1}^{n-j-1} kc\right) + (n-j)c \\
&\le \sum_{k=j+1}^{n} ka + \left(\sum_{k=j+1}^{n} b_k\right) + \sum_{k=1}^{n-j-1} kc + \sum_{k=1}^{n-j} c \\
&= \left(\sum_{k=1}^{n-j} (ka + b_{k+j} + c)\right) + (n-j)ja + \sum_{k=1}^{n-j-1} kc \\
&\le C^*_{G_2} + (n-j)ja + \frac{1}{2}(n-j)(n-j-1)c \\
&\le C^*_{G_2} + [(n-j)j + \frac{1}{2}(n-j)(n-j-1)] \max(a,c) \\
&\le C^*_{G_2} + \frac{1}{2}n(n-1)\cdot \max(a,c)
\end{aligned}$$

So,

$$\begin{aligned}
\sum_{k=1}^{n} C_k(S) &= \sum_{k\in G_1} C_k(S) + \sum_{k\in G_2} C_k(S) \\
&\le C^*_{G_1} + C^*_{G_2} + \frac{1}{2}n(n-1)\cdot \max(a,c) \\
&\le \sum_{k=1}^{n} C_k(S^*) + \frac{1}{2}n(n-1)\cdot \max(a,c) \ .
\end{aligned}$$

By (3.1), $\sum_{k=1}^{n} C_k(S^*) \ge n^2 a + \frac{1}{2}n(n+1)c$. Therefore,

$$\frac{\sum_{k=1}^{n} C_k(S)}{\sum_{k=1}^{n} C_k(S^*)} \le 1 + \frac{\frac{1}{2}n(n-1)\cdot \max(a,c)}{n^2 a + \frac{1}{2}n(n+1)c} < 1 + \begin{cases} \frac{1}{1+\frac{2a}{c}} & \text{if } a \le c \\ \frac{1}{2+\frac{c}{a}} & \text{if } a > c \end{cases} \ .$$

□

Corollary 3.2.5 *Let S be a schedule that schedules jobs in an arbitrary order. If* $a > c$, *then* $\frac{\sum_{k=1}^{n} C_k(S)}{\sum_{k=1}^{n} C_k(S^*)} < \frac{3}{2}$; *if* $a = c$, *then* $\frac{\sum_{k=1}^{n} C_k(S)}{\sum_{k=1}^{n} C_k(S^*)} < \frac{4}{3}$; *and if* $a < c$, *then* $\frac{\sum_{k=1}^{n} C_k(S)}{\sum_{k=1}^{n} C_k(S^*)} < 2$.

Proof : The correctness follows directly from (3.3) and (3.4). □

It is unknown whether the bounds in the above theorem are tight or not. For the case that $a = c = 1$, there are examples approaching the $\frac{4}{3}$ bound. If $n = 5$, let the b_i's be $2, 3, 4, 5, 6$. If b_i's are used to represent the jobs, then the optimal order of the jobs is $(4, 6, 3, 5, 2)$ with total completion time 40, while the worst scheduling order is $(6, 5, 4, 3, 2)$ with total completion time 50. If $n = 9$, let the b_i's be $4, 5, 6, \ldots, 12$. The optimal order is $(8, 9, 10, 12, 5, 11, 7, 4, 6)$ with completion time 126, while the worst order is $(12, 11, 10, 9, 8, 7, 6, 5, 4)$ with total completion time 162.

Now consider simple algorithms that improve upon the previous bounds. The next two theorems show that if the jobs are scheduled in nondecreasing order of the processing times of the slave tasks, then one can obtain better bounds.

Theorem 3.2.6 *Suppose* $a \geq c$. *Let S be a schedule that schedules jobs in nondecreasing order of the processing times of the slave tasks. Then,*

$$\frac{\sum_{k=1}^{n} C_k(S)}{\sum_{k=1}^{n} C_k(S^*)} < 1 + \frac{1}{4 + \frac{2c}{a}} , \tag{3.5}$$

which is a tight bound. If $a = c$, $\frac{\sum_{k=1}^{n} C_k(S)}{\sum_{k=1}^{n} C_k(S^*)} < \frac{7}{6}$ *and if* $a > c$, $\frac{\sum_{k=1}^{n} C_k(S)}{\sum_{k=1}^{n} C_k(S^*)} < \frac{5}{4}$.

Proof : Suppose $b_1 \leq b_2 \leq \ldots \leq b_n$. Since S schedules the jobs in this order, for any $1 \leq i \leq n$, $r_i = \max(na, ia + b_i)$. Therefore, $r_i \leq r_{i+1}$, which implies that $C_i < C_{i+1}$. Let j be the last job that finishes before the first idle time. The jobs will be divided into two subsets G_1 and G_2 as in the proof of Theorem 3.2.4. Let j and all the jobs that finish before j be in G_1. By the above analysis, $k \in G_1$ if and only if $k \leq j$. Because there is no idle time before j finishes, $C_k(S) = na + kc$ for all $k \leq j$. Thus, $\sum_{k \in G_1} C_k(S) = C^*_{G_1}$.

The remaining jobs are in G_2. Since there is idle time before c_{j+1} starts, we have $C_{j+1} = r_{j+1} + c = (j+1)a + b_{j+1} + c$. By assumption, $a \geq c$ and $b_{j+1} \leq b_{j+2}$. Therefore,

$C_{j+1} \leq (j+2)a + b_{j+2} = r_{j+2}$. Thus, $C_{j+2} = \max(C_{j+1}, r_{j+2}) + c = r_{j+2} + c$. Similarly, for any $j+1 \leq k \leq n$, one can derive that $C_k = r_k + c = ka + b_k + c$. Thus,

$$\begin{aligned}
\sum_{k=1}^{n} C_k(S) &= \sum_{k\in G_1} C_k(S) + \sum_{k\in G_2} C_k(S) \\
&= C^*_{G_1} + \sum_{k=j+1}^{n} (ka + b_k + c) \\
&= C^*_{G_1} + \left[\sum_{k=1}^{n-j} (ka + b_{k+j} + c)\right] + j(n-j)a \\
&\leq C^*_{G_1} + C^*_{G_2} + j(n-j)a \\
&\leq \sum_{k=1}^{n} C_k(S^*) + \frac{1}{4}n^2a \ .
\end{aligned}$$

By (3.1), $\sum_{k=1}^{n} C_k(S^*) \geq n^2 a + \frac{1}{2}n(n+1)c$. Therefore,

$$\frac{\sum_{k=1}^{n} C_k(S)}{\sum_{k=1}^{n} C_k(S^*)} \leq 1 + \frac{\frac{1}{4}n^2a}{n^2 a + \frac{1}{2}n(n+1)c} < 1 + \frac{1}{4 + \frac{2c}{a}} \ .$$

It is easy to see that if $a = c$, $\frac{\sum_{k=1}^{n} C_k(S)}{\sum_{k=1}^{n} C_k(S^*)} < \frac{7}{6}$, and if $a > c$, $\frac{\sum_{k=1}^{n} C_k(S)}{\sum_{k=1}^{n} C_k(S^*)} < \frac{5}{4}$.

To show that the bound is tight when $a = c$, set half of the jobs with $b_i = (\frac{1}{2}n-1)a$ and the other half with $b_i = (\frac{3}{2}n - 1)a$. For example, if $n = 6$ and $a = c = 1$, let the b_i's be $2, 2, 2, 8, 8, 8$. The optimal schedule will schedule the jobs with $b_i = 8$ first, then schedule the jobs with $b_i = 2$.

If $a > c$, let $b_1 = c$, $b_2 = a$ and $b_i = (i-1)(a+c)$ for $3 \leq i \leq n$. Then the optimal schedule schedules the jobs in nonincreasing order of the b_i's and the total completion time is $n^2a + \frac{1}{2}n(n+1)c$. However, scheduling the jobs in nondecreasing order of the b_i's gives the total completion time $\frac{5}{4}n^2a + \frac{1}{2}n(n+1)c$. □

It is clear that the schedule obtained by scheduling jobs in nondecreasing order of the processing times of the slave tasks must be an order preserving schedule. Since the optimal canonical and order preserving schedule can not have a total completion time smaller than $n^2a + \frac{1}{2}n(n+1)c$, the above bound also applies to canonical and order preserving schedules.

Corollary 3.2.7 *Given n jobs such that $a_i = a$ and $c_i = c$ for all $1 \leq i \leq n$. Suppose that $a > c$. Then, in $O(n \log n)$ time, one can find a canonical and order preserving schedule with total completion time at most $\frac{5}{4}$ of the minimum among all canonical and order preserving schedules.*

The case $a < c$ is considered next.

Theorem 3.2.8 *Suppose $a < c$. Let S be the schedule that schedules jobs in nondecreasing order of the processing times of the slave tasks. Then*

$$\frac{\sum_{k=1}^{n} C_k(S)}{\sum_{k=1}^{n} C_k(S^*)} < 1 + \frac{1}{2 + \frac{c}{a}} < \frac{4}{3} \ . \tag{3.6}$$

If $4a \leq c$, then the bound is $\frac{7}{6}$.

Proof : Suppose that $b_1 \leq b_2 \leq \ldots \leq b_n$. Let S be the schedule that schedules jobs in this order. The jobs are divided into groups according to their completion times. Those jobs that complete before the first idle interval are in G_0. Those jobs that complete after the first interval and before the second idle interval are in G_1. Those jobs that complete after the second interval and before the third idle interval are in G_2, and so on. Suppose there are $m + 1$ groups. Let the numbers of jobs in these groups be x_0, ..., x_m, respectively.

Let $S^*_{G_i}$ be the schedule that minimizes $\sum_{k \in G_i} C_k$ among all schedules of these n jobs. Let $C^*_{G_i} = \sum_{k \in G_i} C_k(S^*_{G_i})$. Since the jobs in G_0 complete one by one immediately after all a tasks finish under S, $\sum_{k \in G_0} C_k(S) = C^*_{G_0}$. For G_i, $1 \leq i \leq m$, let $y_i = \sum_{j=0}^{i-1} x_j$, i.e., y_i is the number of jobs scheduled before jobs of G_i. It will be shown in the following that $\sum_{j \in G_i} C_j(S) \leq C^*_{G_i} + x_i \cdot y_i \, a$.

Suppose that the jobs finish in the order $j_1, j_2, \ldots, j_{x_i}$ under $S^*_{G_i}$. Then $C_{j_1}(S^*_{G_i}) \geq a + b_{j_1} + c$ and $C_{j_k}(S^*_{G_i}) \geq C_{j_1}(S^*_{G_i}) + (k-1)c$, $1 \leq k \leq x_i$. So

$$C^*_{G_i} = \sum_{k=1}^{x_i} C_{j_k}(S^*_{G_i}) \geq \sum_{k=1}^{x_i} [C_{j_1}(S^*_{G_i}) + (k-1)c] \ .$$

By Theorem 3.1.1, S is order preserving. So job y_i+1 must finish first among all jobs of G_i. Because there is idle time before c_{y_i+1}, it must be true that $C_{y_i+1}(S) = (y_i+1)a + b_{y_i+1} + c$.

By assumption, $b_{y_i+1} \le b_{j_1}$. Therefore, $C_{y_i+1}(S) \le y_i a + (a + b_{j_1} + c) \le y_i a + C_{j_1}(S^*_{G_i})$. Since there is no idle time between jobs of G_i under S, for any other job $j \in G_i$, it must be true that $C_j(S) = C_{y_i+1}(S) + (j - y_i - 1)c$. Thus,

$$\begin{aligned}
\sum_{j \in G_i} C_j(S) &= \sum_{j=y_i+1}^{y_i+x_i} [C_{y_i+1}(S) + (j - y_i - 1)c] \\
&\le \sum_{j=y_i+1}^{y_i+x_i} [y_i a + C_{j_1}(S^*_{G_i}) + (j - y_i - 1)c] \\
&= x_i \cdot y_i a + \sum_{k=1}^{x_i} [C_{j_1}(S^*_{G_i}) + (k-1)c] \\
&= x_i \cdot y_i a + C^*_{G_i} \ .
\end{aligned}$$

Therefore,

$$\begin{aligned}
\sum_{k=1}^{n} C_k(S) &= \sum_{k \in G_0} C_k(S) + \sum_{i=1}^{m} \sum_{k \in G_i} C_k(S) \\
&< C^*_{G_0} + \sum_{i=1}^{m} \left(C^*_{G_i} + x_i \cdot y_i a \right) \\
&= \left(\sum_{i=0}^{m} C^*_{G_i} \right) + \left(\sum_{i=1}^{m} \sum_{j=0}^{i-1} x_i x_j a \right) \\
&\le \sum_{k=1}^{n} C_k(S^*) + \left(\sum_{i=1}^{m} \sum_{j=0}^{i-1} x_i x_j a \right) \\
&\le \sum_{k=1}^{n} C_k(S^*) + \frac{1}{2} \cdot \left(\sum_{i=0}^{m} x_i \right)^2 a \\
&= \sum_{k=1}^{n} C_k(S^*) + \frac{1}{2} n^2 a \ .
\end{aligned}$$

Thus,

$$\frac{\sum_{k=1}^{n} C_k(S)}{\sum_{k=1}^{n} C_k(S^*)} \le 1 + \frac{\frac{1}{2} n^2 a}{\sum_{k=1}^{n} C_k(S^*)} \le 1 + \frac{\frac{1}{2} n^2 a}{n^2 a + \frac{1}{2} n(n+1)c} < 1 + \frac{1}{2 + \frac{c}{a}} < \frac{4}{3} \ ,$$

and if $4a \le c$, then $\frac{\sum_{k=1}^{n} C_k(S)}{\sum_{k=1}^{n} C_k(S^*)} < \frac{7}{6}$. □

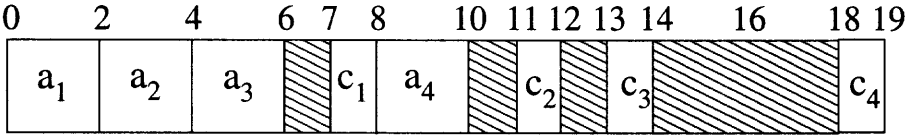


Figure 3.1 Illustration of Algorithm 1: jobs are $(2,5,1)$, $(2,7,1)$, $(2,7,1)$ and $(2,8,1)$.

3.3 Approximation Algorithms for No-wait-in Makespan

In this section two algorithms will be proposed to minimize the makespan of no-wait-in schedules in the special case; i.e. $a_i = a$ and $c_i = c$ for all i. The first algorithm is for the case when $a \geq c$, while the second algorithm is for the case when $a < c$.

Algorithm 1 - $a \geq c$

1. Sort the jobs in nondecreasing order of the processing times of the slave tasks. Suppose the sorted jobs are in the order of $1, 2, \ldots, n$.

2. Schedule task a_1 at time 0, b_1 at time a and c_1 at time $a + b_1$.

3. Repeat until all jobs are scheduled (see Figure 3.1):

   Suppose the jobs $1, 2, \ldots, i-1$ have been scheduled. Find the first idle interval $[t_1, t_2)$ before C_{i-1} such that $t_2 - t_1 = a$, schedule a_i at t_1 on the master machine, b_i at t_2 on the slave machine and c_i at $t_2 + b_i$ on the master machine. If no such idle interval exists, schedule a_i at time C_{i-1}, b_i at $C_{i-1} + a$ and c_i at $C_{i-1} + a + b_i$.

Theorem 3.3.1 *Let S be a schedule produced by Algorithm 1 for a given set of n jobs with preprocessing tasks a and postprocessing tasks c and $a \geq c$. Then S is a feasible no-wait-in schedule, and*

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

Proof : Suppose $b_1 \leq b_2 \leq \ldots \leq b_n$. The first step is to show that S is a feasible no-wait-in schedule. For convenience, let $t_a(i)$, $t_b(i)$ and $t_c(i)$ be the times that a_i, b_i and c_i start in S, respectively. By the algorithm, $t_b(i) = t_a(i) + a$, $t_c(i) = t_a(i) + a + b_i$. Thus, it is enough to show that S is a feasible schedule.

It is sufficient to show that after the jobs $1, 2, \ldots, i-1$ are scheduled, the intervals $[t_a(i), t_a(i) + a)$ and $[t_c(i), t_c(i) + c)$ are both idle. By the algorithm, it must be the case that $[t_a(i), t_a(i) + a)$ is idle. Besides, since the task a_i is scheduled to the first idle interval $[t_a(i), t_a(i) + a)$, it must be true that $t_a(i-1) + a \leq t_a(i)$. Recall that $t_c(i-1) = t_a(i-1) + a + b_{i-1}$ and $t_c(i) = t_a(i) + b_i + a$. Since $b_{i-1} \leq b_i$ and $c \leq a$,

$$C_{i-1} = t_c(i-1) + c = t_a(i-1) + a + b_{i-1} + c \leq t_a(i) + b_i + a = t_c(i) \ .$$

So, every task c_i starts after the task c_{i-1} completes, which means that the interval $[t_c(i), t_c(i) + c)$ must be idle.

By the above analysis, $C_{i-1} < C_i$ in S. Therefore, $C_{max}(S) = C_n = t_a(n) + a + b_n + c$. The first step is to show that $t_a(n) \leq 2\, C_{max}(S^*)$.

If there is no idle time before $t_a(n)$, then this is obviously true. Otherwise, there must be some idle intervals before $t_a(n)$. By the algorithm, the length of each idle interval before $t_a(n)$ must be less than a; additionally, there are two types of intervals: those that overlap with b_i's and those that are between C_i and $t_c(i+1)$ for some i, where $1 \leq i \leq n-2$. Let I_1 denote the total length of first type of idle intervals and I_2 denote the total length of the second type of idle intervals. Because only no-wait-in schedules are considered, the first type of idle intervals are inevitable even in an optimal schedule. There are at most $(n-1)$ idle intervals of the second type all of which are smaller than a. Hence,

$$\begin{aligned} t_a(n) &= \text{non-idle time before } t_a(n) + I_1 + I_2 \\ &< C_{max}(S^*) + I_2 \\ &< C_{max}(S^*) + (n-1)a \\ &< 2\, C_{max}(S^*) \ . \end{aligned}$$

On the other hand, $a + b_n + c \leq C_{max}(S^*)$. Therefore,

$$C_{max}(S) = t_a(n) + a + b_n + c < 3\, C_{max}(S^*) \ .$$

□

Note that if $a = c$, then $(n-1)a < \frac{1}{2} C_{max}(S^*)$. By the above analysis, $t_a(n) < \frac{3}{2} C_{max}$. Thus, $C_{max}(S) < \frac{5}{2} C_{max}(S^*)$. If $a = c = 1$ and all task times are integers, then there is no idle time of the second type before $t_a(n)$, so $t_a(n) < C_{max}(S^*)$. Therefore, $C_{max}(S) = t_a(n)+1+b_n+1 \leq 2\, C_{max}(S^*)$. The bound of 2 can be achieved asymptotically. Consider $n = 2m + 1$ jobs, where $a_i = c_i = 1$ for all $1 \leq i \leq n$, $b_i = 1$ for $1 \leq i \leq 2m$, and $b_{2m+1} = 4m$. The optimal schedule schedules job $2m + 1$ first, and all other jobs completely overlapping with b_{2m+1}. The ratio between the algorithm and the optimal solution for this instance is $1 + \frac{4m}{4m+2}$, which approaches 2 when m is large enough.

Note that Algorithm 1 actually produces order preserving schedules. Since the minimum order preserving and no-wait-in makespan cannot be smaller than the minimum no-wait-in makespan, one can obtain the following corollary.

Corollary 3.3.2 *Let* S *be a schedule produced by Algorithm 1 for a given set of* n *jobs with preprocessing tasks* a *and postprocessing tasks* c *and* $a \geq c$. *Then* S *is an order preserving and no-wait-in schedule and*

$$\frac{C_{max}(S)}{C_{max}(S^*)} \leq 3 \ ,$$

among all order preserving and no-wait-in schedule S^*.

Algorithm 2 - $a < c$

1. Sort the jobs in nondecreasing order of the processing times of the slave tasks. Suppose the sorted jobs are in the order of $1, 2, \ldots, n$.

2. Schedule task a_1 at time 0, b_1 at time a and c_1 at time $a + b_1$.

3. Repeat until all jobs are scheduled (see Figure 3.2):

   Suppose jobs $1, 2, \ldots, i-1$ have been scheduled. Find the first idle interval $[t_1, t_2)$ after $t_a(i-1)+a$ and before C_{i-1} such that $t_2 - t_1 = a$ and $t_2 + b_i \geq C_{i-1}$. Schedule a_i at t_1 on the master machine, b_i at t_2 on the slave machine and c_i at $t_2 + b_i$ on

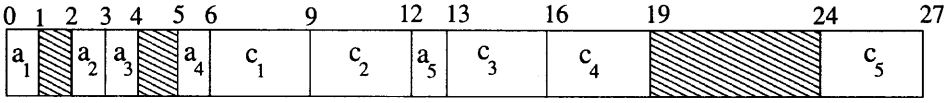


Figure 3.2 Illustration of Algorithm 2: jobs are $(1,5,3)$, $(1,6,3)$, $(1,9,3)$, $(1,10,3)$ and $(1,11,3)$.

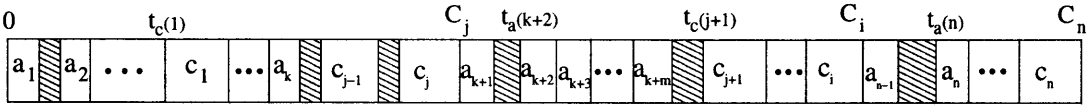


Figure 3.3 Illustration of the proof of Theorem 3.3.3. The idle interval between c_{j-1} and c_j has length less than a; the idle interval between a_{k+1} and a_{k+2} has length less than $c-a$ and the idle interval between a_{k+m} and c_{j+1} has length less than c.

the master machine. If no such idle interval exists, schedule a_i at time C_{i-1}, b_i at $C_{i-1}+a$ and c_i at $C_{i-1}+a+b_i$.

Theorem 3.3.3 *Let* S *be a schedule produced by Algorithm 2 for a given set of* n *jobs with preprocessing tasks* a *and postprocessing tasks* c *and* $a < c$. *Then* S *is a feasible no-wait-in schedule, and*

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

Proof : One can use a similar argument as in the proof of Theorem 3.3.1 to show that S is a feasible no-wait-in schedule.

Assume that $b_j \leq b_{j+1}$ for $1 \leq j \leq n-1$. By the algorithm, $C_1 < C_2 < \ldots < C_n$. Suppose that $C_i \leq t_a(n) < C_{i+1}$ for some i. In order to bound $C_{max}(S)/C_{max}(S^*)$, first consider the length of the idle times between C_j and $t_c(j+1)$ for $1 \leq j \leq i-1$. There are three cases depending on the number of a tasks scheduled between C_j and $t_c(j+1)$. Assume that a_k is the last task scheduled before $t_c(j)$. See Figure 3.3 for an illustration.

1. There is no a task scheduled between C_j and $t_c(j+1)$.

   In this case the whole period between C_j and $t_c(j+1)$ is idle. It will be proved by contradiction that $t_c(j+1) - C_j < a$.

Consider the time when job $k+1$ is going to be scheduled. By the algorithm, the jobs $1, \ldots, k$ have been scheduled at this time. Because a_k is scheduled before c_j, $t_a(k) + a \leq t_c(j)$. By assumption, $b_k \leq b_{k+1}$. Thus,

$$C_k = t_a(k) + a + b_k + c \leq t_c(j) + b_k + c = C_j + b_k < C_j + a + b_{k+1} \ .$$

Suppose that $t_c(j+1) - C_j \geq a$. Let $t_1 = C_j$ and $t_2 = C_j + a$. Then $t_2 \leq t_c(j+1)$. Thus, $[t_1, t_2)$ is an idle interval whose length is equal to a. According to the above inequality, $t_2 + b_{k+1} > C_k$. By the algorithm, a_{k+1} will be scheduled by t_1, which contradicts the assumption that a_{k+1} is scheduled after $t_c(j+1) \geq t_2$. Therefore, it must be true $t_c(j+1) - C_j < a$. In other words, the idle interval has length less than a which is less than c.

2. There is a single task a_{k+1} scheduled between C_j and $t_c(j+1)$.

   By the analysis in Case 1, a_{k+1} must be scheduled immediately after C_j. Thus, the only idle interval between C_j and $t_c(j+1)$ is $[C_j + a, t_c(j+1))$. It will be shown in the following that $t_c(j+1) - (C_j + a) < c$.

   Let $t_1 = C_j + a$ and $t_2 = t_c(j+1)$. Consider the time when the job $k+2$ is scheduled. By assumption, a_{k+2} is scheduled after t_2. Since $[t_1, t_2)$ is an idle interval, by the algorithm, there are only two possible reasons that a_{k+2} is not scheduled during the interval: (1) the length of the idle interval is less than a or (2) $t_2 - t_1 \geq a$ but $t_2 + b_{k+2} < C_{k+1}$. Since $C_{k+1} = t_1 + b_{k+1} + c$ and $b_{k+2} \geq b_{k+1}$, $t_2 - t_1 < c$. By the assumption, $a < c$. Therefore, in both cases,

   $$t_c(j+1) - (C_j + a) = t_2 - t_1 < c \ . \tag{3.7}$$

3. There are several a tasks $a_{k+1}, \ldots, a_{k+m}$ scheduled between C_j and $t_c(j+1)$.

   As in Case 2, the task a_{k+1} must be scheduled immediately after C_j. So there are at most m disjoint idle intervals during the period from C_j and $t_c(j+1)$. It will be shown that the length of each of these idle intervals is less than c.

First consider the length of the idle interval between $t_a(k+p)+a$ and $t_a(k+p+1)$, where $1 \leq p \leq m-1$. Without loss of generality, suppose that this interval has length greater than 0. If $(t_a(k+p)+a)+c > t_c(j+1)$, then the length of the idle interval is obviously less than c. So one may assume that $(t_a(k+p)+a)+c \leq t_c(j+1)$.

Let $t_2 = (t_a(k+p)+a)+c$ and $t_1 = t_2 - a = t_a(k+p)+c$. Consider the time when the job $k+p+1$ is being scheduled. It is clear that the interval $[t_1, t_2)$ must be idle at this moment. Since $b_{k+p+1} \geq b_{k+p}$, it is true that

$$t_2 + b_{k+p+1} = [(t_a(k+p)+a)+c] + b_{k+p+1} \geq t_a(k+p)+a+c+b_{k+p} = C_{k+p} \ .$$

Thus, by the algorithm, the task a_{k+p+1} must be scheduled at or before t_1; that is,

$$t_a(k+p+1) \leq t_1 = t_a(k+p)+c \ , \tag{3.8}$$

which means that

$$t_a(k+p+1) - (t_a(k+p)+a) \leq c-a < c \ . \tag{3.9}$$

For the interval $[(t_a(k+m)+a), t_a(j+1))$, one can use the same argument as in Case 2 to show that the length of the idle time during this period is less than c.

Now, one can bound the length of the interval $[t_c(1), t_a(n))$. The interval consists of three types of smaller intervals: (i) the intervals occupied by a tasks, (ii) the intervals occupied by c tasks and (iii) idle intervals.

Suppose that $a_1, a_2, \ldots, a_q$ are the a tasks scheduled before $t_c(1)$. Then, there are $(n-1-q)$ a tasks scheduled during the interval $[t_c(1), t_a(n))$. Therefore, the total length of the type (i) intervals is $(n-1-q)a$. Recall that it is assumed that $C_i \leq t_a(n) < C_{i+1}$. So the number of c tasks during the interval $[t_c(1), t_a(n))$ is i and the total length of the type (ii) intervals is ic.

By Cases 1, 2 and 3, the length of the idle interval after each a task is at most c. Since there are $(n-1-q)$ tasks during the interval $[t_c(1), t_a(n))$, the total length of the idle intervals after these a tasks is at most $(n-1-q)c$. If there is no a task between C_j

and C_{j+1} for $1 \leq j \leq i-1$, then there is an idle interval between C_j and C_{j+1} with length at most a. Since there are i jobs that finish before $t_a(n)$, the total length of the idle intervals between the jobs C_j and C_{j+1} with no a task between them and $1 \leq j \leq i-1$ is at most ia.

Thus, the length of the interval $[t_c(1), t_a(n))$ is

$$\begin{aligned} t_a(n) - t_c(1) &= \text{total length of type (i) intervals} + \text{total length of type (ii) intervals} \\ &\quad + \text{total length of type (iii) intervals} \qquad (3.10) \\ &\leq (n-1-q)a + ic + [(n-1-q)c + ia] \ . \end{aligned}$$

Next, bound $t_c(1)$. By the above assumption, $a_1, a_2, \ldots, a_q$ are scheduled before C_1. Using a similar analysis as in Cases 2 and 3, one can show that $t_a(j+1) - t_a(j) \leq c$ (see (3.8)) for $1 \leq j \leq q-1$, and $t_c(1) - (t_a(q) + a) < c$ (see (3.7)). Therefore,

$$t_c(1) = a + b_1 \leq t_a(q) + (t_c(1) - t_a(q)) \leq (q-1)c + (c+a) = qc + a \ . \qquad (3.11)$$

By (3.11) and (3.10),

$$\begin{aligned} t_a(n) &= t_c(1) + (t_a(n) - t_c(1) \\ &= t_c(1) + [(n-1-q)a + ic + ((n-1-q)c + ia)] \\ &= (a + b_1) + [(n-1-q)a + ic + ((n-1-q)c + ia)] \\ &\leq (qc + a) + [(n-1-q)a + ic + (n-1-q)c + ia] \\ &= (n-q+i)a + (n-1+i)c \\ &\leq 2n\,a + 2n\,c \\ &= 2n(a+c) \\ &\leq 2C_{max}(S^*) \ . \end{aligned}$$

Hence,

$$C_{max} = t_a(n) + a + b_n + c \leq 2\,C_{max}(S^*) + (a + b_n + c) \leq 3\,C_{max}(S^*) \ .$$

□

It is not known whether there is a set of jobs achieving a ratio of 3. However, one can show that the ratio can approach 2 asymptotically. Let $n = 2m + 1$, $a = 1$ and $c = 1 + \varepsilon$, where ε is arbitrarily small. Suppose $b_i = 1 + \varepsilon$ for $1 \leq i \leq 2m$ and $b_{2m+1} = m(4+3\varepsilon)$. The makespan of the schedule produced by Algorithm 2 is $2m(4+3\varepsilon)+(2+\varepsilon)$. However, the optimal schedule schedules job $2m + 1$ first and all other jobs completely overlapping with b_{2m+1}. Thus, the optimal makespan is $m(4 + 3\varepsilon) + (2 + \varepsilon)$. If m is very large, the ratio between the algorithm and the optimal solution can approach 2 arbitrarily closely.

Note that Algorithm 2 produces schedules that are order preserving. Since the minimum order preserving and no-wait-in makespan cannot be smaller than the minimum no-wait-in makespan, Theorem 3.3.3 implies the following corollary.

Corollary 3.3.4 *Let* S *be a schedule produced by Algorithm 2 for a given set of* n *jobs with preprocessing tasks* a *and postprocessing tasks* c *and* $a < c$. *Then* S *is an order preserving and no-wait-in schedule and*

$$\frac{C_{max}(S)}{C_{max}(S^*)} \leq 3 \ ,$$

among all order preserving and no-wait-in schedule S^*.

CHAPTER 4

APPROXIMATION ALGORITHMS: GENERAL CASES

In this chapter, the total completion time minimization problem will be considered under various scenarios. This includes single-master systems, multi-master systems, and distinct preprocessing and postprocessing master systems. Since the problem is strongly NP-hard, the focus will be on approximation algorithms. For each type of system, both the case when all jobs have the same release time and the case when the jobs have different release times will be considered. Algorithms are designed to approximate the best preemptive or non-preemptive schedules in these cases. The organization of this chapter is as follows. In Sections 4.3 - 4.5, algorithms are designed to minimize total completion times of preemptive schedules. Section 4.3 is devoted to the single-master case, Section 4.4 is devoted to the multi-master case, and Section 4.5 is devoted to the case of distinct preprocessor and postprocessor. In Section 4.6, conversions from preemptive schedules into non-preemptive schedules will be considered. In section 4.7, linear programming relaxation approach is applied to systems with distinct preprocessor and postprocessor.

4.1 Preliminaries

An α-*approximation* algorithm for makespan (or total completion time) is an algorithm that for any set of jobs generates a schedule S whose makespan (or total completion time) is at most α times the optimal makespan (or total completion time). It is an (α, β)-*approximation* algorithm if it is an α-approximation algorithm for makespan and at the same time a β-approximation algorithm for total completion time. For a schedule S, if $C_{max}(S) \leq \alpha C^*_{max}$ and $C(S) \leq \beta C^*$, then S is said to be an (α, β)-schedule.

The Shortest-Processing-Time (SPT) rule, which always runs the job with the least processing time, and the Shortest-Remaining-Processing-Time (SRPT) rule [107, 111],

which always runs the job with the least remaining processing time, are two well-known algorithms for minimizing total completion time. Usually, the SPT rule is used to generate non-preemptive schedules, while the SRPT rule is used to generate preemptive schedules. Suppose each job consists of a single task. If all jobs are available at time 0, then the SPT rule is optimal for total completion time in the single-machine or multi-machine environment. If the release times are arbitrary, then the SRPT rule is optimal for a single machine and it is a 2-approximation (see [95]) in the multi-machine environment.

In this chapter, both rules will be *adapted* for the problems in master slave model. Both are applied to generate *preemptive* schedules. A scheduling decision is made when an a task or a c task completes so that a master machine becomes free, or when a new a task or a c task becomes available. At any such time instant, the SPT rule schedules, from the set of available tasks (including those that have been preempted but have not yet completed), the one with the smallest processing time. Depending on how one chooses from the set of available jobs, one can obtain the SPT_a rule and the SPT_c rule. Specifically, in the SPT_a rule, preemption occurs only among the a tasks and the preemption is based on the length of a_i. In the SPT_c rule, preemption occurs only among the c tasks and the preemption is based on the length of c_i. On the other hand, the SRPT rule schedules, from the set of available tasks, the one with the smallest remaining processing time. Similarly, one can define the $SRPT_a$ rule and the $SRPT_c$ rule. Both the SPT rule and the SRPT rule may generate schedules with *migration* when there are multiple machines, i.e., after interrupted on one machine, a task is resumed on a different machine.

In this chapter, the jobs are frequently sorted in certain nondecreasing order of some parameters x_i of job i, e.g. $x_i = c_i$. For convenience, in this chapter $x_i < x_j$ is used as long as x_i comes before x_j in the sorted order, even though it may be the case that $x_i = x_j$. As in the previous chapters, one can always assume that the b tasks are scheduled immediately when they become available. Thus one can focus on the schedule of a tasks and c tasks only.

4.2 New Results and Techniques

Preemptive relaxation and linear programming relaxation are two important techniques for getting constant-factor approximations for total completion time of non-preemptive schedules; see [95, 59, 14, 45, 109, 108]. Most of these algorithms work by first constructing a relaxed solution, either a preemptive schedule or a linear programming relaxation. These relaxations are then used to obtain an ordering of the jobs, and the jobs are list scheduled (i.e., no unforced idle time) in this order. More about these methods will be discussed in Section 4.6.

The advantage of the preemptive relaxation is that usually there are very efficient algorithms to generate optimal or near-optimal schedules. In most cases, these algorithms (both preemptive and non-preemptive) can be implemented to run in an online fashion, see [95] and [59]. The linear programming relaxation, however, is more time consuming and can only be implemented to run in an offline fashion. On the other hand, it provides better approximation guaranties in some cases. Furthermore, the method usually works even if one wants to optimize the total weighted completion time; see [59, 14, 45, 109, 108].

Both approaches will be used in this chapter. In all cases, efficient algorithms are first developed to generate preemptive offline or online schedules with good approximation. These schedules are shown to have small makespan as well. Then, by applying the ideas in [95] and [16] to the master slave models, these preemptive schedules can be converted into non-preemptive schedules with certain degradation in the quality of approximation. In the case when there are distinct preprocessing masters and postprocessing masters, linear programming relaxation is also used. It is shown that non-preemptive schedules obtained in this way sometimes have better performance guaranties than those obtained by the preemptive relaxation approach. The results are summarized in Tables 4.1, 4.2, 4.3 and 4.4, where e is the base of natural logarithm.

Table 4.1 New Results for Single-Master System

	preemptive	non-preemptive
$r_i = 0$	$(\frac{3}{2}, 2)$, canonical	$(\frac{5}{2}, \frac{2e}{e-1})$
$r_i = 0$	$(2, 2)$, non-canonical	$(3, 4)$
$r_i \geq 0$	$(2, 2)$, online and non-canonical	$(3, 4)$

Table 4.2 New Results for Multi-Master System, All Schedules are Non-Canonical

	preemptive	non-preemptive
$r_i = 0$	$(2, 2)$, no migration	$(4, 4)$
$r_i \geq 0$	$(2, 2)$, offline with no migration	$(4, 4)$
$r_i \geq 0$	$(3, 2)$, online with migration	-

4.3 Single-master

This section assumes that there is a single master. Canonical schedules are studied first, and then non-canonical schedules. Finally, the case when jobs have different release times will be studied. For convenience, throughout this chapter, let $A = \sum_{j=1}^{n} a_j$, $B = \sum_{j=1}^{n} b_j$ and $C = \sum_{j=1}^{n} c_j$.

4.3.1 Canonical Preemptive Schedules

First, two lower bounds of the total completion time in canonical schedules are developed. Suppose there are n jobs 1, 2, ..., n. For canonical schedules, whether preemption is allowed or not, one can derive the following lower bound

$$C^* \geq nA + \sum_{j=1}^{n} \sum_{c_i \leq c_j} c_i \ , \tag{4.1}$$

Table 4.3 New Results for Distinct Preprocessor and Postprocesor System, $m_1 = m_2 = 1$

	preemptive	non-preemptive	
		preemptive relaxation	LP relaxation
$r_i = 0$	$(3,2)$	$(4, \frac{2e}{e-1})$	$(3,5)$
$r_i \geq 0$	$(3,2)$, offline	$(4, 4 + \frac{2e}{e-1})$	$(4,6)$
$r_i \geq 0$	$(3,2)$, online	$(4, 4 + \frac{2e}{e-1})$	-

Table 4.4 New results for distinct preprocessor and postprocesor, $m_1 \geq 1$ and $m_2 \geq 1$

	preemptive	non-preemptive	
		preemptive relaxation	LP relaxation
$r_i = 0$	$(4,2)$, with migration	$(4, \frac{14}{3})$	$(4,6)$
$r_i \geq 0$	$(4,3)$, offline with migration	$(5,13)$	$(5,7)$
$r_i \geq 0$	$(4,3)$, online with migration	$(5,13)$	-

which assumes that there is no idle time in the schedule and that the c tasks are scheduled in ascending order of their lengths. Another lower bound is

$$C^* \geq \left(\sum_{j=1}^{n} \sum_{a_i \leq a_j} a_i \right) + B + C \ , \tag{4.2}$$

which assumes that jobs are scheduled in increasing order of the a_i, and that the b tasks and the c tasks are scheduled immediately after they are available. Finally, the following trivial lower bound which is simply the summation of all the processing times holds for any schedule.

$$C^* \geq A + B + C \tag{4.3}$$

This follows from the observation that $C_j \geq a_j + b_j + c_j$. Summing j from 1 to n gives the result in (4.3).

In canonical schedules, all the a tasks are scheduled first. Since all the a tasks are available at time 0 and the c tasks cannot start until all the a tasks finish, there is no need to preempt the a tasks. Hence, only a c task can be preempted by another c task in this case.

Algorithm Canonical-SPT_c: Schedule the a tasks in an arbitrary order without preemption. After all the a tasks finish, schedule the available c tasks by the SPT_c rule.

Theorem 4.3.1 *Algorithm Canonical-SPT_c generates a $(2,2)$ canonical preemptive schedule in $O(n \log n)$ time.*

Proof : Let C_{a_j} denote the time a_j completes. Then at time $t_j = \max(A, (C_{a_j} + b_j))$, all the a tasks finish and the c_j is available to be scheduled. Since $C_{a_j} \leq A$, it must be true that $t_j \leq A + b_j$. According to the algorithm, if there is another available task c_i that hasn't finished at time t_j and $c_i < c_j$, then c_j has to wait until c_i finishes. Also, during the execution of c_j, if there is another task $c_i < c_j$ that becomes available, then c_i preempts c_j. In both cases, it is said that c_j is *delayed* by c_i. Let C_j be the completion time of c_j in the schedule generated by Algorithm Canonical-SPT_c. Then,

$$C_j = t_j + c_j + \sum_{c_i \text{ delays } c_j} c_i \leq (A + b_j + c_j) + \sum_{c_i < c_j} c_i \ .$$

The total completion time is

$$\sum_{j=1}^{n} C_j \leq \sum_{j=1}^{n} \left(A + b_j + c_j + \sum_{c_i < c_j} c_i \right) = \left(nA + \sum_{j=1}^{n} \sum_{c_i < c_j} c_i \right) + (B + C) < 2C^* \ ,$$

where the last inequality comes from the lower bounds (4.1) and (4.3).

The above approximation ratio is tight. Consider this example: for $1 \leq i \leq n-1$, $a_i = c_i = \varepsilon$, $b_i = a + (n-1)\varepsilon$, $a_n = c_n = a$ and $b_n = \varepsilon$. The optimal canonical schedule schedules $a_1, a_2, \ldots, a_{n-1}$ first and then followed by a_n. The total completion time is about $(n+1)a$. However, if one schedules a_n first followed by $a_1, \ldots, a_{n-1}$, then the total completion time is about $2na$. (For this example, the optimal preemptive schedule has the same total completion time as the optimal non-preemptive schedule.)

It has been shown in [106] that any canonical schedule without preemption is a 2-approximation for makespan. Since preemption among the c tasks can not increase the makespan, the schedule generated by Algorithm Canonical-SPT_c has makespan at most two times the optimal. □

Let $S_1 = \{i : a_i \leq c_i\}$ and $S_2 = \{i : a_i > c_i\}$. Suppose the jobs in S_1 are arranged in increasing order of the b's and the jobs in S_2 are arranged in decreasing order of the b's. In [106], it was shown that the canonical schedule in which the a tasks of S_1 are scheduled before the a tasks of S_2 has makespan at most $3/2$ times the optimal schedule. If the a tasks are scheduled in this order in Algorithm Canonical-SPT_c, then one still gets a $3/2$-approximation for makespan, since preemption on the available c tasks will not increase the makespan.

Corollary 4.3.2 *There is an* $O(n \log n)$ *time algorithm that generates a* $(3/2, 2)$ *canonical preemptive schedule.*

Note that when preemption occurs, algorithm Canonical-SPT_c uses SPT rule, instead of the SRPT rule. This is for the purpose of analysis only. In practice, one can use the SRPT rule to get a better approximation for total completion time.

4.3.2 Non-canonical Preemptive Schedules

In this model, the a tasks and the c tasks can be scheduled alternatively. A lower bound on the C^* can be obtained by assuming all the b tasks have length 0:

$$C^* \geq \sum_{j=1}^{n} \sum_{a_i + c_i \leq a_j + c_j} (a_i + c_i) \tag{4.4}$$

Algorithm Non-canonical-SPT_{a+c}: For any two jobs, if $(a_j + c_j) < (a_i + c_i)$, then both a_j and c_j are said to have higher priority than a_i and c_i. At any time, if the master processor is free for assignment, assign the available task with the highest priority. If a new task becomes available and has higher priority than the currently running task, the new task preempts the currently running task.

Theorem 4.3.3 *Algorithm Non-canonical-*SPT_{a+c} *generates a* $(2,2)$ *preemptive schedule for a single-master system when* $r_j = 0$ *for all job* j.

Proof : Since all the a tasks are ready at time 0, there is no need for an a task to preempt another task. Thus, preemption occurs only between a higher priority c task and a lower priority c task or a lower priority a task. Let C_{a_j} denote the completion time of a_j in the schedule generated by Algorithm Non-canonical-SPT_{a+c}. If none of the a tasks is preempted by a c task, then C_{a_j} would be $a_j + \sum_{a_i+c_i<a_j+c_j} a_i$. Because of preemption, C_{a_j} can be delayed by some higher priority c tasks. In other words, c_i can delay a_j only if $a_i + c_i < a_j + c_j$. At time $t_j = C_{a_j} + b_j$, the task c_j becomes available. According to the algorithm, c_j can only be delayed by a task c_i such that $a_i + c_i < a_j + c_j$. Note that if c_i delays a_j, it will not delay c_j again. Thus, one can bound the completion time C_j:

$$\begin{aligned}
C_j &\le C_{a_j} + b_j + c_j + \sum_{c_i \text{ delays } c_j} c_i \\
&= \left(\sum_{a_i+c_i<a_j+c_j} a_i + \sum_{c_i \text{ delays } a_j} c_i \right) + a_j + b_j + c_j + \left(\sum_{c_i \text{ delays } c_j} c_i \right) \\
&\le \left(\sum_{a_i+c_i<a_j+c_j} a_i \right) + \left(\sum_{a_i+c_i<a_j+c_j} c_i \right) + (a_j + b_j + c_j) \\
&= \left(\sum_{a_i+c_i<a_j+c_j} (a_i + c_i) \right) + (a_j + b_j + c_j) \ .
\end{aligned}$$

Therefore, the total completion time is

$$\begin{aligned}
\sum_{j=1}^{n} C_j &\le \sum_{j=1}^{n} \left(\left(\sum_{a_i+c_i<a_j+c_j} (a_i + c_i) \right) + a_j + b_j + c_j \right) \\
&\le \left(\sum_{j=1}^{n} \sum_{a_i+c_i<a_j+c_j} (a_i + c_i) \right) + (A + B + C) \qquad \text{by (4.4) and (4.3)} \\
&\le 2\,C^* \ .
\end{aligned}$$

To bound the makespan of the schedule, pick the last job j such that c_j is scheduled immediately when it is ready. Such a job always exists since the job i with the highest priority always satisfy this criterion. Then, the interval $I_1 = [0, C_{a_j})$ and the interval after

c_j starts till the end, i.e., $I_2 = [(C_{a_j} + b_j), C_{max})$, contain no idle time. Denote their lengths by $|I_1|$ and $|I_2|$, respectively. Then $|I_1| + |I_2| \leq C^*_{max}$. Thus, the makespan is

$$C_{max} = |I_1| + b_j + |I_2| \leq C^*_{max} + b_j < 2\,C^*_{max}$$

□

4.3.3 Arbitrary Release Times

When arbitrary release times are present, it is not meaningful to have canonical schedules any more. Thus, only non-canonical schedules will be considered. The lower bound (4.4) still holds for this case. Let $R = \sum_{i=1}^{n} r_i$. Then, a trivial lower bound for the minimum total completion time of any schedule is

$$C^* \geq \sum_{i=1}^{n} (r_i + a_i + b_i + c_i) = R + A + B + C \tag{4.5}$$

Observe that Algorithm Non-canonical-SPT_{a+c} makes no assumptions about the release time of a job. So, one can still apply Algorithm non-canonical-SPT_{a+c} when jobs have arbitrary release times. Unlike the case when all jobs have the same release times, in this case, a higher priority a task may also delay a lower priority a task or c task.

Theorem 4.3.4 *Algorithm Non-canonical-SPT_{a+c} generates a $(2, 2)$ preemptive online schedule for a single-master system even when the jobs have arbitrary release times.*

Proof : One can first bound the total completion time of the schedule generated by Algorithm Non-canonical-SPT_{a+c}. Let C_{a_j} be defined as before. Then,

$$C_{a_j} = r_j + \left(\sum_{a_i \text{ delays } a_j} a_i + \sum_{c_i \text{ delays } a_j} c_i \right) + a_j$$

and

$$\begin{aligned} C_j &= C_{a_j} + b_j + c_j + \left(\sum_{a_i \text{ delays } c_j} a_i + \sum_{c_i \text{ delays } c_j} c_i \right) \\ &= r_j + \left(\sum_{a_i \text{ delays } a_j} a_i + \sum_{c_i \text{ delays } a_j} c_i \right) + a_j + b_j + c_j + \sum_{a_i \text{ delays } c_j} a_i + \sum_{c_i \text{ delays } c_j} c_i \end{aligned}$$

$$\leq \left(\sum_{a_i+c_i<a_j+c_j} (a_i + c_i) \right) + (r_j + a_j + b_j + c_j) \ ,$$

where the last inequality comes from the fact that the two sets of tasks delaying a_j and c_j are disjoint, and they all have higher priority than a_j and c_j. Similarly, one can bound the total completion time

$$\begin{aligned} \sum_{j=1}^{n} C_j &\leq \sum_{j=1}^{n} \left(\left(\sum_{a_i+c_i<a_j+c_j} (a_i + c_i) \right) + (r_j + a_j + b_j + c_j) \right) \\ &\leq \left(\sum_{j=1}^{n} \sum_{a_i+c_i<a_j+c_j} (a_i + c_i) \right) + (R + A + B + C) \qquad \text{by (4.4) and (4.5)} \\ &< 2\,C^* \ . \end{aligned}$$

To bound the makespan, one picks the last job j such that c_j starts immediately after it is ready. Then the intervals $I_1 = [r_j, C_{a_j})$ and $I_2 = [(C_{a_j} + b_j), C_{max})$ must be both busy, and the total length $|I_1| + |I_2|$ of these two intervals is at most $\sum_{j=1}^{n} a_j + \sum_{j=1}^{n} c_j = A + C \leq C^*_{max}$. Thus,

$$C_{max} = r_j + |I_1| + b_j + |I_2| \ \leq \ (r_j + b_j) + C^*_{max} \ \leq \ 2\,C^*_{max} \ .$$

To conclude the proof, note that Algorithm Non-canonical-SPT_{a+c} schedules jobs in an online fashion. □

4.4 Multi-master

This section assumes that there are $m \geq 2$ masters, each of which is capable of processing both the a tasks and the c tasks.

4.4.1 Non-canonical Preemptive Schedules

Let us assume that the jobs are indexed in *nondecreasing* order of $a_j + c_j$. That is, $a_j + c_j \leq a_{j+1} + c_{j+1}$ for $1 \leq j \leq n-1$.

Algorithm FAM-SPT_{a+c}: (1) Assign the jobs in order of $n, n-1, \ldots, 1$ using FAM (First Available Machine, see [106]) rule: associate each machine i with a variable T_i. Initially all machine are available and $T_i = 0$. Pick the next unassigned job j and assign it to the machine i with the smallest T_i and after the assignment T_i is increased by an amount of $a_j + c_j$. Repeat this procedure until all jobs are assigned. (2) Apply Algorithm Non-canonical-SPT_{a+c} to each master machine to schedule the jobs assigned to it

Theorem 4.4.1 *Algorithm FAM-SPT_{a+c} generates a $(2,2)$ preemptive schedule for multi-master systems without migration when all jobs have release time 0.*

Proof : Without loss of generality, we may assume that $n = mk$ for some integer k. Otherwise, one can add dummy jobs with $a_i = b_i = c_i = 0$. For convenience, one can reindex the jobs assigned to each machine p in the form of (p, q) such that $a_{(p,q)} + c_{(p,q)} \leq a_{(p,q+1)} + c_{(p,q+1)}$. A lower bound of the total completion time comes from the fact that Algorithm FAM-SPT_{a+c} is optimal if $b_{(p,q)} = 0$ for every job (p, q).

$$C^* \geq \sum_{p=1}^{m} \sum_{q=1}^{k} (k - q + 1)(a_{(p,q)} + c_{(p,q)}) \tag{4.6}$$

Fix a machine p. Let $C_{(p,q)}$ denote the completion time of job (p, q) and $B_p = \sum_{q=1}^{k} b_{(p,q)}$. Following the analysis of Algorithm non-canonical-SPT_{a+c} in Section 4.3.2,

$$\sum_{q=1}^{k} C_{(p,q)} \leq \left(\sum_{q=1}^{k} (k - q + 1)(a_{(p,q)} + c_{(p,q)}) \right) + B_p \ .$$

Thus, the total completion time is

$$\begin{aligned} \sum_{p=1}^{m} \sum_{q=1}^{k} C_{(p,q)} &\leq \sum_{p=1}^{m} \left(\sum_{q=1}^{k} (k - q + 1)(a_{(p,q)} + c_{(p,q)}) + B_p \right) \\ &\leq \sum_{p=1}^{m} \left(\sum_{q=1}^{k} (k - q + 1)(a_{(p,q)} + c_{(p,q)}) \right) + B \qquad \text{by (4.6) and (4.3)} \\ &< 2\,C^* \ . \end{aligned}$$

Now analyze the makespan of the schedule. It is easy to see that

$$C^*_{max} \geq \frac{1}{m} \sum_{j=1}^{n} (a_j + c_j) \ .$$

Suppose the job with the maximum completion time among all jobs is assigned to machine p. Let l be the last job assigned to p. If there is no idle time on machine p, then the makespan is

$$\begin{aligned} C_{max} &= \sum_{j \text{ scheduled on } p, j \neq l} (a_j + c_j) + (a_l + c_l) \\ &\leq \frac{1}{m} \sum_{j=n}^{l+1} (a_j + c_j) + (a_l + c_l) \\ &\leq \frac{1}{m} \sum_{j=n}^{l} (a_j + c_j) + \frac{m-1}{m} (a_l + c_l) \\ &\leq (2 - \frac{1}{m}) C^*_{max} \end{aligned}$$

Otherwise, let l' be the last job on machine p so that $c_{l'}$ is scheduled immediately after it is ready. Define $C_{a_{l'}}$ as before. Then machine p is busy during the intervals $I_1 = [0, C_{a_{l'}})$ and $I_2 = [(C_{a_{l'}} + b_{l'}), C_{max})$. The makespan is

$$C_{max} = |I_1| + b_{l'} + |I_2| \leq \sum_{j \text{ scheduled on } p} (a_j + c_j) - (a_{l'} + c_{l'}) + (a_{l'} + b_{l'} + c_{l'})$$

By the above analysis, $\sum_{j \text{ scheduled on } p} (a_j + c_j)$ is at most $\frac{1}{m} \sum_{j=n}^{l} (a_j + c_j) + \frac{m-1}{m} (a_l + c_l)$ and by the ordering of the jobs one has $(a_l + c_l) < (a_{l'} + c_{l'})$. Thus

$$C_{max} \leq \frac{1}{m} \sum_{j=n}^{l} (a_j + c_j) + (a_{l'} + b_{l'} + c_{l'}) \leq 2\, C^*_{max}$$

□

4.4.2 Arbitrary Release Times

Offline schedule without migration In this case, one can still apply Algorithm FAM-SPT$_{a+c}$. However, note that to assign jobs to machines, Algorithm FAM-SPT$_{a+c}$ requires that one has full knowledge of all the jobs at the beginning. Thus, Algorithm FAM-SPT$_{a+c}$ is an offline algorithm. One can combine the arguments in Sections 4.3.3 and 4.4.1 to get the following theorem.

Theorem 4.4.2 *Algorithm FAM-*SPT_{a+c} *generates a* $(2,2)$ *offline preemptive schedule without migration for multi-master systems when jobs have arbitrary release times.*

Proof : Fix a machine p.

$$\sum_{q=1}^{k} C_{(p,q)} \leq \left(\max_{1\leq q\leq k} r_{(p,q)} + \sum_{q=1}^{k} (k-q+1)(a_{(p,q)} + c_{(p,q)}) \right) + B_p \ .$$

Thus, the total completion time is

$$\begin{aligned}
\sum_{p=1}^{m}\sum_{q=1}^{k} C_{(p,q)} &\leq \sum_{p=1}^{m} \left(\sum_{q=1}^{k} (k-q+1)(a_{(p,q)} + c_{(p,q)}) + B_p + \max_{1\leq q\leq k} r_{(p,q)} \right) \\
&\leq \sum_{p=1}^{m} \left(\sum_{q=1}^{k} (k-q+1)(a_{(p,q)} + c_{(p,q)}) \right) + B + R \qquad \text{by (4.6) and (4.5)} \\
&< 2C^* \ .
\end{aligned}$$

Now consider the makespan. Suppose the job with the maximum completion time among all jobs is assigned to machine p. Let l' be the last job on machine p so that $c_{l'}$ is scheduled immediately after it is ready. Define $C_{a_{l'}}$ as before. Then the machine p is busy during the intervals $I_1 = [r_{l'}, C_{a_{l'}})$ and $I_2 = [(C_{a_{l'}} + b_{l'}), C_{max})$. The the makespan is

$$\begin{aligned}
C_{max} &= r_l + |I_1| + b_l + |I_2| \\
&< \left(\sum_{\text{j assigned to machine p}} (a_j + c_j) \right) + (r_{l'} + b_{l'}) \\
&= \left(\sum_{\text{j assigned to machine p}} (a_j + c_j) \right) - (a_{l'} + c_{l'}) + (r_{l'} + a_{l'} + b_{l'} + c_{l'}) \ .
\end{aligned}$$

Let l be the last job assigned to machine p according to the FAM rule. As in the previous section one can show that

$$\begin{aligned}
\sum_{\text{j assigned to machine p}} (a_j + c_j) &\leq \left(\sum_{j=n}^{l+1} (a_j + c_j) \right) + (a_l + c_l) \\
&\leq \left(\frac{1}{m} \sum_{j=1}^{n} (a_j + c_j) \right) + \frac{m-1}{m} (a_l + c_l) \ .
\end{aligned}$$

Since l is the last job assigned to machine p, $(a_l + c_l) \leq (a_{l'} + c_{l'})$. It is known that $C^*_{max} > (r_{l'} + a_{l'} + b_{l'} + c_{l'})$, thus

$$C_{max} \leq= \left(\frac{1}{m}\sum_{j=1}^{n}(a_j + c_j)\right) + (r_{l'} + a_{l'} + b_{l'} + c_{l'}) \leq 2\,C^*_{max} \ .$$

□

Online schedule with migration In this case, one can apply Algorithm non-canonical-SPT_{a+c} to multi-master systems. Note that if a new task becomes available and one or more currently running tasks have lower priority than the new task, then the task with the lowest priority will be preempted.

Theorem 4.4.3 *Algorithm non-canonical-*SPT_{a+c} *generates a* $(3, 2)$ *online preemptive schedule with migration on multi-master systems when jobs have arbitrary release times.*

Proof : The proof needs another lower bound for multi-master systems. Consider the instance I defined by $(a_i/m, b_i, c_i/m)$, $1 \leq i \leq n$, on a single-master system. Then, by the bound of (4.4), its total completion time is at least $\sum_{a_i+c_i \leq a_j+c_j}(a_i + c_i)/m$. This can be shown to be also a lower bound for the instance II defined by (a_i, b_i, c_i), $1 \leq i \leq n$, on m masters. Let S^* be an optimal schedule of these n jobs. Then, starting from time 0, for each time unit, if a_i or c_i is scheduled in this unit on some machine, then one can schedule $1/m$ of the task to the single master in the same time unit. It is easy to see that the constraint of release time are preserved and that the interval between the finish time of a_i and the start time of c_i is either the same or increased. Thus one obtains a valid schedule of the instance I on the single-master system. Therefore,

$$C^* \geq \sum_{j=1}^{n} \sum_{a_i+c_i \leq a_j+c_j} (a_i + c_i)/m \ , \tag{4.7}$$

which is a lower bound of the original instance defined by (r_i, a_i, b_i, c_i).

Let C_{a_j} denote the completion time of a_j in the schedule generated by Algorithm non-canonical-SPT_{a+c}. After a_j is released at r_j, it may be delayed by other tasks with higher priority before it completes. That is, a_i or c_i can delay a_j only if $a_i + c_i < a_j + c_j$. It

is easy to see that there is no idle time on any machine during the interval $I_1 = [r_j, (C_{a_j} - a_j))$; otherwise, a_j would have completed earlier. Furthermore, only tasks with higher priority than j can be scheduled during this interval. At time $t_j = C_{a_j} + b_j$, the task c_j becomes ready. Similarly, there is no idle time on any machine during the interval $I_2 = [C_{a_j} + b_j, (C_j - c_j))$, and only tasks with higher priority than j can be scheduled during this interval. Note that the task sets in the intervals I_1 and I_2 are disjoint. Let $|I_1|$ and $|I_2|$ denote the lengths of I_1 and I_2, respectively. Then, it must be true that $|I_1| + |I_2| \leq \sum_{a_i+c_i<a_j+c_j} (a_i + c_i)/m$. Thus, one can bound the completion time C_j as follows:

$$\begin{aligned} C_j &\leq r_j + |I_1| + a_j + b_j + |I_2| + c_j \\ &\leq \left(\sum_{a_i+c_i<a_j+c_j} (a_i + c_i)/m \right) + (r_j + a_j + b_j + c_j) \ . \end{aligned}$$

The total completion time is

$$\begin{aligned} \sum_{j=1}^{n} C_j &\leq \sum_{j=1}^{n} \left(\left(\sum_{a_i+c_i<a_j+c_j} (a_i + c_i)/m \right) + (r_j + a_j + b_j + c_j) \right) \\ &\leq \left(\sum_{j=1}^{n} \sum_{a_i+c_i<a_j+c_j} (a_i + c_i)/m \right) + (R + A + B + C) \qquad \text{by (4.7) and (4.5)} \\ &< 2\,C^* \ . \end{aligned}$$

For the makespan of the schedule, let k be the job with the maximum completion time among all jobs. Let l be the last job among all jobs so that c_l is scheduled immediately after it is ready. Define C_{a_l} as before. Then, all machines must be busy during the intervals $I_1 = [r_l, C_{a_l} - a_l)$ and $I_2 = [(C_{a_l} + b_l), C_{max} - c_k)$. The total length of the two intervals is $|I_1| + |I_2| \leq C^*_{max}$. Therefore, the makespan is

$$C_{max} = r_l + |I_1| + a_l + b_l + |I_2| + c_k = (r_l + a_l + b_l) + (|I_1| + |I_2|) + c_k < 3\,C^*_{max} \ .$$

□

4.5 Distinct Preprocessing and Postprocessing Masters

In this section, the model with distinct preprocessing masters and postprocessing masters will be discussed. Different from the previous cases, here an a task can only be preempted by another a task and a c task can only be preempted by another c task. In all cases, Algorithm $SRPT_a$-SPT_c will be applied to obtain a preemptive schedule.

Algorithm $SRPT_a$-SPT_c**:** Schedule the available a tasks using the $SRPT_a$ rule on the preprocessing master. Schedule the available c tasks using the SPT_c rule on the postprocessing master.

Let m_1 and m_2 denote the numbers of preprocessing masters and postprocessing masters, respectively. First the simple case $m_1 = m_2 = 1$ will be studied.

Theorem 4.5.1 *Algorithm* $SRPT_a$-SPT_c *generates a* $(3,2)$ *online preemptive schedule when* $m_1 = m_2 = 1$ *and* $r_j \geq 0$ *for all* j.

Proof : First consider the total completion time. Let $C^*_{a_j}$ be the time a_j finishes in an optimal schedule. Then,

$$C^* \geq \sum_{j=1}^{n} (C^*_{a_j} + b_j + c_j) = \left(\sum_{j=1}^{n} C^*_{a_j} \right) + B + C \ .$$

Let C_{a_j} be the time a_j finishes in the schedule obtained by Algorithm $SRPT_a$-SPT_c. Since the $SRPT_a$ rule is optimal if $b_j = c_j = 0$, Algorithm $SRPT_a$-SPT_c must have the minimum $\sum_{j=1}^{n} C_{a_j}$ among all possible schedules. That is

$$\sum_{j=1}^{n} C_{a_j} \leq \sum_{j=1}^{n} C^*_{a_j} \ .$$

Thus, the total completion time is at most

$$\sum_{j=1}^{n} \left(C_{a_j} + b_j + c_j + \sum_{c_i \text{ delays } c_j} c_i \right) \leq \sum_{j=1}^{n} (C_{a_j} + b_j + c_j) + \sum_{j=1}^{n} \sum_{c_i < c_j} c_i \leq 2C^* \ .$$

For the makespan, consider the last job l such that c_l runs immediately when it is available at time $C_{a_l} + b_l$. There is no idle time in the interval $I_1 = [r_l, C_{a_l})$ and the interval

$I_2 = [(C_{a_l} + b_l), C_{max})$. The length of each interval is at most C^*_{max}. Therefore, the makespan is

$$C_{max} = r_l + |I_1| + b_l + |I_2| \leq 3\, C^*_{max} \ .$$

□

Theorem 4.5.2 *Algorithm* $SRPT_a$-SPT_c *generates a* $(4, 2)$ *preemptive schedule with migration when* $m_1 \geq 1$, $m_2 \geq 1$ *and* $r_j = 0$ *for all* j .

Proof : Since all a tasks are available at time 0, then the $SRPT_a$ rule is the same as the SPT_a rule. As mentioned before, the SPT_a rule is optimal when the b tasks and the c tasks have zero length; that is, it minimizes the total completion time of the a tasks. Let $C^*_{a_i}$ be the finish time of a_i in an optimal schedule, and let C_{a_i} be the finish time of a_i in the schedule generated by Algorithm $SRPT_a$-SPT_c. Then, as in the case of $m_1 = m_2 = 1$, a lower bound of the total completion time is

$$C^* \geq \sum_{i=1}^{n} (C^*_{a_i} + b_j + c_j) \ = \ \left(\sum_{i=1}^{n} C^*_{a_i}\right) + B + C \ \geq \ \left(\sum_{i=1}^{n} C_{a_i}\right) + B + C \qquad (4.8)$$

When the task c_j is ready, it can be delayed by a task c_i only if $c_i < c_j$. The length of the interval $[(C_{a_j} + b_j), C_j - c_j)$ is at most $\sum_{c_i<c_j} \frac{c_i}{m_2}$, since all postprocessing masters must be busy and can only run the task c_i such that $c_i < c_j$ during this interval. Hence the total completion time is at most

$$\sum_{j=1}^{n} C_j \leq \sum_{j=1}^{n} (C_{a_j} + b_j + c_j + \sum_{c_i<c_j} \frac{c_i}{m_2}) \ = \ \left(\sum_{j=1}^{n} C_{a_j} + B + C\right) + \sum_{j=1}^{n} \sum_{c_i<c_j} \frac{c_i}{m_2} \ \leq \ 2\, C^* \ ,$$

where the last inequality comes from (4.7) and (4.8).

Now the makespan is shown to be at most four times the optimal makespan. As before, let k be the job with the maximum completion time, and let l be the last job such that the task c_l runs immediately after it is ready at $C_{a_l} + b_l$. Then the intervals $I_1 = [0, C_{a_l} - a_l)$ and $I_2 = [(C_{a_l} + b_l), (C_{max} - c_k))$ must be both busy. And $|I_1| \leq \sum_{a_j<a_l} a_j/m_1 < C^*_{max}$ and $|I_2| \leq \sum_{c_j<c_l} c_j/m_2 < C^*_{max}$. Therefore,

$$C_{max} = |I_1| + (a_l + b_l) + |I_2| + c_k < 4\, C^*_{max}$$

□

Theorem 4.5.3 *Algorithm* $SRPT_a$-SPT_c *generates a* $(4,3)$ *preemptive schedule with migration when* $m_1 \geq 1$, $m_2 \geq 1$ *and* $r_j \geq 0$ *for all* j.

Proof : Let $C^*_{a_j}$ be the completion time of a_j in an optimal schedule. As in the last section, one can get

$$C^* \geq \left(\sum C^*_{a_j}\right) + B + C$$

Let C_{a_j} be the completion time of a_j in the schedule generated by Algorithm $SRPT_a$-SPT_c. As mentioned before, the $SRPT_a$ rule is a 2-approximation when $b_j = c_j = 0$. Thus, it must true that

$$\sum C_{a_j} \leq 2 \sum C^*_{a_j}$$

and

$$\begin{aligned}
\sum_{j=1}^{n} C_j &= \sum_{j=1}^{n} \left(C_{a_j} + b_j + c_j + \sum_{c_i < c_j} c_i/m_2 \right) \\
&\leq \left(\sum_{j=1}^{n} C_{a_j} \right) + B + C + \sum_{j=1}^{n} \sum_{c_i < c_j} c_i/m_2 \\
&\leq \sum C^*_{a_j} + \left(\sum C^*_{a_j} + B + C \right) + \sum_{j=1}^{n} \sum_{c_i \leq c_j} c_i/m_2 \\
&\leq 3\, C^*
\end{aligned}$$

Now consider the makespan. As before, let k be the job with the maximum completion time, and let l be the last job such that c_l runs immediately after it is ready at time $C_{a_l} + b_l$. Then the intervals $I_1 = [r_l, C_{a_l} - a_l)$ and $I_2 = [(C_{a_l} + b_l), (C_{max} - c_k))$ must be both busy. And $|I_1| \leq \sum_{a_j < a_l} a_j/m_1 < C^*_{max}$ and $|I_2| \leq \sum_{c_j < c_l} c_j/m_2 < C^*_{max}$. Therefore,

$$C_{max} = r_l + |I_1| + (a_l + b_l) + |I_2| + c_k < 4\, C^*_{max}$$

□

4.6 Converting Preemptive Schedules into Non-preemptive Schedules

As mentioned before, non-preemptive schedules can be obtained by converting preemptive schedules. Our approach is based on the techniques introduced by Phillips et al. in [95], and improved by Chekuri et al. in [16]. For completeness, their approaches are described in the following.

The model studied in [95] consists of one or more identical machines and n jobs. For this model, a general approach of converting a preemptive schedule S into a non-preemptive schedule S' has been given in [95]. The idea is to form a list of jobs in increasing order of their completion times in S and then list schedule the jobs in this list one by one, respecting their release times.

Let C_j and C'_j denote the completion time of job j in S and S', respectively. Suppose the jobs are indexed such that $C_i < C_{i+1}$. Then they showed that $C'_j \leq 2C_j$ for a single machine environment and $C'_j \leq 3C_j$ for a multi-machine environment. Let $r^j_{max} = \max_{1 \leq i \leq j} r_i$. The result is based on the observations that (1) $C_j > r^j_{max}$, (2) $C_j \geq \sum_{i=1}^{j} p_i$ in the single-machine case, and $C_j \geq \sum_{i=1}^{j} p_i/m$ in the multi-machine case, where p_j is the processing time of job j, (3) $C'_j \leq r^j_{max} + \sum_{i=1}^{j} p_i$ if there is one machine and $C'_j \leq r^j_{max} + (\sum_{i=1}^{j-1} p_i/m) + p_j$ if there are two or more machines. These results imply that if S is a β-approximation for total completion time, then S' is a 2β-approximation for total completion time in the single-machine environment, and a 3β-approximation for total completion time in the multi-machine environment. In addition, this conversion also yields an online non-preemptive algorithm if the preemptive schedule can be generated online: simply simulate the algorithm for preemptive schedule, start a job j if j completes in the preemptive schedule or put it in the waiting queue if the machine is busy.

Now, consider the makespan which is equal to the completion time of job n in S'. It is easy to see that $C'_n \leq C_n + \sum_{j=1}^{n} p_j \leq C_n + C^*_{max}$. Therefore, if S is an α-approximation for makespan, i.e., $C_{max}(S) = C_n \leq \alpha C^*_{max}$, then $C_{max}(S') = C'_n \leq (1+\alpha)\, C^*_{max}$. In other words, S' is a $(1+\alpha)$-approximation for makespan.

Later, Chekuri et al. [16] improved the above results for total completion time. They designed a deterministic $O(n^2)$ time *offline* algorithm such that the schedule obtained has total completion time at most $\frac{e}{e-1} \approx 1.58$ times that of the preemptive schedule. They also gave a randomized *online* algorithm which generates schedules having *expected* completion time $\frac{e}{e-1}$ times that of the preemptive schedule. The difference between the two approaches in [16] and [95] is how to obtain the list of jobs. Given S and a parameter $\lambda \in (0, 1]$, let $C_j^\lambda(S)$, the λ-point of j, be the time when λp_j (a λ-fraction) of job j is completed. Instead of forming a list based on $C_j(S)$, now form a list based on $C_j^\lambda(S)$. A λ-schedule is a non-preemptive schedule obtained by list scheduling jobs in increasing order of $C_j^\lambda(S)$, possibly introducing idle time to account for the release times. It is easy to see that the algorithm given by [95] is a λ-schedule with $\lambda = 1$. It is also clear that a λ-scheduler can be made to be an on-line algorithm if the underlying preemptive algorithm is an on-line algorithm.

The main result in [16] is that for each given instance, there exists a λ, the best λ, such that the λ-schedule has total completion time at most $\frac{e}{e-1}$ times that of S, and has makespan at most $(1 + \lambda)$ times that of S. One can obtain such a λ-schedule by finding the best λ in $O(n^2)$ time offline deterministically; or one can obtain, through a randomized on-line algorithm, a schedule whose *expected* total completion time is at most $\frac{e}{e-1}$ times that of S and whose makespan is at most $(1 + \lambda)$ times that of S.

In the multi-machine case, Chakrabarti et al. [14] showed that the convert procedure given in [95] has a bound of $7/3$ for total completion time, instead of 3 times that of S.

The following sections describe how to convert preemptive schedules generated in Sections 4.3-4.5 to non-preemptive schedules. The difficulty of the conversion in the master slave model is that one needs to respect not only the release time of a_i, $1 \leq i \leq n$, but also respect the constraint that the interval between the finish time of a_i and the start time of c_i has length at least b_i.

4.6.1 Single Master and Multi-Master Systems

First consider the single master systems.

Theorem 4.6.1 *In $O(n^2)$ time, one can obtain a $(\frac{5}{2}, \frac{2e}{e-1})$ non-preemptive canonical schedule when there is a single master and $r_j = 0$ for all j.*

Proof : Let S be a preemptive canonical schedule of n jobs obtained by applying Corollary 4.3.2. Let S_a be the partial schedule of S during the interval $(0, A]$, and S_c be the partial schedule of S_c during the interval $(A, C_{max}]$.

Clearly S_a contains all a tasks only. By the Algorithm Canonical-SPT$_c$, there is no preemption in S_a. Let C_{a_j} be the completion time of a_j. It is easy to see that the partial schedule S_c contains all c tasks only and it can be seen as a preemptive schedule of n tasks on a single machine where each task j has a "release time" $\max(A, C_{a_j} + b_j)$ and processing time c_j.

To convert S into a non-preemptive schedule S', one fixes S_a and convert S_c to a non-preemptive schedule S'_c of c tasks by using the approach of [16]. Let C_j and C'_j be the completion time of c_j in S and S', respectively. As mentioned at the beginning of the section, it has been shown in [16] that $C'_j \leq \frac{e}{e-1} C_j$ and $C'_j \leq C_j + C_{max}$. Since S is a $(\frac{3}{2}, 2)$ canonical schedule, the obtained schedule is a $(\frac{5}{2}, \frac{2e}{e-1})$ non-preemptive canonical schedule. This concludes the proof. □

Theorem 4.6.2 *In $O(n \log n)$ time, one can obtain a $(3, 4)$ online non-preemptive schedule when there is a single master and $r_j \geq 0$ for all job j.*

Proof : Let S be a non-canonical preemptive schedule S. One can get a λ-schedule S', $\lambda = 1$, similarly as [95]: (1) Sort all tasks (both a tasks and c tasks) in increasing order of their completion times. (2) List schedule these tasks on the master machine, with the constraint that each task a_j must start after r_j and the interval between the time a_j finishes and the time c_j starts is at least b_j.

For the purpose of analysis, one can visualize the above procedure as follows (see Figure 4.1): For each task, a_j or c_j, remove all but the last scheduled piece of the task. Suppose the last piece of a_j and c_j have length k_{a_j} and k_{c_j}, respectively. Now process the tasks one by one in the scheduling order of their last pieces in S. If the current task is a_j, complete its last piece by inserting an extra piece with length $(a_j - k_{a_j})$ immediately after the last piece of a_j; at the same time push backward in time all the last pieces of the tasks which finish after a_j in S by an amount of $(a_j - k_{a_j})$. If the task is c_j, complete its last piece by inserting an extra piece with length $(c_j - k_{c_j})$ immediately after the last piece of c_j; at the same time push backward in time all the last pieces of the tasks which finish after c_j in S by an amount of $(c_j - k_{c_j})$. Let the schedule be S''. It is easy to see that during this process, the two constraints, (a) each task a_j is scheduled after r_j and (b) the interval between the time a_j finishes and the time c_j starts is at least b_j, are not violated. Thus S'' is still a feasible schedule.

Now one pushes all jobs forward in time as much as possible without changing the order of the tasks, or violating the constraints (a) and (b) mentioned above. The result is exactly the schedule S'. Since a task c_j can only be moved back by processing times associated with tasks finished earlier than c_j in S, we have $C'_j \leq 2C_j$ and $C'_j \leq C_j + \sum_{i=1}^{n}(a_i + c_i) \leq C_j + C_{max}$, where C'_j is the completion time of c_j in S'. This implies that if one takes the $(2,2)$ non-canonical schedule in Theorem 4.3.4, then one can get a $(3,4)$ non-preemptive schedule. Furthermore, it can be implemented online if the preemptive schedule is online. □

The following theorem shows how to get offline non-preemptive schedules for multi-master systems. It is not known how to obtain an online non-preemptive schedule.

Theorem 4.6.3 *When there are multi-masters, one can obtain a* $(4,4)$ *non-preemptive offline schedule.*

Proof : Let S be the (2,2)-schedule generated by Algorithm FAM-SPT_{a+c}. Then S has no migration. One can obtain the non-preemptive schedule S' by converting the schedule

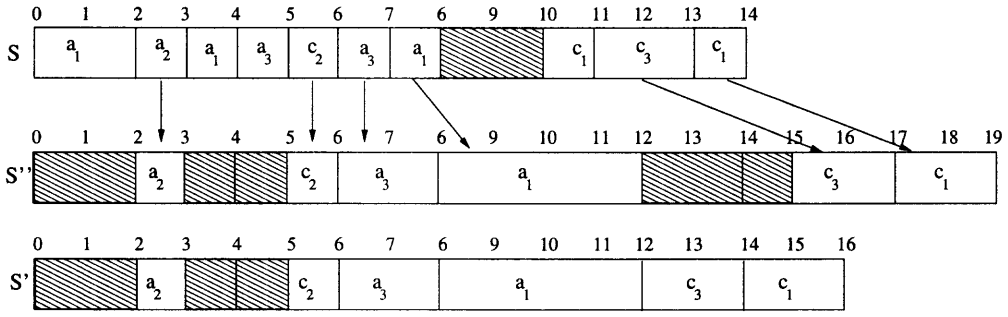


Figure 4.1 Convert a preemptive schedule into a non-preemptive schedule: S is a preemptive schedule of three jobs $J_1(0,4,2,2)$, $J_2(2,1,2,1)$, $J_3(4,2,4,2)$; S'' is the non-preemptive schedule obtained from S by completing the last piece of each task into a whole piece; S' is the non-preemptive schedule obtained from S'' by removing unnecessary idle intervals in S''.

on each machine separately in the same way as described in the proof of Theorem 4.6.2. Thus, the total completion time of S' is at most two times that of S. Since S is a 2-approximation for total completion time, S' is a 4-approximation for total completion time. For the makespan, $C'_j \leq C_j + \max_{i=1}^{m} \Delta_i$ where Δ_i is the total length of the a tasks and the c tasks assigned on machine i. In Section 4.4, it has been shown that $\Delta_i \leq 2\,C^*_{max}$. Thus, $C'_j \leq C_j + \max_{i=1}^{m} \Delta_i \leq C_j + 2\,C^*_{max}$. Since S is 2-approximation for makespan, S' is a 4-approximation for the makespan. This concludes the proof. □

4.6.2 Distinct Preprocessors and Postprocessors

This subsection considers the case when there are m_1 preprocessors and m_2 postprocessors.

Theorem 4.6.4 *In* $O(n^2)$ *time, one can obtain a* $(4, \frac{2e}{e-1})$ *non-preemptive schedule when* $r_j = 0$ *for all* j, *and* $m_1 = m_2 = 1$.

Proof : Let S be the $(3,2)$-schedule generated by Algorithm $SRPT_a$-SPT_c. Since $r_j = 0$ for all j, there is no preemption on the single preprocessor. Let C_{a_j} be the completion time of a_j in S. To get a non-preemptive schedule, one can fix the schedule of the a tasks and do the conversion simply on the single postprocessor using exactly the approach given in [16],

respecting the "release time" of task c_j $(C_{a_j} + b_j)$. Following exactly the same argument, one can show that S' is a $(4, \frac{2e}{e-1})$ non-preemptive schedule. □

When the release times are arbitrary, one needs to do the conversion carefully so as to make sure that the difference between the finish time of a_j and the start time of c_j is at least b_j.

Theorem 4.6.5 *When* $m_1 = m_2 = 1$, *and* $r_j \geq 0$, *one can obtain a* $(4, 4 + \frac{2e}{e-1})$ *non-preemptive offline schedule or an online non-preemptive schedule with expected performance* $(4, 4 + \frac{2e}{e-1})$.

Proof : Let S be the $(3,2)$-schedule generated by Algorithm $SRPT_a$-SPT_c. Let C_j be the completion time of job j in S. Let C_{a_j} be the completion time of a_j in S.

The conversion consists of two steps. First one can remove the preemptions among the a tasks to get the best (offline) λ-schedule of the a tasks on the single preprocessor by using the approach of [16]. Let C'_{a_j} be the new completion time of a_j. Then, one must have $C'_{a_j} \leq \frac{e}{e-1} C_{a_j}$. Now fix the schedule of a tasks, and remove the preemptions among the c tasks to get a λ-schedule of the c tasks, where $\lambda = 1$, and make sure the interval between C'_{a_j} and the start time of c_j is at least b_j for each job j.

Suppose the jobs are indexed so that $C_i < C_{i+1}$. Then, $C_j \geq \max(\max_{i=1}^{j} r_i, \max_{i=1}^{j} b_i, \sum_{i=1}^{j} a_i, \sum_{i=1}^{j} c_i)$. Let the new schedule be S' and let the completion time of c_j in S' be C'_j. Then

$$C'_j \leq C'_{a_j} + \max_{1 \leq i \leq j} b_i + \sum_{i=1}^{j} c_i \leq \frac{e}{e-1} C_{a_j} + 2 C_j \leq (2 + \frac{e}{e-1}) C_j .$$

Thus, $\sum_{j=1}^{n} C'_j \leq (2 + \frac{e}{e-1}) \sum_{j=1}^{n} C_j$. By assumption, S is 2-approximation for total completion time. Thus, S' is $(4 + \frac{2e}{e-1})$-approximation for total completion time.

For the makespan,

$$C'_j \leq \max_{1 \leq i \leq j} r_i + \sum_{C_{a_i} \leq C_{a_j}} a_i + \max_{1 \leq i \leq j} b_i + \sum_{i=1}^{j} c_i \leq 4 C^*_{max} .$$

Thus, S' is a $(4, 4 + \frac{2e}{e-1})$ non-preemptive offline schedule. Similarly as in [16], one can also get an online schedule whose expected performance is $(4, 4 + \frac{2e}{e-1})$. This concludes the proof. □

Theorem 4.6.6 *In* $O(n \log n)$ *time, one can obtain a* $(4, 14/3)$ *non-preemptive schedule when* $r_j = 0$ *for all* j, $m_1 \geq 1$ *and* $m_2 \geq 1$.

Proof : Let S be the $(4, 2)$-schedule generated by Algorithm $SRPT_a$-SPT_c. Since all jobs have the same release time, no preemption occurs on the preprocessors. Now fix the schedule of the a tasks. Let C_{a_j} be the completion time of a_j in S. The task c_j can be seen as a task with release time $(C_{a_j} + b_j)$. The conversion is performed on the postprocessors using the approach given in [95], subject to the constraint that c_j can not start earlier than its "release time" $(C_{a_j} + b_j)$. Then by [14], the total completion time of S' is at most $\frac{7}{3}$ times that of S, i.e., S' is a $\frac{14}{3}$-approximation for total completion time.

Now consider the makespan. Suppose the jobs are indexed such that $C_i \leq C_{i+1}$ in S. By the algorithm, for any $1 \leq i \leq n$, $C_{a_i} + b_i \leq \sum_{k \neq i} a_k/m_1 + (a_i + b_i) \leq 2\, C^*_{max}$ and $\sum_{i=1}^{j} c_i/m_2 \leq C^*_{max}$. In S', the latest time the postprocessors become busy is $\max_{i \leq j}(C_{a_i} + b_i)$. Therefore, $C'_j \leq \max_{i \leq j}(C_{a_i} + b_i) + \sum_{i=1}^{j-1} c_i/m_2 + c_j \leq 4C^*_{max}$. □

Theorem 4.6.7 *In* $O(n \log n)$ *time, one can obtain a* $(5, 13)$ *non-preemptive online schedule when* $m_1 \geq 1$ *and* $m_2 \geq 1$.

Proof : Let S be a $(4, 3)$-schedule generated by Algorithm $SRPT_a$-SPT_c. The conversion consists of two steps. First we use the approach of [95] to remove preemptions among the a tasks to get a $\lambda = 1$ schedule of the a tasks, respecting the release times of the a tasks. Let C'_{a_j} be the new completion time of a_j. By the result of [14], one must have $C'_{a_j} \leq \frac{7}{3} C_{a_j}$,. Now, fix the schedule of the a tasks. Each task c_j can be seen as a task with a release time $(C'_{a_j} + b_j)$. Next, one can remove preemptions among the c tasks to get a λ-schedule of the

c tasks, where $\lambda = 1$, and make sure that the interval between C'_{a_j} and the start time of c_j is at least b_j for each job j.

Let C_j be the completion time in the preemptive schedule. Suppose the jobs are indexed such that $C_i < C_{i+1}$ in S. Then, $C_j \geq \max_{1\leq i\leq j}(C_{a_i} + b_i + c_i)$ and $C_j > \sum_{i=1}^{j} c_i/m_2$. Let C'_j be the new completion time of job j. Then

$$C'_j \leq \max_{1\leq i\leq j}(C'_{a_i} + b_i) + \sum_{1\leq i<j} \frac{c_i}{m_2} + c_j \leq \max_{1\leq i\leq j}(\tfrac{7}{3} C_{a_i} + b_i) + \sum_{1\leq i<j} \frac{c_i}{m_2} + c_j \leq \tfrac{13}{3} C_j .$$

Thus, $\sum_{j=1}^{n} C'_j \leq \frac{13}{3} \sum_{j=1}^{n} C_j$. By assumption, S is a 3-approximation for total completion time, thus C'_j is a 13-approximation for total completion time.

For the makespan, $C^*_{max} \geq \max(A/m_1, C/m_2, \max(a_j + b_j + c_j))$. Therefore,

$$\begin{aligned} C'_j &\leq \max_{1\leq k\leq n} r_k + \left(\sum_{C_{a_k}<C_{a_j}} a_k/m_1 \right) + (a_j + c_j) + \max_{C_i<C_j} b_j + \left(\sum_{1\leq i<j} \frac{c_i}{m_2} \right) \\ &\leq 5\, C^*_{max} \end{aligned}$$

Then S' is a $(5, 13)$-schedule. □

4.7 Linear Programming: Distinct Preprocessors and Postprocessors

As mentioned before, another important approach for NP-hard scheduling problems is to formulate the problem as a linear programming problem. This method has the advantage that it also works for total weighted completion time. However, its disadvantage is that it takes relatively long time to obtain a solution. In this section approximation algorithms are presented for the case when there are distinct preprocessing and postprocessing processors. Each job j is allowed to have a weight w_j.

The basis of the approximation algorithms in this section is a linear programming relaxation that uses as variables the completion times of the a tasks and the c tasks. For each task j, define C_{a_j} and C_j to be the completion time of a_j and c_j, respectively. The total weighted completion time minimization problem can be formulated as follows:

$$\text{minimize} \sum_{j=1}^{n} w_j C_j \tag{4.9}$$

subject to

$$C_{a_j} \geq r_j + a_j \tag{4.10}$$

$$C_j \geq C_{a_j} + b_j + c_j \tag{4.11}$$

$$C_j \geq C_k + c_j \text{ or } C_k \geq C_j + c_k \quad \text{for each pair } j \neq k \tag{4.12}$$

$$C_{a_j} \geq C_{a_k} + a_j \text{ or } C_{a_k} \geq C_{a_j} + a_k \quad \text{for each pair } j \neq k \tag{4.13}$$

4.7.1 $m_1 = m_2 = 1$

The difficulty with the above characterization is the so-called "disjunctive" constraints (4.12)-(4.13), which are not linear inequalities and cannot be modeled using linear inequalities. Instead, one can use a class of valid inequalities for any feasible schedules (maybe preemptive), introduced by Queyranne [98] and Wolsey [119]. Suppose a set N of n tasks, denoted by $1, 2, \ldots, n$, are scheduled on a single machine. Let p_j be the processing time of j and C_j be the completion time of j in any feasible schedule. Then, the following inequality is valid for any subset $X \subseteq N$.

$$\sum_{j \in X} p_j C_j \geq \frac{1}{2}\left(\sum_{j \in X} p_j^2 + \left(\sum_{j \in X} p_j\right)^2\right) \quad \text{for each } X \subseteq N \tag{4.14}$$

The key to the quality of the approximation deriving from the above relaxations is the following lemma.

Lemma 4.7.1 *([59],[108]) Let* $\overline{C}_1, \overline{C}_2, \ldots, \overline{C}_n$ *satisfy* (4.14), *and assume without loss of generality that* $\overline{C}_1 < \overline{C}_2 < \ldots < \overline{C}_n$. *Then, for each job* $j = 1, \ldots, n$,

$$\overline{C}_j \geq \frac{1}{2} \sum_{k=1}^{j} p_k$$

A feasible solution $\overline{C}_1 < \overline{C}_2 < \ldots < \overline{C}_n$ satisfying (4.14) need not correspond to a feasible schedule. Lemma 4.7.1 states that merely satisfying the constraints (4.14) is sufficient to obtain a relaxation of this: $\overline{C}_j \geq \frac{1}{2}\sum_{k=1}^{j} p_k$.

Queyranne [98] has shown that the linear program with constraints (4.14) is solvable in polynomial time via the ellipsoid algorithm; the key observation is that there is a polynomial time separation algorithm for the exponentially large class of constraints given by (4.14).

In our model, one can apply the above constraints to the preprocessing master and the postprocessing master, respectively.

$$\sum_{k\in X} a_k C_{a_k} \geq \frac{1}{2}\left(\sum_{j\in X} a_j^2 + \left(\sum_{j\in X} a_j\right)^2\right) \quad \text{for each } X \subseteq N \tag{4.15}$$

$$\sum_{k\in X} c_k C_k \geq \frac{1}{2}\left(\sum_{j\in X} c_j^2 + \left(\sum_{j\in X} c_j\right)^2\right) \quad \text{for each } X \subseteq N \tag{4.16}$$

Algorithm List-Schedule-Guided-by-LP-Single-Master: First obtain an optimal solution to the linear program formed by (4.9), (4.10), (4.11), (4.15) and (4.16). Denote the completion time of the jobs by $\overline{C}_1, \ldots, \overline{C}_n$. Then one can form a schedule by scheduling both the a tasks and the c tasks in increasing order of $\overline{C}_j$ under the condition that a_j can not start until r_j, and the interval between the start time of c_j and the finish time of a_j is at least b_j, i.e., c_j can not start until b_j finishes.

Theorem 4.7.2 *Suppose that* $m_1 = m_2 = 1$. *Then Algorithm List-Schedule-Guided-by-LP-Single-Master produces a* $(3,5)$*-schedule when all jobs have the same release time and* $(4,6)$*-schedule when each job has an arbitrary release time. Furthermore, the* a *tasks and the* c *tasks complete in the same order.*

Proof : Let S be the schedule obtained by Algorithm List-Schedule-Guided-by-LP-Single-Master. Let C_{a_j} be the completion time of a_j in S. By the way the a tasks are

scheduled, we have $C_{a_j} \leq \max_{k\leq j}\{r_k\} + \sum_{k\leq j} a_k$ since the latest time the preprocessor becomes busy is $\max_{k\leq j}\{r_k\}$. Let C_j be the completion time of job j in S. Because the c tasks are scheduled in the same order as the a tasks, it is possible that c_j can not start even after b_j finishes at $C_{a_j} + b_j$. This is because c_{j-1} may not have completed yet. However, the time that c_j needs to wait after it is ready is at most $\max_{k\leq j}\{b_k\} + \sum_{k\leq j-1} c_k$. Thus,

$$\begin{aligned}
C_j &\leq C_{a_j} + \max_{k\leq j}\{b_k\} + \sum_{k=1}^{j-1} c_k + c_j \\
&\leq \left(\max_{k\leq j}\{r_k\} + \sum_{k=1}^{j} a_k\right) + \max_{k\leq j}\{b_k\} + \sum_{k=1}^{j} c_k \\
&\leq \max_{k\leq j}\{r_k\} + 2\overline{C}_{a_j} + \max_{k\leq j}\{b_k\} + 2\overline{C}_j \qquad \text{by Lemma 4.7.1} \\
&\leq \max_{k\leq j}\{r_k\} + 5\overline{C}_j
\end{aligned}$$

Thus, if all jobs have the same release times, then $C_j \leq 5\overline{C}_j$; otherwise, $C_j \leq 6\overline{C}_j$. This implies that S is a 5-approximation for total completion time when $r_j = 0$ for all j, and a 6-approximation when $r_j \geq 0$. This is so even if each job j has a weight.

For the makespan,

$$\begin{aligned}
C_j &\leq C_{a_j} + \max_{k\leq j} b_k + \sum_{k=1}^{j} c_k \\
&\leq \left(\max_{k\leq j}\{r_k\} + \sum_{k=1}^{j} a_k\right) + \max_{k\leq j}\{b_k\} + \sum_{k=1}^{j} c_k \\
&\leq \max_{k\leq j}\{r_k\} + 3C^*_{max}
\end{aligned}$$

Thus, if all jobs have the same release times, then $C_j \leq 3C^*_{max}$; otherwise, $C_j \leq 4C^*_{max}$. □

4.7.2 $m_1 \geq 1$ and $m_2 \geq 1$

If the jobs are scheduled on m machines, then similar valid inequalities hold which were also observed by Queyranne [99].

$$\sum_{k\in X} p_k C_k \geq \frac{1}{2m}\left(\sum_{j\in X} p_j^2 + \left(\sum_{j\in X} p_j\right)^2\right) \quad \text{for each } X \subseteq N \tag{4.17}$$

One can derive similar lemma as in the single machine case.

Lemma 4.7.3 *Let* $\overline{C}_1, \overline{C}_2, \ldots, \overline{C}_n$ *satisfy* (4.17), *and assume without loss of generality that* $\overline{C}_1 < \overline{C}_2 < \ldots < \overline{C}_n$. *Then, for each job* $j = 1, \ldots, n$,

$$\overline{C}_j \geq \frac{1}{2m} \sum_{k=1}^{j} p_k$$

In our model, one can apply the above constraints to the m_1 preprocessing masters and the m_2 postprocessing masters, respectively.

$$\sum_{k \in X} a_k C_{a_k} \geq \frac{1}{2m_1} \left(\sum_{j \in X} a_j^2 + \left(\sum_{j \in X} a_j \right)^2 \right) \quad \text{for each } X \subseteq N \tag{4.18}$$

$$\sum_{k \in X} c_k C_k \geq \frac{1}{2m_2} \left(\sum_{j \in X} c_j^2 + \left(\sum_{j \in X} c_j \right)^2 \right) \quad \text{for each } X \subseteq N \tag{4.19}$$

Algorithm List-Schedule-Guided-by-LP-Multi-Master: First obtain an optimal solution to the linear program given by (4.9), (4.10), (4.11), (4.18) and (4.19). Denote the completion time of the jobs by $\overline{C}_1, \ldots, \overline{C}_n$. Schedule both the a tasks and the c tasks one by one in nondecreasing order of $\overline{C}_j$ under the condition that: a_j can not start until r_j, and the interval between the finish time of a_j and the start time of c_j is at least b_j. In other words, c_j can not start until b_j finishes.

Theorem 4.7.4 *Suppose that* $m_1 \geq 1$ *and* $m_2 \geq 1$. *Then Algorithm List-Schedule-Guided-by-LP-Multi-Master produces a* $(4, 6)$*-schedule when all jobs have the same release time and* $(5, 7)$*-schedule when each job has an arbitrary release time.*

Proof: Let S be the schedule obtained by Algorithm List-Schedule-Guided-by-LP-Multi-Master. Let C_{a_j} be the completion time of task a_j in S. By the way the a tasks are scheduled, the latest time the preprocessor becomes busy is $\max_{k \leq j}\{r_k\}$. Then a_k, $k \leq j-1$, are scheduled one by one. By time $t \leq \max_{k \leq j}\{r_k\} + \sum_{k \leq j-1} a_k/m_1$, the first $j-1$ tasks must be finished and there will be an idle machine to process the j^{th} task. Thus, $C_{a_j} \leq \max_{k \leq j}\{r_k\} + \sum_{k \leq j-1} a_k/m_1 + a_j$. Let C_j be the completion time of job j in S. Because

the c tasks are scheduled in the same order as the a tasks, it is possible that c_j can not start even after b_j finishes at $C_{a_j} + b_j$, because c_{j-1} may not have completed yet. However, the time that c_j needs to wait after it is ready is at most $\max_{k\le j}\{b_k\} + \sum_{k\le j-1} c_k/m_2$. Thus,

$$\begin{aligned} C_j &\le C_{a_j} + \max_{k\le j}\{b_k\} + \frac{1}{m_2}\sum_{k=1}^{j-1} c_k + c_j \\ &\le \left(\max_{k\le j}\{r_k\} + \frac{1}{m_1}\sum_{k=1}^{j-1} a_k + a_j\right) + \max_{k\le j}\{b_k\} + \frac{1}{m_2}\sum_{k=1}^{j-1} c_k + c_j \\ &\le \max_{k\le j}\{r_k\} + 2\max_{k\le j-1}\overline{C}_{a_k} + \max_{k\le j}\{b_k\} + 2\overline{C}_j + (a_j + c_j) \\ &\le \max_{k\le j}\{r_k\} + 6\overline{C}_j \end{aligned}$$

Thus, if all jobs have the same release times, then $C_j \le 6\overline{C}_j$; otherwise, $C_j \le 7\overline{C}_j$. Thus, S is a 6-approximation for total completion time when $r_j = 0$ for all j, and a 7-approximation when $r_j \ge 0$. This is so even if each job j has a weight w_j.

Now consider the makespan.

$$\begin{aligned} C_j &\le C_{a_j} + \max_{k\le j}\{b_k\} + \frac{1}{m_2}\sum_{k=1}^{j-1} c_k + c_j \\ &\le \left(\max_{k\le j}\{r_k\} + \frac{1}{m_1}\sum_{k=1}^{j-1} a_k + a_j\right) + \max_{k\le j}\{b_k\} + \frac{1}{m_2}\sum_{k=1}^{j-1} c_k + c_j \\ &\le \max_{k\le j}\{r_k\} + \frac{1}{m_1}\sum_{k=1}^{j-1} a_k + \max_{k\le j}\{b_k\} + \frac{1}{m_2}\sum_{k=1}^{j-1} c_k + (a_j + c_j) \\ &\le \max_{k\le j}\{r_k\} + 4C^*_{max} \end{aligned}$$

Thus, if all jobs have the same release times, then $C_j \le 4C^*_{max}$; otherwise, $C_j \le 5C^*_{max}$. □

PART II

NETWORK DESIGN PROBLEMS

CHAPTER 5

POLYNOMIAL-TIME APPROXIMATION SCHEMES FOR THE EUCLIDEAN SURVIVABLE NETWORK DESIGN PROBLEM

5.1 Introduction

This chapter considers the geometric version of the *survivable network design problem.* The input vertices are assumed to be points in $\mathbb{R}^d$ and the cost of each link is equal to the Euclidean distance between its endpoints (which is a good approximation in many applications, since often the "installation" and the "service" cost is roughly proportional to the length of the link [91]).

The focus is on two most basic variants of the geometric survivable network design problem: (1) SMT problem in which $r_v \in \{0, 1\}$ for any $v \in V$ and (2) $\{0, 1, 2\}$-vertex- and -edge-connectivity problem in which $r_v \in \{0, 1, 2\}$ for all $v \in V$. Note that SMT problem is a special case of $\{0, 1, 2\}$-vertex- and -edge-connectivity problem.

The arguments provided by Grötschel *et al.* [53] (see also [91, 113]), suggest that the second special case in the above models well many applications of the survivability problem, e.g., the problem of designing survivable fiber telephone networks [91, 113]. In the case of fiber communication networks for telephone companies, network topologies with connectivity requirements in $\{0, 1, 2\}$ provide an adequate level of survivability for the distinguished central nodes of connectivity type 2. Simply, most failures usually can be repaired relatively quickly and, as statistical studies have revealed, it is unlikely that a second failure will occur for their duration.

5.1.1 Related Works

There has been a lot of research on the survivable network design problem. Typically, the research addresses either practical heuristics and algorithms (see, e.g., [21, 50, 51, 53, 53, 91, 113]) or the general problem for arbitrary networks (see, e.g., [36, 37, 41, 64,

117]), or the problem restricted to very specific networks. In particular, the result due to Jain [64] gives a polynomial-time 2-approximation algorithm for the edge-connected survivable network design problem (for *arbitrary connectivity requirements*). Also, polynomial-time 2-approximation algorithms for arbitrary networks in the case $r_{v,u} \in \{0, 1, 2\}$ for every v, u have been recently presented [36, 37]. There is no other good polynomial-time approximation algorithm specialized for the geometric version of the survivability problem except the case when $r_v \in \{0, 1\}$ for every v [97]. If $r_v \in \{0, 1\}$ for every v and $U = \{v \in V : r_v = 1\}$, then one can easily show that a minimum spanning tree of U guarantees the approximation ratio of 2 in any metric space (and thus, in particular, in any Euclidean space $\mathbb{R}^d$). Importantly, in this case the geometric survivability problem is a generalization of the classical *Euclidean (complete) Steiner tree problem* (see, e.g., [44, 62, 63, 97, 118]). The Euclidean (complete) Steiner tree problem for a finite set of points S in $\mathbb{R}^d$ is to construct a minimum length tree whose vertex set consists of all points in S and possibly some other points in $\mathbb{R}^d$. Thus, the latter problem can be regarded as a survivable network design problem on an infinite vertex domain, i.e., $V = \mathbb{R}^d$ and $r_v = 1$ for any $v \in S$, and $r_v = 0$ otherwise. By the celebrated results due to Arora [3] and Mitchell [90] (see also [101]), the Euclidean (complete) Steiner tree problem admits a polynomial-time approximation scheme for any constant d.

5.1.2 New Contributions

The *first polynomial-time approximation schemes* (PTASs) are designed for the two aforementioned basic variants of the geometric survivable network design problem.

First, the simplest case in which $r_v \in \{0, 1\}$ for any vertex $v \in V$ is considered, that is, the Steiner tree problem. An algorithm is designed such that that for any constant d and any constant ε, it returns a Steiner tree whose cost is at most $(1 + \varepsilon)$ times larger than the minimum. The algorithm runs in time $\mathcal{O}(n \log n)$. For general d and ε, its running time is $\mathcal{O}(n \log n\,(d/\varepsilon)^{\mathcal{O}(d)}) + \mathcal{O}(n\,(d/\varepsilon)^{(d/\varepsilon)^{\mathcal{O}(d^2)}})$.

Next, the case when $r_v \in \{0, 1, 2\}$ for any vertex $v \in V$ is considered; this is the classical problem investigated thoroughly by Grötschel and Monma *et al.* [50, 51, 52, 53, 91, 113]. The algorithm for the Steiner tree problem is extended to design an algorithm that, for any constant d and any constant ε, returns a graph satisfying all the vertex (or edge, respectively) connectivity requirements and having the cost at most $(1 + \varepsilon)$ times the minimum. The algorithm runs in time $\mathcal{O}(n \log n)$. When d and ε are allowed to vary arbitrarily, its running time is $\mathcal{O}(n \log n\,(d/\varepsilon)^{\mathcal{O}(d)}) + \mathcal{O}(n\,(d/\varepsilon)^{(d/\varepsilon)^{\mathcal{O}(d^2)}})$.

Finally, observe that the techniques yield also a PTAS for the multigraph variant where the edge-connectivity requirements satisfy $r_v \in \{0, 1, \ldots, k\}$ and $k = \mathcal{O}(1)$.

All the polynomial-time approximation schemes in this chpater follow an approach similar to those used in the recent PTASs for finding TSP, (complete) Steiner trees, and minimum-cost biconnected spanning subgraph in Euclidean graphs, see [3, 26, 101]. However, there are many important differences that make the new results significantly more complicated. First of all, one has to deal with the restriction of the Steiner points to the set given *a priori* (unlike in the minimum-cost Euclidean (complete) Steiner trees problem, in which Steiner points are allowed to be any points in $\mathbb{R}^d$). Furthermore, one has to deal with non-uniform connectivity requirements. The substantial differences and complications occur in the so called filtering phase and searching phase (dynamic programming).

All the PTASs are randomized and achieve the promised approximation guarantees and running time on the average. However, all these algorithms can be *derandomized* in a way similar to that used by Rao and Smith in [101]. The derandomization preserves the running time of $\mathcal{O}(n \log n)$ for constant d and ε.

5.2 Definitions

Following are some notations and definitions for Euclidean (geometric) graphs that will be used in this chapter.

A *Euclidean* graph, which frequently will be called in this chapter a *graph*, is a pair $G = (P, E)$, where P is a set of points in a Euclidean space $\mathbb{R}^d$ and E is a subset of the pairs of points in P. Every Euclidean graph is weighted and the *cost* of edge (x, y) is equal to the Euclidean distance between points x and y. The *cost of the graph* is the sum of the costs of its edges. Additionally, Euclidean multigraphs, which are as Euclidean graphs but may contain parallel edges, are also considered. Consistently with the definition, the edges of a Euclidean graph or multigraph $G = (P, E)$ are in one-to-one correspondence with the straight-line segments (in $\mathbb{R}^d$) connecting the incident vertices. (Such graphs are frequently called straight-line graphs in the literature.) Sometimes, for technical reasons, it is also allowed to *bend* some edges. A bent edge between a pair of points in P will be identified with a *straight-line path* (a path consisting of straight-line segments connecting the points).

Spanners for general graphs have been defined in Chapter 1. In this chapter, geometric spanners of a set of points are considered.

Definition 5.2.1 (Geometric spanners) *Let P be a set of points in $\mathbb{R}^d$. A graph G on P is called a geometric* t-spanner *of P, $t \geq 1$, if for any pair of points $p, q \in P$ there exists a path in G from p to q of length at most t times the Euclidean distance between p and q.*

Since only geometric spanners are considered. For simplicity, a geometric spanner is simply referred to as a spanner. The following result is proven in [56][1].

Lemma 5.2.2 [56] *Let P be a set of n points in $\mathbb{R}^d$, where d is a constant. Let ε be any positive constant. Then, there exists an $\mathcal{O}(n \log n)$-time algorithm that finds a $(1+\varepsilon)$-spanner of P having constant maximum degree and the total cost of order of the minimum spanning tree of P.*

[1]Notice that the algorithms used in Lemma 5.2.2 are in the so-called real RAM model, which is the algebraic decision tree model extended with indirected addressing [56]. If one assumes the algebraic decision tree model, then the running time will be by a factor of $\log n / \log \log n$ larger [56].

For arbitrary d *and* ε*, the running time of this algorithm is* $\mathcal{O}(\tau_{d,\varepsilon} \cdot n + d \cdot n \cdot \log n)$ *and the resulting* $(1+\varepsilon)$*-spanner of* P *has maximum degree upper bounded by* $\xi_{d,\varepsilon}$ *and the total cost upper bounded by* $\xi_{d,\varepsilon}$ *times the cost of the minimum spanning tree of* P*, where* $\tau_{d,\varepsilon} = (d/\varepsilon)^{\mathcal{O}(d)}$ *and* $\xi_{d,\varepsilon} = (d/\varepsilon)^{\mathcal{O}(d)}$.

Definition 5.2.3 Leapfrog property ([30]) *Let* $t \geq t' > 1$. E *is a set of line segments in* d*-dimensional space.* E *satisfies the* (t', t) *leapfrog property if the following is true for every possible subset* $S = \{(u_0, v_0), (u_1, v_1), \ldots, (u_m, v_m)\}$ *of* E*:*

$$t' \cdot |u_0 v_0| < \sum_{1 \leq i \leq m} |u_i v_i| + t \cdot (|v_m u_0| + \sum_{0 \leq i \leq m-1} |v_i u_{i+1}|) \ .$$

Informally, this definition says that if there exists an edge between u_0 and v_0, then any path not including (u_0, v_0) must have length greater than $t' \cdot |u_0 v_0|$.

Lemma 5.2.4 ([30]) *If a set of line segments* E *in* d*-dimensional space satisfies the* (t', t)*-leapfrog property where* $t \geq t' > 1$*, then the weight of* E *is* $\mathcal{O}(wt(MST)$ *where* $wt(MST)$ *is the cost of a minimum spanning tree connecting the endpoints of* E*. The constant implicit in the* $\mathcal{O}$*- notation depends on* t *and* d.

Definition 5.2.5 (Steiner trees) *Let* P_1 *be a set of points in* $\mathbb{R}^d$*. A Euclidean tree is called a* Steiner tree *of* P_1 *if its vertex set includes all the points* P_1*. All the vertices of a Steiner tree of* P_1 *outside* P_1 *are called its* Steiner points*. If the Steiner points are restricted to a point set* P_0*, the tree is called a* Steiner tree of P_1 with respect to P_0 *and the points in* P_0 *are called* Steiner point candidates.

Definition 5.2.6 (Euclidean (complete) Steiner trees) *Let* P *be a set of points in* $\mathbb{R}^d$*. The* Euclidean (complete) Steiner minimal tree *of* P *is a Steiner tree of* P *having the minimum cost.*

Definition 5.2.7 (Steiner minimum trees (SMT)) *Let* $P = P_0 \cup P_1$ *be a point set in the Euclidean space* $\mathbb{R}^d$. *A Steiner tree of* P_1 *with respect to* P_0 *having the minimum cost will be called a* Steiner minimum tree (SMT) *of* P_1 *with respect to* P_0.

Observe the difference between the definition of Euclidean (complete) Steiner tree and Steiner minimum tree; in this chapter the abbreviation SMT is used only to denote a Steiner minimum tree.

As mentioned before, the focus in this chapter is on the variant of the problem when $r_p \in \{0, 1, 2\}$ for any $p \in P$. This problem is called the $\{0, 1, 2\}$-*vertex- or -edge-connectivity* problem, depending on whether the vertex-connected or the edge-connected version of the problem is considered. (Notice that the $\{0, 1, 2\}$-vertex- and -edge-connectivity problem includes the SMT problem in which $r_p \in \{0, 1\}$ for any $p \in P$(see Definition 5.2.7).) Furthermore, one can repeat the arguments used in [26] (which were also used earlier in [38, Section 3]) to show that in metric spaces $\{0, 1, 2\}$-vertex-connectivity and $\{0, 1, 2\}$-edge-connectivity are essentially equivalent (the arguments in [26] and [38] were given only for biconnectivity vs. two-edge-connectivity).

Lemma 5.2.8 *Let* P_0, P_1, P_2 *be any three sets of points in a metric space. Let* H *be a multigraph with the vertex set* $P_0 \cup P_1 \cup P_2$ *such that for any pair of vertices* $u \in P_i$ *and* $v \in P_j$ *there are at least* $\min\{i, j\}, 0 \leq i, j \leq 2$, *edge-disjoint paths from* u *to* v *in* H. *Then, in linear time, one can transform* H *into a graph* G *without increasing the total cost such that for any pair of vertices* $u \in P_i$ *and* $v \in P_j$ *there are at least* $\min\{i, j\}$ *internally vertex-disjoint paths form* u *to* v *in* G.

Proof : Initially set $G = H$ and a sequence of modification will be performed to transform G to get the required properties. It is easy to see that if $u \in P_0 \cup P_1$, then graph G already has the required properties with respect to u. Therefore one only has to deal with points $u \in P_2$ all of which are in the same two-edge-connected component of G. If this single component is also a block, it is done; otherwise pick any two edges (v, u) and (v, w) such

that u and w are in different blocks, in other words, v is an articulation point. Replace edges (v, u) and (v, w) by (u, w). Now, u and w will be in the same block. By the triangle inequality, the cost of the new block will not be larger than the sum of the older two blocks. One does this replacement until all points of P_2 are in the same block. Then remove all parallel edges from the graph, and G becomes a graph has a cost not greater than that of H, and for any pair of vertices $u \in P_i$ and $v \in P_j$ there are at least $\min\{i, j\}$ internally vertex-disjoint paths form u to v in G.

Using standard techniques this transformation can be performed in time linear in the number of edges in H. □

This lemma allows one to concentrate only on the $\{0, 1, 2\}$-edge-connectivity problem, and to allow the output to be given in a form of a multigraph.

Partitioning the space. An important component of the approximation algorithms in this chapter is a partitioning scheme introduced by Arora in [3] and later extended in [25, 26].

Definition 5.2.9 (Dissection, 2^d-ary tree) *Given a set* S *of points in* $\mathbb{R}^d$, *a* bounding box *of* S *is the smallest* d*-dimensional axis-parallel cube* L^d *containing the points in* S. *A* (2^d-ary) dissection *[3](see Figure 5.1) of* S *is the recursive partitioning of the cube into smaller sub-cubes, called* regions. *Each* region U^d *of volume* > 1 *is recursively partitioned into* 2^d *regions* $(U/2)^d$. *A* 2^d-ary tree *(for a given* 2^d*-ary dissection) is a tree whose root corresponds to* L^d, *and whose other non-leaf nodes correspond to the regions containing at least two points from* S *(see Figure 5.1). For a non-leaf node* v *of the tree corresponding to a region* R, *its children in the tree correspond to the* 2^d *regions that partition* R *in the dissection.*

For any d*-vector* $\mathbf{a} = (a_1, \ldots, a_d)$, *where all* a_i *are integers* $0 \leq a_i \leq L$, *the* **a**-shifted dissection *[3, 25] of a set* X *of points in the cube* L^d *in* $\mathbb{R}^d$ *is the dissection of the set* X^* *in the cube* $(2L)^d$ *in* $\mathbb{R}^d$ *obtained from* X *by transforming each point* $\mathbf{x} \in X$ *to* $\mathbf{x} + \mathbf{a}$. *A* random shifted dissection *of a set of points* X *in a cube* L^d *in* $\mathbb{R}^d$ *is an* **a***-shifted*

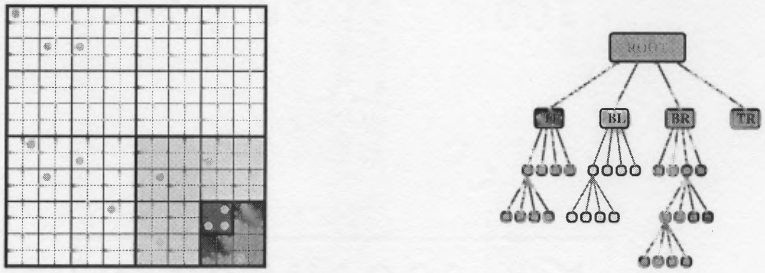


Figure 5.1 Dissection of a bounding cube in $\mathbb{R}^2$ (left) and the corresponding 2^2-ary tree (right). In the tree, the children of each node are ordered from left to right: **T**op/**L**eft square, **B**ottom/**L**eft square, **B**ottom/**R**ight square, and **T**op/**R**ight square.

dissection of X *with* $\mathbf{a} = (a_1, \ldots, a_d)$ *and the elements* $a_1, \ldots, a_d$ *chosen independently and uniformly at random from* $\{0, 1, \ldots, L\}$.

In this chapter a special class of geometric graphs with respect to a given dissection [26] will also be studied.

Definition 5.2.10 (**r-locally-light graphs**) *A graph is* r-locally-light *[26] with respect to a shifted dissection if for each region in the dissection there are at most* r *edges having exactly one endpoint in the region. The crossings on the border of the region generated by these edges are called relevant crossings.*

The main reason of introducing this class of graphs is that while it is $\mathcal{NP}$-hard to solve the $\{0, 1, 2\}$-vertex- or -edge-connectivity problem (or even the minimum-cost Steiner tree problem) for arbitrary Euclidean graphs, one can show how to solve this problem efficiently for r-locally-light graphs in Section 5.8.

5.3 Steiner Minimum Tree Problem

In the attempt to provide an efficient approximation scheme for the $\{0, 1, 2\}$-connectivity problem in Euclidean graphs, the SMT problem will be considered first. The new approach can be seen as a combination of the approach of Arora [3] and Rao and Smith [101] developed to design a PTAS for the TSP problem with the approach of Czumaj and Lingas [26]

developed to design a PTAS for k-connectivity problems. The new algorithms is based on efficient implementations of the following three procedures.

Filtering: *Let* P_0 *and* P_1 *be sets of points in* $\mathbb{R}^d$ *and let* t *be any positive real number. Find a subset* X *of* P_0 *that satisfies the following two properties:*

- *The cost of the SMT of* P_1 *with respect to* X *is at most* $1+t$ *times the cost of the SMT of* P_1 *with respect to* P_0.
- *The cost of the minimum spanning tree of* $X \cup P_1$ *is upper bounded by* $\lambda_{d,t}$ *times the cost of the minimum spanning tree of* P_1, *where* $\lambda_{d,t}$ *is a function of* d *and* t *only* ($\lambda_{d,t}$ *will be set to* $2^{\mathcal{O}(d^4)}/t^{\mathcal{O}(d)}$).

Lightening: *Let* P_0 *and* P_1 *be sets of points in* $\mathbb{R}^d$ *and let* t *be any positive number. Let* G *be any* $(1+t)$*-spanner of* $P_0 \cup P_1$ *satisfying the so called* $(t', 1+t)$*-leapfrog property [56] for* $1 < t' < 1+t$. *Modify* G *to obtain an* r*-locally-light graph with the vertex set* $P_0 \cup P_1$ *that has as its subgraph a Steiner tree of* P_1 *with respect to* P_0 *whose cost is at most* $(1+2t)$ *times the cost of the SMT of* P_1 *with respect to* P_0.

Searching: *Let* P_0 *and* P_1 *be sets of points in* $\mathbb{R}^d$ *and let* r *be any positive integer. Let* G *be any* r*-locally-light graph on* $P_0 \cup P_1$. *Find a minimum-cost Steiner tree of* P_1 *with respect to* P_0 *that is a subgraph of* G.

5.4 Filtering for SMT

In this section it will be shown how to perform the filtering phase efficiently. First it will be proved that the following algorithm finds a subset X of P_0 that satisfies the required filtering property with $t = \frac{3}{2}\varepsilon$.

1. Build a $(1+\varepsilon)$-spanner S on P_1 with $n \cdot \xi_{d,\varepsilon}$ edges whose total cost is upper bounded by $\xi_{d,\varepsilon}$ times the cost of the minimum spanning tree of P_1, where $\xi_{d,\varepsilon} = (d/\varepsilon)^{\mathcal{O}(d)}$, as in Lemma 5.2.2.

2. For each edge e of the spanner whose cost exceeds the $|P_1|^{-4}$ fraction of the cost of minimum spanning tree of P_1, circumscribe a d-dimensional ball $B\langle e\rangle$ with the center at the middle of e and of radius $R\langle e\rangle$ equal to ρ_d/ε times the length of the edge, where ρ_d is a function depending only on d, $\rho_d = 2^{\mathcal{O}(d^3)}$.

3. Let Y be a subset of P_0 that includes all points contained in the constructed balls and possibly some other points in P_0 at distance at most $4\,R\langle e\rangle$ from the center of such a ball $B\langle e\rangle$.

4. For each ball $B\langle e\rangle$ define a (rectilinear) d-dimensional cube $C\langle e\rangle$ of side length $8\,R\langle e\rangle$ that is co-centric with the ball $B\langle e\rangle$. Within each cube $C\langle e\rangle$ introduce a grid with interspacing $|e|\,\varepsilon^2/(8\,\Delta\,\rho_d\,\sqrt{d})$, where Δ is the bound of maximum degree of any MST of n points in dimension d . Let set X be initially empty. Repeatedly, in the increasing length order of the edges e of S, assign each point $p \in Y$ associated with ball $B\langle e\rangle$ to the closest point of the grid $C\langle e\rangle$. For each grid point in $C\langle e\rangle$ if there is at least one point $p \in Y$ assigned to it, add one such a point to X.

Lemma 5.4.1 (SMT Filtering) *For any point sets P_0 and P_1 in $\mathbb{R}^d$ and any positive real number ε, the subset X of P_0 satisfies the filtering property.*

In order to prove Lemma 5.4.1. One has to prove that the set $X \subseteq P_0$ obtained in the above construction satisfies the following two properties:

1. The cost of the SMT of P_1 with respect to X is at most $1 + \frac{3}{2}\varepsilon$ times the cost of the SMT of P_1 with respect to P_0.

2. The cost of the minimum spanning tree of $X \cup P_1$ is upper bounded by $2^{\mathcal{O}(d^4)}/\varepsilon^{\mathcal{O}(d)}$ times the cost of the minimum spanning tree of P_1.

In the following, the set X is first shown to satisfy the first property and then it is shown to satisfy the second property. The proof is rather long and will be presented in a general form that can eventually be used in some further applications.

5.4.1 First Filtering Property

The proof needs a couple of auxiliary lemmas. Following is a standard result about the upper bound for the cost of the minimum spanning trees contained in a d-dimensional ball (for a proof, see, e.g., [77]).

Lemma 5.4.2 *Let P be a set of n points in $\mathbb{R}^d$ contained in a d-dimensional ball of radius r. Then, the minimum spanning tree of P has cost upper bounded by $r \cdot n^{1-1/d} \cdot \frac{2d}{(d-1)(1-n^{-1/d})}$.*

In particular, if $n \geq 2^d$ and $d \geq 2$ then the cost of the MST of P is upper bounded by $8rn^{1-1/d}$. □

The second lemma gives an upper bound for the maximum degree in a minimum spanning tree (and an SMT) of a point set in $\mathbb{R}^d$.

Lemma 5.4.3 [102] *Let P be a set of n points in $\mathbb{R}^d$. Then, any minimum spanning tree of P has maximum degree Δ upper bounded by $c^d = 2^{O(d)}$, where c is an absolute constant.* □

From now on, Δ will be used to denote the upper bound for the maximum degree of any minimum spanning tree of a set of points in $\mathbb{R}_d$ (though we will not use in the notation the dependence on d).

Now, a simple lemma will be proved which shows some basic property of SMT (which holds also for MST) and paths in spanners, and in general, in arbitrary connected graphs.

Lemma 5.4.4 *Let P_0 and P_1 be disjoint point sets in $\mathbb{R}^d$. Let G be any graph connected with respect to P_1 (for example, a spanner on P_1) and let T be an SMT of P_1 with respect to P_0. Let u and v be any two points in P_1 and let $p_{u \rightsquigarrow v}$ be the shortest path in G between u and v. Let e be any edge in T of length ℓ. If after removal of e from T, points u and v are in different connected components of the resulting forest, then at least one edge on the path $p_{u \rightsquigarrow v}$ is not shorter than ℓ.*

Proof : The proof is by contradiction. Suppose after removal of e from T two subtrees T_1 and T_2 with $u \in T_1$ and $v \in T_2$ are obtained and suppose that all edges on $p_{u \rightsquigarrow v}$ in G are shorter than ℓ. Since u and v are in different subtrees, then there must exist an edge xy on the path $p_{u \rightsquigarrow v}$ such that $x \in T_1$, $y \in T_2$. By the assumption, $|xy| < \ell$. Therefore, $T_1 + T_2 + xy$ forms a Steiner tree of P_1 with respect to P_0 whose cost is smaller than the cost of T. But this contradicts the assumption that T is an SMT of P_1 with respect to P_0. □

Next lemma is concerned with balls containing no points from P_1. It will be shown that in that case in any SMT any co-centric ball of a slightly smaller radius has the number of Steiner points upper bounded by $2^{\mathcal{O}(d^2)}$. Notice that a similar lemma is used by Rao and Smith in [101] and the proof is this chapter uses many ideas from [101]. The main differences are caused by the fact that in this chapter the Steiner trees considered have the vertex set in a subset of $P_0 \cup P_1$ while the proof in [101] worked for $P_0 = \mathbb{R}^d$. This implies, among other, that Rao and Smith could use many well known properties of such Steiner trees (for example, in that case, the maximum degree in a minimum-cost Steiner tree is 3).

Lemma 5.4.5 *Let P_0 and P_1 be any disjoint point sets in $\mathbb{R}^d$. Let $\mathbb{B}_O$ and $\mathbb{B}_I$ be any two co-centric d-dimensional balls in $\mathbb{R}^d$ of radius 1 and r, $0 < r < 1$, respectively. Then for any SMT of P_1 with respect to P_0, if $\mathbb{B}_O$ contains no points from P_1 then $\mathbb{B}_I$ contains at most $(16\,e\,d\,\Delta)^{4d} = 2^{\mathcal{O}(d^2)}$ Steiner points, i.e., SMT points from P_0.*

Proof : First fix an SMT of P_1 with respect to P_0 and denote it by T. Let Δ denote the maximum degree of Steiner points in the SMT. (Notice that Lemma 5.4.3 ensures that $\Delta \leq c^d$ for certain constant c.) It can be assumed without loss of generality, that every Steiner point is of degree at least 3. Hence, also, the number of points in P_1, which will be denoted by n, is bigger than the number of Steiner points. Let s denote the number of Steiner points contained in $\mathbb{B}_I$; the goal is to prove that $s \leq (16\,e\,d\,\Delta)^{4d}$.

Let x be the center of balls $\mathbb{B}_I$ and $\mathbb{B}_O$. For any $R \in \mathbb{R}_{\geq 0}$, let B_R denote the d-dimensional ball of radius R with center at x. For any $R \in \mathbb{R}_{\geq 0}$, let N_R denote the number

of places where T touches the border of B_R, i.e., the number of edges that have exactly one endpoint contained in B_R. Next, let S_R denote the number of Steiner points contained in B_R, and let T_R denote the portion of T contained in B_R. Note that by assumption, for any $0 < R \leq 1$, $N_R \leq \Delta \cdot S_R$.

Pick any R and R^* such that $r \leq R < R^* \leq 1$. Notice that if $N_r < 2^d$, then $s < N_r < 2^d$ and hence the lemma is proved. So, suppose that $N_r \geq 2^d$, and thus $N_R \geq 2^d$. Observe that since B_{R^*} contains no points from P_1, B_R contains S_R Steiner points, and all Steiner points are of degree at least 3, thus $N_{R^*} \geq S_R + 2 > S_R$. (This follows from the fact that T_{R^*} contains at least S_{R^*} "internal" nodes of degree at least 3 and N_{R^*} "leaves" which are of degree 1.) Furthermore, notice that the length of T_{R^*} is at least $(R^* - R) \cdot S_R$. On the other hand, T_{R^*} contains at least S_R Steiner points and additional N_{R^*} points on its border. Therefore, by Lemma 5.4.2,

$$(R^* - R) \cdot S_R < 8 \cdot R^* \cdot (N_{R^*} + S_R)^{1-1/d} < 16\,(N_{R^*})^{1-1/d} \ . \tag{5.1}$$

Let $\delta = \frac{1}{\ln \ln n}$ and $\Gamma = d \ln \ln n$. For any integer i, $0 \leq i \leq d \ln \ln n = \Gamma$, let $R_i = 1 - \frac{i \cdot \delta}{\Gamma}$. Consider the balls B_{R_i} for $0 \leq i \leq \Gamma$. Now apply Inequality (5.1) to $B_{R_0}, B_{R_1}, B_{R_2}, \ldots, B_{R_\Gamma}$ to obtain

$$\frac{\delta}{\Gamma} \cdot S_{R_1} < 16 \cdot (N_{R_0})^{1-1/d} \ , \ \frac{\delta}{\Gamma} \cdot S_{R_2} < 16 \cdot (N_{R_1})^{1-1/d} \ , \cdots \frac{\delta}{\Gamma} \cdot S_{R_\Gamma} < 16 \cdot (N_{R_{\Gamma-1}})^{1-1/d} \ .$$

Recall that $N_{R_0} = N_1$. Therefore, one can combine the inequality $N_R \leq \Delta \cdot S_R$ that holds for all $0 < R \leq 1$, with all the inequalities above to obtain

$$S_{R_\Gamma} < (16\,\Gamma/\delta)^{\sum_{j=0}^{\Gamma-1}(1-1/d)^j} \cdot \Delta^{\sum_{j=1}^{\Gamma-1}(1-1/d)^j} \cdot N_1^{(1-1/d)^\Gamma} \ .$$

Therefore, since $N_1 \leq n$, one can get

$$S_{R_\Gamma} \leq e \cdot (16 \cdot \Gamma \cdot \Delta/\delta)^d = e \cdot (16 \cdot \Delta \cdot d \cdot (\ln \ln n)^2)^d \ .$$

A more general form of this result is stated as follows.

> *Let* $r + \delta \leq R \leq 1$ *and let* N_R *be the number of places in which* T *touches the border of* B_R. *Then* $S_{R-\delta}$, *the number of Steiner points in* $T_{R-\delta}$, *is upper bounded by* $e \cdot (16 \cdot \Delta \cdot d \cdot (\ln \ln N_R)^2)^d$.

Since $N_R \leq \Delta \cdot S_R$ for any $0 < R \leq 1$, one can apply the claim above to obtain the following bound

$$S_{1-(l+1)\delta} \leq e \cdot (16 \cdot \Delta \cdot d \cdot (\ln \ln N_{1-l\delta})^2)^d \leq e \cdot (16 \cdot \Delta \cdot d \cdot (\ln \ln(\Delta \cdot S_{1-l\delta}))^2)^d$$

that holds for any integer l, $0 \leq l < \frac{1-r}{\delta}$.

Notice that if $S_{1-l\delta} \leq \Delta$ for any integer l, $0 \leq l < \frac{1-r}{\delta}$, then (by Lemma 5.4.3) the lemma is proven. Therefore, one can assume that $S_{1-l\delta} > \Delta$, and hence the upper bound above can be simplified to get

$$S_{1-(l+1)\delta} \leq e \cdot (16 \cdot \Delta \cdot d \cdot (\ln \ln((S_{1-l\delta})^2))^2)^d \leq (16 \cdot e \cdot \Delta \cdot d \cdot \ln \ln S_{1-l\delta})^{2d} .$$

This implies that for certain $l = \log^* N_1 + \mathcal{O}(1)$, one has

$$S_{1-(l+1)\delta} \leq (16 \cdot e \cdot \Delta \cdot d)^{4d} .$$

This yields $s \leq S_{1-(l+1)\delta} \leq (16\, e\, \Delta\, d)^{4d} = 2^{\mathcal{O}(d^2)}$. □

First filtering property of set Y Now Y will be proved to satisfy the first property. Similarly as in the proof of Lemma 5.4.5, some of the arguments are similar to those used by Rao and Smith [101] in their PTAS for the Euclidean (complete) Steiner tree problem. This time, however, the differences caused by the fact that Rao and Smith worked with $P_0 = \mathbb{R}^d$ are even bigger. Therefore, most of the details of the following proof are completely new.

Lemma 5.4.6 [2] *Let P_0 and P_1 be two disjoint point sets in $\mathbb{R}^d$. Let S be a $(1+\frac{1}{4}\varepsilon)$-spanner of P_1. For each edge e in S let $B\langle e\rangle$ be the ball with the center at the midpoint of e and with the radius equal to ρ_d/ε times the length of the edge e, where $\rho_d = 2^{\mathcal{O}(d^3)}$ Let Y [3] be the subset of P_0 consisting of all points that are contained in at least one of the balls $B\langle e\rangle$, $e \in S$. Let G be a $(1+\frac{1}{2}\varepsilon)$-spanner of $Y \cup P_1$.*

---

[2] *This and the next Lemma is not exactly the same as the first property, they implies the first property.*

[3] *This Y is a little different from what is defined in Section 5.4. Here we assume that every edge in the spanner S has a ball surround it, however this assumption will not affect the final proof for X later.*

Then, there is a subgraph H *of* G *that is connected with respect to vertices in* P_1 *and whose cost is upper bounded by* $(1+\varepsilon)$ *times the minimum-cost Steiner tree of* P_1 *with respect to* P_0.

Proof : Let T be the SMT of P_1 with respect to P_0. It can be assumed without loss of generality, that every Steiner point in T is of degree greater than 2. The approach is to construct the subgraph H of G using the following two-step procedure:

1. For every edge uv in T with $u, v \in Y \cup P_1$, H contains the shortest path in G between u and v.

2. After Step 1 H may be disconnected with respect to P_1; make H connected by adding additional edges having total minimum weight.

It is easy to see that the obtained graph H is connected with respect to $Y \cup P_1$ (and so, with respect to vertices in P_1 too). What will be proven below is that its cost is upper bounded by $(1+\varepsilon)$ times the cost of T. The proof consists of two parts. Because of the spanner property, the graph H obtained in Step 1 has its cost upper bounded by $(1+\frac{1}{2}\varepsilon)$ times the cost of the edges used in this step. Therefore, what really has to be proven is that the second part of graph H has an upper bound such that the total cost of H is bounded by $(1+\varepsilon)$ times the cost of T.

Take any edge uv in T, let ℓ denote its length, and let t be the midpoint of uv. Since it is known how to deal with the edges in T in which $u, v \in Y \cup P_1$, only the case when $u \notin Y \cup P_1$ needs to be considered. Remove edge uv from T and let SMT_u and SMT_v be the obtained two subtrees of T such that u is contained in SMT_u and v is contained in SMT_v. For any point $x \in \mathbb{R}^d$ and any real R, let B_R^x denote the ball with center at x and of radius R. Fix $k = 4\,d\,\log_2(16\,e\,\Delta\,d) = \Theta(d^2)$ and $r = 16\,\ell\,k\,\Delta^k/\varepsilon = 2^{\Theta(d^3)} \cdot \ell/\varepsilon$, such that $\rho_d = (3+\varepsilon)\,r\,\varepsilon/\ell$. Let T_u and T_v, respectively, denote the subgraph of SMT_u and SMT_v, respectively, induced by the vertices being at distance at most k edges from u and

v, respectively. In the following it will be shown that (assuming $u \notin Y \cup P_1$) there must be an edge xy in at least one of T_u and T_v whose length is greater than or equal to r/k.

- First it will be shown that it is impossible that at the same time all edges in T_u are of length smaller than $2r/k$, SMT_u contains a point $x \in P_1$ that is contained in B^t_{2r}, and SMT_v contains a point $y \in P_1$ that is contained in B^t_{2r}. (An equivalent case is when u is exchanged with v, that is, when all edges in T_v are of length smaller than $2r/k$, SMT_u contains a point $x \in P_1$ that is contained in B^t_{2r} and SMT_v contains a point $y \in P_1$ that is contained in B^t_{2r}.)

  If all edges in T_u are of length smaller than $2r/k$ then T_u is contained in B^t_{2r}. Since x and y are disconnected after removal of edge uv, Lemma 5.4.4 implies that S has at least one edge e on a shortest path $p_{x \rightsquigarrow y}$ between x and y whose length $|e|$ is at least ℓ.

  Now it will be shown that path $p_{x \rightsquigarrow y}$ cannot have a vertex outside $B^t_{(2+\varepsilon)r}$. Indeed, if path $p_{x \rightsquigarrow y}$ were not contained in $B^t_{(2+\varepsilon)r}$, then there were a point z in $p_{x \rightsquigarrow y}$ that is not contained in $B^t_{(2+\varepsilon)r}$. The length of the path $p_{x \rightsquigarrow y}$ is not shorter than $|xz| + |zy|$. But this contradicts the definition of the spanner, because the length of $p_{x \rightsquigarrow y}$ is assumed to be upper bounded by $1 + \varepsilon/4$ times the length of xy, and one can easily show (see, e.g., Lemma 5.10.1) that

  $$(1 + \varepsilon/4) \cdot |xy| \leq |xz| + |zy| \leq \text{ length of } p_{x \rightsquigarrow y} \ .$$

  So, now it is known that path $p_{x \rightsquigarrow y}$ must be contained in $B^t_{(2+\varepsilon)r}$. This in particular means that edge e is contained in $B^t_{(2+\varepsilon)r}$. Notice that the endpoints of edge e belong to P_1. Therefore, by the construction of set Y, all the points from P_0 that are contained in the ball $B\langle e \rangle$ (having the center at the midpoint of e and radius $\rho_d \cdot |e|/\varepsilon$) belong to Y. Now, since the length of e is greater than or equal to ℓ, the radius of $B\langle e \rangle$ is at least $\rho_d \ell/\varepsilon$. Since $\rho_d \ell/\varepsilon = (3 + \varepsilon) r$, observe that $B\langle e \rangle$ contains the entire ball B^t_r.

Therefore points u and v must belong to $Y \cup P_1$. This is a contradiction, because it is assumed that $u \notin Y \cup P_1$.

- Otherwise, there either must be edges in each of T_u and T_v of lengths greater than or equal to $2r/k$, or SMT_u must contain no point from P_1 that is contained in B^t_{2r}, or SMT_v must contain no point from P_1 that is contained in B^t_{2r}.

  Consider the case when SMT_u contains no point from P_1 that is contained in B^t_{2r}. Then, either T_u is contained in B^t_r or there is an edge in T_u of length greater than or equal to r/k. First show that T_u cannot be contained in B^t_r. Indeed, if T_u were contained in B^t_r, then B^t_r would contain more than $|T_u| > 2^k = (16\,e\,d\,\Delta)^{4d}$ Steiner points. On the other hand, it is known that B^t_{2r} contains no points from P_1 in SMT_u. By Lemma 5.4.5, this yields a contradiction. Hence, T_u cannot be contained in B^t_r. This in turn, implies that one of T_u and T_v, say T_u, must contain an edge of length greater than or equal to r/k.

Thus, one can conclude that there is an edge xy in T_u whose length is greater than or equal to $r/k = 16\,\ell\,\Delta^k/\varepsilon$. The cost of edge uv will be "charged" to xy in Step 1 of the construction of H.

Let K_1 be the set of edges in T having both endpoints in $Y \cup P_1$ and let K_2 denote the set of the remaining edges in T. Pick any edge e in T and denote its cost by L. Observe that e may be charged to by at most $2\,(2\,\Delta^k - 1) < 4\,\Delta^k$ "short" edges. Notice further, that the cost of each such a short edge is upper bounded by $L\,\varepsilon/(16\,\Delta^k)$. (Indeed, by construction, a short edge of length ℓ is charged to edge e only if $L \geq 16\,\ell\,\Delta^k/\varepsilon$.) Therefore, the total cost of all edges in T charged to e is upper bounded by $\frac{1}{4}\,\varepsilon\,L$. Furthermore, notice that only the edges in K_2 are charged in this way. This implies that the total length of all edges in K_2 is upper bounded by $\frac{1}{4}\,\varepsilon$ times the total cost of T.

Let H_1 denote the subgraph of H obtained in Step 1 and H_2 denote the subgraph of H obtained in Step 2 of the construction. Notice that the subgraph of T induced by the

edges in K_2 is a forest. Pick any tree τ in this forest and let $\mathcal{L}$ be the set of vertices in τ belonging to $\mathcal{L} \subseteq Y \cup P_1$. Clearly, τ is a Steiner tree of $\mathcal{L}$. It is well known that the cost of a minimum spanning spanning tree for any set of points (in any metric space) is upper bounded by twice the cost of the SMT. Therefore, the cost of a minimum spanning tree of $\mathcal{L}$ is upper bounded by twice the cost of τ. Hence, since the cost of H_2 is upper bounded by the sum of the costs of minimum spanning trees of $\mathcal{L}$ over all trees τ in K_2, the cost of H_2 is upper bounded by twice the cost of the edges in K_2. By the discussion above, the total length of all edges in K_2 is upper bounded by $\frac{1}{4}\,\varepsilon$ times the total cost of T. Thus, the cost of H_2 is upper bounded by $\frac{1}{2}\,\varepsilon$ times the total cost of T.

To summarize, the graph H consists of graph H_1 and H_2. By the spanner property, the cost of H_1 is upper bounded by $(1 + \frac{1}{2}\,\varepsilon)$ times the total cost of T. By our arguments above, the cost of H_2 is upper bounded by $\frac{1}{2}\,\varepsilon$ times the total cost of T. This implies that the total cost of H is upper bounded by $(1 + \varepsilon)$ times the total cost of T, and hence yields the proof of the lemma. □

First filtering property of set X. Lemma 5.4.6 gives "almost" a subset of P_0 we want to obtain in the Filtering phase. However, the so obtained set Y does not have to satisfy the second Filtering property, that is, that the cost of a minimum spanning tree of $Y \cup P_1$ is proportional to the cost of a minimum spanning tree of P_1. The problem is that if there are too many points in Y, even the cost of the minimum spanning tree of Y can be arbitrary large. Therefore, for investigations one can choose a subset of Y that is obtained by a "sparsification" of Y. One can slightly modify the proof of Lemma 5.4.6 to obtain the following result.

Lemma 5.4.7 *Let P_0 and P_1 be two point sets in $\mathbb{R}^d$. Let S be an $(1 + \frac{1}{4}\,\varepsilon)$-spanner of P_1. For each edge e of the spanner whose cost exceeds the $|P_1|^{-4}$ fraction of the cost of minimum spanning tree of P_1, let $B(e)$ be the ball with the center at the midpoint of e and with the radius equal to ρ_d/ε times the length of the edge e, where $\rho_d = 2^{\mathcal{O}(d^3)}$. For each*

ball $B\langle e\rangle$, *let* Y_e *be the subset of* P_0 *consisting of all points that are contained in it. Let* X_e *be an arbitrary subset of* Y_e *such that if* $y \in Y_e - X_e$ *then that there is* $x \in X_e$ *such that* $|yx| \leq |e|\,\varepsilon^2/(8\,\Delta\,\rho_d\,\sqrt{d})$. *Let* $X^* = \bigcup_{e\in S} X_e$ *and let* G *be a* $(1+\frac{1}{2}\,\varepsilon)$*-spanner of* $X^* \cup P_1$.

Then, there is a subgraph H *of* G *that is connected with respect to vertices in* P_1 *and whose cost is upper bounded by* $(1+\frac{3}{2}\,\varepsilon + o(1))$ *times the minimum-cost Steiner tree of* P_1 *with respect to* P_0.

Proof : The proof is a slight modification of the proof of Lemma 5.4.6. Therefore, only the places where that proof has to be modified will be referred.

Suppose first that even for the very short edges e of S the balls $B\langle e\rangle$ and the sets Y_e, X_e induced by them have been created.

One follows the arguments from the proof of Lemma 5.4.6. One needs to modify them only when there is a path $p_{x\rightsquigarrow y}$ that is contained in $B^t_{(2+\varepsilon)\,r}$ and therefore there is an edge e of length greater than or equal to ℓ that is contained in $B^t_{(2+\varepsilon)\,r}$ and has its both endpoints in P_1. In this case the contradiction will not be obtained as in the proof of Lemma 5.4.6. However, observe that since the length of e is less than or equal to $2\,(2+\varepsilon)\,r$ and since the ball $B\langle e\rangle$ contains u and v (and hence both points are in Y_e), there must exist two points (that might be identical) $w, z \in X_e$ with $|uw| \leq |e|\,\varepsilon^2/(8\,\Delta\,\rho_d\,\sqrt{d})$ and $|vz| \leq |e|\,\varepsilon^2/(8\,\Delta\,\rho_d\,\sqrt{d})$. Thus, one can "modify" all edges incident to u and v to have their endpoint in w and z, respectively. It is easy to see that each such a modification may increase the cost of the graph by at most

$$\Delta\,|uw| + \Delta\,|vz| \;\leq\; 2\,\Delta\,\frac{|e|\,\varepsilon^2}{8\,\Delta\,\rho_d\,\sqrt{d}} \;\leq\; \frac{(2\,(2+\varepsilon)\,r)\,\varepsilon^2}{4\,\rho_d} \;=\; \frac{2\,(2+\varepsilon)\,\rho_d\,\ell\,\varepsilon^2}{(3+\varepsilon)\,\varepsilon\,4\,\rho_d} \;\leq\; \ell\,\varepsilon/2\ .$$

Therefore, the construction of H can be modified as follows:

1. For every edge uv in T with $u, v \in X^* \cup P_1$, H contains the shortest path in G between u and v.

2. Otherwise, for every other edge uv in T, if there is an edge e in S (and hence, with both endpoints in P_1) such that u and v belong to Y_e, then pick the closest points

$w, z \in X^*$ for u and v, respectively. Then, modify T by moving all edges incident to u and v to have their endpoint in w and z, respectively. If any new edge has both endpoints in $X^* \cup P_1$ then do as in Step 1.

3. Afterwards H may be still disconnected with respect to P_1; make H connected by adding additional edges having total minimum weight.

One can apply the same arguments as in the proof of Lemma 5.4.6 to show that the cost of the edges in H created in Steps 1 and 3 is upper bounded by $(1 + \varepsilon)$ times the minimum-cost Steiner tree of P_1 with respect to P_0. On the other hand, by the arguments above, the total cost of the modifications performed in Step 2 is upper bounded by $\sum_{(uv)\in T} |uv|\, \varepsilon/2$, which is $\frac{1}{2}\varepsilon$ times the minimum-cost Steiner tree of P_1 with respect to P_0.

To obtain the claimed result, it remains to move all the points in X_e for the edges e whose costs do not exceed $|P_1|^{-4}$ of the cost of the minimum spanning tree of P_1 to an endpoint of e. Such movements can increase the total cost of G by $O(|P_1|^{-3}\,(d/\varepsilon)^{\mathcal{O}(d)}\, n)$ (since the total number of edges in the spanner is $(d/\varepsilon)^{\mathcal{O}(d)}\, n$) times the cost of the minimum spanning tree which with increasing $|P_1|$ is arbitrarily small in comparison to the cost of a minimum Steiner tree of P_1 with respect to P_0. □

The following is immediate corollary of Lemma 5.4.7.

Corollary 5.4.8 *Under the assumptions of Lemma 5.4.7, for any $Z \supseteq X^*$, the cost of the SMT of P_1 with respect to Z is at most $1 + \frac{3}{2}\varepsilon$ times than the cost of the SMT of P_1 with respect to P_0.* □

5.4.2 Second Filtering Property

In order to prove that set X satisfies the second filtering property, the following Lemma will be proved first.

Lemma 5.4.9 *Let P_0 and P_1 be two point sets in $\mathbb{R}^d$. Let S be an $(1 + \frac{1}{4}\varepsilon)$-spanner of P_1. For each edge e of the spanner whose cost exceeds the $|P_1|^{-4}$ fraction of the cost of*

minimum spanning tree of P_1, *let* $B(e)$ *and* $\widehat{B(e)}$ *be the balls with the center at the midpoint of* e *and with the radius equal to* $|e| \cdot \rho_d/\varepsilon$ *and* $4 \cdot |e| \cdot \rho_d/\varepsilon$, *respectively, where* $\rho_d = 2^{\mathcal{O}(d^3)}$. *For each ball* $B(e)$, *let* Y_e *be the subset of* P_0 *consisting of all points that are contained in it, and let* $\widehat{Y}_e$ *be any subset of* P_0 *containing all points in* Y_e *and possibly some other points contained in the ball* $\widehat{B(e)}$. *Let* $\widehat{X_e}$ *be any subset of* $\widehat{Y}_e$ *such that (i) if* $y \in \widehat{Y}_e - \widehat{X_e}$ *then that there is* $x \in \widehat{X_e}$ *such that* $|yx| \leq |e|\,\varepsilon^2/(8\,\Delta\,\rho_d\sqrt{d})$ *and (ii) if* $x, y \in \widehat{X_e}$ *then* $|xy| \geq |e|\,\varepsilon^2/(8\,\Delta\,\rho_d\sqrt{d})$. *Let* $\widehat{X} = \bigcup_{e \in S} \widehat{X_e}$.

Then, there is a spanning tree of $\widehat{X} \cup P_1$ *whose total cost is upper bounded by*

$$(\rho_d/\varepsilon) \cdot \left(\mathcal{O}\left(\sqrt{d}\,\Delta\,\rho_d^2/\varepsilon^3\right)\right)^{d-1} = 2^{\mathcal{O}(d^4)}/\varepsilon^{\mathcal{O}(d)}$$

times the cost of the spanner S.

Proof : First a spanning graph $\mathcal{T}$ of $\widehat{X} \cup P_1$ will be constructed. First of all, $\mathcal{T}$ contains a minimum spanning tree of P_1. Then, for every edge $e \in S$ we find a minimum spanning tree T_e of $\widehat{X_e}$ (for convention, if $\widehat{X_e}$ is undefined, it will be treated as an empty set), connect it to any of the endpoints of edge e, and add the obtained tree to $\mathcal{T}$. It is easy to see that so defined graph $\mathcal{T}$ is a spanning tree of $\widehat{X} \cup P_1$. Therefore, the cost of the minimum spanning tree of $\widehat{X} \cup P_1$ is upper bounded by the cost of $\mathcal{T}$ and therefore the attention will be focused on estimating the cost of $\mathcal{T}$.

Fix an arbitrary edge e in S. Note that if $x, y \in \widehat{X_e}$, then $|xy| \geq |e|\,\varepsilon^2/(8\sqrt{d}\,\Delta\,\rho_d\sqrt{d})$. Furthermore, all points in $\widehat{X_e}$ are contained in a ball of radius upper bounded by $4\,|e|\,\rho_d/\varepsilon$. This immediately implies that the size of $\widehat{X_e}$ is $\left(\mathcal{O}\left(\sqrt{d}\,\Delta\,\rho_d^2/\varepsilon^3\right)\right)^d$. Hence, by Lemma 5.4.2, the cost of T_e is $(4|e|\,\rho_d/\varepsilon) \cdot \left(\mathcal{O}\left(\sqrt{d}\,\Delta\,\rho_d^2/\varepsilon^3\right)\right)^{d-1}$. Therefore, if $\mathrm{MST}(P_1)$ denotes the cost of the minimum spanning tree of P_1 and by $\mathrm{COST}(S)$ the cost of the spanner S, one can obtain the following upper bound for the total cost of $\mathcal{T}$:

$$\mathrm{MST}(P_1) + \sum_{e \in S} (4|e|\rho_d/\varepsilon) \cdot \left(\mathcal{O}\left(\sqrt{d}\,\Delta\rho_d^2/\varepsilon^3\right)\right)^{d-1} < (4\rho_d/\varepsilon) \cdot \left(\mathcal{O}\left(\sqrt{d}\,\Delta\rho_d^2/\varepsilon^3\right)\right)^{d-1} \cdot \mathrm{COST}(S)\ .$$

□

Since X satisfies the two properties of $\widehat{X}$, let $\widehat{X} = X$ and apply X to the above lemma, then there is a spanning tree of $X \cup P_1$ whose total cost is upper bounded by $(4\rho_d/\varepsilon) \cdot \left(\mathcal{O}\left(\sqrt{d}\,\Delta\,\rho_d^2/\varepsilon^3\right)\right)^{d-1} = 2^{\mathcal{O}(d^4)}/\varepsilon^{\mathcal{O}(d)}$ times the cost of the spanner S. Recall that the X is built upon the spanner whose cost is bounded by $\xi_{d,\varepsilon} = (d/\varepsilon)^{\mathcal{O}(d)}$ times $\mathrm{MST}(P_1)$, one can easily see that the cost of Minimum Spanning tree of $X \cup P_1$ is bounded by: $\lambda_{d,\varepsilon} = (2^{\mathcal{O}(d^4)}/\varepsilon^{\mathcal{O}(d)}) \cdot (d/\varepsilon)^{\mathcal{O}(d)} = \mathcal{O}(2^{\mathcal{O}(d^4)}/\varepsilon^{\mathcal{O}(d)})$ times the cost of Minimum Spanning tree of P_1 $\mathrm{MST}(P_1)$

5.4.3 Complexity of SMT-Filtering

This section shows how to implement SMT-Filtering phase. The construction of the set X described above has been specially tuned to enable an efficient implementation. The following lemma will be proved first.

Lemma 5.4.10 *The filtering algorithm for SMT can be implemented to determine the set* X *in time* $(d/\varepsilon)^{\mathcal{O}(d)} \cdot n \log n$, *where* $n = |P_0 \cup P_1|$.

Proof: By Lemma 5.2.2, Step 1 can be implemented in time $(d/\varepsilon)^{\mathcal{O}(d)} \cdot n + \mathcal{O}(d \cdot n \cdot \log n)$.

To implement Steps 2, 3, i.e., to determine the set Y, one can use a core data structure for *approximate point location in equal balls* due to Indyk and Motwani [67]. For a given approximation factor $c \geq 1$ and a given real r, they designed a static data structure for a set of points $S \subseteq \mathbb{R}^d$, called (c, r)-PLEB, such that for any point $q \in \mathbb{R}^d$, if S contains a point within distance r from q then (c, r)-PLEB outputs a point $q \in S$ that is promised to be within a distance at most $c\,r$ from q. Indyk and Motwani [67, Theorem 3] show that there is an algorithm for $(2, r)$-PLEB (in the Euclidean d-dimensional metric) that after an $\mathcal{O}(|S|\,2^{\mathcal{O}(d)})$-time preprocessing achieves $\mathcal{O}(d)$ query time. Note that in the algorithm above, the costs of the spanner edges from which the balls originate fall into a logarithmic number of intervals of the form $[2^i l_0, 2^{i+1} l_0)$, where l_0 is their minimum cost (which is lower bounded by the cost of the minimum spanning tree of P_1 over $|P_1|^4$). To determine Y, for $i = 0, \ldots, \mathcal{O}(\log n)$, one can build the $(2, 2^{i+1} l_0\, \rho_d/\varepsilon)$-PLEB data structure

Remark 5.5.2 *The key property of the construction in Lemma 5.5.1 is that the obtained graph* H *has* $P_0 \cup P_1$ *as its vertex set. This distinguishes it from previous constructions, e.g., [3, 101], where a related graph* H *was allowed to use arbitrary points outside* $P_0 \cup P_1$. *This property is critical in the approximation of SMT and other connectivity problems (in contrast to, e.g., TSP approximation). Indeed, suppose that* H *has as a vertex a point* $q \notin P_0 \cup P_1$ *which is a cut-vertex in* H *and that* H *contains some tree* T *to be pruned to a Steiner tree for* P *(or its subset). Then, if the degree of* q *in* T *is very high, it might be impossible to remove* q *from* T *to obtain a tree of the cost as good (or almost as good) as the cost of* T *(in contrast, TSP on a superset of* P *can be easily modified to obtain a TSP on* P *without any cost increase). Observe that this construction allows to* bend *the edges, that is, an edge between two points* x *and* y *in* P *is a path of straight-line segments between* x *and* y, *see the discussion at the beginning of Section 5.2.* ♠

Now the main ideas behind the proof Lemma 5.5.1 will be discussed. As mentioned, this lemma uses (almost directly) results already developed in [26]. Therefore the results presented here are given without proofs.

Theorem 3.1 from [26] is the key technical theorem in that paper.

Lemma 5.5.3 [26] *Let* ε *and* λ *be any positive but otherwise arbitrary reals (that is, they may depend also on other parameters). Let* k *be an arbitrary positive integer. Let* P *be an arbitrary point set in* $\mathbb{R}^d$. *Let* G *be a spanner for* P *that has* $n\,(d/\varepsilon)^{O(d)}$ *edges, has total cost* $\mathrm{COST}(G)$, *and that satisfies the* $(t, 1+\frac{1}{2}\varepsilon)$*-leapfrog property, where* $1 < t \leq 1+\frac{1}{2}\varepsilon$. *Choose a shifted dissection uniformly at random. Then, one can modify* G *to a graph* H *with vertex set* P *such that*

- H *is* τ*-locally-light with respect to the shifted dissection chosen, where* $\tau = 2^{O(d)} \cdot k^2 + (d/\varepsilon)^{O(d)} + d \cdot (\mathcal{O}(\lambda \cdot d^{3/2}))^d$, *and*
- *there exists a* k*-edge-connected multigraph* M *which is a spanning subgraph of* H *with possible parallel edges (of multiplicity at most* k*) whose expected cost (over the*

for all the center points of spanner edges having cost in the interval $[2^i l_0, 2^{i+1} l_0)$, and then query it with all the points in P_0. The time needed for the construction of the logarithmic number of PLEB data structures is $\mathcal{O}(\log n)$ times the number of spanner edges (which is $n\,\xi_{d,\varepsilon} = n\,(d/\varepsilon)^{\mathcal{O}(d)}$) and the time needed for the $|P_0|$ queries is $\mathcal{O}(\log n) \times |P_0| \times \mathcal{O}(d)$ [67]. Hence, the total time required in Steps 2 and 3 is $\mathcal{O}(n\,(d/\varepsilon)^{\mathcal{O}(d)}\,\log n)$.

Observe that during the construction of Y, one can also insert each point accounted to Y into a list corresponding to the smallest ball it belongs to (approximately) by processing the $\mathcal{O}(\log n)$ PLEB queries without increasing the asymptotic time performance. These lists are useful in the implementation of Step 4. Simply, for each of the balls $B\langle e\rangle$, and for each point p in the corresponding list $L\langle e\rangle$, one checks whether or not the point p after rounding off to the nearest point on the grid $C\langle e\rangle$ is already marked. If not, one marks the grid point and add p to X. It is easy to see that in this way Step 4 can be implemented in time $\mathcal{O}((d/\varepsilon)^{\mathcal{O}(d)} \cdot n)$. □

5.5 Lightening for SMT

For the Lightening phase, the framework developed in [26] will be used to transform spanners into r-locally-light graphs maintaining connectivity properties.

Lemma 5.5.1 *Let P_0 and P_1 be sets of points in $\mathbb{R}^d$ and let $\varepsilon > 0$. Let $r = (d/\varepsilon)^{\mathcal{O}(d^2)}$. Let G be any $(1+\frac{1}{4}\varepsilon)$-spanner of $P_0 \cup P_1$ that has $n\,(d/\varepsilon)^{\mathcal{O}(d)}$ edges, whose total cost is upper bounded by $(d/\varepsilon)^{\mathcal{O}(d)}$ times the cost of the minimum spanning tree of P_1 and that satisfies the $(t, 1+\frac{1}{4}\varepsilon)$-leapfrog property, where $1 < t < 1+\frac{1}{4}\varepsilon$. Then, one can transform G to obtain a graph H with vertex set $P_0 \cup P_1$ (i) that is r-locally-light and (ii) that contains as its subgraph a Steiner tree of P_1 with respect to P_0 whose cost is at most $(1+\frac{1}{2}\varepsilon)$ times the cost of the SMT of P_1 with respect to P_0. Moreover, this transformation can be performed in time $\mathcal{O}(d^{3/2} n \log n) + n\,(2^{d^{\mathcal{O}(d)}} + (d/\varepsilon)^{\mathcal{O}(d)})$.*

random choice of the shifted dissection) is upper bounded by $(1+\frac{1}{2}\,\varepsilon+\frac{\mathrm{COST}(G)}{\lambda\cdot \mathrm{MST}(P)})$ *times the minimum-cost* k*-edge-connected multigraph spanning* P*, where* $\mathrm{MST}(P)$ *denotes the cost of the minimum spanning tree of* P.

Moreover, the modification can be performed in time $\mathcal{O}(d^{3/2}\cdot n\cdot \log n)+n\cdot(2^{d^{\mathcal{O}(d)}}+(d/\varepsilon)^{\mathcal{O}(d)})$. □

Remark 5.5.4 *Notice that the original theorem in [26] the so-called "isolation property" of the spanner was required. However, the proof of that theorem can be easily modified to spanners satisfying the "leapfrog property" (see [56] for a precise definition). The reason of this change is that in [26] the authors were using the spanner construction due to Arya* et al. *[5]. Unfortunately, it has been shown (see, e.g., [56]) that there is a serious flaw in the construction due to Arya* et al. *The only known existing construction of "optimal" spanners is due to Gudmundsson* et al. *[56], see Lemma 5.2.2. This construction satisfies the leapfrog property and therefore we gave a modified version of Theorem 3.1 from [26].* ♠

Remark 5.5.5 *Some brief intuitions why the leapfrog property is required (and why it can replace the isolation property) in the proof of Lemma 5.5.3will be given here . As it is shown in [25, Lemma 2.3] (a predecessor paper of [26]), with a very little cost increase one can transform any spanner (or any* k*-edge-connected multigraph) into a graph that is "almost"* τ*-locally-light. By "almost" it is meant that the number of "short" relevant crossings is at most* $\tau/2$*, where a "short" crossing of a facet of side length* L *is of the length* $\mathcal{O}(\sqrt{d}\cdot L)$*. However, the number of longer crossings may be significantly larger. But if the input spanner satisfies the leapfrog property then one can show that the number of "long" relevant crossings is small. This allows to prove that the construction leads to an* τ*-locally-light graph.* ♠

Remark 5.5.6 *Actually, Lemma 5.5.3 requires that the input points are slightly "perturbed" and are so-called "well-rounded." This modification of the input points is used in all papers following Arora's framework (see, e.g., [3, 26, 101]). It requires to move all input points (by a very tiny vector) to a certain grid in* $\mathbb{R}^d$. *Although formally this step is very important in the algorithm, for simplicity of presentation it will be neglected.* ♠

Remark 5.5.7 *Finally notice that the key feature of the construction in Lemma 5.5.3 is that the obtained graph* H *has* P *as its vertex set. This is the key property that distinguishes it from previous construction, for example, as in [3, 101]. This property is not required if one wants to find an approximation, for example, for TSP, but it is critical when dealing with the SMT problem and other connectivity problems. Indeed, suppose that* H *is using some new point* $q \notin P$ *and this vertex is a cut-vertex in* H *and suppose that* H *contains a certain tree* T *that we would like to use as a Steiner tree for* P *(or its subset). Then, if the degree of* q *in this tree is very high, then it might be impossible to remove* q *from* T *to obtain a tree of the cost as good (or almost as good) as the cost of* T. *(Notice that this is easy to be done in the case of TSP, as in [3, 101], because TSP on a superset of* P *can be easily modified to obtain a TSP on* P *without any cost increase.) Additionally, this construction allows to* bend *the edges, that is, an edge between two points* x *and* y *in* P *is a path of straight-line segments between* x *and* y, *see the discussion at the beginning of Section 5.2.* ♠

Keeping in mind the remarks above, notice that the proof of Lemma 5.5.3 is actually much stronger. The proof of Lemma 5.5.3 is performed by a sequence of removals of certain edges from and inserting new edges to G. The idea behind inserting the edges is that they are required to keep the same connectivity of the obtained graph as of G. This does not mean only global connectivity, but also the local one (up to connectivity k). Therefore, the proof (without any modification) of Lemma 5.5.3 leads to the following much stronger result (that is stated here without a proof).

Lemma 5.5.8 *Let ε and λ be any positive but otherwise arbitrary reals (that is, they may depend also on other parameters). Let k be an arbitrary positive integer. Let P be an arbitrary point set in $\mathbb{R}^d$. Let G be a $(1+\frac{1}{2}\varepsilon)$-spanner for P that has $n\,(d/\varepsilon)^{\mathcal{O}(d)}$ edges, has total cost $\mathrm{COST}(G)$, and that satisfies the $(1+\frac{1}{2}\varepsilon, t)$-leapfrog property, where $1 < t \leq 1+\frac{1}{2}\varepsilon$. Choose a shifted dissection uniformly at random. Then, one can modify G to a graph H with vertex set P such that*

- *H is τ-locally-light with respect to the shifted dissection chosen, where $\tau = 2^{\mathcal{O}(d)} \cdot k^2 + (d/\varepsilon)^{\mathcal{O}(d)} + d \cdot (\mathcal{O}(\lambda \cdot d^{3/2}))^d$,*
- *there exists a multigraph M which is a spanning subgraph of H with possible parallel edges (of multiplicity at most k) such that*
  1. *for every pair of points x, y in P, if there are τ_{xy} edge-disjoint paths between x and y in G then there are at least $\min\{\tau_{xy}, k\}$ edge-disjoint path between x and y in M,*
  2. *the expected cost of M (over the random choice of the shifted dissection) is upper bounded by $(1+\frac{1}{2}\varepsilon+\frac{\mathrm{COST}(G)}{\lambda\cdot\mathrm{MST}(P)})$ times the minimum-cost multigraph spanning P satisfying the above property 1, where $\mathrm{MST}(P)$ denotes the cost of the minimum spanning tree of P.*

Moreover, the modification can be performed in time $\mathcal{O}(d^{3/2} \cdot n \cdot \log n) + n \cdot (2^{d^{\mathcal{O}(d)}} + (d/\varepsilon)^{\mathcal{O}(d)})$. □

This lemma directly implies Lemma 5.5.1 after setting the parameters appropriately. Indeed, pick any $(1+\frac{1}{2}\varepsilon)$-spanner G of $P_0 \cup P_1$ that satisfies the assumptions of Lemma 5.5.1. Pick $\lambda = \frac{2\cdot\mathrm{COST}(G)}{\varepsilon\cdot\mathrm{MST}(P)} = (d/\varepsilon)^{\mathcal{O}(d)}$ and set $k = 1$. Apply Lemma 5.5.8 to G to construct the promised graph H. Now, it is easy to verify that H satisfies the requirement of Lemma 5.5.1, which completes the proof of Lemma 5.5.1.

5.6 Searching for SMT

The approach in the Searching phase is to apply dynamic programming to find an optimal Steiner tree in a r-locally-light graph.

Lemma 5.6.1 *Let P_0 and P_1 be sets of jointly n points in $\mathbb{R}^d$ and let r be any integer. Let G be an r-locally-light (with respect to a certain given shifted dissection) graph on $P_0 \cup P_1$. Then, in time $\mathcal{O}(n \cdot r^{\mathcal{O}(2^d r)})$ one can find a minimum-cost Steiner tree T of P_1 with respect to P_0 that is a subgraph of G.*

This lemma is proved by showing how to use dynamic programming to find a minimum-cost Steiner tree T of P_1 with respect to P_0. It will be shown that the running time of the construction is $\mathcal{O}(n \cdot r^{\mathcal{O}(2^d r)})$. The dynamic programming procedure is essentially the same as the one used in PTAS algorithms for TSP and the Euclidean complete Steiner tree problems due to Arora [3, 101], but for the sake of completeness it is presented here in details.

It should be mentioned here that like [3, 25, 26, 101], in the dynamic programming procedure, the input is assumed to be a well-rounded point set, which is done by Perturbation to the original points. Like mentioned before, for simplicity of presentation it will be neglected.

First define the subproblem in a region R of the dissection. *Let S be a set containing $m \leq r$ relevant crossings on the facets of R. Given a partition $(S_1, S_2, \ldots, S_k)$, $1 \leq k \leq m$ of S, find a minimum-cost steiner forest that (i) consist of k trees, (ii) the i-th tree T_i of the forest contains all the crossings (called portals in [3, 101]) in S_i as their leaves, and (iii) all the trees of the forest collectively contain all points $p \in P_1$ in this region.* If no such forest exists for this partition, this partition is said to be *invalid*.

The tree T is found in a bottom-up fashion. First a minimum-cost Steiner forest for each leaf region is found, then the minimum cost forests for each non-leaf region by combining the optimal solutions of its child regions. The T we are looking for is the minimum-cost forest for the root region.

1. For the subproblems of a leaf region, there is at most one point in the region. The computation is trivial, one can easily compute the optimal solution in $\mathcal{O}(r)$. There are three cases, depending on the number and type of the point:

   - No point, therefore no relevant crossings at all. The forest is empty, and the cost is always 0.
   - One point $p \in P_1$. There are only several ways of partition of S that are valid, and p must be contained in one of the tree, which makes the total cost of the forest for the partition minimum.
   - One point $p \in P_0$. Almost the same as the second case, except that here p is a Steiner candidate, so p is not required to be contained in the forest. Specifically when $k = m$, the optimal solution is a forest with m trees each containing only one crossing and the total cost is 0 .

2. For each maximal sequence of regions $R_1, \ldots R_q$ such that for $i = 1, \ldots, q-1$, R_{i+1} is the only child region of R_i that has points in it, the minimum cost forest within R_1 can be easily computed from that within R_q just by extending the latter along the edges that cross the facets of R_q.

3. For the subproblems arising from a non-leaf region with more than one point in it, it will be shown that one can compute all its optimal solutions each with respect to a particular partition of the crossings in time $\mathcal{O}(r^{\mathcal{O}(2^d r)})$.

   Each forest in parent region R is a combination of forests from its child regions $R_1, R_2, \ldots, R_{2^d}$ respectively. So the minimum-cost forest in R can be found by enumerating all combinations of forests from its child region. The combination is done through overlapping crossings on the shared facets of child regions. If there is no mismatched crossings, a forest in the parent region is obtained. This forest also defines the crossings set S on the border of R and a partition of S. For a specific set S

and a specific partition of S, there may be many combinations correspond to it, one needs to find the one that gives the minimum cost forest.

Note that if a crossing shared by two child regions is relevant for one child, but not for another. That is there is an edge which comes from a point outside the region, gets through one of its child and ends in another child. The combination is still valid. The segment of the edge in the former child is not included in the computation for the child, but it needs to be added to the cost of the parent region.

There are $\mathcal{O}(r^r)$ forests for each child region, so the time complexity of combination is $\mathcal{O}(r^{(2^d r)}$

For a shifted dissection, it has $\mathcal{O}(n)$ leaf regions with a point in it, and has only $\mathcal{O}(n)$ non-leaf regions with more than one point in them, so the time complexity of the whole problem is $O(n \cdot r^{\mathcal{O}(r)})$.

5.7 Polynomial-Time Approximation Scheme for SMT

Now, it will be shown how to combine all the arguments from Sections 5.4 – 5.6 to obtain a PTAS for the Euclidean SMT problem. The input to the problem consists of two sets P_0 and P_1 of points in $\mathbb{R}^d$ of total size $|P_0| + |P_1| = n$. The goal is to find a Steiner tree of P_1 with respect to P_0 whose cost is less than or equal to $1 + \varepsilon$ times the minimum.

First apply Lemma 5.4.1 to find a subset X of P_0 having the promised properties (with $t = \frac{1}{4}\varepsilon$). Then, take a $(1 + \frac{1}{4}\varepsilon)$-spanner G for $X \cup P_1$ and apply Lemma 5.5.1 to modify G in order to obtain an r-locally-light graph H that has as a subgraph a Steiner tree of P_1 with respect to X whose cost is upper bounded by $(1 + \frac{1}{2}\varepsilon)$ times the cost of the SMT of P_1 with respect to X. Finally, apply Lemma 5.6.1 to find a minimum-cost Steiner tree of P_1 with respect to X that is a subgraph of H and output it. This leads to the following theorem.

Theorem 5.7.1 *There is a polynomial-time approximation scheme for the Steiner Minimum Problem in Euclidean space* $\mathbb{R}^d$. *In particular, for any sets of points* P_0 *and* P_1 *in* $\mathbb{R}^d$ *of total size* n, *in time* $\mathcal{O}(n \log n\,(d/\varepsilon)^{\mathcal{O}(d)}) + \mathcal{O}(n\,(d/\varepsilon)^{(d/\varepsilon)^{\mathcal{O}(d^2)}})$ *one can find a Steiner tree of* P_1 *with respect to* P_0 *whose cost is at most* $1+\varepsilon$ *times the minimum.*

For constant d *and* ε, *the running time of this algorithm is* $\mathcal{O}(n \log n)$. □

5.8 {0, 1, 2}-Connectivity Problem

One can extend the algorithm from the previous section to obtain a polynomial-time approximation scheme for the {0, 1, 2}-Connectivity Problem in Euclidean graphs. Actually, special attention has been paid to present the algorithm for the SMT problem in a form extendable to include the {0, 1, 2}-Connectivity Problem.

Due to Lemma 5.2.8, one may consider only the {0, 1, 2}-edge-connectivity problem and allow the output to be given in a form of a multigraph. The algorithm uses similar three phases as the algorithm for the SMT problem. The Filtering phase is essentially the same as for the SMT problem except that one works on the spanner of $P_1 \cup P_2$ now in order to find X and Y. Similarly, the Lighting phase, is essentially the same as for the SMT problem, see Section 5.8.1. Searching phase is the only phase that is completely different and rather tricky, but still one can implement this phase efficiently by using the idea of *connectivity type* of the multigraph within a region of the dissection, the detailed description of the dynamic procedure and the proof of its correctness is in Section 5.8.2.

5.8.1 Lightening for {0, 1, 2}-Edge-Connectivity

For {0, 1, 2}-edge-connectivity, one can argue analogously as in the proof of the Lightening Lemma for SMT.

Lemma 5.8.1 *Let* P_0, P_1 *and* P_2 *be sets of points in* $\mathbb{R}^d$. *Let* $\varepsilon > 0$ *and* $r = (d/\varepsilon)^{\mathcal{O}(d^2)}$. *Let* G *be any* $(1+\frac{1}{2}\varepsilon)$*-spanner of* $P_0 \cup P_1 \cup P_2$ *that has* $n\,(d/\varepsilon)^{\mathcal{O}(d)}$ *edges, whose total cost upper bounded by* $(d/\varepsilon)^{\mathcal{O}(d)}$ *times the cost of the minimum spanning tree of* $P_1 \cup P_2$

and that satisfies the $(t, 1+\frac{1}{2}\varepsilon)$ *-leapfrog property, where* $1 < t < 1+\frac{1}{2}\varepsilon$. *Then, one can transform* G *to obtain a graph* H *with vertex set* $P_0 \cup P_1 \cup P_2$ *(i) that is* τ*-locally-light and (ii)there is a sub-multigraph* M *whose induced graph is* H, *and it satisfies the connectivity requirement of every vertex in* M, *the cost of* M *is at most* $(1+\varepsilon)$ *times the cost of the minimum-cost multigraph having the same connectivity property. Moreover, this transformation can be performed in time* $\mathcal{O}(d^{3/2} \cdot n \cdot \log n) + n \cdot (2^{d^{\mathcal{O}(d)}} + (d/\varepsilon)^{\mathcal{O}(d)})$.

It is easy to transform the arguments used in the proof of Lemma 5.5.1 and in Section 5.5 to prove the above claim. Pick any $(1+\frac{1}{2}\varepsilon)$-spanner G of $P_0 \cup P_1 \cup P_2$ that satisfies the assumptions. One can still use Lemma 5.5.8 here, let $\lambda = \frac{2 \cdot \text{COST}(G)}{\varepsilon \cdot \text{MST}(P)} = (d/\varepsilon)^{\mathcal{O}(d)}$ and $k = 2$, construct the graph H, and it is easy to see H meets the requirement of the claim.

Remark 5.8.2 *The arguments above can be also easily modified to include the Lightening Lemma for arbitrary* $\{0, 1, \ldots, k\}$*-edge-connectivity in the same time complexity.*

5.8.2 Dynamic Programming for $\{0, 1, 2\}$-Edge-Connectivity

The goal of searching phase for $\{0, 1, 2\}$-edge-connectivity is using dynamic programming to solve the following problem: Let $P = P_0 \cup P_1 \cup P_2$ be a well-rounded point set in $\mathbb{R}^d$, where P_i is the set of points whose connectivity requirement is i. Given an arbitrary shifted dissection, let G be an Euclidean graph on P that is τ-locally-light with respect to this dissection. Find the minimum-cost $\{0, 1, 2\}$-edge-connected multigraph H on P for which the induced graph is a subgraph of G.

Before describing the procedure of dynamic programming, some definitions will be introduced first.

For any region R of the dissection, after duplicating some of the edges of G, a *multigraph within* R is the part of the multigraph contained within R resulting from the removal of all edges that cross R but have no endpoint inside R (see, Figure 5.2 (a–b)). The multigraph within R has two types of points, those from P, which will be called the *input*

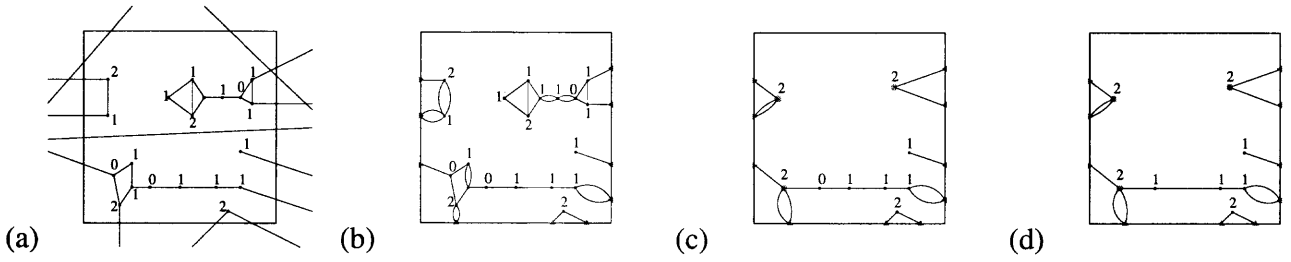


Figure 5.2 Illustration of connectivity type construction: (a) A part of graph G inside a region R, (b) a multigraph M_R within R, (c) contraction of two-edge-connected components in M_R, and (d) contraction of paths in M_R.

points, and those defined by the crossings of the edges with the border of P, which will be called the *border points*.

The idea of *connectivity type* introduced in [25, 26] will be used here, but different and more subtle characterizations are needed. The *connectivity type* of a multigraph is the ("pseudo"-)forest obtained by the following steps (See Figure 5.2 for an illustration).

- Contracting each maximal two-edge-connected component in the multigraph induced by its non-leaf vertices to a single vertex, and associate a connectivity requirement with this vertex which equals to the maximum connectivity requirement among all vertices of the component.

- Contracting each maximal path composed of input and/or contracted points of degree two into the single edge with the endpoints being the first and the last vertex at the path, and associating with each endpoint of the new edge the connectivity requirement equal to the maximum connectivity requirement among all the points on the path.

It can happen that a "leaf" in such a pseudo-forest have two edges connecting it with its parents(thus, there are possible cycles of length two). This explains the name "pseudo-forest". This issue shall be ignored and the notation shall be slightly abused by calling such "pseudo- forests" as forest in the continuation.

From the definition above, it is known that a connectivity type is a forest and two multigraphs are said to have the same connectivity type if the forests are isomorphic. For a region with at most r relevant crossings, each forest obtained from a multigraph whose induced graph is the graph within the region has at most r leaves, thus less than $\mathcal{O}(2r)$ vertices, so there are at most $r^{\mathcal{O}(r)}$ such forests, and therefore there are at most $r^{\mathcal{O}(r)}$ connectivity types.

Dynamic programming. The dynamic programming procedure will determine for each region and for each possible connectivity type, the minimum-cost multigraph of this type within the region.

1. For each leaf region, each possible multiset S of at most r crossings on the facets of the region, since there is at most one point in the region, the minimum-cost multigraph of each connectivity type corresponding to S can be easily computed in time $r^{\mathcal{O}(r)}$.

2. Like dynamic programming in SMT, for each maximal sequence of regions $R_1, \ldots R_q$ such that for $i = 1, \ldots, q-1$, R_{i+1} is the only child region of R_i that has points in it, the minimum cost multigraph of each connectivity type within R_1 can be easily computed from those within R_q in constant time. Since there are $\mathcal{O}(r^r)$ connectivity types, so the total time needed is $\mathcal{O}(r^r)$.

3. Let R be any other non-leaf region R. Let $Q_1, Q_2, \ldots Q_{2^d}$ be an arbitrary connectivity types for the child regions $R_1, R_2, \ldots R_{2^d}$ respectively. These connectivity types(pseodo-forests) are combined through the overlapping crossings and then eliminate these vertices from the resulting graph and add single or double edges crossing the common facets at these vertices. If a crossing on a common facet is a relevant crossing in one child region, but not in the other, then the combination is valid. If a valid graph is obtained from the combination, then its connectivity type is computed by performing the necessary contraction. By enumerating all combinations, the

minimum-cost of each connectivity type and its corresponding multigraph in R can be found.

The total running time of the combination, contraction is obviously linear in the total size of these connectivity types, that is $\mathcal{O}(2^d r)$. Since there are at most $\mathcal{O}(r^{\mathcal{O}(r)})^{2d}$ possible combinations, the computation for a non-leaf region takes $\mathcal{O}(2^d r)\cdot(r^{\mathcal{O}(r)})^{2d} = r^{\mathcal{O}(2^d r)}$.

4. Finally after the minimum-cost multigraph of each connectivity type for the root region is computed, the optimal multigraph for the original problem can be easily found by just taking the one with minimum cost among those connectivity types which consist of a tree with only one vertex having connectivity requirement of 2 and some isolated vertices belonging to P_0.

From the analysis above, it is easy to see that the complexity of the searching phase is $\mathcal{O}(n \cdot r^{\mathcal{O}(2^d r)})$.

Correctness of the dynamic programming. Now, a sketch will be given to prove that the multigraph obtained satisfies the connectivity requirements and has minimum cost. First the multigraph will be shown to meet the connectivity requirement of each input point. This is obviously true if one looks at connectivity type of this multigraph found. It is a tree together with some points of P_0, so all points of $P_1 \cup P_2$ are connected, furthermore only one vertex having connectivity requirement of 2, this means all points belong to P_2 are contained in a two-edge-connected components represented by this vertex.

Next, it will be proved that its cost is minimum among all such multigraphs. It is enough to show that the solution is optimal in each region, i.e., in each region the sub-multigraph of the solution within it has minimum cost among all multigraphs within this region having the same connectivity type. This can be proven by contradiction. Suppose the solution is not optimal in some regions. Take any such a region at the lowest level. Replace the sub-multigraph in this region of the solution by the optimal solution having

the same connectivity type in this region. It is easy to see that a new multigraph satisfying the connectivity requirements but with lower cost than the solution is obtained, but this is contradict to the fact that the solution has minimum cost among all multigraphs having the the same connectivity type.

Theorem 5.8.3 *There exists a polynomial-time approximation scheme for the* $\{0,1,2\}$*-vertex/edge-connectivity problem in Euclidean space* $\mathbb{R}^d$*. For* $\varepsilon > 0$ *and any sets of points* P_0, P_1, P_2 *in* $\mathbb{R}^d$ *of total size* n*, the algorithm in time* $n \log n\,(d/\varepsilon)^{\mathcal{O}(d)} + n\,(d/\varepsilon)^{(d/\varepsilon)^{\mathcal{O}(d^2)}}$ *finds a graph on* $P_1 \cup P_2$ *with possible Steiner points in* P_0 *that satisfies the connectivity requirement for any pair of points and whose cost is at most* $1+\varepsilon$ *times the minimum.*

For constant d *and* ε*, the running time of this algorithm is* $\mathcal{O}(n \log n)$. □

5.9 Extensions

1. All the arguments from Section 5.8 except the dynamic programming part can be easily generalized to include $\{0,1,\ldots,k\}$*-edge-connectivity for multigraphs.* Regarding the dynamic programming, one can combine the dynamic programming for the k-edge-connectivity from [25] with the method of dealing with non-uniform connectivity requirements. Thus, the variant of the geometric survivability problem, where the edge-connectivity requirements satisfy $r_v \in \{0,1,\ldots,k\}$ and $k = \mathcal{O}(1)$, admits also a PTAS. The running time is however, significantly greater than $\mathcal{O}(n \log n)$, though it is still polynomial for constant d and ε.

2. Using the same arguments as in [3], one can extend the PTASs to include other ℓ_p^d metrics as well.

3. The PTASs could be also extended to an infinite domain for Steiner candidate points (e.g., if $P_0 = \mathbb{R}^d$ one has the Euclidean complete Steiner problem) provided one can determine the set Y (or its good approximation) as fast as claimed in Lemma 5.4.10.

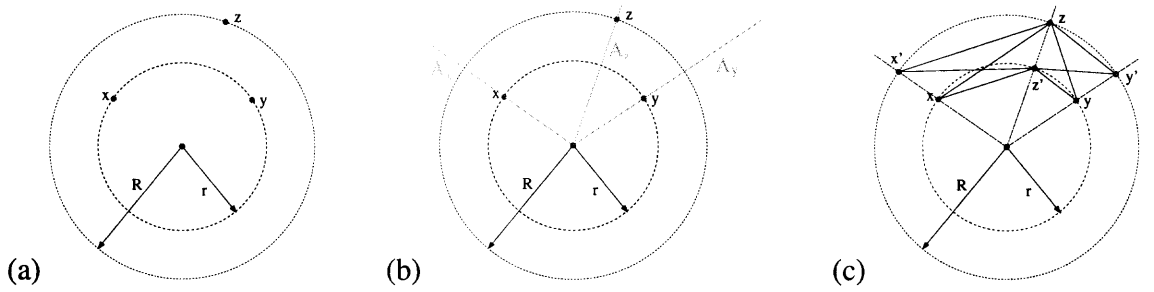


Figure 5.3 Illustration to the proof of Lemma 5.10.1. (a) An example of input configuration. (b) Rays A_x, A_y, and A_z. (c) Construction of x', y', z' (notice that the same construction is valid even when A_z does not lie between A_x and A_y).

5.10 Auxiliary Claims

Lemma 5.10.1 *Let* r *and* ρ *be any positive real numbers and let* $R = (1+\rho)\cdot r$. *Let* B_r *and* B_R *be two co-centric* d*-dimensional balls of radius* r *and* R, *respectively. If* x *and* y *are points contained in* B_r *and* z *is not contained within of* B_R *then*

$$(1+\rho)\cdot|xy| \leq |xz| + |yz| \ .$$

Proof : First notice that it is sufficient to prove the lemma for $d = 2$. Furthermore, it is easy to see that it is enough to consider the case when x and y are on the boundary of B_r (otherwise one could move the triangle $\triangle(x,y,z)$ to have x and y on the boundary of B_r while still having z not contained within B_R) and z is on the boundary of B_R (otherwise, one could move z to the boundary of B_R without changing $|xy|$ and with decreasing $|xz|+|yz|$).

Now, see Figure 5.3 for an illustration for the remaining of the proof. Draw three rays A_x, A_y, and A_z from the center of B_r through points x, y, and z, respectively (see Figure 5.3 (b)). Define x' and y' to be the points on the intersection of the boundary of B_R and A_x, A_y, respectively. Let z' be the point on the ray A_z that lies on the boundary of B_r. (See Figure 5.3 (c) for an illustration.) By the Thales theorem

$$(1+\rho)\cdot|xy| = |x'y'| \ . \tag{5.2}$$

Furthermore, from the triangle inequality one can get

$$|x'y'| \leq |x'z'| + |y'z'| \ . \tag{5.3}$$

Now, consider the quadrilateral $\langle x, x', z, z' \rangle$ (see Figure 5.3 (c)). Since the segments xz' and $x'z$ are parallel, $\langle x, x', z, z' \rangle$ is a trapezoid. Furthermore, $\angle(x, x', z) = \angle(x', z, z')$ and $\angle(z', x, x') = \angle(z, z', x)$. Therefore

$$|xz| = |x'z'| \ . \tag{5.4}$$

The same arguments imply

$$|yz| = |y'z'| \ . \tag{5.5}$$

Thus, one can now summarize all our inequalities to obtain

$$(1+\rho) \cdot |xy| \overset{\text{ineq.(5.2)}}{=} |x'y'| \overset{\text{ineq.(5.3)}}{\leq} |x'z'| + |y'z'| \overset{\text{ineq.(5.4),(5.5)}}{=} |xz| + |yz| \ .$$

□

CHAPTER 6

APPROXIMATION SCHEMES FOR MINIMUM 2-EDGE-CONNECTED AND BICONNECTED SUBGRAPHS IN PLANAR GRAPHS

6.1 Introduction

This chapter considers approximation algorithms for the most basic case of the survivable network design problem in which the resulting subgraphs should be resistant to the removal of a *single* edge or vertex. The two classical problems considered here are to find a minimum 2-edge-connected (2-EC) spanning subgraph (a 2-ECSS), or a 2-vertex-connected (2-VC or biconnected) spanning subgraph (a 2-VCSS) of planar graphs.

As mentioned before, these problems are max-SNP-hard [25], even for unweighted graphs or when duplicate edges are allowed; therefore one can not expect a PTAS. But this does not preclude a PTAS for restricted classes of graphs: in particular, a PTAS exists for both problems in *geometric graphs* of constant dimension [25]. The goal in this chapter is to design PTASs for the two problems in planar graphs.

Many polynomial time approximation algorithms are already known for these problems, see the survey [70] and more recent advances [19, 39, 40, 66, 74]. In unweighted graphs, the best currently known approximation ratios are 5/4 for 2-ECSS problem [66], and a 4/3 for the 2-VCSS problem [116]. For both problems in weighted planar graphs, the best known subexponential-time approximation guarantee is still 2 [73, 94]. All these approximaitons are achieved by polynomial-time algorithms working for general weighted graphs.

This chapter describes a PTAS for the minimum 2-edge-connectivity problem and a PTAS for the minimum biconnectivity problem, both for unweighted planar graphs. More generally, when the planar graph has edge weights the algorithms approximately solve the minimum-cost problems in time $n^{O(\gamma/\varepsilon)}$, where γ is the ratio of the total edge cost to the

optimum solution cost. This is a PTAS when γ is bounded; note that γ is bounded (by 3) when the edge weights are uniform.

The new general approach resembles the approximation schemes for metric-TSP in planar graphs [4, 47, 48]. It uses a separator theorem, hierarchical decomposition, and dynamic programming. The new separator finds low-cost cycles in a planar graph so that after contracting those cycles (and committing their edges to the approximate solution), the remaining graph has a logarithmic size vertex separator. Using this, the input graph G is recursively divided into pieces, forming a decomposition tree $\mathcal{T}$ of logarithmic depth. Each piece has a logarithmic number of "portal" vertices connecting it to the rest of G. For each piece, one can enumerate all the different ways that some subgraph of G (outside this piece) may influence the connectivity constraints within this piece. These are called the *external types* of the piece, and one can show that the number of such types is a simple exponential in the number of portals. For each piece in $\mathcal{T}$ and for each external type, one must find a near minimum cost subgraph H of the piece, so that H together with the external type can meet the global connectivity constraints. These problems are solved by dynamic programming, working up $\mathcal{T}$ from the leaves to the root G.

In the following, 2-EC denotes "2-edge-connected", 2-VC denotes "2-vertex-connected" (or biconnected), 2-ECSS denotes "2-edge-connected spanning subgraph", 2-VCSS denotes "2-vertex-connected spanning subgraph", and $c(H)$ denotes the total edge cost of a subgraph H. The main results are summarized in the following two theorems.

Theorem 6.1.1 *Let $\varepsilon > 0$, let G be a 2-EC planar graph with edge costs, and let* OPT *be the minimum cost of a 2-ECSS in G. There is an algorithm taking inputs G and ε, running in time $n^{O(c(G)/(OPT\cdot\varepsilon))}$, and producing a 2-ECSS H in G such that $c(H) \leq (1+\varepsilon)\cdot OPT$.*

Theorem 6.1.2 *Let $\varepsilon > 0$, let G be a 2-VC planar graph with edge costs, and let* OPT *be the minimum cost of a 2-VCSS in G. There is an algorithm taking inputs G and ε, running in time $n^{O(c(G)/(OPT\cdot\varepsilon))}$, and producing a 2-VCSS H in G such that $c(H) \leq (1+\varepsilon)\cdot OPT$.*

As remarked above, each claimed algorithm is a PTAS when the ratio $\gamma = c(G)/OPT$ is bounded. In particular, Theorems 6.1.1 and 6.1.2 imply a PTAS for the *unweighted* minimum 2-edge-connectivity and the minimum biconnectivity problems.

Corollary 6.1.3 *Let $\varepsilon > 0$, let G be a 2-EC planar graph, and let OPT be the minimum number of edges of a 2-ECSS in G. There is an algorithm taking inputs G and ε, running in time $n^{O(1/\varepsilon)}$, and producing a 2-ECSS H in G that has at most $(1+\varepsilon) \cdot OPT$ edges.*

Corollary 6.1.4 *Let $\varepsilon > 0$, let G be a 2-VC planar graph, and let OPT be the minimum number of edges of a 2-VCSS in G. There is an algorithm taking inputs G and ε, running in time $n^{O(1/\varepsilon)}$, and producing a 2-VCSS H in G that has at most $(1+\varepsilon) \cdot OPT$ edges.*

In the following, Theorem 6.1.1 will be discussed first, then the new ideas required for Theorem 6.1.2 will be presented. Throughout the chapter, it is assumed that graphs are undirected, without self-loops but possibly with parallel edges. Each edge e has a nonnegative cost c_e; a subgraph or minor H inherits edge costs from its parent graph, and $c(H)$ denotes the total edge cost of H.

6.2 Cuts and k-EC Types

Suppose $G = (V, E)$ is a graph and S_1, S_2 are disjoint subsets of its vertex set $V = V(G)$. S_1 and S_2 are *separated* if there is no path in G from a vertex of S_1 to a vertex of S_2. An edge set $F \subseteq E$ *separates* S_1 and S_2 if they are separated in $G - F$; F is said to be an *edge cut* for S_1 and S_2. Similarly, a *vertex cut* for S_1 and S_2 is some $U \subseteq V - (S_1 \cup S_2)$ such that S_1 and S_2 are separated in $G - U$. Let $Cut^e_G(S_1, S_2)$ denote a minimum size edge cut, and $C^e_G(S_1, S_2) = |Cut^e_G(S_1, S_2)|$ is the *edge capacity* between S_1 and S_2. $Cut^v_G(S_1, S_2)$ and $C^v_G(S_1, S_2)$ are defined similarly. Such min cuts can be computed efficiently using max-flow. For $P \subseteq V$, a (vertex or edge) cut is said to *crosses* P if it separates some S_1 and S_2 such that $S_1 \cup S_2 = P$.

A *cycle* has no repeated vertex, but it may consist of two vertices joined by two parallel edges. For $e \in E$, G/e denotes the graph obtained by contracting e (that is, identifying its endpoints). If C is a cycle of G, G/C is the graph obtained by contracting the edges of C. After contraction self-loops are discarded, but parallel edges are retained. A *minor* of graph G is a graph obtained from G through a series of such edge contractions and edge/vertex deletions.

Definition 6.2.1 *A* bipartition *of a set* P *is a pair of nonempty subsets* $\{S_1, S_2\}$ *such that* $S_1 \cup S_2 = P$ *and* $S_1 \cap S_2 = \emptyset$.

Suppose G *is a graph,* $P \subseteq V(G)$*, and* k *is a positive integer. The* $(k\text{—}EC, P)$-type *of* G *is a table* t *indexed by bipartitions* $\{S_1, S_2\}$ *of* P*, and holding the values* $t(S_1, S_2) = \min(k, C^e_G(S_1, S_2))$.

Suppose t_1 *and* t_2 *are* $(k\text{—}EC, P)$*-types of two graphs sharing the vertex subset* P*; they are* compatible *iff* $t_1(S_1, S_2) + t_2(S_1, S_2) \geq k$ *for all* $\{S_1, S_2\}$.

Intuitively, the $(k\text{—}EC, P)$-type describes how P is crossed by edge cuts using less than k edges. Usually only the type when G is $(k\text{—}EC, P)$*-safe* is interesting, meaning that all edge cuts of G not crossing P use at least k edges. The relevance of such types to the k-ECSS problem follows from this simple claim.

Claim 6.2.2 *Suppose* H_1 *and* H_2 *are graphs with disjoint edge sets, and* $V(H_1) \cap V(H_2) = P$*. Then* $H_1 \cup H_2$ *is* k*-EC iff:*

1. H_1 *is* $(k\text{—}EC, P)$*-safe,*
2. H_2 *is* $(k\text{—}EC, P)$*-safe, and*
3. *the* $(k\text{—}EC, P)$*-types of* H_1 *and* H_2 *are compatible.*

The third condition above can be abbreviated by saying that H_1 and H_2 (or H_1 and the type of H_2, or vice versa) are *compatible*.

Suppose G_1 and G_2 are edge disjoint graphs with $V(G_1) \cap V(G_2) = P$. To solve the k-ECSS problem in $G_1 \cup G_2$, it suffices to do the following:

1. For each possible type t_1 of a subgraph of G_1, find a min-cost (k—EC, P)-safe spanning subgraph H_2 of G_2 compatible with t_1.
2. For each possible type t_2 of a subgraph of G_2, find a min-cost (k—EC, P)-safe spanning subgraph H_1 of G_1 compatible with t_2.
3. Consider all pairs of H_1 from Step 1 and H_2 from Step 2. Return the min-cost compatible pair.

A similar approach is used in the PTAS of Theorem 6.1.1. It is necessary to have in particular a polynomial bound on the number of distinct subgraph types considered is needed. One may succinctly represent the (k—EC, P)-type of G by a smaller graph $t(G)$ which contains P and has the same type. In particular a minor of G contains P as long as no vertex of P is deleted, nor two vertices of P are contracted together.

In the special case of $k = 2$, one can construct $t(G)$ from G by applying the following rules, until none apply:

1. If a cycle C has a chord (an edge $e \notin E(C)$ connecting two vertices of C), delete the chord.
2. If a cycle C has at most one vertex in P, contract C to a point.
3. If a vertex $v \notin P$ has degree 2, contract it with a neighbor.

The correctness of the above follows from the observation that all 0-edge cuts and 1-edge cuts of P are invariant under the above rules. Note that if G is planar, then so is $t(G)$. Also if G is (2—EC, P)-safe, then so is $t(G)$. For the application considered in this chapter, one needs to consider the situation where G is embedded in a disk with the vertices of P on the boundary. In the next two lemmas the size of $t(G)$ and the total number of possible (2—EC, P)-types induced by subgraphs of G are bounded.

Lemma 6.2.3 *Suppose* G *is a* (2—EC, P)*-safe planar graph embedded in the disk, with the vertices of* P *on the disk boundary. Then* $t(G)$ *is a planar graph embedded in the same way, with* $O(|P|)$ *vertices.*

Proof : By considering the three rules used to form $t(G)$, it is also a planar $(2-EC, P)$-safe graph embedded in the disk with P on the boundary. Every internal face f of $t(G)$ has at least two portals. If f has exactly two portals, one draws an arc e_f inside f between those two portals. If f has $d \geq 3$ portals, one draws a cycle of d arcs within f connecting the portals. These arcs form an outerplanar graph A on the portals.

One can claim that A has no parallel edges. Suppose instead that two portals $p, q \in P$ are connected by two parallel arcs a_1 and a_2, from faces f_1 and f_2. Since all faces must involve at least two portals, one can choose a_1 and a_2 consecutive at p, so that f_1 and f_2 share at least one edge. Now consider the part of $t(G)$ drawn between a_1 and a_2: it has no cycles (by rule 2) and is connected, so it is a tree. Because $t(G)$ is $(2-EC, P)$-safe, it is a path from p to q. By rule 3 it must be an edge directly between p and q. But then it is a chord between f_1 and f_2, so it should have been deleted by rule 1.

Therefore A is a simple outerplanar graph on vertex set P, so it has less than $2|P|$ arcs. Further if one add arcs from the outer faces of $t(G)$ (those faces bounded by a segment of the boundary), there are at most three parallel arcs per pair of portals, therefore at most $6|P|$ arcs. For each $p \in P$, its degree in $t(G)$ is at most the number of adjacent arcs; therefore the sum of the degree of p in $t(G)$, over all $p \in P$, is at most $\ell = 12|P|$.

Now if one erases each portal and an infinitesimal neighborhood around it, the graph $t(G)$ is transformed into a forest (by rule 2) with ℓ leaves, and all internal vertices of degree at least 3 (by rule 3). Then $t(G)$ has less than ℓ vertices not in P, or in other words $t(G)$ has less than $13|P|$ vertices overall. □

Lemma 6.2.4 *With G and P embedded as in the previous lemma, the number of distinct $(2-EC, P)$-types defined by subgraphs of G is $2^{O(|P|)}$.*

Proof : Let H be a subgraph of G containing P. By trimming H, one can make it $(2-EC, P)$-safe without changing its $(2-EC, P)$-type. In the previous proof it is shown that $t(H)$ can be described by a planar forest T with at most $12|P|$ leaves, internal vertices

of degree at least three, and each leaf labeled by some $p \in P$, where the labels for a given p are on consecutive leaves. By standard tree counting techniques, there are $2^{O(|P|)}$ such graphs, and therefore at most that many distinct $(2-EC, P)$-types. □

6.3 Planar Separators

Suppose one needs to approximately solve the 2-ECSS problem in a planar graph G embedded on a sphere. If a low-cost simple cycle C in G can be found, then one may divide the problem into subproblems by contracting C. This follows from two observations:

Fact 6.3.1 *For any subgraph* H *of* G *containing* C, H *is a 2-ECSS in* G *iff* H/C *is a 2-ECSS in* G/C. *(This does not use planarity.)*

Fact 6.3.2 *When* C *is contracted, the sphere pinches into two spheres kissing at the new contracted vertex (a cut-point). Therefore the 2-ECSS problem in* G/C *is equivalent to two disjoint 2-ECSS problems, one on each sphere.*

Therefore to approximately solve the 2-ECSS problem in G, one may contract C and approximately solve the two independent 2-ECSS subproblems. Then lift the edges of those two solutions back to G and add the edges of C. The obtained graph is a 2-ECSS in G. The additive error of this solution (the difference between its cost and the optimal cost) is at most the sum of the errors of the two subproblems plus $c(C)$.

One can not always luckily find a light cycle which does a good enough job of separating G, therefore a more general kind of separator combining cycles with a *Jordan cut* is considered: a Jordan cut of G is a closed Jordan curve in the embedding of G that does not cross (intersect the interior of) any edge. Given a Jordan cut J, every edge is either in the interior or the exterior of J, but those vertices and faces intersected by J are not counted in either the interior or the exterior of J.

The following theorem is from [10] which is a modification of Miller's planar separator theorem, as already used for the planar TSP [4, 47, 89, 10].

Theorem 6.3.3 *Let* G *be a connected planar graph on* $n \geq 3$ *vertices embedded in the plane. Suppose* G *has non-negative weights on its vertices, edges and faces, and non-negative costs on its edges. Let* W *be the total weight of the graph and let* M *be its total cost and assume that no edge has weight more than* (3/4)W.

Then, for any positive integer k, *one can find a subgraph* F *of* G *and a closed Jordan curve* J *in* $\mathcal{O}(n)$ *time such that:*

1. F *is the union of at most two vertex-disjoint simple cycles (maybe none). The total cost of the edges on each cycle is at most* M/k. *If* F *contains two cycles* A *and* B, *then* $\mathtt{int}(B) \subset \mathtt{int}(A)$.

2. *The interior of* B *and the exterior of* A *(if they exist) both have weight at most* W/2.

3. *Denote by* G′ *the embedded graph that results after deleting the interior of* B *and the exterior of* A *(if they exist) and contracting each cycle in* F *to a vertex of weight 0. Then* J *is a Jordan curve through the new contracted vertices, which intersects edges of* G′ *only at their endpoints. The interior and exterior of* J *both have weight at most* (3/4)W.

4. *The set of vertices of* G′ *on* J *has size* $\mathcal{O}(k)$.

The three possible types of the separator (according to the number of cycles in F) are illustrated in Figure 6.1. Note that it is necessary to assume that each edge weighs at most (3/4)W. If a graph has an edge e that has weight larger than (3/4)W and has cost exceeding M/k, then no such separator would exist. Because of its high cost e cannot be on any of the cycles in F and due to its high weight it cannot be in the contracted interior of B or in the contracted exterior of A. Hence e is also an edge in G′ and any Jordan curve in the contracted graph G′ has e either in its interior or exterior and so would not satisfy the third property.

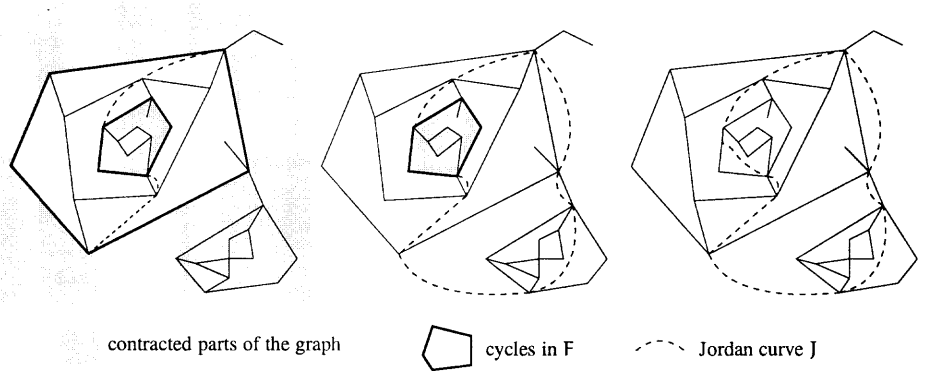


Figure 6.1 The three different types of the separator. In the first two cases the Jordan curve is closed after the contraction.

When the above theorem is applied, Q is said to be the set of new[1] *portal* vertices introduced by the separator. The original graph G has been divided into at most four parts of weight at most $(3/4)W$: the interior of the cycle B denoted by G_B (with B contracted), the exterior of the cycle A denoted by G_A (with A contracted), the interior of J, and the exterior of J. Let G_1 denote Q together with the subgraph of G' interior to J, and let G_2 denote Q together with the subgraph of G' exterior to J. In this way $G_1 \cup G_2 = G'$, $E(G_1) \cap E(G_2) = \emptyset$, and $V(G_1) \cap V(G_2) = Q$. In the inherited embedding of G_1 (or G_2) all vertices of Q appear on a single new face which is called a *portal face*; the old faces that intersected J are gone. When solving 2-ECSS in G, one will see that the subproblems in G_A and G_B become independent 2-ECSS problems, but the subproblems in G_1 and G_2 are dependent because they share Q.

6.4 The 2-ECSS Algorithm

Let G_0 be the input which is an embedded planar 2-EC graph with n vertices, non-negative edge costs, and a parameter $\varepsilon > 0$. The algorithm assigns weight 1 to each vertex and weight 0 to each face. By existing approximation algorithms one estimates $OPT(G_0)$, the

[1]"P" is reserved to denote all portals in a graph, new or old.

minimum cost of a 2-ECSS, within a constant factor. Fix an integer $k = \Theta((\gamma/\varepsilon)\log n)$, where $\gamma = c(G_0)/OPT(G_0)$.

A rooted decomposition tree $\mathcal{T}$ is built from G_0 as follows. Each node of $\mathcal{T}$ stores an embedded planar graph G, which has edge costs, vertex/face weights, and some distinguished subset P of "portal" vertices. The root of $\mathcal{T}$ stores G_0 itself, with no portals. Each node of $\mathcal{T}$ has at most four children, defined inductively as follows.

Let G be the graph stored at a node of $\mathcal{T}$, and let W be its total vertex/face weight. If W is $O(k^2)$, then this node is a leaf of $\mathcal{T}$. Otherwise, apply Theorem 6.3.3 to partition G into at most four pieces (interior of B and exterior of A, and G_1, G_2), each of total weight at most $(3/4)W$. Uncontracted portal vertices from G remain as portals in each piece where they appear; the graphs G_1 and G_2 each get at most k new portals, the set Q. In G_1 and G_2, assign a weight of $W/(16k)$ to each new portal, and weight $W/16$ to the new portal face. By the Separator Theorem, the number of new portals obtained in this phase is at most k, thus the weight of each of G_1 and G_2 is at most $\frac{3}{4}W + k\cdot(\frac{1}{16k}W) + \frac{1}{16}W = \frac{7}{8}W$. In the graph G_A and G_B, assign the new (non-portal) vertex weight 1 and all the the remaining vertices have the same weight as in G. Since each child has weight at most constant fraction of the weight of the parent, the tree $\mathcal{T}$ has depth $O(\log n)$ and size $O(n \log n)$. Furthermore, if G_0 is 2-edge connected, by the separator properties, each G in $\mathcal{T}$ is (2−EC, P)-safe.

By the construction, P is the set of portals in G which have been introduced by some Jordan cut but not yet cut off by a cycle contraction or another Jordan cut. Now, consider the number of portals and faces of the smaller graphs. Since G_A and G_B do not have new portals and portal faces, one needs consider G_1 and G_2 only. The construction ensures that the portals that G_1 and G_2 inherited from G are always heavier than the new portals in the vertices of J, and the inherited portal faces are always heavier than the new portal face. This implies that in G_1 and G_2, every portal has weight at least $\frac{1}{16k}W$, and every portal face has weight at least $\frac{1}{16}W$. Since G_1 and G_2 each has weight at most $\frac{7}{8}W$, one can conclude that G_1 and G_2 each contains at most 14k portals and at most 13 portal faces. Each portal

face contains a hole made by a Jordan cut at some ancestor of G in $\mathcal{T}$. Note that a Jordan cut might cut (simply) across an existing portal face, in which case some old portals may appear on the new portal face, but this still counts as a single portal face. Or in terms of an embedding on a sphere with holes, all old holes crossed by the Jordan cut disappear, with segments of their boundaries incorporated into the one new hole boundary.

G is connected via P to the rest of G_0 (really a pinched and contracted version of G_0) which can be embedded as disjoint pieces, one in each portal face of G. Therefore the $(2\text{–EC}, P)$-type imposed on P by the rest of G_0 decomposes into independent types, one in each portal face of G. By applying Lemma 6.2.4 to each portal face, one may bound and enumerate the $2^{O(|P|)} = n^{O(\gamma/\varepsilon)}$ different $(2\text{–EC}, P)$-types that may be imposed on P by the rest of G_0. Call this list the list of *external types* for G. It is more efficient, although not essential, if each external type t of G is represented as a planar graph of size $O(|P|)$ (see Lemma 6.2.3) embedded in the portal faces of G. In particular at the root of $\mathcal{T}$ the input graph G_0 has no portals, and therefore it has the "empty" external type t_0.

Having computed $\mathcal{T}$ and these external type lists, one may now define a set of subproblems that need to be approximately solved:

Definition 6.4.1 *For G in $\mathcal{T}$ (with portal set P) and an external type t for G, the* subproblem (G, t) *is this: find a min-cost $(2\text{–EC}, P)$-safe spanning subgraph H of G which is compatible with t, or else declare that (G, t) is infeasible (no such H).*

Checking feasibility is simple: just check whether t is compatible with G itself. The total number of subproblems (over all choices of G and t) is $n^{O(\gamma/\varepsilon)}$, and the subproblem (G_0, t_0) is the original 2-ECSS problem. The subproblems will be approximately solved starting at the leaves of $\mathcal{T}$ and finishing at the root. Using dynamic programming, the solutions are stored to avoid recomputation.

In the base case, G is a leaf of $\mathcal{T}$ and has size $N = O(k^2)$. Then enumerative method can be applied based on Lipton-Tarjan separators [85] to exactly solve such sub-

problems in $2^{O(\sqrt{N})} = n^{O(\gamma/\varepsilon)}$ time (this may be regarded as a continuation of our method, using Jordan cuts without cycle contractions).

Otherwise G is not a leaf, and has up to four children in $\mathcal{T}$ as found by Theorem 6.3.3. The external type t is decomposed into independent external types, one for each portal face of G.

Let G_C denote either G_A or G_B and let t_C be the subtype of t induced by the portal faces of G_C. Lookup the solution H_C to the subproblem (G_C, t_C) and lift the edges of H_C and the edges of C to be part of the approximate solution H for the (G, t) subproblem.

Now consider the two remaining children G_1 and G_2. As in Theorem 6.3.3, let G' be what is left of G after the (up to two) cycles are contracted and their interior or exterior is pinched; so $G_1 \cup G_2 = G'$. Let t' denote the external type induced by t in the portal faces of G'. Not knowing the optimal choice of external types t_1 and t_2 for G_1 and G_2, one needs to try them all. That is, for every pair (t_1, t_2) where subproblems (G_1, t_1) and (G_2, t_2) were found feasible, one lookups their solutions H_1 and H_2 and check whether $H' = H_1 \cup H_2$ is compatible with t'. The cheapest compatible H' found are taken, and its edges are lifted back to G. These edges of H', together with the C and H_C edges mentioned earlier, comprise the approximate solution H for the (G, t)-subproblem.

Although G' is not actually associated with a node of $\mathcal{T}$, note that one can still speak sensibly of the (G', t') subproblem as defined above. In fact it would be a simple matter to reformulate $\mathcal{T}$ as a binary tree including G': at each internal node of $\mathcal{T}$ one would either pinch one cycle, or apply a Jordan cut.

Analysis The above algorithm solves $n^{O(\gamma/\varepsilon)}$ subproblems, each in $n^{O(\gamma/\varepsilon)}$ time, so the total running time is $n^{O(\gamma/\varepsilon)}$.

Consider a feasible subproblem (G, t). By planarity, t decomposes into independent types in each portal face, and these faces cannot cross a cycle; therefore each cycle-pinched subproblem (G_C, t_C) and the remaining subproblem (G', t') are all feasible. Tak-

ing the external type on G_1 induced by $G_2 \cup t'$ as t_1, we see that (G_1, t_1) is feasible. Supposing (by induction) that the algorithm found some solution H_1 for (G_1, t_1), then $H_1 \cup t'$ induces an external type t_2 on G_2 such that (G_2, t_2) is also feasible. Therefore by induction up $\mathcal{T}$, the algorithm finds some solution for each feasible (G, t)-subproblem.

Now suppose H is the solution that the algorithm finds for a feasible subproblem (G, t). Define the *error* on (G, t) as the difference between the cost $c(H)$ and the minimum possible cost. For each pinched cycle C (up to two), by Facts 6.3.1 and 6.3.2 subproblem (G, t) will inherit the error of (G_C, t_C) plus an additional additive error of at most $c(C)$.

After pinching cycles, the remaining error of (G, t) is that from (G', t'). Recall $G' = G_1 \cup G_2$; let H^* be the unknown optimal solution for (G', t'). Let t_1^* denote the external type of G_1 induced by $(H^* \cap G_2) \cup t'$, and similarly let t_2^* denote the external type of G_2 induced by $(H^* \cap G_1) \cup t'$. Then (G_i, t_i^*) has the optimal solution $H^* \cap G_i$ (for $i = 1, 2$), and (t_1^*, t_2^*) is a compatible type pair considered by the algorithm; if these two subproblems are solved optimally, the solution cost would be $c(H^*)$. Therefore the error on (G', t') is at most the sum of the errors on (G_1, t_1^*) and (G_2, t_2^*), even though one might not actually find the best solution H' using this pair. Therefore error terms simply add at a Jordan cut.

Therefore the total error of the root problem (G_0, t_0) is at most the sum of $c(C)$ over all cycles contracted in $\mathcal{T}$. It is easy to see that for any level of $\mathcal{T}$, the total edge cost of that level is at most $c(G_0)$. Therefore the total edge cost of all cycles contracted on that level is $O(c(G_0)/k)$. Summing over all $O(\log n)$ levels of $\mathcal{T}$, the total error from all levels of $\mathcal{T}$ is $O((c(G_0)/k) \log n)$. By an appropriate choice of the leading constant defining k, this is at most $\varepsilon \cdot \mathrm{OPT}(G_0)$. Therefore the final solution has cost at most $(1 + \varepsilon)\mathrm{OPT}(G_0)$, proving Theorem 6.1.1.

6.5 The 2-VCSS Algorithm

The main idea of the 2-VCSS algorithm is similar to the 2-ECSS algorithm. Given the input plane graph G_0, the separator theorem is applied to decompose G_0 hierarchically into small pieces. For each piece, types are used to enumerate the different ways that the "rest" of the graph may influence the connectivity constraints within this piece. Then, dynamic programming is used to approximately find the minimum subgraph compatible for each type of each piece. Similarly one can prove that the number of external types of each piece is a simple exponential in the number of portals (Lemma 6.5.4), yielding the same running time analysis. Again the only source of error is in the weight of the separating cycle edges, yielding the same error analysis.

However, there are some difficulties preventing one from using the same technique to solve the 2-VCSS problem. The principle difficulty is cycle contraction. In the 2-ECSS algorithm, the decomposition, type definition and dynamic programming are all performed on the minors of G_0, and it is shown that this is sufficient. But it is no longer reasonable to contract the cycles in the 2-VCSS problem, because this changes the problem (in particular, it may introduce a false cutvertex). Therefore for each node in $\mathcal{T}$, a pair of graphs are kept, the compressed subgraph G as defined in the 2-ECSS algorithm and its corresponding *uncompressed* subgraph $\mathbb{G}$, that is the subgraph of G_0 induced by all vertices which appear (after cycle contractions) as some vertex in G. As before, the separator theorem is still applied to G, and the decomposition of $\mathbb{G}$ is obtained naturally. But the type and dynamic programming will be defined on $\mathbb{G}$.

The uncompressed graph $\mathbb{G}$ also contains portal vertices and portal faces, which are the preimages of the portals and portal faces in G. Specifically, let J be the Jordan cut when the separator theorem is applied to G. Let p be a vertex on J. If p is mapped to a single vertex of $\mathbb{G}$, then p is a new portal that will be contained in the subgraphs of $\mathbb{G}$. Otherwise, p is mapped to many vertices which are on a series of cycles in $\mathbb{G}$, then at most 2 of these vertices will be specified as new portals of $\mathbb{G}$. These two vertices are where the Jordan cut J

would intersect $\mathbb{G}$ if the cycles represented by p are uncontracted. Thus there are still $O(k)$ portals in $\mathbb{G}$. Each portal appears on some portal face in $\mathbb{G}$ corresponding to the portal faces of G, and the portal faces identify where "the rest of G_0" would appear in the embedding. The edges of each separating cycle will appear in $\mathbb{G}$ as *hard edges* which are committed to the final solution as in the 2-ECSS algorithm.

The notion of types must also be redeveloped in the biconnected context, so that one may again use types to characterize the possible counterparts of an uncompressed graph; this is the point of Lemma 6.5.2 below. For convenience, types are simply represented as graphs (rather than abstract cut tables represented by graphs, as in Section 6.2). Following is a definition analogous to "(k−EC, P)-safe" graphs.

Definition 6.5.1 *For a graph* $G = (V_G, E_G)$ *with a portal set* $P \subseteq V_G$*, a graph* $H = (V_H, E_H)$ *is called a* counterpart *(of* G*) if* $V_H \cap V_G \subseteq P$*. The graph* $G \cup H$ *is obtained by taking the union of the vertices and edges of the* G *and* H *and keeping the multiplicity of each vertex and edge at most* 1*. The graph* G *is called* (2−VC, P)-safe *if it has a counterpart* H *such that* $G \cup H$ *is biconnected.*

6.5.1 Types of (2-VC, P)-Safe Planar Graphs

The type of a (2−VC, P)-safe graph is a *graph* describing a simplified characterization of the biconnectivity information of the portals in the graph.

Definition of the type. The definition of type is operational and it is oriented towards Lemmas 6.5.2 and 6.5.4 below. Let $H = (V_H, E_H)$ be any (2−VC, P)-safe graph (does not have to be planar) with a distinguished portal set P and a distinguished set of hard edges $E_{H,T}$. The type $t(H)$ of H is defined by performing a series of operations on H. See Figure 6.2 for an illustration.

1. Let H_τ be the graph consisting all portals of P and all the edges of $E_{H,\tau}$. From H_τ, construct a simplified graph $\widehat{H_\tau}$ which is a forest formed by a collection of (possibly connected) stars[2]:

   (a) All vertices of H_τ are included in $\widehat{H_\tau}$.

   (b) Every portal of P is marked as a *super-portal.*

   (c) For each vertex $q \notin P$, if it is a cutvertex or an end vertex of an isolated edge in H_τ, mark it as a super-portal in $\widehat{H_\tau}$.

   (d) For each block of H_τ (hard block) with at least two vertices, create a new vertex in $\widehat{H_\tau}$ as a *super-portal* and connect by an edge the super-portal to every vertex in the block.

   (e) Assign distinct IDs to all super-portals.

2. Replace the subgraph H_τ of H by the graph $\widehat{H_\tau}$ and remove those vertices that have no ID and are adjacent only to a super-portal (notice that the removed vertices are all connected only to vertices of a hard block of H_τ). Let the obtained graph be $\widehat{H}$.

3. For each block of $\widehat{H}$ that is not a bridge, if it has exactly two cutvertices and does not contain a super-portal, contract it into a single vertex.

4. Repeat the following two operations until none apply.

   (a) Remove all chords in any cycle of $\widehat{H}$.

   (b) For each path π of $\widehat{H}$ with no super-portals as internal vertices and all internal vertices with degree exactly 2, *contract* π to an edge with the same end vertices as π.

---

[2]Notice a similarity of this construction to the standard construction of cutvertex-trees, see, e.g., [32].

5. Let H' be the graph obtained after Steps 1–4. The type $t(H)$ is a graph with some vertices *labeled* (having IDs, these are vertices corresponding to super-portals and will be called portal nodes). It is constructed from H' as follows:

   (a) Add each cutvertex x in H' to $t(H)$, and refer to x as a *cutvertex* in $t(H)$. If it is a super-portal p in H', then it is a portal-node in $t(H)$ with the ID of p.

   (b) For each block in H' that does not contain any super-portals, contract it to a vertex in $t(H)$. This vertex in $t(H)$ is referred as a *block-vertex*.

   (c) For each block in H' containing exactly one super-portal p, contract this block to a block-vertex. If the block is not a bridge and p is not a cutvertex in H', then this block-vertex is a portal node in $t(H)$ with the ID of p.

   (d) Connect a block-vertex by an edge in $t(H)$ to each cutvertex adjacent to it or to the endvertices of the bridge if the block-vertex is obtained from the bridge in H'.

   (e) If a block has two or more super-portals, then keep the block in $t(H)$ as it is in H'.

   (f) Finally, for each path π of $t(H)$ with no portal-nodes as internal vertices and all internal vertices with degree exactly 2, *contract* π to a single edge with the same end vertices as π.

Note the concept of *super-portal* can be applied to any graph H that contains portals and hard edges, and it is completely determined by the portal set and the hard edge set of H.

Properties of the types. Some basic properties of the types are listed here. First of all, the number of labels of $t(H)$ is identical to the number of super-portals of H. Secondly, notice that the type $t(H)$ is uniquely defined. Let H_1 and H_2 be two $(2-VC, P)$-safe graphs on the same vertex set and with the same set of portals and hard edges. Then H_1 and H_2

are said to have *the same type* if $t(H_1)$ and $t(H_2)$ are identical with respect to their IDs. Thirdly, all vertices that are not portal-nodes have degree at least 3. Next, notice that if H is biconnected and the portal set and hard edges are empty, then $t(H)$ contains a single vertex (block-vertex). Finally, it is easy to see that the connections among portals in H are preserved and presenteed by the connections of portal nodes in $t(H)$. These properties can be summarized in the following lemma.

Lemma 6.5.2 *Let* G *and* H *be two* $(2-VC, P)$*-safe graphs that have the same set of portals* P *and the same set of hard edges* E_T. *Let* $t(H)$ *be the type of* H. *Then for any spanning subgraph* G' *of* G *that contains all hard edges in* E_T, $t(G')$ *and* $t(H)$ *have the same set of portal-nodes with respect to IDs, and* $G' \cup H$ *is biconnected if and only if* $t(G') \cup t(H)$ *is 2-edge connected and the cutvertices of* $t(G') \cup t(H)$ *are block-vertices of* $t(H)$ *or* $t(G')$ *or the super-portals resulting from contractions of hard blocks as defined in the type.*

Let H be a counterpart of G with the same set of portals and hard edges, if $G \cup H$ is biconnected, then G is said to be compatible with $t(H)$. To check whether G is compatible with $t(H)$, by the above lemma, it is enough to check whether $t(G) \cup t'(H)$ is 2-edge connected and the cutvertices of $t(G) \cup t(H)$ are block-vertices of $t(G)$ or $t(H)$ or super-portals representing hard blocks of G and H.

Next, the number of types, specifically, the number of *external types* of $(2-VC, P)$-safe *plane* graphs is considered. Let G be a $(2-VC, P)$-safe plane graph with a hard edge set E_T and a portal face set F. Let the super-portal set of G determined by P and E_T be P_s. If H be a counterpart of G that can be embedded in F, [3] then the type $t(H)$ of H is said to be an external type of G with respect to F. The focus will be on the counterpart H of G that also is $(2-VC, P)$-safe and shares the same set of super-portals P_s (i.e., H and G have the same set of portals and hard edges). Then, the type $t(H)$ is called an external type of G with respect to P_s and F. The goal is to show that the number of external types of G with

[3]That is, H is drawn on the plane such that all vertices and edges of H are contained in the portal faces F (including the boundaries of all faces in F).

respect to P_s and F is small, that is, it is only exponential in $|P_s|$. The following lemma shows a type of a plane graph H can always maintain some topology of H.

Lemma 6.5.3 *Let $H = (V, E)$ be a planar graph with a set of portals $P \subseteq V$ that is embedded on the plane. Then, one can draw $t(H)$ on the plane such that $t(H)$ is planar and each portal node of $t(H)$ is drawn at the same location as one [4] of the corresponding portals in H.*

Proof : Every single operation performed on H while constructing $t(H)$ satisfies this property, and hence any sequence of such operations satisfies it too. □

Following is the the main lemma bounding the number of types.

Lemma 6.5.4 *Let $G = (V_G, E_G)$ be a $(2-VC, P)$-safe plane graph with a hard edge set E_τ, a super-portal set P_s and a portal face set F. G is embedded in a way such that all super-portals of P_s will be on the border of the portal faces if the graph G_τ consisting of all portals and hard edges is replaced with $\widehat{G_\tau}$ as in the definition of type. Then the number of external types of G with respect to P_s and F induced by $(2-VC, P)$-safe graphs is at most $2^{\mathcal{O}(|P_s|)}$.*

Proof : Let H be any $(2-VC, P)$-safe plane graph that has super-portal set P and is embedded in F. For each portal face F_i, let the subgraph of H embedded in F_i be H_i. Because H is plane, H_i is disjoint with H_j for $i \neq j$. Correspondingly, the subgraphs of $t(H)$ in different faces are also disjoint. Hence the type of the subgraph in each face can be considered independently.

Fix any face $F_i \in F$ and the subgraph H_i of H in F_i. Let its type be t_i. Now consider the structure of t_i. The vertices in t_i whose degree is at most one are going to be called as *leaves*. A leaf must be a portal node of P, because all vertices that are not portal nodes have degree at least 3 by the construction of $t(H)$. Secondly, there may be some parallel edges

[4] *A portal node in $t(H)$ may corresponds to many portals because the hard edges are compressed.*

in t_i which must be between two portal nodes (super-portals), because otherwise it will be contracted according to Step 4a-4b of the definition. If t_i contains a block, then the block must be a cycle. This is because by assumption and Lemma 6.5.3, all portal nodes are on the boundary of F_i, thus can not be inside a block. But after Step 1-3, no vertex will remain inside the cycle. Furthermore, by construction of t_i, each such cycle in t_i contains at least two portal nodes and all non-portal vertices on the cycle must have degree at least 3.

For the purpose of counting, one can transform t_i into a forest with only $|\mathcal{O}(P_i)|$ leaves. If one erases each portal node and an infinitesimal neighborhood around it, then a forest will be obtained with all internal nodes have degree at least 3 with no ID assigned. The leaves correspond to portals-nodes, they have the same ID as the corresponding portal-nodes. It can be claimed that the number of the leaves is $\mathcal{O}(P_i)$[5]. Thus, there are at most $2^{\mathcal{O}(|P_i|)}$ such forests.

Hence, the number of types t_i in face F_i is proved to be $2^{\mathcal{O}(|P_i|)}$. Because t_i and t_j are independent for $i \neq j$, the total number of possible external types with respect to P and F is $2^{\sum_{i=1}^{|F|} 2^{\mathcal{O}(|P_i|)}}$. This completes the proof. □

Let $(\mathbb{H}, H)$ be a pair stored in a node of $\mathcal{T}$ where $\mathbb{H}$ is the uncompressed $(2-VC, P)$-safe graph. Suppose $\mathbb{H}$ has a portal set P and a hard edge set E_τ. Let t be an external type of $\mathbb{H}$. $\mathbb{H}$ is said to be compatible with t if $t(\mathbb{H}) \cup t$ is 2-edge connected and the cutvertices of $t(\mathbb{H}) \cup t$ are block-vertices of $t(\mathbb{H})$ or t or super-portals representing hard blocks.

By Lemma 6.5.4, the number of possible external types is determined by the number of super-portals of $\mathbb{H}$. Now it will be shown that the number of super-portals in each subgraph obtained from separator theorem is $\mathcal{O}(k)$. By definition, the number of super-portals of $\mathbb{H}$ is at most the number of portals in it plus the number of separating cycles (hard cycles) and the cutvertices between these cycles. Because the depth of the decomposition

[5]The operation can be seen as two steps. First break the cycles at the portals on the cycle by erasing only the neighborhood on the cycle edge. Thus one gets at most $2|P_i|$ new leaves, and the obtained graph is a forest with internal vertices degree at least three except those portal nodes. Therefore, the number of edges of the forest is $\mathcal{O}(|P_i|)$. To make all portal nodes be leaves, one can do the same operation to the internal portal nodes and gets the final forest.

tree is at most $\mathcal{O}(\log n)$ and at each node at most 2 "cycles" are introduced [6], the number of hard "cycles" is at most $\mathcal{O}(\log n)$. Hence the number of super-portals of $\mathbb{H}$ is $\mathcal{O}(k) + \mathcal{O}(\log n) = \mathcal{O}(k)$. By Lemma 6.5.4, $\mathbb{H}$ has at most $2^{O(k)} = n^{\mathcal{O}(\gamma/\varepsilon)}$ external types.

6.5.2 Recursive Decomposition

In this section, a procedure is presented to build a decomposition tree T in a way suitable to solve the 2-VCSS problem in planar graphs. The procedure is similar as for the 2-ECSS problem. However at each node of T, a pair of graphs are kept, the compressed subgraph G as defined in the 2-ECSS algorithm and its corresponding *uncompressed* subgraph $\mathbb{G}$. In the root the input graph G_0 and $\mathbb{G}_0 = G_0$ are stored. Let ε be a positive real, fix certain $k = \Theta((\gamma/\varepsilon)\log n)$ for the rest of this section.

Let $\mathbb{G}$ and G be the pair of graphs stored at some node of T. $\mathbb{G}$ has a set of portals, a set of portal faces F and a set of hard edges. The Separator Theorem is applied to the compressed graph G to obtain up to two cycles $\mathcal{F}$ and a Jordan cut J having at most k vertices. This decomposes G (explicitly) and $\mathbb{G}$ (implicitly) into at most 4 smaller graphs:

- For each cycle C in $\mathcal{F}$, take the subgraph of G contained inside (outside if the cycle is A) C together with the cycle C. One can contract C to a single vertex (all edges incident to any vertex of C are now incident to the new vertex with all self-loops removed) and call the resulting graph G_C. Assign weight 1 to the new vertex.

  There are *no new* portals in G_C, but G_C may inherit some old portals and portals faces from G. The weight of G_C is at most $\frac{3}{4}W$, where W is the weight of G.

- Let G' be defined as in the Separator Theorem. One needs to define two other compressed graphs G_1 and G_2 that are the interior and the exterior of the Jordan cut J in G', respectively. All vertices in J are added to the set of portals of G_1 and G_2. Additionally, the algorithm assigns the weight of $\frac{1}{16k}W$ to each new portal, and weight $\frac{1}{16}W$ to the new portal face. Thus the weight of each of G_1 and G_2 is at most $\frac{7}{8}W$.

[6] They may not be simple cycles in $\mathbb{H}$, but a set of cycles glued together

While G is decomposed explicitly, the decomposition of $\mathbb{G}$ is obtained implicitly. Corresponding to G_C, $\mathbb{G}_C$ which is G_C but no cycles are contracted. Besides, the edges on C will be new hard edges in G_C.

Before defining $\mathbb{G}_1$ and $\mathbb{G}_2$ which correspond to G_1 and G_2, respectively, one needs to first define the new portals. Intuitively, these are the vertices where the Jordan cut J would intersect the uncompression graph $\mathbb{G}$. Let p be a portal on the Jordan cut J. If p is mapped to a single vertex of G, then p itself is a new portal of $\mathbb{G}$. Otherwise, p is mapped to many vertices which are on a series of cycles in $\mathbb{G}$. Consider those edges E that are incident to some vertex on some of these cycles and are in the interior of J in G. Assume it is not empty; otherwise consider those edges in the exterior of J in G. Let e_1 be the leftmost edge in E for the fixed embedding. Suppose e_1 is incident to u which is on one of the cycles. Then u will be designated as a new portal in $\mathbb{G}$. If there are edges of E not incident to u, pick the rightmost edge e_2 in the embedding. Suppose e_2 is incident to v which is on one of the cycles. Then v is also designated as the new portal. In this way, for each new portal mapped to a cycle in G, at most 2 new portals are defined.

Now, $\mathbb{G}_1$ and $\mathbb{G}_2$ would be the graphs inside and outside J respectively, with no cycle contracted. Further, they will not only inherit some old portals and hard edges from $\mathbb{G}$, but also have all the new hard edges and portals defined as above. Additionally, besides the inherited portal faces, both G_1 and G_2 have a new portal face.

By the Separator Theorem and the definition of $(2-VC, P)$-safe, it is easy to see that if $\mathbb{G}$ is $(2-VC, P)$-safe, then $\mathbb{G}_C$ and $\mathbb{G}_1$ and $\mathbb{G}_2$ are also $(2-VC, P)$-safe with respect to their portals.

Now consider the number of portals and faces. As for the 2-ECSS, by the way of setting the weight of new portals and new portal faces, one can show that G_1 and G_2 each contains at most 14k portals and at most 13 portal faces. The number of portal faces in $\mathbb{G}_1$ and $\mathbb{G}_2$ are the same as that of G_1 and G_2. But the number of portals in $\mathbb{G}_1$ and $\mathbb{G}_2$ may be larger because two portals in $\mathbb{G}_1$ and $\mathbb{G}_2$ may be mapped to a single portal in G_1 and

G_2. However, these portals must be on the cycles in G_1 and G_2 found by the Separator Theorem. Each time the Separator Theorem is applied, at most 2 "cycles" (also called hard cycles) are introduced in the uncompression graph G. [7] On the other hand, each time the recursive call reduces the weight of the graph by a constant ratio, thus, the depth of the recursion is at most $\mathcal{O}(\log n)$. Hence, the number of hard "cycles" is at most $\mathcal{O}(\log n)$. Therefore, the number of portals in $\mathbb{G}_1$ and $\mathbb{G}_2$ is at most the number of portals of G_1 and G_2 plus $2\mathcal{O}(\log n)$, i.e., $\mathcal{O}(k)$. The number of super-portals of $\mathbb{G}_1$ or $\mathbb{G}_2$ is at most the number of portals in them plus the number of hard cycles and the cutvertices between these cycles, which is $\mathcal{O}(k) + \mathcal{O}(\log n) = \mathcal{O}(k)$. Since $\mathbb{G}_1$ and $\mathbb{G}_2$ have constant number of portal faces, by Lemma 6.5.4, they have at most $2^k = n^{\mathcal{O}(\frac{1}{\varepsilon})}$ external types each. Similarly it can be shown that G_C has constant number of portal faces, $\mathcal{O}(k)$ super-portals, and $n^{\mathcal{O}(\frac{1}{\varepsilon})}$ external types with respect to P_C and its portal faces.

6.5.3 Dynamic Programming

After the decomposition, the next step of the algorithm is using dynamic programming to solve the problem. Let $\mathbb{G}$ be a plane graph with a distinguish set of hard edges. A subgraph $\mathbb{H}$ of $\mathbb{G}$ is said to be *consistent* if $\mathbb{H}$ is a spanning subgraph of $\mathbb{G}$ and contains all hard edges of $\mathbb{G}$. The subproblems are defined as follows and the goal is to solve the subproblems approximately.

Definition 6.5.5 *For a pair* $(G, \mathbb{G})$ *in a node of* $\mathcal{T}$ *and an external type* t *for* $\mathbb{G}$*, the* subproblem $(\mathbb{G}, t)$ *is to find a min-cost* $(2\text{—VC}, P)$*-safe consistent subgraph* $\mathbb{H}$ *of* $\mathbb{G}$ *which is compatible with* t*, or else declare that* $(\mathbb{G}, t)$ *is infeasible (no such* $\mathbb{H}$*).*

As before, the total number of subproblems over all choices of $\mathbb{G}$ and t is $2^{O(k)} = n^{\mathcal{O}(\gamma/\varepsilon)}$. The subproblem $(\mathbb{G}_0, t_0)$ is the original 2-VCSS problem, where t_0 is empty and

[7] They may not be simple cycles in G, but be glued with a set of other cycles from previous recursive calls.

G_0 is the input graph with no portals and no hard edges. Each vertex in G_0 has weight 1, each face and edge has weight 0.

As for the 2-ECSS problem, the problems associated with the leaves where the compressed G has size $N = O(k^2)$ are considered first. All hard edges of $\mathbb{G}$ are included in the solution to $(\mathbb{G}, t)$. The subproblem instance are solved exactly in $2^{O(\sqrt{N})} = n^{O(\gamma/\varepsilon)}$ time by applying Lipton-Tarjon Theorem [85] to G (explicitly) and $\mathbb{G}$ (implicitly).

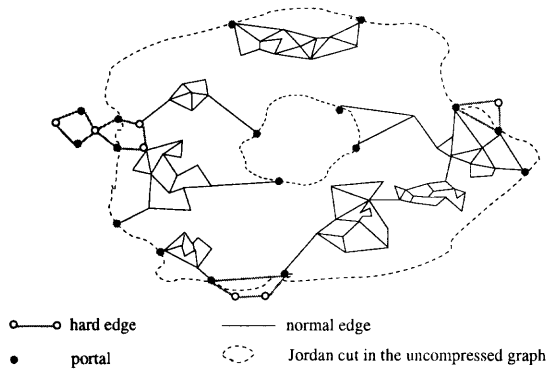
Let $(\mathbb{G}, t)$ be a subproblem associated with an internal node of T based on the solutions to the subproblems associated with the children nodes. Suppose the problem instance for the children of $\mathbb{G}$, $\mathbb{G}_A$, $\mathbb{G}_B$, $\mathbb{G}_1$ and $\mathbb{G}_2$, have been solved.

Let G_C represent either G_A or G_B. Let $t(C)$ be the subgraph of t that is induced by the super-portals of $\mathbb{G}_C$. The minimum subgraph $\mathbb{H}_C$ of $\mathbb{G}_C$ compatible with type $t(C)$ can be found by looking up the solution to the problem instance of $\mathbb{G}_C$.

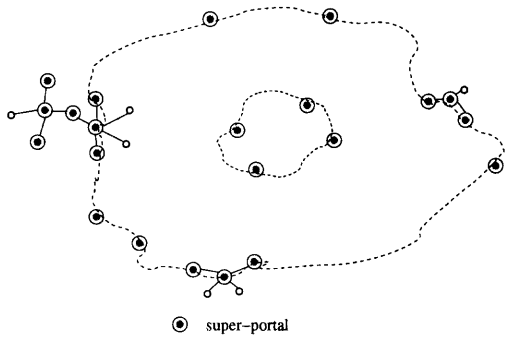
Let $t' = t - t(C)$. The solution $\mathbb{H}'$ to $(\mathbb{G}', t)$ can be found by combining the solutions to $(\mathbb{G}_1, t_1)$ and $(\mathbb{G}_2, t_2)$ where $t_1 \cup t_2 = t'$. Then, $\mathbb{H}_C \cup \mathbb{H}'$ is approximately the minimum consistent subgraph of G for external type t.

Analysis of the algorithm As the algorithm for 2-ECSS, the only source of error comes from the cycles in the the separator, i.e., the hard edges. Using similar analysis, one can show that by selecting an appropriate constant in $k = \Theta((\gamma/\varepsilon)\log n)$, the error is at most ε times the optimal.

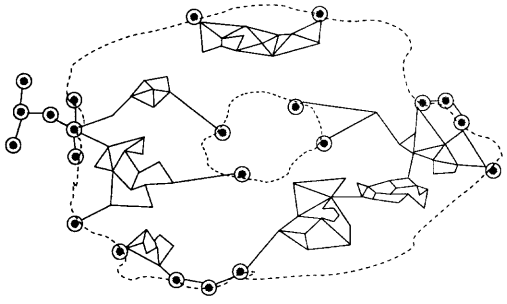
For the time complexity, the time spent in the decomposition phase is mainly for the run of the Separator Theorem which is applied at most $\mathcal{O}(n \log n)$ times, so the total time is $\mathcal{O}(n^2 \log n)$. In the dynamic programming, there are $n^{\mathcal{O}(\gamma/\varepsilon)}$ subproblems, each can be solved in $n^{\mathcal{O}(\gamma/\varepsilon)}$ time. So, the running time for the whole algorithm is $n^{\mathcal{O}(\gamma/\varepsilon)}$. This proves Theorem 6.1.2.



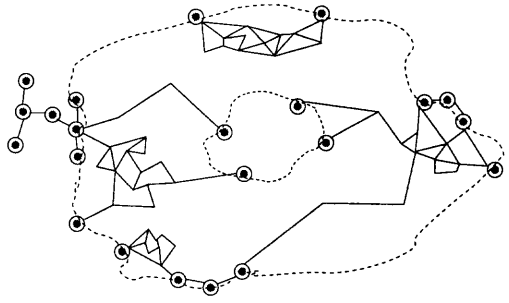
(a) A (2−VC, P)-safe graphH



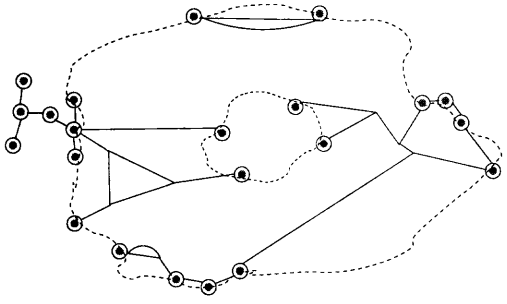
(b) $\widehat{H_T}$ obtained after step 1



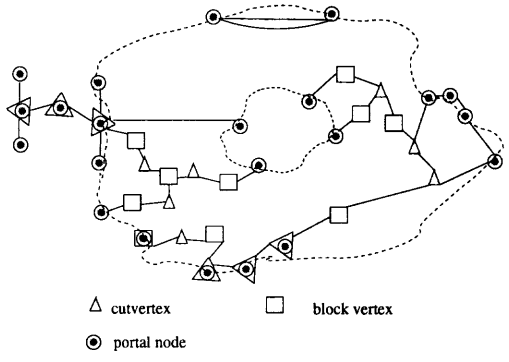
(c) $\widehat{H}$obtained after step 2



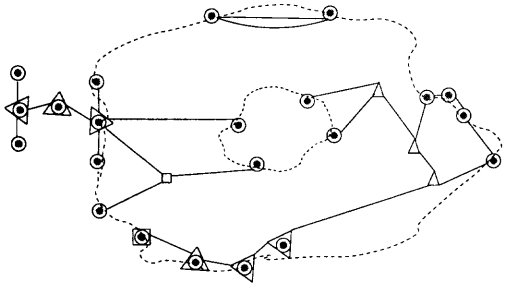
(d) $\widehat{H}$ after step 3



(e) H′obtained after step 4



(f) graph obtained after step 5a-5f



(g) the type t(H) of H

Figure 6.2 Type of a (2−VC, P)-safe graph used in the 2-VCSS Algorithm .

CHAPTER 7

APPROXIMATION SCHEMES FOR MINIMUM 2-CONNECTED SPANNING SUBGRAPHS IN WEIGHTED PLANAR GRAPHS

7.1 Introduction

In this chapter the approximation algorithms for the 2-ECSS and 2-VCSS problem in arbitrary weighted planar graphs are studied. A standard relaxation of the 2-ECSS problem is also considered which is to find a minimum weight 2-EC spanning sub-*multigraph* (2-ECSSM) H of G, meaning that an edge of G can be used multiple times in H (consequently its weight is also counted multiple times in H). Another classical extension is the *1-2-connectivity problem*: each vertex v is assigned a connectivity type $r_v \in \{1,2\}$. The problem is to find a minimum weight spanning subgraph such that for any pair of vertices $v, u \in V$, there are at least $r_{uv} = \min\{r_u, r_v\}$ edge-disjoint or vertex-disjoint paths between v and u. The 1-2-edge-connectivity is denoted by $\{1,2\}$-EC, and 1-2-vertex-connectivity by $\{1,2\}$-VC. The relaxed 1-2-edge-connectivity problem where each edge may be used more than once is also considered.

7.1.1 Related Results

All the problems mentioned above have been extensively studied in the literature. Since all these problems are $\mathcal{NP}$-hard, the main research has been devoted to designing efficient approximation algorithms, see the survey [70] and more recent advances [19, 39, 40, 66, 74]. In general, one would prefer to design a PTAS. However, all the problems considered in this chapter are max-SNP-hard [25]. Therefore they do not have a PTAS unless $\mathcal{P} = \mathcal{NP}$. But this does not preclude a PTAS for restricted classes of graphs: indeed, in the previous chapter, it has been shown that there are PTASs for both 2-ECSS and 2-VCSS problems in *unweighted planar graphs* as. In fact, the approximation schemes of in Chapter 6 allow *weighted* planar graphs, but then the algorithms will either run in time $n^{O\left(\frac{w(G)}{\varepsilon \cdot OPT}\right)}$ to

ensure an $(1+\varepsilon)$-approximate solution or they run in polynomial time with an approximation guarantee of $1+O(\frac{w(G)}{\varepsilon \cdot OPT})$. Since the ratio $w(G)/OPT$ could be arbitrarily large, these algorithms are in general not PTAS's for weighted planar graphs.

For both the 2-ECSS and 2-VCSS problems in weighted planar graphs, the best known polynomial-time or even quasi-polynomial-time approximation guarantee is still 2 [73, 94], which is achieved by polynomial-time algorithms working for general weighted graphs. The best known result for {1,2}-ECSS in weighted planar graphs is a 2-approximation algorithm, due to Jain [64], which in fact solves the more general problem where $r_v \leq k$ for any k. For the weighted {1,2}-VCSS problem, Fleischer [36] gives a 2-approximation algorithm, which actually solves the {0,1,2}-VCSS problem. A PTAS for the geometric version of these problems is presented in [27].

7.1.2 New Contributions and Techniques

Efficient approximation schemes for all the above mentioned problems in weighted planar graphs will be presented. These approximation algorithms depend in a crucial way on the new construction of *light spanners* for planar graphs.

Let G be a weighted graph. Let $d_G(u,v)$ denote the weighted shortest path distance between the vertices u and v in G. An s-*spanner* of G is a spanning subgraph H of G such that $d_H(u,v) \leq s \cdot d_G(u,v)$ for all u, v. A spanner provides an approximate representation of the shortest path metric (1-connectivity) in G, but it may be much lighter than G.

Althöfer et al. [1] designed a simple greedy algorithm that for an arbitrary graph G computes an s-spanner H of G for any $s > 1$. In the case of *planar* graphs, it is shown in [1] that this spanner has weight $w(H) \leq (1+2/(s-1))MST(G)$, where $MST(G)$ is the weight of a minimum spanning tree in G. Since $MST(G) \leq OPT$ for all the problems considered, this bounds the ratio $w(H)/OPT$ in terms of just s. If all weighted graphs in a graph family have spanners with such a bound on $w(H)/OPT$ (depending only on s), then the family is said to have *light spanners* for this problem. Light spanners are known to be very

useful for solving various optimization problems on graphs. For example, planar graphs have light spanners for metric-TSP: the first step in the metric-TSP PTAS for weighted planar graphs [4] is to replace the input graph with an accurate enough s-spanner (using [1]), thus effectively bounding $w(G)/OPT$ for the remainder of the algorithm. Spanners are also used in complete geometric graphs to design efficient PTAS's for geometric TSP and related problems [101], and to design PTAS's for the 2-edge and 2-vertex-connectivity problems [26, 27].

By combining the spanner constructed in [1] with the planar separator decomposition approach tuned to analyze 2-connected graphs in Chapter 6, it will be shown that one can design a PTAS for the 2-ECSSM problem and a PTAS for the $\{1,2\}$-ECSSM problem. However, this approach of replacing the input graph with an s-spanner fails for the 2-ECSS and 2-VCSS problems. The reason is that a spanner does not have to be 2-connected, thus the spanner may not contain the optimal or a near optimal solution in most cases. Naturally, one may think to use light *fault-tolerant* spanners (see, e.g., in [82]), which are subgraphs that persist as s-spanners even after deleting a constant number of vertices or edges. Unfortunately, this concept is not useful for weighted planar graphs, since simple examples show that light fault-tolerant spanners do not exist in weighted planar graphs, not even for a single edge deletion.

To solve the problem mentioned above, the main contribution is described: a new greedy spanner construction which produces a light planar spanner with certain desirable properties. Specifically, given a weighted planar graph G, a connected spanning subgraph A of G and $s > 1$, it computes an s-spanner H of G. H contains A as a subgraph and has total weight $w(H) = O(1/(s-1) \cdot w(A))$. Thus if the algorithm is fed with α-approximate solutions H to the various connectivity problems in a weighted planar graph G, then an $O(\alpha/(s-1))$-approximation H^* for that problem is obtained where H^* is an s-spanner for G at the same time. Furthermore, one can show that while H^* need not contain

an $(1+\varepsilon)$-approximate solution S, the number of edges of S "crossing" each face of H^* (Lemma 7.4.2) is bounded.

Using the new spanner construction technique and the planar separator decomposition, one can design approximation schemes for the 2-ECSS and 2-VCSS, {1,2}-ECSS and {1,2}-VCSS problems, which find solutions with weight at most $(1+\varepsilon) \cdot OPT$ in $n^{O(\log n \log(1/\varepsilon)/\varepsilon)}$ time; these are *quasi-polynomial time approximation schemes* (QPTAS's).

Organization. First a PTAS for the 2-ECSSM problem is presented in Section 7.2. This section contains also a description of the main algorithmic approach used in our approximation schemes, which is a combination of the use of spanners, a recursive approach driven by a variant of the planar separator theorem, and dynamic programming. Next, in Sections 7.3 and 7.4, the new construction of spanners is described and the special properties of the spanners are discussed. In Section 7.5, quasi-polynomial approximation schemes for the 2-ECSS and the 2-VCSS problems are presnted. Finally in Section 7.6, {1,2}-ECSS and {1,2}-VCSS problems are considered : a PTAS for the {1,2}-ECSSM problem and a QPTAS for each of the {1,2}-ECSS and {1,2}-VCSS problems are presented.

7.2 PTAS for the 2-ECSSM Problem

Let G be a connected weighted graph. A 2-ECSSM H of G is a spanning sub-multigraph of G in which edges can have some multiplicity and in which every pair of vertices is connected by at least two edge-disjoint paths. Note that G may not have any multiple edges at all. If an edge is used multiple times in H, its weight also contributes multiple times to the weight of H. Since it never helps to use an edge more than twice, one may cap all edge multiplicities at two. Following is a PTAS for this problem which runs in $n^{O(1/\varepsilon^2)}$ time.

Given G and $\varepsilon > 0$, the algorithms chooses s so that $s^2 \leq 1+\varepsilon$. First an s-spanner H in G is computed by the greedy spanner algorithm [1], with weight $w(H) =$

$O((1/\varepsilon) \cdot \mathrm{OPT})$. Now it will be shown that there is a $(1+\varepsilon)$-approximate 2-ECSSM that uses only edges from H. Suppose S^* is an optimal 2-ECSSM in G with $w(S^*) = \mathrm{OPT}$. Now S^* is modifed such that it uses only edges from H. For each edge e of S^* not in H, remove e and add a shortest path from H of total weight at most $s \cdot w(e)$. When the path is added, the edges are added with multiplicity, but capped at two. The result of all these modifications is another 2-ECSSM S, using only edges from H, each edge used at most twice, with $w(S) \leq s \cdot \mathrm{OPT}$.

Next the algorithm applies the 2-ECSS s-approximation algorithm from Chapter 6 to the graph H', which is H with each edge duplicated. In summary, one can get the following theorem.

Theorem 7.2.1 *Let $\varepsilon > 0$ and let G be a connected weighted planar graph with n vertices. There is an algorithm running in time $n^{O(1/\varepsilon^2)}$ that outputs a 2-ECSSM of G whose weight is at most $(1+\varepsilon)$ times the minimum.*

Why this technique fails for other problems: The above technique does not work for other problems considered in this chapter, because in these problems, we are not allowed to duplicate edges from G in the output graph. Instead, our approximation schemes must consider the possibility that the near-optimal S needs some "extra" edges from outside the spanner. In Sections 7.3 and 7.4 a new type of light planar spanners are developed and the number and arrangement of those extra edges outside the spanner are limited.

7.3 Augmented Planar Spanners

In this section a new greedy algorithm is presented to construct s-spanners in weighted planar graphs, resembling the standard greedy algorithm [1] for general graphs. Just as in the standard algorithm, the algorithm takes a connected weighted graph G and a parameter $s \geq 1$, and produces an s-spanner H. Unlike the general algorithm, our G must be planar, and for each edge e of G not in H it is guaranteed that $s \cdot w(e)$ is at least the length of

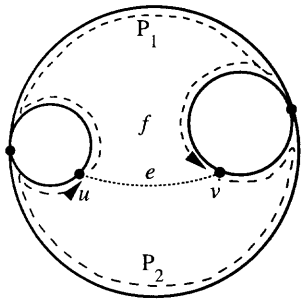


Figure 7.1 A non-simple face f in H, a chord e, and walks P_1 and P_2.

some path in the face of H containing e. The new algorithm is also provided with a third argument: a "seed" spanning subgraph A, containing edges that must appear in H. In Section 7.4 A will be used to enforce some 2-connectivity properties in the spanner.

Suppose G is a weighted *plane graph* (that is, an embedded planar graph) and H is a subgraph. A *chord* e of H is an edge of G not in H. Note that H and e inherit embeddings from G. For each chord e $w_H(e)$ is defined to be the length of the shortest walk connecting the endpoints of e, along the boundary of the face of H containing e.

More precisely, if the endpoints of e are disconnected in H, then define $w_H(e) = +\infty$. Otherwise e connects two vertices in a component of H, and e is embedded in some face f of this component. The boundary of f is a cyclic walk of (oriented) edges, with total weight $w(f)$; note that a cut-edge may appear twice in the boundary (once per orientation), and its weight would then count twice in $w(f)$. Similarly a cut-vertex may appear multiple times. The edge e splits the boundary sequence into two walks P_1 and P_2, both connecting the endpoints of e, with $w(P_1)+w(P_2) = w(f)$. Now define $w_H(e) = \min(w(P_1), w(P_2))$ (see Figure 7.1).

Given G, s, and A as above, $H = \mathrm{Augment}(G, s, A)$ is computed as follows:

Augment(G, s, A):

$H \leftarrow A$

for all edges e of G in non-decreasing $w(e)$ order **do**

if e is not in H **and** $s \cdot w(e) < w_H(e)$ **then**

add e to H

return H

Note $A \subseteq H \subseteq G$. If A is empty (has all vertices of G but no edges), then this is like the general greedy spanner algorithm [1], except that w_H replaces d_H.

Theorem 7.3.1 *Let* G *be a weighted plane graph,* $s > 1$, *and* A *a spanning subgraph of* G. *Then* $H = \mathtt{Augment}(G, s, A)$ *is an* s*-spanner of* G. *If* A *is connected, then* $w(H) \leq (1 + 2/(s-1)) \cdot w(A)$.

Proof : To show that H is an s-spanner it suffices to show that each edge of G is s-approximated in H. For e not in H, at the moment it was rejected it must be the case that $w_H(e) \leq s \cdot w(e)$. Note that $w_H(e)$ may only decrease after that, so $d_H(e) \leq w_H(e) \leq s \cdot w(e)$ at the end of the algorithm.

For the second part one needs to show that the weight of all edges in H but not A is at most $(2/(s-1)) \cdot w(A)$. Suppose e is such an edge; then e is not a cut edge in H since A is a connected spanning subgraph. Therefore e is bounded by two distinct faces. Let f be either face bounding e. First one can claim that $w(f) > (1 + s) \cdot w(e)$. To see this, consider the *last* edge e' added to f whose boundary consists of a path P plus e'. Since e' is added to H, one must have $s \cdot w(e') < w_H(e')$ and $w_H(e') \leq w(P)$. Adding $w(e')$ to both sides of $s \cdot w(e') < w(P)$, and noting $w(e) \leq w(e')$, one gets the claim.

For each face f of A, let E_f be the set of edges in H crossing the interior of f. Since the sum of $w(f)$ over all faces of A is $2 \cdot w(A)$, it suffices to show that $w(E_f) \leq (1/(s-1)) \cdot w(f)$. Note that the edges dual to E_f define a tree on the faces of H inside f. Orient this dual tree away from some arbitrarily chosen root: now for each $e \in E_f$, an adjacent face f_e of H (only the root was not picked) is chosen. For each $e \in E_f$ it is known $w(f_e) - 2 \cdot w(e) > (s-1) \cdot w(e)$, from the previous paragraph. Summing these inequalities over all $e \in E_f$, one gets at most $w(f)$ on the left hand side, and exactly $(s-1) \cdot w(E_f)$ on the right. □ □

7.4 Spanners and 2-EC Subgraphs

Suppose a weighted plane 2-EC graph G is given, where the goal is to find an $(1+\varepsilon)$-approximate 2-ECSS. First one can construct an auxiliary subgraph H^*, as follows:

1. Compute a 2-approximate 2-ECSS A, in polynomial time.
2. Compute $H^* = \texttt{Augment}(G, \sqrt{2}, A)$.

The constant $\sqrt{2}$ here is not critical, just convenient. By Theorem 7.3.1, H^* is a 12-approximate 2-ECSS. Below it will be shown that for every $\varepsilon > 0$, this H^* has nice intersection properties with some $(1+\varepsilon)$-approximate 2-ECSS in G.

Given a face f in H^*, the chords of f are the edges of G embedded inside this face, according to G's embedding. A *face-edge* e of f is an abstract edge connecting two vertices of f; unlike a chord, a face-edge is not necessarily an edge of G. (If vertices appear more than once on f, one must specify which appearances is the endpoints of e.) The face edge e is said to *crosses* a chord c if: c is a chord of the same face f, their endpoints are distinct vertex appearances on f, and they appear in cyclic "ecec" order around the boundary of f. Note that e may be embedded inside f so e intersects only the crossed chords.

Suppose S is a 2-ECSS in G, and an edge c of S is not in H^*. Then c is a chord of some face f of H^*. Let P_c be the path in f connecting the endpoints of c, such that $w(P_c) \leq \sqrt{2} \cdot w(c)$. Then the *chord move at* c is the following modification of S: add to S all the edges of P_c that were not already in S, and remove from S any chords inside the cycle $c \cup P_c$ (see Figure 7.2(a)). Since H^* is 2-EC, the cycle has no repeated edges, and therefore S is still a 2-ECSS after the chord move. The chord move is *improving* if $w(S)$ decreases; this happens whenever $w(P_c)$ (or $\sqrt{2} \cdot w(c)$) is less than the weight of the discarded chords. Any non-trivial chord move brings S closer to H^* (in Hamming distance), thus at most $O(n)$ improving chord moves apply to any given 2-ECSS S.

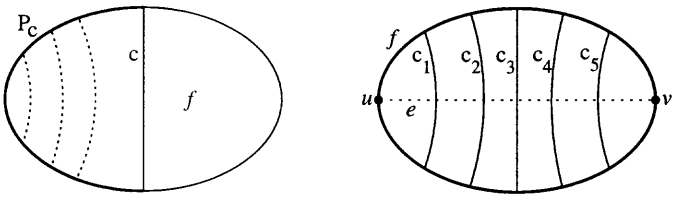


Figure 7.2 (a) Face f of H^* (oval) with chord c, path P_c (bold), and chords removed from S by the chord move at c (dotted). (b) Face f with a face-edge e (dashed) crossed by five chords from S.

Lemma 7.4.1 *Let* e *be a face edge of a face* f *in* H^* *and* S *be a 2-ECSS of* G. *Suppose* C *is a set of edges of* S *all of which are chords crossing* e, $\sqrt{2} \cdot \min_{c \in C} w(c) > \max_{c \in C} w(c)$, *and no chord in* C *gives an improving chord move. Then* $|C| \leq 4$.

Proof : If not, S has five chords crossing e as in Figure 7.2(b). But then there is an improving chord move at c_3, since the discarded chords ($\{c_1, c_2\}$ or $\{c_4, c_5\}$) weigh more than $\sqrt{2} \cdot w(c_3)$. □

Now one can argue that by accepting a small additive error in the 2-ECSS, one may assume it has only a small number of chords crossing a given face-edge:

Lemma 7.4.2 *Suppose* G *and* H^* *are as above,* $\varepsilon > 0$, S *is a 2-ECSS,* f *is a face in* H^*, *and* e *is a face-edge in* f. *Then there exists a 2-ECSS* S' *such that* $w(S') \leq w(S) + \varepsilon \cdot w(C'_f)$, *where* C'_f *is the set of edges of* S' *that are chords crossing* e, $C'_f \subseteq S$, *and* $|C'_f| = O(\log(1/\varepsilon))$.

Proof : First one can suppose that S has no improving chord move at a chord crossing e, since such a move could only remove some chords crossing e. Let C_f be the set of chords in S crossing e. Arrange C_f in "left to right" order, according to how they intersect e. Let $c_0 \in C_f$ be the chord with maximum weight. Say that a chord $c \in C_f$ is *short* if $w(c) \leq \varepsilon \cdot w(c_0)/(2\sqrt{2})$. Now if there are short chords to the left of c_0, perform a chord move at the rightmost one, c_l. Similarly if there are short chords to the right of c_0, perform

a chord move at the leftmost one, c_r. S' is the result of these (at most) two chord moves; note that C'_f contains no short chords except possibly c_l and c_r.

Map each non-short chord $c \in C'_f$ to the real number $\log(w(c_0)/w(c))$, a point in the real interval $I = [0, \log(1/\varepsilon) + 3/2]$. Note that two edges can be mapped to the same semi-open subinterval of I of length $1/2$ only if the heavier edge has weight less than $\sqrt{2}$ times that of the lighter edge. By Lemma 7.4.1, at most four edges can be mapped into the same subinterval of I of length $1/2$. This implies $|C'_f| = O(\log(1/\varepsilon))$.

The chord moves in f increased $w(S')$ by at most $\sqrt{2}(w(c_l) + w(c_r)) \leq \varepsilon \cdot w(c_0)$, which is at most $\varepsilon \cdot w(C'_f)$. □ □

Remarks: In the 2-VCSS case, the initial A should be a 2-approximate 2-VCSS, so that H^* is a 12-approximate 2-VCSS. Then in the chord move the cycle has no repeated vertices, therefore S remains a 2-VCSS after the move. In Lemmas 7.4.1 and 7.4.2, the only properties of H^* that are needed were that it was 2-EC (or 2-VC), and that $w_{H^*}(e) \leq \sqrt{2} \cdot w(e)$ for each chord e.

7.5 Approximation Schemes for the 2-ECSS and 2-VCSS Problems

In this section it will be shown how to use our new spanner construction to find quasi-polynomial time approximation schemes for the 2-ECSS and the 2-VCSS problems in weighted planar graphs. Following is the QPTAS for 2-ECSS problem.

A similar framework as that in the PTAS for 2-ECSSM problem in Section 7.2 is used. But instead of using the spanner constructed as in [1], now one may use the augmented spanner H^* as constructed in Section 7.4.

First Theorem 6.3.3 is applied to H^* with $k = \Theta(\log n/\varepsilon)$ to decompose H^*. However, different from the PTAS for the 2-ECSSM problem, H^* may not contain a near-optimal solution of the 2-ECSS problem. Thus one cannot work on the pieces of H^* directly. Fortunately, Lemma 7.4.2 guarantees that there exists a near-optimal solution with

at most $O(k \log(1/\varepsilon))$ edges crossing the Jordan curve J. These crossing edges can be guessed by trying all $n^{O(k \log(1/\varepsilon))}$ possibilities. The guessed edges are added to the corresponding pieces, and the vertices of H^* along J together with the endpoints of the guessed edges determine the set of portals P for the new pieces. For each possible new piece with the guessed edges, weights are assigned to the new portals such that each new piece has cost at most constant fraction of H^* and $O(k)$ portals. Then the new pieces are recursively decomposed.

As in the PTAS for the 2-ECSSM problem, for each piece edge-connecti-vity types are defined to describe how these portals may be connected outside one of the pieces in a $(1+\varepsilon)$-approximate solution. The number of types for each piece is $2^{O(k \log(1/\varepsilon))}$, exponential in the number of portals. Then, dynamic programming is used to solve the subproblems as before and the cycle edges are committed to the solution.

The approximation scheme for the 2-VCSS problem is similar, and only the differences are mentioned here: first, H^* is redefined as remarked at the end of Section 7.4, and then vertex-connectivity types are defined using the same techniques as in Chapter 6.

The error of the final solution comes from two sources. First, the edges of the cycles that arose from the application of the separator theorem to the solution are added. Since each piece in the decomposition has weight at most constant fraction of its parent weight, the depth of the recursive calls is $O(\log n)$. As before, the total error per recursive level is $O((w(H^*)/k) \log n)$, where $k = \Theta(\log n/\varepsilon)$ and $w(H^*) = O(OPT/\varepsilon)$. By an appropriate choice of the leading constant defining k, this is at most $(\varepsilon/2) \cdot OPT$.

Moreover, each time a face of H^* (or its pieces) is cut by a Jordan curve, $O(\log(1/\varepsilon))$ crossing edges are guessed. If these edges are guessed optimally (they were edges in some original optimal S^*), then by Lemma 7.4.2 an additive error of at most $\varepsilon/2$ times the weight of these guessed edges is paid. Summing over the entire assembly of a possible solution, the total of these errors is at most $(\varepsilon/2) \cdot OPT$.

The dominating factor in the running time comes from trying all $n^{O(k\log(1/\varepsilon))}$ possibilities for the guessed edges. The weights of the subproblems are only a constant times the weight of their respective parents and therefore a pure recursive approach (without dynamic programming) leads to a time bound of $T(n) \leq n^{O(k\log(1/\varepsilon))}T(c \cdot n)$ $(0 < c < 1)$, with solution $n^{O((1/\varepsilon)\cdot\log(1/\varepsilon)\cdot\log^2 n)}$. This bound may be improved by a logarithmic factor in the exponent by using dynamic programming and by a more careful count of subproblems. The following lemma proved in [10] bounds the number of ways how a graph can be decomposed regardless of its weight scheme.

Lemma 7.5.1 *Let* G *be a planar graph on* n *vertices with non-negative edge costs, embedded in the plane, and a parameter* $k \geq 1$. *Then one can find a list of* $O(n^2)$ *separations of* G, *such that for any valid weight scheme of the vertices, edges and faces of* G, *some separation in this list satisfies the properties of Theorem 6.3.3.*

Lemma 7.5.1 shows that a piece is partitioned in only $O(n^2)$ different ways, no matter how many arrangements of the vertices along the Jordan curve and incident with the "guessed" edges. This implies the following lemma.

Lemma 7.5.2 *The total number of distinct pieces (contracted subgraphs) of the original* H^* *that occur during our recursive decomposition is* $n^{O(\log n)}$. *Therefore the number of distinct subproblems (a piece,* $|P| = O(k\log(1/\varepsilon))$ *portals selected in the piece, and an external connectivity type on those portals) is* $n^{O(\log n)}n^{O(|P|)}2^{O(|P|)} = n^{O(k\log(1/\varepsilon))}$.

Theorem 7.5.3 *Let* $\varepsilon > 0$ *and let* G *be a 2-EC (2-VC) weighted planar graph with* n *vertices. There is an algorithm running in time* $n^{O(\log n\cdot\log(1/\varepsilon)/\varepsilon)}$ *that outputs a 2-ECSS (2-VCSS)* H *of* G *such that* $w(H) \leq (1+\varepsilon)\cdot \mathrm{OPT}$.

7.6 Extensions to the $\{1,2\}$-Connectivity Problem

In this section, one can extend the previous results to the $\{1,2\}$-connectivity problems in weighted planar graphs. One needs to focus on the algorithm for the $\{1,2\}$-ECSS problem

only. The algorithm for the $\{1,2\}$-VCSS problem can be obtained similarly. The algorithms presented are modifications of the respective algorithms in Sections 7.2 and 7.5.

First, consider the $\{1,2\}$-ECSSM problem, which is a relaxed version of the $\{1,2\}$-ECSS problem where duplicate edges are allowed. As in Section 7.2, it can be shown that there is a $(1+\varepsilon)$-approximate $\{1,2\}$-ECSSM that uses only edges from a light $(1+\varepsilon)$-spanner H. So instead of G, one can work on H with duplicated edges.

The main difference from Section 7.2 is the dynamic programming part. The connectivity types need to be redefined to reflect the non-uniform connectivity requirement. For this, one can use the connectivity type construction in Chapter 5. Informally, the main difference is that each time a 2-connected component or path is contracted, the highest connectivity requirement among all contracted vertices is assigned to the new vertex. This increases the number of types from $2^{O(|P|)}$ to $2^{O(2|P|)}$, where P is the set of portals in the given graph. Again a PTAS is obtained with running time $n^{O(1/\varepsilon^2)}$.

Now consider the $\{1,2\}$-ECSS problem. First a 2-approximate solution A can be found using algorithms from [64] (or [36] for $\{1,2\}$-VCSS). Then A is augmented into a light spanner H^* as in Section 7.4. Using similar arguments as in the proof of Lemma 7.4.2, one can show that there is a $(1+\varepsilon)$-approximate $\{1,2\}$-ECSS S so that for each picked face-edge e, only $O(\log(1/\varepsilon))$ edges of S cross e. Now redefine the connectivity types as above. Finally, dynamic programming is used to solve the problem. The running time is still dominated by the number of subproblems $n^{O(\log n \cdot \log(1/\varepsilon)/\varepsilon)}$. Hence, a QPTAS is obtained in this case.

The results in this section are summarized as follows.

Theorem 7.6.1 *Let $\varepsilon > 0$ and let G be a weighted planar graph with n vertices. There is an algorithm running in time $n^{O(1/\varepsilon^2)}$ that outputs a $\{1,2\}$-ECSSM of G whose weight is at most $(1+\varepsilon)\cdot \mathrm{OPT}$.*

Theorem 7.6.2 *Let* $\varepsilon > 0$ *and let* G *be a weighted planar graph with* n *vertices. There is an algorithm running in time* $n^{O(\log n \cdot \log(1/\varepsilon)/\varepsilon)}$ *that outputs a* $\{1,2\}$*-ECSS* H *of* G *such that* $w(H) \leq (1+\varepsilon) \cdot \mathrm{OPT}$.

Theorem 7.6.3 *Let* $\varepsilon > 0$, *and let* G *be a weighted planar graph with* n *vertices. There is an algorithm running in time* $n^{O(\log n \cdot \log(1/\varepsilon)/\varepsilon)}$ *that outputs a* $\{1,2\}$*-VCSS* H *of* G *such that* $w(H) \leq (1+\varepsilon) \cdot \mathrm{OPT}$.

CHAPTER 8

FAULT-TOLERANT GEOMETRIC SPANNERS

8.1 Introduction

This chapter considers geometric fault-tolerant spanners in Euclidean spaces which are formally defined as follows.

Definition 8.1.1 (Fault-tolerant spanners [83]) *Let* V *be a set of* n *points in a metric space,* $t \geq 1$ *a real, and* k *a non-negative integer. A graph* $G = (V, E)$ *is called a* k-vertex fault-tolerant t-spanner for V, *denoted by* (k, t)-*VFTS, if for any subset* V' *of* V *of size at most* k, *the graph* $G \setminus V'$ *is a* t-*spanner for the point set* $V \setminus V'$.

Definition 8.1.2 (Edge fault-tolerant spanners [83]) *Let* V *be a set of* n *points in a metric space,* $t \geq 1$, *and* k *an integer. A graph* $G = (V, E)$ *is called a* k-edge fault-tolerant t-spanner for V, *denoted by* (k, t)-*EFTS, if for any subset* E' *of* E *of size at most* k *and for any pair of points* p *and* q *in* V, *the graph* $G \setminus E'$ *contains a* pq-*path of total length at most* t *times the length of a shortest path between* p *and* q *in the graph* $K_V \setminus E'$.

One can show (see [83] and [86]) that a (k, t)-VFTS is also a (k, t)-EFTS. Therefore, from now on, the focus will be on vertex fault-tolerant spanners.

8.1.1 Previous Results

This problem of efficiently constructing good fault tolerant spanners has been proposed recently by Levcopoulos et al. [83]. They presented in [83] three algorithms constructing k-vertex fault-tolerant spanners. Their first algorithm is based on the observation that the $(k+1)$-power of a t-spanner is a (k, t)-VFTS (an s-power of a graph G is a graph with the same vertex set as G and it contains an edge between any pair of vertices that are connected by a path in G with at most s edges). Therefore, if one starts with the spanner construction from

[55], then in time $\mathcal{O}(n \log n) + n\, 2^{\mathcal{O}(k)}$ one can obtain a (k, t)-VFTS of maximum degree $2^{\mathcal{O}(k)}$ and the total cost of $2^{\mathcal{O}(k)}$ times the cost of a MST. For a constant k, this construction leads to a fault-tolerant spanner with asymptotically optimal parameters. The other two algorithms described by Levcopoulos et al. [83] use the *well-separated pair decomposition* [13]. It is shown that a k-vertex fault-tolerant spanner can be constructed (i) in $\mathcal{O}(n \log n + k^2 n)$ time with $\mathcal{O}(k^2 n)$ edges, or (ii) in $\mathcal{O}(n k \log n)$ time with $\mathcal{O}(n k \log n)$ edges. Neither the maximum degree nor the total cost of the fault-tolerant spanner is bounded in these two algorithms.

In a follow-up paper, Lukovszki [86] gave a construction of (k, t)-VFTSs with the optimal number of edges $\mathcal{O}(n k)$; the running time of this algorithm is $\mathcal{O}(n \log^{d-1} n + n k \log\log n)$. Lukovszki presented also a construction of (k, t)-VFTSs with the maximum degree of $\mathcal{O}(k^2)$ and investigated fault-tolerant spanners that allow the use of Steiner points.

The main idea of that construction in [86] is to take a high-quality (that is, with constant maximum degree and total cost proportional to the cost of MST, see [5, 55]) t-spanner and then put around each point k Steiner points, so that the distance between the point and any Steiner point associated with it is infinitesimally small. Each point is connected with all Steiner points associated with it. Furthermore, whenever there is an edge in the spanner between a pair of input points x and y, then this edge is left in our fault-tolerant spanner and k new edges are created by connecting in a matching the Steiner points associated with x and y. It is not difficult to see that the obtained graph has exactly $k n$ additional Steiner points, the degree of each vertex (either input point or a Steiner one) is upper bounded by $\mathcal{O}(k)$, and that the total cost of the graph is $\mathcal{O}(k)$ times the cost of minimum spanning tree. Furthermore, one can show that if one removes any set of at most k vertices from that graph, then for any pair of vertices corresponding to the remaining points there is a path connecting these points whose length is upper bounded by $t + \varepsilon$ times the Euclidean distance between the points.

8.1.2 New Contributions

The main open problem left in [83] and [86] is whether there exist fault-tolerant spanners having good bounds for the maximum degree and the total cost. The best bounds obtained in the prior constructions were the k-fault-tolerant spanners having the maximum degree of $\mathcal{O}(k^2)$ by Lukovszki [86], and the one having the total cost of $2^{\mathcal{O}(k)}$ times the cost of an MST by Levcopoulos et al. [83].

The first result in this paper resolves that problem and gives a construction of *optimal* (k,t)*-VFTSs*.

Theorem 8.1.3 *Let* V *be a set of points in* $\mathbb{R}^d$. *Let* $t > 1$ *and let* k *be a positive integer. Then, one can construct a* (k,t)*-VFTS for* V *that has maximum degree* $\mathcal{O}(k)$ *and whose total cost is* $\mathcal{O}(k^2 \cdot W_{MST})$, *where* W_{MST} *denotes the cost an* MST *of* V. *The constants implicit in the* $\mathcal{O}$*-notation depend on* t *and* d.

Notice that by the arguments above this result implies the identical result for k-edge fault-tolerant spanners.

It is not difficult to see that the spanner promised in Theorem 8.1.3 has asymptotically *optimal bounds both for the maximum degree and the total cost.* Indeed, since a (k,t)-VFTS ((k,t)-EFTS) has to be $k+1$-edge connected, every vertex must have degree $\Omega(k)$. To see that the total cost must be $\Omega(k^2 \cdot W_{MST})$ in the worst-case, consider the following construction (see also [83]), see Figure 8.1. Suppose that k is even. Let c_1 and c_2 be two points with $|c_1c_2| = 1$ and let $r \ll 1/n$. Consider n points such that $k/2$ of them are contained in a ball B_1 of radius r centered at c_1 and the remaining $n - k/2$ points are contained in another ball B_2 of radius r with the center at c_2. Since any MST of these n points has only a single edge between B_1 and B_2, $W_{MST} = \mathcal{O}(1)$. However, since the minimum degree of any k-vertex (or k-edge) fault-tolerant spanner is $k+1$, every vertex in B_1 has to be connected to at least $k/2 + 2$ vertices contained in ball B_2. Therefore, there are $\Omega(k^2)$ edges between B_1 and B_2, and the cost of any k-vertex (or k-edge) fault-tolerant spanner is $\Omega(k^2 \cdot W_{MST})$.

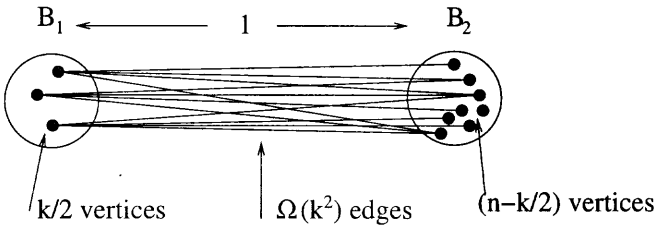


Figure 8.1 Any VFTS for points in B_1 and B_2 must have weight at least $\Omega(k^2)$, while the MST has weight $\mathcal{O}(1)$.

The construction in Theorem 8.1.3 is a generalization of the greedy algorithm that has been used before to construct spanners [1, 5, 15, 30, 55]. The main contribution in this context is the first, precise analysis of the fault-tolerant spanners obtained in that construction.

The next contribution gives an efficient construction of good fault-tolerant spanners. The construction from Theorem 8.1.3 gives fault-tolerant spanners having optimal parameters, but it does not lead to an efficient algorithm for constructing the spanners. The following theorem shows that one can construct very efficiently a fault-tolerant spanner whose total cost is just slightly larger than optimal (and the maximum degree remains optimal).

Theorem 8.1.4 *Let V be a set of n points in $\mathbb{R}^d$. Let $t > 1$ and let k be a positive integer. Then, in time $\mathcal{O}(n\,k\log^d n + n\,k^2\log k)$, one can construct a directed (k,t)-VFTS that has maximum degree $\mathcal{O}(k)$ and whose total cost is $\mathcal{O}(k^2\cdot\log n\cdot W_{\mathrm{MST}})$, where W_{MST} denotes the cost of an* MST *of V. The constants implicit in the $\mathcal{O}$-notation depend on t and d.*

Our efficient algorithm from Theorem 8.1.4 is based on a new, interesting property of Euclidean graphs that gives a sufficient condition (characterization) for graphs to be (k,t)-VFTSs.

8.2 Preliminaries

In this chapter, many algorithms investigated will consider pairs of points (edges) in some sequential order. For convenience, the total order on the costs of the edges in E is introduced, such that for two edges $(x, x'), (y, y') \in E$, edge (x, x') is *shorter* than edge (y, y') if either $|xx'| < |yy'|$ or $|xx'| = |yy'|$ and edge (x, x') is taken by the algorithm before edge (y, y').

As mentioned before, *both, directed and undirected Euclidean graphs* will be considered. Throughout the chapter (x, y) will denote *both*, an undirected and a directed edge from x to y. In the first part of the chapter, in Section 8.3, *undirected* graphs and undirected fault-tolerant spanners are analyzed, while in the second part of the chapter, in Section 8.4, for convenience, *directed* graphs and their spanners will be considered. Notice however that this distinction is really for convenience only, since any undirected spanner can be converted to directed one by replacing each edge with two directed edges; similarly any directed spanner can be transformed to be undirected by making every edge undirected and then removing the parallel edges between all pairs of vertices.

8.2.1 Menger's Theorem and Its Consequences

The following lemma that follows easily from Menger's theorem (see [11, Chapter III, Theorem 5]) will be used later.

Lemma 8.2.1 *Let $G = (V, E)$ be an undirected graph. Let $v, u \in V$. Let $X \subseteq (V \setminus \{v, u\})$ be a set of s vertices such that for each $x \in X$*

- *either $(v, x) \in E$ or there are s internally vertex-disjoint vx-paths in G, and*
- *either $(x, u) \in E$ or there are s internally vertex-disjoint xu-paths in G.*

Then, there are s internally vertex-disjoint vu-paths in G. In particular, if one removes any $s - 1$ vertices in $V \setminus \{v, u\}$ from G, then the obtained graph still contains a vu-path. □

One can easily extend the above lemma to obtain the following.

Lemma 8.2.2 *Let* $G = (V, E)$ *be an undirected graph. Let* $v, u \in V$. *Let* $E^+ = \{(v_1, u_1), \ldots, (v_s, u_s)\}$ *be such that*

- *for every* i, $1 \leq i \leq s$, *either* $(v_i, u_i) \in E$, *or* $v_i = u_i$ *and* $v_i \neq v, u$,
- *all* u_i *and* v_i *that are neither* u *nor* v *are pairwise distinct,*
- *for every* i, $1 \leq i \leq s$, *either* $(v, v_i) \in E$ *or there are* s *internally vertex-disjoint* vv_i*-paths in* G, *and*
- *for every* i, $1 \leq i \leq s$, *either* $(u_i, u) \in E$ *or there are* s *internally vertex-disjoint* u_iu*-paths in* G.

Then, there are s *internally vertex-disjoint* vu*-paths in* G. *In particular, if one removes any* $s - 1$ *vertices in* $V \setminus \{v, u\}$ *from* G, *then the obtained graph still contains a* vu*-path.* □

8.3 k-Vertex Fault-Tolerant Spanners of Low Degree and Low Cost

In this section, the following algorithm will be analyzed to show it constructs (k, t)-VFTSs of both low degree and low cost.

k-Greedy Algorithm

Input: A complete *undirected* Euclidean graph $G = (V, E)$, integer $k \geq 0$, real $t > 1$

Output: A (k, t)-VFTS $G' = (V, E')$ for V

```
E' = ∅
G' = (V, E')
for each edge (u, v) ∈ E taken in nondecreasing order by length do
    if G' = (V, E') does not have k + 1 internally vertex-disjoint t-spanner uv-paths
        then E' = E' ∪ {(u, v)}
             G' = (V, E')
output G' = (V, E')
```

Claim 8.3.1 *The* k*-Greedy Algorithm constructs a* (k, t)*-VFTS.*

Proof : Let V' be any subset of V having size at most k. First prove that $G' \setminus V'$ is a t-spanner for $V \setminus V'$.

Pick any pair of points u, v in $V \setminus V'$. One has to show that $G' \setminus V'$ has a t-spanner uv-path. Clearly, if $(u, v) \in E'$, then this is true. So suppose that $(u, v) \notin E'$. Then, according to the algorithm, the only reason for not including edge (u, v) in E' is that there are $(k+1)$ internally vertex-disjoint t-spanner uv-paths in G'. Therefore, since $|V'| \leq k$, there is at least one such path in $G' \setminus V'$. □

Remark 8.3.2 *Notice that Claim 8.3.1 holds even if the edges are taken in an arbitrary order (that is, not necessarily nondecreasing).*

Furthermore, this claim holds not only for Euclidean graphs and the assumption that G *is a complete graph can be weakened. For general graphs, a graph* $G^* = (V^*, E^*)$ *is said to be a* (k^*, t^*)*-VFTS of a weighted graph* $G = (V, E)$ *if* $V^* = V$, $E^* \subseteq E$, *and for any* $V' \subseteq V$ *with* $|V'| \leq k^*$, *for any pair of vertices* $v, u \in V \setminus V'$, *the graph* $G^* \setminus V'$ *contains a* vu*-path of total length at most* t^* *times the length of the shortest* vu*-path in* $G \setminus V'$. *Then, the proof of Claim 8.3.1 implies that if one begins the* k*-Greedy Algorithm with an arbitrary* $(k+1)$*-connected weighted graph* $G = (V, E)$, *then the obtained graph* G' *will be a* (k, t)*-VFTS of* G.

8.3.1 Analyzing the Maximum Degree

This section is devoted to prove that the (k, t)-VFTS constructed by the k-Greedy Algorithm has maximum degree $\Theta(k)$. The analysis is in a similar spirit as the analysis of the greedy algorithm for normal spanners, see, e.g., [1, 15].

Following is an auxiliary claim that will be used in later analysis.

Claim 8.3.3 *Let* $0 < \theta < \frac{\pi}{4}$ *and suppose that* t *is chosen such that* $t \geq \frac{1}{\cos\theta - \sin\theta}$. *Let* $G' = (V, E')$. *Let* u, v, x *be any three points in* V *with* $(u, v), (u, x) \in E'$ *and* $\measuredangle(vux) \leq \theta$. *Suppose further that* $|uv| \leq |ux|$. *Then, for each* t*-spanner* vx*-path* p *in* G' *the path consisting of the edge* (u, v) *followed by the path* p *is a* t*-spanner* ux*-path.*

Proof : Let p be any t-spanner vx-path in G'. Let q be the ux-path obtained by taking the edge (u, v) followed by p. Then

$$\text{length}(q) \;=\; |uv| + \text{length}(p) \;\leq\; |uv| + t \cdot |vx| \;. \tag{8.1}$$

Furthermore, if z denotes the point on the segment ux such that $\measuredangle(uzv) = \measuredangle(vzx) = \pi/2$, then

$$|vx| \;\leq\; |vz| + |zx| \;=\; |ux| + (|vz| - |uz|) \;. \tag{8.2}$$

Next, observe that $|vz| = |uv| \cdot \sin(\measuredangle(zuv)) \leq |uv| \cdot \sin\theta$ and $|uz| = |uv| \cdot \cos(\measuredangle(zuv)) \geq |uv| \cdot \cos\theta$. Therefore, combining these two identities with (8.1) and (8.2), one obtains

$$\begin{aligned} \text{length}(q) \;&\leq\; |uv| + t \cdot |vx| \;\leq\; |uv| + t \cdot (|ux| + |uv| \cdot (\sin\theta - \cos\theta)) \\ &=\; t \cdot |ux| + |uv| \cdot (1 - t\,(\cos\theta - \sin\theta)) \;. \end{aligned}$$

Hence, by the assumption that $t \geq 1/(\cos\theta - \sin\theta)$, one can conclude the claim that $\text{length}(q) \leq t \cdot |ux|$. □

This claim can be used to prove the following result.

Claim 8.3.4 *Let $0 < \theta < \frac{\pi}{4}$ and $t \geq \frac{1}{\cos\theta - \sin\theta}$. Let $G' = (V, E')$ be the output of the k-Greedy Algorithm for V. For any $u \in V$, let $\mathcal{C}_u$ be any cone in $\mathbb{R}^d$ with the apex at u and the angular diameter[1] at most θ. Then, G' has at most $k+1$ edges incident to u that are contained in the cone $\mathcal{C}_u$.*

Proof : Let $E'_{\mathcal{C}_u}$ be the set of edges in G' incident to u that are contained in the cone $\mathcal{C}_u$. Let (u, v) be the longest edge in $E'_{\mathcal{C}_u}$. The proof is to show that if there are more than k edges in $E'_{\mathcal{C}_u}$ that are shorter than (u, v), then there are $k+1$ t-spanner uv-paths in G', each using only edges shorter than $|uv|$. This will imply that the k-Greedy Algorithm would not add edge (u, v) to E', which contradicts to the fact that (u, v) is an edge of G'.

Suppose there are $k+1$ edges in $E'_{\mathcal{C}_u}$ that are shorter than (u, v). Then one can prove the existence of $k+1$ internally vertex-disjoint uv-paths such that each path uses

[1] *The angular diameter of a cone C in $\mathbb{R}^d$ having its apex at point $p \in \mathbb{R}^d$ is defined as the maximum angle between any two vectors $\overrightarrow{px}$ and $\overrightarrow{py}$, $x, y \in C$.*

edges in G' that are shorter than (u, v). Consider first any edge $(u, z) \in E'_{\mathcal{C}_u}$ such that $(v, z) \in E'$. By Claim 8.3.3, the uv-path consisting of the edges $(u, z), (z, v)$ is of length upper bounded by $t \cdot |uv|$.

Next, consider any edge $(u, z) \in E'_{\mathcal{C}_u}$ such that $(v, z) \notin E'$. Then, since the k-Greedy Algorithm has not taken edge (v, z) to E', there must exist $k + 1$ internally vertex-disjoint t-spanner vz-paths, and each edge on these paths is shorter than (v, z), which is less than (u, v). Therefore, by Claim 8.3.3, there exist $k + 1$ t-spanner uv-paths between u and v such that all paths begin with edge (u, z) and then are internally vertex-disjoint.

Summarizing, there are $k+1$ vertices $z_1, z_2, \ldots, z_{k+1}$ such that for each i, $1 \leq i \leq k+1$, (i) $(u, z_i) \in E'_{\mathcal{C}_u}$ and (ii) either $(z_i, v) \in E'$ and $|uz_i| + |z_iv| \leq t \cdot |uv|$, or G' contains $k + 1$ t-spanner uv-paths between u and v such that all paths begin with edge (u, z_i) and then are internally vertex-disjoint, and each edge on each path is shorter than (u, v). Therefore, one can apply Lemma 8.2.1 to conclude that G' contains at least $k+1$ internally vertex-disjoint t-spanner uv-paths, each path using only edges shorter than (u, v). This, however, contradicts to the fact that (u, v) is an edge of G', and hence, this completes the proof of the claim. □

In [120], it was shown that there is a constant $c > 0$ such that for any point $p \in \mathbb{R}^d$ and any angle θ, $0 < \theta < \pi$, there is always a collection $\mathcal{C}$ of $\mathcal{O}((c/\theta)^{d-1})$ cones in $\mathbb{R}^d$ having the apex at point p such that (i) $\bigcup_{C \in \mathcal{C}} C = \mathbb{R}^d$, and (ii) each cone $C \in \mathcal{C}$ has the angular diameter at most θ. One can incorporate this upper bound for the number of cones in $\mathcal{C}$ with Claims 8.3.1 and 8.3.4 to obtain the following lemma.

Lemma 8.3.5 *Let V be any point set in a Euclidean space $\mathbb{R}^d$. Let k be any non-negative integer and let t be any real number, $t > 1$. Then, the k-Greedy Algorithm returns a (k, t)-VFTS for V having maximum degree of $\mathcal{O}((c/\theta)^{d-1} k)$, where $\cos\theta - \sin\theta \geq \frac{1}{t}$. In particular, if the dimension d and t are constant, then the maximum degree is $\Theta(k)$.* □

8.3.2 Upper Bound for the Cost of Spanners Generated by the k-Greedy Algorithm

The most difficult and challenging part of the proof of Theorem 8.1.3 is the analysis of the total cost of the spanner generated by the k-Greedy Algorithm. In order to bound the cost of that spanner, one needs to first introduce the concept of "*leapfrog property*" [29], which yields a bound for the total cost of a set of edges in terms of the relative position of these edges in Euclidean space.

Definition 8.3.6 *(Leapfrog property [29]) Let* $t \geq 1$. *Let* $G = (V, E)$ *be a Euclidean graph. A set* $E^* \subseteq E$ *satisfies the* t-leapfrog property *if, for every* $s \geq 2$, *for every subset* $E^+ = \{(p_1, q_1), \ldots, (p_s, q_s)\} \subseteq E^*$ *it holds that*

$$t \cdot |p_1 q_1| < \sum_{i=2}^{s} |p_i q_i| + t \cdot \left(|q_s p_1| + \sum_{i=1}^{s-1} |q_i p_{i+1}| \right) .$$

The following result is shown in [29, 31] (see also [30, Theorem 3]).

Claim 8.3.7 *Let* t *be a constant greater than* 1. *Let* $G = (V, E)$ *be a Euclidean graph. If a set* $E^* \subseteq E$ *satisfies the* t*-leapfrog property then the total cost of the edges in* E^* *is* $\mathcal{O}(\mathsf{MST}_{E^*})$, *where* MST_{E^*} *is the cost of an* MST *connecting the endpoints of* E^*. *The constant implicit in the* $\mathcal{O}$*-notation depends on* t *and* d. □

The following result gives a tight upper bound for the cost of (k, t)-VFTSs generated by the k-Greedy Algorithm. This is one of the main results of this chapter; together with Lemma 8.3.5, it directly yields Theorem 8.1.3.

Lemma 8.3.8 *Let* $G = (V, E)$ *be a* (k, t)*-VFTS of a point set* V *generated by the* k*-Greedy Algorithm. Then, the cost of* G *is at most* $\mathcal{O}(k^2 \cdot W_{\mathrm{MST}})$. *The constant implicit in the* $\mathcal{O}$*-notation depends on* t *and* d.

Proof : The idea of the proof is to partition the edges of G into $\mathcal{O}(k^2)$ groups and then show that the cost of edges in each group is upper bounded by $\mathcal{O}(W_{\mathrm{MST}})$.

First partition E into disjoint sets $E_1, E_2, \ldots$ such that each E_i is a maximal set of edges and any two edges in the set have an angle at most θ where θ satisfies $t \geq$

$1/(\cos\theta - \sin\theta)$. By the discussion before (see also [120]), there are $\mathcal{O}((c/\theta)^{d-1})$ such disjoint sets $E_1, E_2, \ldots$.

Next, partition each E_i into sets $E_{i1}, E_{i2}, \ldots$ such that each E_{ij} satisfies the t-leapfrog property. For each edge $e = (v, u) \in E$, let $V_e \subseteq V$ be a minimum vertex set such that after removal of all vertices of V_e from G, there will be no t-spanner vu-paths consisting only of edges *shorter than* e. Since G is a (k, t)-VFTS generated by the k-Greedy Algorithm, besides e itself, there are at most k internally vertex-disjoint t-spanner vu-paths having all edges shorter than e. Therefore, by Menger's theorem one gets $|V_e| \leq k$.

Fix a set E_i. Let S_e be a subset of E_i containing edges that are shorter than e and that are incident to a vertex in V_e. Then, define the sets $E_{i1}, E_{i2}, \ldots$ by picking the edges $e \in E_i$ one by one and adding e to any set E_{ij} that does not contain any edge in S_e. One can claim that $\mathcal{O}(k^2)$ sets E_{ij} are sufficient in the construction and that each set E_{ij} satisfies the t-leapfrog property. Indeed, by Claim 8.3.4, each vertex is incident to at most $k+1$ edges in E_i. Therefore, since $|V_e| \leq k$, it must be true that $|S_e| \leq k(k+1)$. Thus, $k(k+1)+1$ sets E_{ij} are sufficient for each i.

Fix a set E_{ij}. Now prove that E_{ij} satisfies the t-leapfrog property. Let $E' = \{(u_0, v_0), (u_1, v_1), \ldots, (u_m, v_m)\}$ be any subset of E_{ij}. The goal is to show that

$$t \cdot |u_0 v_0| < \sum_{s=1}^{m} |u_s v_s| + t \cdot (|v_m u_0| + \sum_{s=0}^{m-1} |v_s u_{s+1}|) .$$

Observe first that this inequality is trivial if either $|u_0 v_0| \leq |v_m u_0|$ or $|u_0 v_0| \leq |v_s u_{s+1}|$ for certain s, $0 \leq s \leq m-1$. Furthermore, it is enough to consider the case when (u_0, v_0) is the longest edge in E'. Therefore, from now on, it will be assumed that (u_0, v_0) is the longest edge in $E' \cup \Big\{\{v_0, u_1\}, \{v_1, u_2\}, \ldots, \{v_{m-1}, u_m\}, \{v_m, u_0\}\Big\}$.

For convenience, let $e = (u_0, v_0)$. Let G_e be the graph obtained from G after removing all vertices in V_e together with their incident edges and then by removing all edges not shorter than (u_0, v_0). By the discussion above and by Menger's theorem, any $u_0 v_0$-path in G_e must have total length greater than $t \cdot |u_0 v_0|$. The goal is to show that

$\sum_{s=1}^{m} |u_s v_s| + t \cdot (|v_m u_0| + \sum_{s=0}^{m-1} |v_s u_{s+1}|)$ is at least the length of certain $u_0 v_0$-path in G_e.

For each $\{x, y\}$ in $\left\{\{v_0, u_1\}, \{v_1, u_2\}, \ldots, \{v_{m-1}, u_m\}, \{v_m, u_0\}\right\}$, either $(x, y) \in E$ or there are $k+1$ internally vertex-disjoint t-spanner xy-paths in G consisting only of edges shorter than (x, y), which is shorter than (u_0, v_0) by assumption. Furthermore, none of the points $u_0, u_1, \ldots, u_m, v_0, v_1, \ldots, v_m$ is in V_e. (Indeed, if, for example, $u_s \in V_e$, then by the assumption no edge incident to u_s should be included in E_{ij}, but this contradicts to the fact that $(u_s, v_s) \in E_{ij}$.)

Therefore, by Menger's theorem there must be a t-spanner xy-path in G_e. Hence, one can create a $u_0 v_0$-path π in G_e as follows: π starts with a t-spanner $v_0 u_1$-path, then uses edge (u_1, v_1), then uses a t-spanner $v_1 u_2$-path, then uses edge (u_2, v_2), ..., then uses edge (u_m, v_m), and finally terminates with a t-spanner $v_m u_0$-path. By the discussion above, π is a $u_0 v_0$-path in G_e. Moreover, one can derive that

$$\begin{aligned} \text{length}(\pi) &\leq t \cdot |v_0 u_1| + |u_1 v_1| + t \cdot |v_1 u_2| + |u_2 v_2| + \cdots |u_m v_m| + t \cdot |v_m u_0| \\ &= \sum_{s=1}^{m} |u_s v_s| + t \cdot (|v_m u_0| + \sum_{s=0}^{m-1} |v_s u_{s+1}|) \ . \end{aligned}$$

However, by the arguments above, G_e contains no t-spanner $u_0 v_0$-path. Therefore,

$$t \cdot |u_0 v_0| < \text{length}(\pi) = \sum_{s=1}^{m} |u_s v_s| + t \cdot (|v_m u_0| + \sum_{s=0}^{m-1} |v_s u_{s+1}|) \ ,$$

which implies the t-leapfrog property of set E_{ij}.

To summarize the discussion, one can partition the set of edges of G into $\mathcal{O}((c/\theta)^{d-1} \cdot k^2)$ sets of edges, each set satisfying the t-leapfrog property. Therefore, by Lemma 8.3.7, this concludes the proof. □

8.4 Efficient Construction of Fault Tolerant Spanners

In the previous section, it has been proved that the k-Greedy Algorithm generates (k, t)-VFTSs with low maximum degree and low total cost. The disadvantage of this algorithm, however, is that it is not known how to implement it efficiently. In this section an alternative

approach to construct fault tolerant spanners will be discussed. For convenience, in this section *directed* graphs and their spanners will be considered.

First the notion of *gap property* and *near parallel edges* is introduced, and then a new sufficient property for graphs to be (k, t)-VFTSs is prented. Then, one can use this characterization to design a simple algorithm that generates good (k, t)-VFTSs in polynomial time. Finally, algorithm is tuned to decrease the running time to $\mathcal{O}((n\,k\log^d n + n\,k^2\log k))$.

8.4.1 Basic Auxiliary Properties

The following are two important notions on directed edges that will be heavily used in the chapter.

Definition 8.4.1 *(Near parallel edges) Two directed edges* (p, q) *and* (x, y) *are called* α-near parallel[2] *if after translating vector* $\overrightarrow{xy}$ *such that* x *coincides with* p*, that is, to the vector* $\overrightarrow{pz}$ *with* $z = y - (x - p)$*, the angle between vectors* $\overrightarrow{pq}$ *and* $\overrightarrow{pz}$ *is upper bounded by* α.

Definition 8.4.2 *(Gap property) Let* $\omega \geq 0$*. Let* $G = (V, E)$ *be a directed Euclidean graph. A set of edges* $E^* \subseteq E$ *satisfies* ω-gap property *if for any two edges* $(v_1, u_1), (v_2, u_2) \in E^*$ *the distance between the heads and the tails of* (v_1, u_1) *and* (v_2, u_2) *is greater than* ω *times the length of the shorter of the two edges, that is,*

$$\min\{|v_1v_2|, |u_1u_2|\} > \omega \cdot \min\{|v_1u_1|, |v_2u_2|\} .$$

The following result is shown in [15]. (The constant implicit in the $\mathcal{O}$-notation does not depend on d.)

Claim 8.4.3 [15] (see also [6, Lemma 1]) *Let* $\omega > 0$*. Let* $G = (V, E)$ *be a directed Euclidean graph. If a set* $E^* \subseteq E$ *satisfies the* ω*-gap property then the total cost of the*

[2]*pairs of* α*-near parallel edges are called "similar directional" in [15].*

edges in E^* *is* $\mathcal{O}(\frac{1}{\omega} \cdot \log |E^*| \cdot MST_{E^*})$, *where* MST_{E^*} *is the cost of an* MST *connecting the endpoints of the edges in* E^*. □

Our next claim states, informally, that if two edges in a directed Euclidean graph are near parallel and close to each other, then one can form a spanner path between the endpoints of the "longer" edge by concatenating the "shorter" edge and the spanner paths between endpoints of the two edges. This claim is essentially proven in [6, Lemma 2].

Claim 8.4.4 *Let* t, α, ω *be real numbers such that* $0 < \alpha < \pi/4$, $0 \leq \omega \leq \frac{1}{2}(\cos\alpha - \sin\alpha - \frac{1}{t})$. *Let* G *be a directed Euclidean graph. Let* (u, v) *and* (x, z) *be two edges in* G *that are* α*-near parallel to each other. Suppose* $|uv| \leq |xz|/\cos\alpha$, *and* $|ux| \leq |vz|$. *If* $|ux| \leq \omega \cdot |uv|$, *then (i)* $|vz| < |xz|$ *and (ii)* $t \cdot |xu| + |uv| + t \cdot |vz| \leq t \cdot |xz|$. □

Remark 8.4.5 *It should be emphasized that Claim 8.4.4 holds not only when the length of edge* (u, v) *is less than or equal to the length of edge* (x, z), *but it also holds when* $|xz| < |uv| \leq |xz|/\cos\alpha$.

Furthermore, notice that Claim 8.4.4 still holds if one changes the assumption $|ux| \leq \omega \cdot |uv|$ *to* $|ux| \leq \omega \cdot \min\{|uv|, |xz|\}$ *since the latter assumption is stronger. Therefore, this claim is true when edges* (u, v) *and* (x, z) *do not satisfy the* ω*-gap property.*

These two observations are important for the efficient algorithm described in Section 8.4.3.

8.4.2 Sufficient Conditions for Being a k-Vertex Fault-Tolerant Spanner

In this section a new sufficient condition for a Euclidean graph to be a (k, t)-VFTS will be presented. Later it will be shown how this condition can be used to obtain an efficient algorithm for constructing (k, t)-VFTSs. The approach is motivated by a similar characterization of spanners developed by Arya and Smid in [6].

Lemma 8.4.6 *Let* t, α, ω *be real numbers such that* $0 < \alpha < \pi/4$ *and* $0 \leq \omega \leq \frac{1}{2}(\cos\alpha - \sin\alpha - \frac{1}{t})$. *Let* $G = (V, E)$ *be a directed Euclidean graph. Suppose that for any two vertices* u *and* v *in* V,

1. *either* $(u, v) \in E$ *or*
2. *there are* $k+1$ *edges* $\{(u_1, v_1), \ldots, (u_{k+1}, v_{k+1})\} \subseteq E$ *such that*
   - *for each* $1 \leq i \leq k+1$, $|u_i v_i| \leq |uv|/\cos\alpha$,
   - *all* u_i *and* v_i *that are neither* u *nor* v *are pairwise distinct,*
   - *for each* $1 \leq i \leq k+1$, *edge* (u_i, v_i) *is* α*-near parallel to* (u, v), *and*
   - *for each* $1 \leq i \leq k+1$, $\min\{|uu_i|, |v_i v|\} \leq \omega \cdot |u_i v_i|$.

Then G *is a* (k, t)*-VFTS for* V.

Notice that in Lemma 8.4.6, it is allowed that some of the u_i and v_i are equal to u or v, but otherwise, all other endpoints of the edges in $\{(u_1, v_1), \ldots, (u_{k+1}, v_{k+1})\}$ must be pairwise distinct.

Proof : In order to prove that G is a (k, t)-VFTS, one need to show that for any two vertices $u, v \in V$, either $(u, v) \in E$ or G contains $k+1$ disjoint t-spanner uv-paths, each uv-path having all edges shorter than $|uv|/\cos\alpha$. The proof is by induction on the rank of the distances between the pairs of points in V.

If $|uv|$ has the minimum distance among all pairs of vertices, then (u, v) must be an edge of E, and hence the claim holds for u, v. Next, one can proceed by induction. Consider a pair of vertices $u, v \in V$. By induction, for all ordered pairs of vertices $x, y \in V$ with $|xy| < |uv|$, either $(x, y) \in E$ or G contains $k+1$ disjoint t-spanner xy-paths, each having all edges shorter than $|xy|/\cos\alpha$. The goal is to prove that either $(u, v) \in E$ or G contains $k+1$ disjoint t-spanner uv-paths in G, each having all edges shorter than $|uv|/\cos\alpha$.

One only has to consider the case when $(u, v) \notin E$. Then, by the lemma's assumption, there exist $k+1$ edges $(u_1, v_1), (u_2, v_2), \ldots, (u_{k+1}, v_{k+1})$ in E such that

(i) for each $1 \leq i \leq k+1$, edge (u_i, v_i) is shorter than $|uv|/\cos\alpha$,

(ii) all u_i and v_i that are neither u nor v are pairwise distinct,

(iii) for each $1 \leq i \leq k+1$, edges (u_i, v_i) and (u, v) are α-near parallel, and

(iv) for each $1 \leq i \leq k+1$, $\min\{|uu_i|, |v_iv|\} \leq \omega \cdot |u_iv_i|$.

Pick any edge (u_i, v_i). Assume without loss of generality that $|uu_i| = \min\{|uu_i|, |v_iv|\}$. Since, (u_i, v_i) and (u, v) are α-near parallel by (iii), $|u_iv_i| \leq |uv|/\cos\alpha$ by (i), and $|uu_i| \leq \omega \cdot |u_iv_i|$ by (iv), Claim 8.4.4 (i) implies that $|v_iv| < |uv|$. Hence, by induction, either $(v_i, v) \in E$ or there are $k+1$ disjoint t-spanner v_iv-paths in G, all using only edges shorter than $|v_iv|/\cos\alpha$. Similarly, since $|uu_i| \leq |v_iv| < |uv|$, either $(u, u_i) \in E$ or G contains $k+1$ disjoint t-spanner uu_i-paths that use only edges shorter than $|uu_i|/\cos\alpha$. Furthermore, by Claim 8.4.4 (ii), each uv-path consisting of a t-spanner uu_i-path (or edge (u, u_i)), edge (u_i, v_i), and a t-spanner v_iv-path (or edge (v_i, v)) is a t-spanner uv-path.

So far it has been proven that there are $k+1$ edges $(u_1, v_1), (u_2, v_2), \ldots, (u_{k+1}, v_{k+1})$ such that for each i, $1 \leq i \leq k+1$, (i) either $(u, u_i) \in E$ or G contains $k+1$ disjoint t-spanner uu_i-paths that use only edges shorter than $|uu_i|/\cos\alpha$, and (ii) either $(v_i, v) \in E$ or there are $k+1$ disjoint t-spanner v_iv-paths in G that use only edges shorter than $|vv_i|/\cos\alpha$. Now, the claim follows directly from the Menger's theorem (for more details, see Lemma 8.2.2). □

Essentially identical arguments can be used to prove the following characterization for edge fault-tolerant spanners.

Lemma 8.4.7 *Let* t, α, ω *be real numbers such that* $0 < \alpha < \pi/4$, $0 \leq \omega \leq \frac{1}{2}(\cos\alpha - \sin\alpha - \frac{1}{t})$. *Let* $G = (V, E)$ *be a Euclidean graph. Suppose that for any two vertices* u *and* v *of* V, *either* $(u, v) \in E$ *or there are* $k+1$ *edges* $\{(u_1, v_1), \ldots, (u_{k+1}, v_{k+1})\} \subseteq E$ *such that*

- *for* $1 \leq i \leq k+1$, $|u_iv_i| \leq |uv|/\cos\alpha$,

- *each edge* (u_i, v_i) *is* α*-near parallel to* (u, v)*, and*
- *for each* $1 \leq i \leq k+1$, $\min\{|uu_i|, |v_iv|\} \leq \omega \cdot |u_iv_i|$.

Then G *is a* (k, t)*-EFTS for* V. □

k-Gap-Greedy Algorithm

Input: A directed complete Euclidean graph $G = (V, E)$, α and ω, and integer k such that $k \geq 0$, $0 < \alpha < \pi/4$, and $0 < \omega < \frac{1}{2}(\cos\alpha - \sin\alpha)$

Output: A (k, t)-VFTS $G' = (V, E')$ for V, where $t = 1/(\cos\alpha - \sin\alpha - 2\omega)$.

$E' = \emptyset$
for each edge $(u, v) \in E$ taken in nondecreasing order by length **do**
  Let E^* be a maximal (in the sense of inclusion) subset of E' such that:
  1. all edges in E' beginning at u or ending at v are contained in E^*,
  2. for every $(x, y) \in E^*$, (x, y) is α-near parallel to (u, v),
  3. for every $(x, y) \in E^*$, $\min\{|ux|, |yv|\} \leq \omega \cdot |xy|$,
  4. for any pair of distinct edges $(x, y), (z, w) \in E^*$, if $x \neq u$ and $y \neq v$, then $x \neq z$ and $y \neq w$.

  if $|E^*| < k + 1$ **then** $E' = E' \cup \{(u, v)\}$
output $G' = (V, E')$

Figure 8.2 The k-Gap-Greedy Algorithm

The characterization of (k, t)-VFTSs in Lemma 8.4.6 almost immediately implies a simple polynomial-time algorithm for constructing such spanners, which is called k-Gap-Greedy Algorithm and describe in a form of a meta-algorithm in Figure 8.2.

The following central lemma describes main properties of the k-Gap-Greedy Algorithm.

Lemma 8.4.8 *The* k*-Gap-Greedy Algorithm outputs a* directed (k, t)*-VFTS for* $t = 1/(\cos\alpha - \sin\alpha - 2\omega)$ *with maximum in-degree and out-degree* $\mathcal{O}(k)$ *and whose total cost is* $\mathcal{O}(\frac{1}{\omega} \cdot k^2 \log n)$ *times the cost of an* MST *for* V. *The constant implicit in the* $\mathcal{O}$*-notation depends on* t *and* d.

Proof : First it will be prove that G' is a (k,t)-VFTS by showing that for any ordered pair of vertices u, v, one of the two conditions of Lemma 8.4.6 is satisfied. If $(u,v) \in E'$, then the first condition is obviously true. Otherwise, $(u,v) \notin E'$ and consider the iteration of the algorithm when (u,v) is chosen. The only reason that (u,v) is not added to E' is that $|E^*| \geq k+1$. But this implies that the second condition of Lemma 8.4.6 is satisfied for (u,v). Therefore, G' is a (k,t)-VFTS by Lemma 8.4.6.

Next, it will be prove that the maximum out-degree and the maximum in-degree of each vertex is $\mathcal{O}(k)$. Let u be any vertex. First prove the out-degree of u is $\mathcal{O}(k)$. Let C_u be any cone in $\mathbb{R}^d$ with the apex at u and the angular diameter at most α. Let E'_{C_u} be the set of edges in G' beginning at u that are contained in the cone C_u. It will be proved that $|E'_{C_u}| \leq k+1$, which immediately implies that the out-degree of u is $\mathcal{O}(k)$. Now analyze the behavior of the algorithm at the moment when $|E'_{C_u}| = k+1$ and the algorithm considers a new edge (u,v) with $v \in C_u$. Observe that in that case one will have $E'_{C_u} \subseteq E^*$, and hence, E^* will be of size at least $k+1$. Therefore, the algorithm will not add the edge (u,v) to the spanner. This implies that $|E'_{C_u}| \leq k+1$, and hence, the out-degree of u is $\mathcal{O}(k)$. One can use essentially identical arguments to prove the in-degree of u is $\mathcal{O}(k)$. (Similar arguments show that u is the head/tail of at most $k+1$ pairwise α-near parallel edges.)

Finally, one needs to prove that G' has small cost. One can proceed similarly as in the proof of Lemma 8.3.8 and first partition E' into disjoint sets $E'_1, E'_2, \ldots$ such that each E'_i is a maximal set of edges which are α-near parallel. By the discussion in the proof of Lemma 8.3.8, there are $\mathcal{O}((c/\alpha)^{d-1})$ such disjoint sets $E'_1, E'_2, \ldots$. Let us fix one set E'_i and divide the edges in E'_i into a minimal number of groups such that the edges in the same group satisfy the ω-gap property. It will be proved that $\mathcal{O}(k^2)$ groups are sufficient. It is enough to show that for any edge $e \in E'_i$ there are at most $\mathcal{O}(k^2)$ edges $e' \in E'_i$ shorter than e such that $\{e, e'\}$ does not satisfy the ω-gap property.

Fix an edge $(u,v) \in E'_i$. Let $\overline{V}^u_{(u,v)} = \{p \in V : \exists (p,q) \in E'_i, |pq| \leq |uv|, 0 \leq |up| \leq \omega \cdot |pq|\}$ and let $\overline{E}^u_{(u,v)}$ be the set of edges in E'_i that (i) begin at vertices in $\overline{V}^u_{(u,v)}$, and (ii) are shorter than $|uv|$. Note that since $\overline{E}^u_{(u,v)} \subseteq E'_i$, all edges in $\overline{E}^u_{(u,v)}$ are pairwise α-near parallel. One can show that $|\overline{E}^u_{(u,v)}| < 2(k+1)^2$ by contradiction. Suppose that $|\overline{E}^u_{(u,v)}| \geq 2(k+1)^2$ and consider the iteration in which the edge (u,v) is picked by the algorithm. Let E^* be the set taken by the k-Gap-Greedy Algorithm when the edge (u,v) is considered. It will be proved that $|E^*| \geq k+1$, which contradicts to the assumption that $(u,v) \in E'$. Let $\mathcal{E}$ be the set of all edges in E' shorter than (u,v) such that for every edge $e \in \mathcal{E}$, (i) e is α-near parallel to (u,v) and (ii) $\{e,(u,v)\}$ does not satisfy the ω-gap property. Note that because the algorithm picks edges in nondecreasing order, the condition 3 of the algorithm implies that none of the edges in E^* satisfies ω-gap property with (u,v). Thus, E^* can be defined as the sum of certain *maximal matching* in $\mathcal{E}$ and the set of all edges in $\mathcal{E}$ that either begin at u or end at v. Therefore, to prove that $|E^*| \geq k+1$ it is sufficient to show that every maximal matching in $\mathcal{E}$ contains at least $k+1$ edges. Furthermore, because of the well known relation between the cardinality of the maximum matching and the minimum cardinality of a maximal matching, it is enough to show that the maximum matching in $\mathcal{E}$ contains at least $2(k+1)$ edges. One can prove this property by considering the edges in $\overline{E}^u_{(u,v)}$. First observe that by the definition $\overline{E}^u_{(u,v)} \subseteq \mathcal{E}$. Therefore, one only must show that $\overline{E}^u_{(u,v)}$ has a matching of size at least $2(k+1)$. Note that there are at most $k+1$ edges in E'_i starting at each vertex as proved above. Since it is assumed that $|\overline{E}^u_{(u,v)}| \geq 2(k+1)^2$, one can conclude that set $\overline{E}^u_{(u,v)}$ must contain at least $2(k+1)$ disjoint edges. Therefore, there is a matching of size at least $2(k+1)$ in $\mathcal{E}$. But this leads to a contradiction, and hence one proved that $|\overline{E}^u_{(u,v)}| < 2(k+1)^2$.

Symmetrically, one can define $\overline{E}^v_{(u,v)}$, and prove that $|\overline{E}^v_{(u,v)}| < 2(k+1)^2$. Therefore, $|\overline{E}^u_{(u,v)} \cup \overline{E}^v_{(u,v)}| < 4(k+1)^2$. Hence one can conclude that one needs at most $4(k+1)^2$ groups of edges from E'_i such that the edges in each group satisfy ω-gap property. Thus, it is proved that one can partition the edges in E' into $\mathcal{O}((c/\alpha)^{d-1}k^2)$ groups such that the

edges in each group satisfy ω-gap property. To conclude the proof one can apply Claim 8.4.3 to obtain an upper bound for the total cost of E' to be $\mathcal{O}((c/\alpha)^{d-1} k^2 \frac{1}{\omega} \log n)$ times the cost of minimum spanning tree of V. □

8.4.3 Efficient Construction of k Fault-Tolerant Spanner

It is easy to implement the k-Gap-Greedy Algorithm in polynomial time, however, direct implementations lead to the running time of $\Omega(n^2)$. This section discusses how that algorithm can be modified to achieve the running time of $\mathcal{O}(n k \log^d n + n k^2 \log k)$ while still returning a spanner having the parameters promised in Lemma 8.4.8. The new approach is similar to the one developed by Arya and Smid [6] to construct spanners with bounded degree and low cost.

The main idea behind the new approach is not to consider all the $\Theta(n^2)$ edges but to have an efficient procedure that will "forbid" certain edges without considering them explicitly during the run of the algorithm. Since every vertex is of degree $\mathcal{O}(k)$, the goal is to ensure that only $\mathcal{O}(k)$ edges incident to any vertex are considered by the algorithm. The rules of inserting an edge in the k-Gap-Greedy Algorithm will be relaxed in order to obtain a more efficient implementation. On one hand, the goal is to maintain the properties of the output spanner required by Lemma 8.4.6 and on the other hand, the aim is to output a spanner having properties described in Lemma 8.4.8. Below the main idea is described behind that relaxation, and the algorithm itself is described in Section 8.4.3.

Main idea behind the modified algorithm A collection of cones of angular diameter α (see, Section 8.3.1) will be used to test whether two edges are α-near parallel to each other. That is, it is assume that there is a collection $\mathcal{C}$ of cones with apexes at the origin and the angular diameter α, so that the cones in $\mathcal{C}$ cover $\mathbb{R}^d$. Then, an edge (u, v) is said to be in a cone C if the vector $v - u \in C$. Using this notion, each time a pair of points u, v with $(u, v) \in C$ are consider, one only needs to look at those edges $(x, y) \in E'$ that are in C.

Since if (u, v) and (x, y) are in the same cone, then they are α-near parallel to each other, this notion allows one to relax testing of α-near parallel edges.

Once the collection $\mathcal{C}$ of cones is fixed, the cones from $\mathcal{C}$ will be considered separately. The spanner is built by defining the edge set E' which will be the union of the edge sets constructed for each cone separately. Let E'_C denote all those edges of E' inside the cone $C \in \mathcal{C}$.

In the k-Gap-Greedy Algorithm, one analyzes the edges in the order of their increasing lengths. This is more complicated (in the sense of efficiency) if one wants to consider the edges being in different cones separately. Therefore, following the approach from [103] (see also [6]), one can approximate the distance between the points in a cone. For each cone C, fix a ray ℓ_C that is incident to the apex of C and that is included in the cone C. Then, the idea of approximating the distance between pairs of points is to use the distance between the projections of the points on the ray ℓ_C. To make this notion more formal, a few definitions are needed. For any cone $C \in \mathcal{C}$ and any point $x \in \mathbb{R}^d$, let C_x be the translation of C so that the apex of C_x is at x, that is, $C_x = \{y \in \mathbb{R}^d : y - x \in C\}$. Similarly, for the ray ℓ_C, let $\ell_{C,x} = \{y \in \mathbb{R}^d : y - x \in \ell_C\}$. Then, one can approximate the distance between the points in the cone C using the following function dist_C:

$$\mathrm{dist}_C(x, y) = \begin{cases} |x\, R_y| & \text{if } y \in C_x \text{ and } R_y \text{ is the orthogonal projection of } y \text{ onto } \ell_{C,x} \\ \infty & \text{if } y \notin C_x \ . \end{cases}$$

The main reason of using this approximation of the distances between pairs of points inside a cone is that it can be efficiently maintained by a dynamic algorithm (see also [6, 103]). Furthermore, it is easy to show that if $\mathrm{dist}_C(x, y) \leq \mathrm{dist}_C(u, v)$, then $|xy| \leq |uv|/\cos\alpha$. Therefore, by Claim 8.4.4 and Lemma 8.4.6, the algorithm remains correct (in the sense that it satisfies the properties from Lemma 8.4.6) even if edge (x, y) is considered before edge (u, v).

Summarizing, the algorithm is going to consider the edges inside each cone in non-decreasing order of their dist_C length.

Let (x, y) be an edge in a cone $C \in \mathcal{C}$ that is to be taken by the algorithm. Observe that if there are already at least $k+1$ edges in C with the head at x then the edge (x, y) will not be taken by the algorithm. Therefore, whenever a vertex x already has $k+1$ outgoing edges in C, then no further edge starting at x in C will be considered. Symmetrically, one only needs to consider at most $k+1$ incoming edges for any point y.

The other reason for rejecting edge (x, y) is that there are many disjoint edges in C which are shorter than (x, y) with respect to dist_C length and are very close to either x or y. Let $x \in V$ be any input point. Two special data structures will be used to maintain a maximal set of disjoint edges in C that have starting or ending points close to x, respectively. These two data structures will be modified not when an edge in C incident to x is considered, but instead, they will be updated each time an edge close to x is inserted into E'_C. To be more precise, at each moment of the algorithm, a set $F_C(x)$ is maintained for each $x \in V$ which contains a set of disjoint edges in E'_C such that for any $(u, v) \in F_C(x)$, $|xu| \leq \omega \cdot |uv|$. Similarly, $H_C(x)$ is maintained to contain a set of disjoint edges in E'_C such that for any $(u, v) \in H_C(x)$, $|vx| \leq \omega \cdot |uv|$.

Now, suppose an edge (u, v) is inserted to E'_C. Let $N^u_{(u,v)}$ and $N^v_{(u,v)}$ be the set of points that are at distance at most $\omega \cdot |uv|$ from u and v, respectively. By the definitions of $F_C(x)$ and $H_C(x)$, one needs to update those points in $N^u_{(u,v)}$ and $N^v_{(u,v)}$ that are "affected" by the edge (u, v). That is, for every point $x \in N^u_{(u,v)}$, if $F_C(x)$ contains no edge incident to either u or v, then one updates $F_C(x)$ by inserting (u, v) to it. Similarly, one updates $H_C(y)$ for every point $y \in N^v_{(u,v)}$.

Notice that the operation of updating the sets $F_C(x)$ and $H_C(x)$ may be very expensive, because both the vertex sets $N^u_{(u,v)}$, $N^v_{(u,v)}$ and the edge sets $F_C(x)$, $H_C(x)$ may be very large. Therefore, one can relax the definition of the sets $F_C(x)$ and $H_C(x)$ by observing that once $F_C(x)$ or $H_C(x)$ is of size larger than or equal to $k+1$, by Lemma 8.4.6, no new edge in C beginning or ending at x, respectively, will have to be added to E'_C. Therefore, two sets of points from V are maintained: V^1_C containing all points that still can be the heads of new

edges in E'_C and V^2_C containing points that still can be the tails of new edges in E'_C. Each time the size of certain $F_C(x)$ is greater than or equal to $k+1$, x is deleted from V^1_C; no further edges in C that begin at x will be considered, $F_C(x)$ will not be updated anymore, nor will x be considered in any further sets $N^u_{(u,v)}$. Similarly, if $H_C(x) \geq k+1$, x is deleted from V^2_C, no further edges in C ending at x will be considered, $H_C(x)$ will not be updated and x will not be considered in any further sets $N^v_{(u,v)}$. In this way, since both $F_C(x)$ and $H_C(x)$ have a size between 0 and $k+1$, the total number of operations for updating the sets $F_C(x)$ and $H_C(x)$ in the entire algorithm (for a given cone C) will be $\mathcal{O}(kn)$.

Observe that once a vertex x has been reported $2(k+1)^2$ times in the sets $N^u_{(u,v)}$, then there are $2(k+1)^2$ edges (u,v) such that $|ux| \leq \omega \cdot |uv|$. Since each vertex is the head of at most $k+1$ edges in C, there must be $2(k+1)$ disjoint edges among all these edges. Therefore, in this case, the size of $F_C(x)$ must be greater than or equal to $k+1$ (see Lemma 8.4.8 for a more detailed discussion on similar arguments). Hence, at this moment x will be deleted and will not be reported in $N^u_{(u,v)}$ for any $u \in V$ any more. For the same reason x will be reported in $N^v_{(u,v)}$ at most $2(k+1)^2$ times. This allows one to conclude that the total size of $N^u_{(u,v)}$ and $N^v_{(u,v)}$ for all vertices u and v in the entire algorithm (for a given cone C) is $4n(k+1)^2$.

To define sets $N^u_{(u,v)}$ one needs to find all points that are at the distance at most $\omega \cdot |uv|$ from u. However, since it is difficult to maintain dynamic data structures to find $N^u_{(u,v)}$ in the Euclidean metric, in the definition of $N^u_{(u,v)}$ one assumes the distances between points according to the L_∞ metric. That is, $N^u_{(u,v)}$ is *redefined* to be the set of all points x with $|ux|_\infty \leq \frac{\omega}{\sqrt{d}}|uv|$. (Here, one needs to output only the points that have not been deleted, that is, those x for which $|F_C(x)| < k+1$.) Since $|ux|_\infty \leq \frac{\omega}{\sqrt{d}}|uv|$ implies that $|ux| \leq \omega \cdot |uv|$, the new definition of $N^u_{(u,v)}$ gives a subset of the set $N^u_{(u,v)}$ defined above. Set $N^v_{(u,v)}$ is redefined symmetrically.

Improved k-Gap-Greedy Algorithm In the previous subsection the main ideas behind the modifications of the k-Gap-Greedy Algorithm are presented. These ideas allow one to design a new algorithm that finds good fault-tolerant spanners efficiently. A more formal description of the algorithm is presented below, on page 185. A formal proof of its correctness will be given first and then an efficient implementation of the algorithm will be discussed.

Properties of the Improved k-Gap-Greedy Algorithm. This section is devoted to a formal proof of the following lemma.

Lemma 8.4.9 *Let* V *be a set of* n *points in* $\mathbb{R}^d$. *Let* α, ω *be real numbers such that* $0 < \alpha < \pi/4$ *and* $0 < \omega < \frac{1}{2}(\cos\alpha - \sin\alpha)$. *Let* $t = 1/(\cos\alpha - \sin\alpha - 2\omega)$. *There is a constant* c *such that the Improved* k*-Gap-Greedy Algorithm outputs a* directed (k,t)*-VFTS with maximum in-degree and out-degree* $\mathcal{O}(k)$ *and whose total cost is* $\mathcal{O}((c/\alpha)^{d-1}\frac{\sqrt{d}}{\omega}k^2\log n)$ *times the cost of a* MST *for* V.

Proof : One needs to prove that the output of Improved k-Gap-Greedy Algorithm has the same properties as the output of k-Gap-Greedy Algorithm after some conditions are relaxed. The proof is similar to that of Lemma 8.4.8.

Let G' be the output of the Improved k-Gap-Greedy Algorithm. First, G' is proved to be a (k,t)-VFTS by showing that G' satisfy the conditions of Lemma 8.4.6. Let u, v be any ordered pair of vertices, and $v - u \in C$ for some $C \in \mathcal{C}$. If $(u,v) \in E'_C$, then the first condition is obviously true. Otherwise, $(u,v) \notin E'_C$ and there are two possible reasons why (u,v) was not inserted to E'_C.

i. (u,v) is considered explicitly in the "while" loop but is not inserted to E'_C.

   In this case, $dist_C(u,v)$ is minimum at the beginning of certain iteration. Since (u,v) is not added to E'_C, there must exist either $k+1$ edges with heads at u or $k+1$ edges with tails at u which are already inserted to E'_C. These edges have length at most

$|uv|/\cos\alpha$, are α-near parallel to (u, v), and they have one common endpoint with (u, v). Therefore, the second condition of Lemma 8.4.6 is satisfied.

ii. (u, v) is deleted when some other pair (x, y) is picked and added to E'_C in the "while" loop.

In this case, the only reason that (u, v) is deleted from E_C is either $|F_C(u)| = k+1$ or $|H_C(v)| = k+1$ after (x, y) is inserted to $F_C(u)$ or $H_C(u)$ respectively. Suppose that $|F_C(u)| = k+1$ and consider the edges in $F_C(u) = \{(x_1, y_1), \ldots, (x_{k+1}, y_{k+1})\}$. By the Improved k-Gap-Greedy Algorithm, edge (x_i, y_i), $1 \leq i \leq k+1$, can be inserted to $F_C(u)$ only if: (a) $|ux_i|_\infty \leq \frac{\omega}{\sqrt{d}}|x_iy_i|$, which implies that $|ux_i| \leq \omega \cdot |x_iy_i|$, and (b) (x_i, y_i) is disjoint with all other edges in $F_C(u)$. Furthermore, since (x_i, y_i) is considered before (u, v) by the algorithm, it must be true that $\mathrm{dist}_C(x_i, y_i) \leq \mathrm{dist}_C(u, v)$, which means $|x_iy_i| \leq |uv|/\cos\alpha$. Therefore, one can combine with (a) and (b), to see that the second condition of Lemma 8.4.6 is satisfied.

To summarize, it has been proved for each pair (x, y), either (x, y) is an edge in G' or there are $k+1$ edges in G' satisfying the second condition of Lemma 8.4.6. Therefore, G' is a (k, t)-VFTS.

Next, it will be proved that the maximum in-degree and out-degree of each vertex is $\mathcal{O}(k)$. The proof is basically the same as the proof of Lemma 8.4.8. Let u be any vertex. According to the Improved k-Gap-Greedy Algorithm it is easy to see at most $k+1$ edges beginning at u can be added to the E'_C for any given cone $C \in \mathcal{C}$. Therefore the out-degree of u in G' is $\mathcal{O}(k)$. Similarly, one can show that the in-degree is $\mathcal{O}(k)$.

Finally, one needs to prove that G' has small cost. Fix a cone C and divide the edges in E'_C into a minimal number of groups such that the edges in the same group satisfy the $\frac{\omega}{\sqrt{d}}$-gap property. It will be proved that $\mathcal{O}(k^2)$ groups suffice. To prove this claim it is enough to show that for any edge $e \in E'_C$ there are at most $\mathcal{O}(k^2)$ edges $e' \in E'_C$ that

are "shorter" than e in terms of the $dist_C$ length and such that $\{e, e'\}$ does not satisfy the $\frac{\omega}{\sqrt{d}}$-gap property.

Fix an edge $(u, v) \in E'_C$. Let $E^u_{(u,v)} = \{(x, y) \in E'_C : dist_C(x, y) < dist_C(u, v)$ and $|ux| \leq \frac{\omega}{\sqrt{d}} \cdot |xy|\}$ and $E^v_{(u,v)} = \{(x, y) \in E'_C : dist_C(x, y) < dist_C(u, v)$ and $|vy| \leq \frac{\omega}{\sqrt{d}} \cdot |xy|\}$. Observe that if an edge (x, y) is "shorter" than (u, v) with respect to the $dist_C$ length and it fails to satisfy the $\frac{\omega}{\sqrt{d}}$-gap property with (u, v), then $(x, y) \in E^u_{(u,v)} \cup E^v_{(u,v)}$. Therefore, to prove the claim it is sufficient to show that $|E^u_{(u,v)} \cup E^v_{(u,v)}| = \mathcal{O}(k^2)$.

First it will be proved that $|E^u_{(u,v)}| = \mathcal{O}(k^2)$. Let $\mathcal{E}^u_{(u,v)} = \{(x, y) \in E'_C : dist_C(x, y) < dist_C(u, v)$ and $|ux|_\infty \leq \frac{\omega}{\sqrt{d}} \cdot |xy|\}$. Note that since $|ux|_\infty \leq |ux|$, $E^u_{(u,v)} \subseteq \mathcal{E}^u_{(u,v)}$. In the following it will be shown that $|\mathcal{E}^u_{(u,v)}| = \mathcal{O}(k^2)$ which directly implies that $|E^u_{(u,v)}| = \mathcal{O}(k^2)$.

There are two observations. First because all edges in $\mathcal{E}^u_{(u,v)}$ are "shorter" than (u, v) with respect to $dist_C$, all these edges are picked and inserted to E'_C by the algorithm before (u, v). Secondly, each time after one of these edges (x, y) is inserted to E'_C, the algorithm inserts (x, y) to $F_C(u)$ if (x, y) is disjoint with edges already in $F_C(u)$. In other words, at the end of each iteration of the Improved k-Gap-Greedy Algorithm $F_C(u)$ keeps a maximal matching of the edges in $\mathcal{E}^u_{(u,v)}$ that have been inserted to E'_C so far.

Now one can prove by contradiction that $|\mathcal{E}^u_{(u,v)}| < 2(k+1)^2$. Suppose that $|\mathcal{E}^u_{(u,v)}| \geq 2(k+1)^2$. It has already been shown that each vertex has out-degree (in-degree) in E'_C of at most $k+1$. Therefore, the maximum matching of $\mathcal{E}^u_{(u,v)}$ must contain at least $2(k+1)$ edges and thus any maximal matching of $E^u_{(u,v)}$ must have at least $(k+1)$ edges. Since $F_C(u)$ is a maximal matching of $\mathcal{E}^u_{(u,v)}$, $|F_C(u)| \geq k+1$. Then, there must exist an edge $(x, y) \in \mathcal{E}^u_{(u,v)}$ such that during the iteration when (x, y) is inserted to E'_C, (x, y) is also inserted to $F_C(u)$, and $|F_C(u)|$ becomes $k+1$. However, the Improved k-Gap-Greedy Algorithm is designed such that in that moment u must have been deleted from V^1_C and all edges with head at u including (u, v) are also deleted from E_C and will not be inserted to E'_C. This contradicts to the fact that $(u, v) \in E'_C$. Thus, it is proved that $|E^u_{(u,v)}| \leq |\mathcal{E}^u_{(u,v)}| < 2(k+1)^2$.

In a similar way, one can prove that $|E^v_{(u,v)}| < 2\,(k+1)^2$. Therefore, $|E^u_{(u,v)} \cup E^v_{(u,v)}| < 4\,(k+1)^2$. Hence, one can partition the edges from E'_i into at most $4\,(k+1)^2$ groups such that the edges in each group satisfy the $\frac{\omega}{\sqrt{d}}$-gap property. Thus, the edges in E' can be partitioned into $\mathcal{O}((c/\alpha)^{d-1}\,k^2)$ groups such that the edges in each group satisfy $\frac{\omega}{\sqrt{d}}$-gap property. To conclude the proof one can apply Claim 8.4.3 to obtain an upper bound for the total cost of E', which is $\mathcal{O}((c/\alpha)^{d-1}\,k^2\,\frac{\sqrt{d}}{\omega}\,\log n)$ times the cost of MST of V. □

Details of the implementation of the Improved k-Gap-Greedy Algorithm. The description of the Improved k-Gap-Greedy Algorithm is on a high level and now it will be disscussed how one can implement that algorithm efficiently. In order to obtain an efficient implementations one must provide efficient data structures that allow one to query for (i) an edge $(u,v) \in E_C$ with minimum dist_C, (ii) the number of edges in E'_C beginning or ending at u, (iii) reporting all points in $N^u_{(u,v)}$ and $N^v_{(u,v)}$, and (iv) for verifying if an edge (u,v) is disjoint to all edges in $F_C(x)$ or $H_C(x)$.

It is easy to see that one can maintain the data structure (ii) reporting the number of edges in E'_C beginning or ending at a given vertex with constant query time and update time. Similarly, one can easily maintain the data structure (iv) verifying if an edge (u,v) is disjoint to all edges in $F_C(x)$ or $H_C(x)$ with $\mathcal{O}(\log k)$ query time and update time. (The bound for the query time follows immediately from the fact that $F_C(x)$ or $H_C(x)$ contains $\mathcal{O}(k)$ edges and one can use a balanced binary search tree to store the endpoints of these edges.)

Arya and Smid [6] gave an efficient data structure supporting queries (i) for an edge in E_C with minimum dist_C. As it is demonstrated (and discussed in details) in [6], the total running time needed to perform all these operations is in our case $\mathcal{O}(n\,k\,\log^d n)$. (This bound follows from the fact that one can use query (i) $\mathcal{O}(k\,n)$ times, and as shown in [6], efficient data structure can be built in time $\mathcal{O}(n\,\log^d n)$ that has constant query time and $\mathcal{O}(\log^d n)$ amortized update time per edge.)

To efficiently report all points in $N^u_{(u,v)}$ and $N^v_{(u,v)}$ one can use dynamic data structure for orthogonal range queries (see [88]). The algorithm only deletes points and therefore the amortized deletion time is $\mathcal{O}(\log^{d-1} n)$ and the query time is $\mathcal{O}(\log^{d-1} n)$ time plus the number of reported points [88]. Since each point is deleted at most once, the total deletion time is $\mathcal{O}(n \log^{d-1} n)$. A query operations is performed on the dynamic data structure each time after an edge is inserted, so the total time is $\mathcal{O}(n\, k \log^{d-1} n)$ for $\mathcal{O}(nk)$ edges. Each vertex is reported at most $\mathcal{O}(k^2)$ times, when reported, one needs to verify whether $F_C(x)$ or $H_C(x)$ can be enhanced by adding an edge, the verification and update takes time $\mathcal{O}(\log k)$, thus the total time for reporting, verification, and update is $\mathcal{O}(n\, k^2 \log k)$.

In summary, one can conclude the discussion in this section with the following lemma.

Lemma 8.4.10 *Let V be a set of n points in $\mathbb{R}^d$. Let α, ω be real numbers such that $0 < \alpha < \pi/4$ and $0 < \omega < \frac{1}{2}(\cos\alpha - \sin\alpha)$. Let $t = 1/(\cos\alpha - \sin\alpha - 2\omega)$. There is a constant c such that the Improved k-Gap-Greedy Algorithm can be implemented to run in time $\mathcal{O}((c/\alpha)^{d-1}(n\, k \log^d n + n\, k^2 \log k))$.* □

One can conclude the discussion in this section with the following main theorem that follows immediately from Lemmas 8.4.9 and 8.4.10.

Theorem 8.4.11 *Let α, ω be real numbers such that $0 < \alpha < \pi/4$, $0 < \omega < \frac{1}{2}(\cos\alpha - \sin\alpha)$. Let V be a set of n points in $\mathbb{R}^d$. Let $t = 1/(\cos\alpha - \sin\alpha - 2\omega)$. There is a constant c such that in $\mathcal{O}((c/\alpha)^{d-1}(n\, k \log^d n + n\, k^2 \log k))$ time the Improved k-Gap-Greedy Algorithm computes a directed (k,t)-VFTS having the maximum degree of $\mathcal{O}((c/\alpha)^{d-1} k)$ and the total cost of $\mathcal{O}((c/\alpha)^{d-1} k^2 (\sqrt{d}/\omega) \log n\, W_{MST})$.* □

To conclude this section, notice that Theorem 8.4.11 implies directly Theorem 8.1.4.

Improved k-Gap-Greedy Algorithm

Input: Set V of n points in $\mathbb{R}^d$, an integer $k \geq 0$, and positive α, ω with $\alpha < \pi/4$ and $\omega < \frac{1}{2}(\cos\alpha - \sin\alpha)$

Output: A (k,t)-VFTS $G' = (V, E')$ for V, where $t = 1/(\cos\alpha - \sin\alpha - 2\omega)$.

let $\mathcal{C}$ be any collection of cones with angular diameter α that cover $\mathbb{R}^d$

for each cone $C \in \mathcal{C}$ **do**

$E_C = \{(u,v) : \text{dist}_C(u,v) < \infty\}$ {E_C contains all edges (u,v) such that $v - u \in C$}

$E'_C = \emptyset$ {E'_C collects the edges in E_C that are to be included in the spanner}

for each $x \in V$ **do** $F_C(x) = \emptyset$

{$F_C(x)$ contains disjoint edges in E'_C that begin at points near to x}

for each $x \in V$ **do** $H_C(x) = \emptyset$

{$H_C(x)$ contains disjoint edges in E'_C that end at points near to x}

$V^1_C = V$ {V^1_C contains all points u in V such that $|F_C(x)| < k+1$ }

$V^2_C = V$ {V^2_C contains all points v in V such that $|H_C(x)| < k+1$ }

while $E_C \neq \emptyset$ **do**

let $(u,v) \in E_C$ be such that $\text{dist}_C(u,v)$ is minimal

if the number of edges beginning at u in E'_C is at least $k+1$ **then**

delete u from V^1_C and delete from E_C all edges beginning at u

if the number of edges ending at v in E'_C is at least $k+1$ **then**

delete v from V^2_C and delete from E_C all edges ending at v

if (u,v) is still in E_C **then** {edge (u,v) will be in the spanner}

$E_C = E_C \setminus \{(u,v)\}$

$E'_C = E'_C \cup \{(u,v)\}$

let $N^u_{(u,v)}$ be the set of points $x \in V^1_C$ with $|ux|_\infty \leq \frac{\omega}{\sqrt{d}}|uv|$

let $N^v_{(u,v)}$ be the set of points $y \in V^2_C$ with $|vy|_\infty \leq \frac{\omega}{\sqrt{d}}|uv|$

for each $x \in N^u_{(u,v)}$ **do**

if (u,v) is disjoint with all edges in $F_C(x)$ **then**

$F_C(x) = F_C(x) \cup \{(u,v)\}$

if $|F_C(x)| \geq k+1$ **then**

delete x from V^1_C and delete from E_C all edges beginning at x

for each $y \in N^v_{(u,v)}$ **do**

if (u,v) is disjoint with all edges in $H_C(y)$ **then**

$H_C(y) = H_C(y) \cup \{(u,v)\}$

if $|H_C(y)| \geq k+1$ **then**

delete y from V^2_C and delete from E_C all edges ending at y

$E' = \bigcup_{C \in \mathcal{C}} E'_C$ output $G' = (V, E')$

CHAPTER 9

CONCLUSIONS

This dissertation studied combinatorial optimizations problems arising from two areas: scheduling and network design. The main focus was to design efficient approximation algorithms for NP-hard problems.

9.1 Scheduling Problems

In the scheduling area, problems in master-slave model have been considered. The master-slave model finds many applications in parallel computer scheduling and industrial settings such as semiconductor testing, machine scheduling, transportation maintenance and parallel computing.

The complexity issues are considered first. It is shown that many makespan and total completion time problems with constraints such as canonical, order preserving and no-wait-in, are NP-hard in the strong sense.

Motivated by the computational complexity, some special cases of the problem are considered. Several efficient algorithms are designed to minimize the total completion time and makespan. These algorithms are proven to have very good approximation ratio. Next more general cases are discussed. A job can have an arbitrary release time and arbitrary processing time. There can be a single master, multi-masters, or distinct preprocessor and postprocessors. Constant approximation algorithms are designed for the total completion time problem. It is shown that these algorithms give good approximation for makespan at the same time.

There are several questions that have not been answered. In this dissertation, it is assumed that there are no precedence constraints among the jobs. These assumption may not hold in some applications. It is worthwhile to develop approximation algorithms which

work well even with precedence constraints. Only the makespan and total completion time are considered so far. Other objectives such as maximum lateness, number of tardy jobs and total tardiness, have not been studied. In view of the NP-hardness of makespan and total completion time, minimizing these objectives must also be NP-hard. In that case, it will be desirable to have good approximation algorithms.

9.2 Network Design Problems

The survivable network design problem is the problem of designing graphs that resist edge and/or vertex removal. This is a fundamental problem in algorithmic graph theory with numerous applications in computer science and operations research. It is well-known that all non-trivial variants of the survivable network design problem are NP-hard and therefore the main research interest lies in the design of efficient approximation algorithms.

First the geometric version of the survivable network design problem is considered. A PTAS is developed for the $\{0,1,2\}$-connectivity problem in which each vertex has a connectivity requirement at most 2. Then it is shown that the techniques can be generalized to other ℓ_p^d metrics and to the $\{0,1,\ldots,k\}$-edge-connectivity problems for multigraphs. PTASs or Quasi-PTASs are designed for the minimum 2-connectivity problem and some of its variations in unweighted or weighted planar graphs.

Then the problem of efficient construction of fault-tolerant spanners are considered. A greedy algorithm is presented which finds the fault-tolerant spanners with both maximum degree and total cost asymptotically optimal. An efficient algorithm is then developed to find a fault-tolerant spanners with asymptotically optimal bound for the maximum degree and almost optimal bound for the total cost.

Two important components used in all the approximation schemes are hierarchical decomposition and dynamic programming. Through the decomposition, a large graph is decomposed into smaller ones which have small interface with the rest of the graph. In the Euclidean space, random shifted dissection is used to decompose a set of points into

regions. The interface between two regions contains only constant number of portals. In planar graphs, a modified separator theorem is developed and applied to decompose the graphs into small pieces. Each small piece has at most logarithmic number of portals. The small interface between subgraphs allows one to solve the problem by dynamic programming in polynomial time.

It would be great if one can extend the above techniques to other related problems. Following are some possible problems.

k-connectivity problem in doubling metrics. The doubling dimension of a metric is the smallest k such that any ball of radius $2r$ can be covered using 2^k balls of radius r. This concept for abstract metrics has been proposed as a natural analog to the dimension of a Euclidean space. Doubling metric appears in several practical applications such as peer-to-peer networks and data analysis. In [114], Talwar showed that for low dimensional metrics there exist quasi-polynomial time approximation schemes for TSP and other optimization problems.

The geometric version of the k-connectivity problem has been considered and polynomial approximation schemes have been designed k-connectivity problem in geometric graphs by Czumaj and Lingas in [26]. Their algorithm is based on the framework of Arora [3]. Like the approach used by Rao and Smith [101] for the TSP problem, their algorithm depends on spanners and some other nontrivial techniques.

For the problem in the low dimension doubling metric, there is no known results about construction of light and sparse spanners. Therefore one can not use the approach as in [26]. However, it is still possible to achieve a Quasi-PTAS by using some novel ideas and those from [114], [27] [26].

Minimum strongly connected subgraph in planar graphs. For the general unweighted graphs, constant approximation algorithms have been given by Khuller et al. in [71, 72].

The existing separator theorems only work for undirected graphs. A PTAS for the problem in planar graphs will heavily depend on light, directed cycle separators.

Forbidden minor 2-ECSS (or 2-VCSS). There are two difficulties here: first, the usual separator theorems do not produce cycles (just trees). Also the appropriate generalization of Lemma 6.2.4 may require a careful application of the Robertson-Seymour theory, which appears difficult. For the special case of bounded-genus graphs, both of these issues look more tractable.

For the fault tolerant spanners, an obvious open question that is left is whether one can design an efficient algorithm that outputs fault-tolerant spanners having properties from Theorem 8.1.3. It is believed that it should be possible to design an $\mathcal{O}(nk \operatorname{polylog} n)$-time algorithm for that problem.

In all existing analysis of fault-tolerant spanners, the cost of fault-tolerant spanners is compared with that of MST. Since a k-fault-tolerant spanner must be $(k+1)$ connected, a natural question is how to compare its cost with the cost of the minimum $(k+1)$ connected spanning subgraphs. One can extend the example in Figure 9.1 to show that the cost ratio between the optimal $(1, 1+\varepsilon)$ fault-tolerant spanner and the minimum $(k+1)$ connected spanning subgraphs can be arbitrarily large in planar graphs. It is believed that the cost ratio for the geometric graphs should be bounded by constant. It is desirable if one could design efficient algorithm to construct such light spanners.

There are many algorithmic applications of spanners, perhaps the most appealing being the recent application in an $\mathcal{O}(n \log n)$-time approximation algorithm for the Euclidean TSP [101]. On the other hand, not many applications of fault-tolerant spanners are known. Actually, even in the most natural application to the k-connectivity problem, the fastest approximation algorithms do not use fault-tolerant spanners [25, 26, 27]. Therefore, investigating the relationship between the connectivity problems in Euclidean graphs and the notion of fault-tolerant spanners would be an interesting topic.

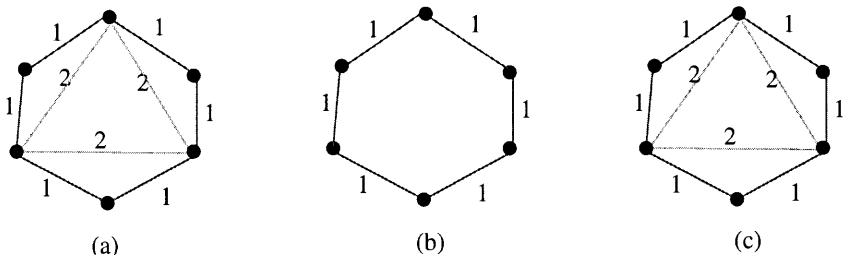


Figure 9.1 (a) A weighted planar graph G, (b) the minimum cost biconnected spanning subgraph of G, (c) the optimal $(1, 1+\varepsilon)$ fault-tolerant spanner which is G itself.

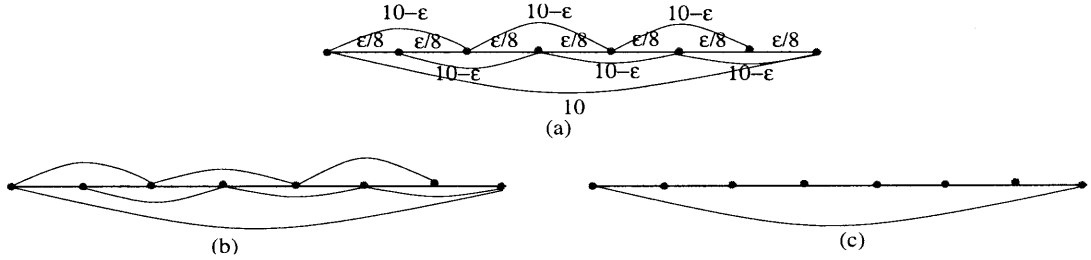


Figure 9.2 (a) A weighted planar graph G, (b) $(1, 1+\varepsilon)$-VFTS (or $(1, 1+\varepsilon)$-EFTS) of G generated by the greedy algorithm with $t = 1+\varepsilon$, and (c) the minimum $(1, 1+\varepsilon)$-VFTS (or $(1, 1+\varepsilon)$-EFTS) of G.

It is tempting (and very interesting) to try to extend the results for geometric fault tolerant spanners to non-Euclidean graphs. One could extend Claim 8.3.1 to hold for arbitrary graphs, that is to show that the graph obtained in the k-Greedy Algorithm is a (k, t)-VFTS for arbitrary (that is, also for non-Euclidean) graphs.

Furthermore, since the k-Greedy Algorithm is an extension of the classical greedy that has been used extensively in the construction of t-spanners, see, e.g., [1], it is plausible to ask whether it produces good quality spanners for arbitrary graphs or for planar graphs. For example, it follows from [1] that if the k-Greedy Algorithm with $k = 0$ is run on a planar graph, then it produces a t-spanner whose total cost is upper bounded by $\mathcal{O}(1/(t-1))$ times the minimum spanning tree cost. However, this result cannot be generalized to larger k. First there are graphs that does not have light spanners at all, see Figure 9.1 for an example. Furthermore, even if there is one, the greedy algorithm can not always produce good spanners, see Figure 9.2.

Due to its potential applications in various situations, ad hoc wireless network has received significant attention over the last few years for its various applications. In an ad hoc network the links between neighboring nodes get up and down from time to time because of mobility of the nodes and the communication capabilities of each node is usually constrained by its limited battery power. This makes fault tolerance of the network one of the fundamental and critical issues. Because the network is distributed, the global algorithms for survivable network design and fault-tolerant spanner in this dissertaion do not work any more. Li et al. [84] studied how to set the transmission radius to achieve the k-connectivity with certain probability for a network of n devices; they also proposed a construction of fault-tolerant spanners. Although the number of links is bounded, the total cost (transmission power) may be unbounded. A natural question is how to assign the transmission power for each node to minimize the total transmission power assignment.

REFERENCES

[1] I. Althöfer, G. Das, D. Dobkin, D. Joseph, and J. Soares. On sparse spanners of weighted graphs. *Discrete & Computational Geometry*, vol. 9, pp. 81-100, 1993.

[2] M. Dell'Amico. Shop problems with two machine and time lags. *Operations Research*, 44, 5, 777-787, 1996.

[3] S. Arora. Polynomial time approximation schemes for Euclidean traveling salesman and other geometric problems. *Journal of the ACM*, 45(5):753-782, 1998.

[4] S. Arora, M. Grigni, D. Karger, P. Klein, and A. Woloszyn. A polynomial time approximation scheme for weighted planar graph TSP. *Proceedings of the of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 33-41, 1998.

[5] S. Arya, G. Das, D. M. Mount, J. S. Salowe, and M. Smid. Euclidean spanners: Short, thin, and lanky. In *Proceedings of the 27th Annual ACM Symposium on Theory of Computing*, pp. 489-498, Las Vegas, NV, May 29 - June 1, 1995. ACM Press, New York, NY.

[6] S. Arya and M. Smid. Efficient construction of a bounded-degree spanner with low weight. *Algorithmica*, 17(1):33-54, January 1997.

[7] K.R. Baker. *Introduction to Sequencing and Scheduling*. John Wiley & Sons, New York, 1974.

[8] M. de Berg, M. van Kreveld, M. Overmars, and O. Schwarzkopf. *Computational Geometry - Algorithms and Applications*. Springer-Verlag, Berlin, 1997.

[9] A. Berger,A. Czumaj, M. Grigni, and H. Zhao. Approximate minimum 2-connected subgraphs in weighted planar graphs. *Manuscript*, 2004.

[10] A. Berger, M. Grigni, and H. Zhao. A well-connected separator for planar graphs. *Manuscript*, 2004.

[11] M. Bollobas. *Modern Graph Theory*. Springer-Verlag, Berlin, 1998.

[12] R.E. Buten and V.Y. Shen. A scheduling model for computer systems with two classes of processors. *Sagamore Computer Conference on Parallel Processing*, pp. 130-138, 1973.

[13] P. B. Callahan and S. R. Kosaraju. A decomposition of multidimensional point sets with applications to k-nearest-neighbors and n-body potential fields. *Journal of the ACM*, 42(1):67-90, January 1995.

[14] S. Chakrabarti, C. Phillips, A. Schulz, D.B. Shmoys, C. Stein and J. Wein. Improved scheduling algorithms for minsum criteria. In *Proceedings of the 23rd International Colloquium on Automata, Languages and Programming*, pp. 646-657, 1996

[15] B. Chandra, G. Das, G. Narasimhan, and J. Soares. New sparseness results on graph spanners. In *Proc. of the 8th Annual ACM Symposium on Computational Geometry*, pp. 192-201, Berlin, Germany, June 10-12, 1992. ACM Press, New York, NY.

[16] C. Chekuri, R. Motwani, B. Natarajan and C. Stein. Approximation techniques for average completion time scheduling. *SIAM Jounal on Computing*, 31(1), pp. 146-166, 2001.

[17] J. Cheriyan, A. Sebö, and Z. Szigeti. An improved approximation algorithm for minimum size 2-edge connected spanning subgraphs. *Proceedings of the of the 6th International Integer Programming and Combinatorial Optimization Conference, LNCS*, 1412:126-136, 1998.

[18] J. Cheriyan, S. Vempala, and A. Vetta. Approximation algorithms for minimum-cost k-vertex connected subgraphs. *Proceedings of the of the 34th ACM Symposium on Theory of Computing*, pp. 306-312, 2002.

[19] J. Cheriyan, S. Vempala, and A. Vetta. An approximation algorithm for the minimum-size k-vertex connected subgraph. *SIAM Journal on Computing*, 32(4):1050-1055, 2003.

[20] P. L. Chew. There is a planar graph as good as the complete graph. In *Proc. of the 2nd Annual ACM Symposium on Computational Geometry*, pp. 169-177, 1986.

[21] S. Chopra and C.-Y. Tsai. A branch-and-cut approach for minimum cost multi-level network design. *Discrete Mathematics*, 242:65-92, 2002.

[22] E. G. Coffman, Jr.(ed.) *Computer and Job-Shop Scheduling Theory*. John Wiley and Sons, New York, 1976.

[23] B. Csaba, M. Karpinski, and P. Krysta. Approximability of dense sparse instances of minimum 2-connectivity, TSP and path problems. *Proc. of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 74-83, 2002.

[24] A. Czumaj, M. Grigni, P. Sissokho, and H. Zhao. Approximation schemes for minimum 2-edge-connected and biconnected subgraphs in planar graphs. In *Proceedings of the 15th Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 489-498, New Orleans, LA, January 11 - 13, 2004. SIAM, Philadelphia, PA.

[25] A. Czumaj and A. Lingas. On approximability of the minimum cost spanning subgraph problem. *Proceedings of the of the 10th Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 281-290, 1999.

[26] A. Czumaj and A. Lingas. Fast approximation schemes for Euclidean multi-connectivity problems. In *Proceedings of the of the 27th Annual International Colloquium on Automata, Languages and Programming*, pp. 856-868, 2000.

[27] A. Czumaj, A. Lingas, and H. Zhao. Polynomial-time approximation schemes for the Euclidean survivable network design problem. *Proceedings of the of the 29th Annual International Colloquium on Automata, Languages and Programming, LNCS*, 2380, pp. 973-984, 2002.

[28] A. Czumaj and H. Zhao. Fault-tolerant geometric spanners. In *Proceedings of the 19th Annual ACM Symposium on Computational Geometry*, pp. 1-10, San Diego, CA, June 8-10, 2003. ACM Press, New York, NY.

[29] G. Das, P. Heffernan, and G. Narasimhan. Optimally sparse spanners in 3-dimensional Euclidean space. In *Proceedings of the of the 9th Annual ACM Symposium on Computational Geometry*, pp. 53-62, San Diego, CA, May 19-21, 1993. ACM Press, New York, NY.

[30] G. Das and G. Narasimhan. A fast algorithm for constructing sparse Euclidean spanners. *International Journal of Computational Geometry and Applications*, 7(4):293-315, 1997.

[31] G. Das, G. Narasimhan, and J. Salowe. A new way to weigh malnourished Euclidean graphs. In *Proc. of the 6th Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 215-222, 1995.

[32] R. Diestel. *Graph Theory.* Springer-Verlag, New York, 2000.

[33] J. Du and J.Y-T. Leung. Minimizing mean flow time in two-machine open shops and flow shops. *Journal of Algorithms*, 14:24-44, 1993.

[34] D. Eppstein. Spanning trees and spanners. In J.-R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, chapter 9, pp. 425-461. Elsevier Science B.V., 1997.

[35] C. G. Fernandes. A better approximation ratio for the minimum size k-edge-connected spanning subgraph problem. *Journal of Algorithms*, 28:105-124, 1988.

[36] L. Fleischer. A 2-approximation for minimum cost $\{0, 1, 2\}$ vertex connectivity. In *Proceedings of the 8th International Integer Programming and Combinatorial Optimization Conference*, volume 2081 of *Lecture Notes in Computer Science*, pp. 115-129, Utrecht, The Netherlands, June 13-15, 2001. Springer-Verlag, Berlin.

[37] L. Fleischer, K. Jain, and D. P. Williamson. An iterative rounding 2-approximation algorithm for the element connectivity problem. In *Proceedings of the 42nd IEEE Symposium on Foundations of Computer Science*, pp. 339-347, Las Vegas, NV, October 14-17, 2001. IEEE Computer Society Press, Los Alamitos, CA.

[38] G. N. Frederickson and J. JáJá. On the relationship between the biconnectivity augmentation and traveling salesman problem. *Theoretical Computer Science*, 19(2), pp. 189-201, 1982.

[39] H. N. Gabow. An ear decomposition approach to approximating the smallest 3-edge connected spanning subgraph of a multigraph. *Annual ACM-SIAM Symposium on Discrete Algorithms* 2002, pp. 84-93.

[40] H. N. Gabow. Better performance bounds for finding the smallest k-edge connected spanning subgraph of a multigraph. *Annual ACM-SIAM Symposium on Discrete Algorithms* 2003, pp. 460-469.

[41] H. N. Gabow, M. X. Goemans, and D. P. Williamson. An efficient approximation algorithm for the survivable network design problem. *Mathematical Programming*, 82(1-2):13-40, 1998.

[42] J. Gao, L. J. Guibas, J. Hershburger, L. Zhang, and A. Zhu. Geometric spanner for routing in mobile networks. In *Proceedings of the 2nd ACM Symposium on Mobile Ad Hoc Networking & Computing (MobiHoc 2001)*, pp. 45-55, Long Beach, CA, October 4-5, 2001.

[43] M.R. Garey and D.S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. Freeman, New York, 1979.

[44] E. N. Gilbert and H. O. Pollak. Steiner minimal trees. *SIAM Journal on Applied Mathematics*, 16(1):1-29, 1968.

[45] M.X. Goemans. Improved approximation algorithms for scheduling with release dates. In *Proceedings of the Eighth ACM-SIAM Symposium on Discrete Algorithms*, pp. 591-598, 1997.

[46] T.F. Gonzalez and S. Sahni. Flowshop and jobshop schedules: complexity and approximation. *Operations Research*, 26, pp. 26-52, 1978.

[47] M. Grigni, E. Koutsoupias, and C. Papadimitriou. An approximation scheme for planar graph TSP. *Proceedings of the of the 36th IEEE Symposium on Foundations of Computer Science*, pp. 640-645, 1995.

[48] M. Grigni and P. Sissokho. Light spanners and approximate TSP in weighted graphs with forbidden minors. *Proceedings of the 13th Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 852-857, 2002.

[49] M. Grötschel. Discrete mathematics in manufacturing. *Proceedings of the of the 2nd International Conference on Industrial and Applied Mathematics*, pp. 119-145, 1991.

[50] M. Grötschel and C. L. Monma. Integer polyhedra arising from certain network design problems with connectivity constraints. *SIAM Journal on Discrete Mathematics*, 3(4):502-523, November 1990.

[51] M. Grötschel, C. L. Monma, and M. Stoer. Computational results with a cutting plane algorithm for designing communication networks with low-connectivity constraints. *Operations Research*, 40(2):309-330, 1992.

[52] M. Grötschel, C. L. Monma, and M. Stoer. Polyhedral and computational investigations for designing communication networks with high survivability requirements. *Operations Research*, 43:1012-1024, 1995.

[53] M. Grötschel, C. L. Monma, and M. Stoer. Design of survivable networks. In M. O. Ball, T. L. Magnanti, C. L. Monma, and G. L. Nemhauser, editors, *Handbooks in Operations Research and Management Science*, volume 7: Network Models, chapter 10, pp. 617-672. North-Holland, Amsterdam, 1995.

[54] M. Grünewald, T. Lukovszki, C. Schindelhauer, and K. Volbert. Distributed maintenance of resource efficient wireless network topologies. In B. Monien and R. Feldmann, editors, *Proceedings of the 8th Euro-Par*, volume 2400 of *Lecture Notes in Computer Science*, pp. 935-946, Paderborn, Germany, August 27-30, 2002. Springer-Verlag, Berlin.

[55] J. Gudmundsson, C. Levcopoulos, and G. Narasimhan. Fast greedy algorithms for constructing sparse geometric spanners. *SIAM Journal on Computing*, 31(5):1479-1500, August 2002.

[56] J. Gudmundsson, C. Levcopoulos, and G. Narasimhan. Improved greedy algorithms for constructing sparse geometric spanners. In M. M. Halldórsson, editor, *Proceedings of the 7th Scandinavian Workshop on Algorithm Theory*, volume 1851 of *Lecture Notes in Computer Science*, pp. 314-327, Bergen, Norway, July 5-7, 2000. Springer-Verlag, Berlin.

[57] J.N.D. Gupta. Two-stage, hybrid flowshop scheduling problem. *Journal of the Operational Research Society*, 38, pp. 359-364, 1988.

[58] L.A. Hall. Approximability of flow shop scheduling. *Mathematical Programming*, 82, pp. 175-190, 1998.

[59] L.A. Hall, A.S. Schulz, D.B. Shmoys and J. Wein, Scheduling to minimize average completion time: Offine and online algorithms. *Mathematics of Operations Research*, 22, pp. 513-544,

[60] D. S. Hochbaum(ed.). *Approximation Algorithms for NP-Hard Problems.* PWS Publishing Company, Boston, MA, 1995.

[61] J.A. Hoogeveen and T. Kawaguchi. Minimizing total completion time in a two machine flowshop: Analysis of special cases. *Mathematics of Operations Research*, 24(4), 887-910, 1999.

[62] F. K. Hwang and D. S. Richards. Steiner tree problems. *Networks*, 22:55-89, 1991.

[63] F. K. Hwang, D. S. Richards, and P. Winter. *The Steiner Tree Problem.* North-Holland, Amsterdam, 1992.

[64] K. Jain. A factor 2 approximation algorithm for the generalized Steiner network problem. *Combinatorica*, 21(1):39-60, 2001. A preliminary version appeared in *Proceedings of the 39th IEEE Symposium on Foundations of Computer Science*, pp. 448-457, Palo Alto, CA, November 8-11, 1998. IEEE Computer Society Press, Los Alamitos, CA.

[65] S.M. Johnson. Optimal two and three-stage production schedules with setup times included. *Naval Research Logistics* Quarterly, 1, pp. 61-68, 1954.

[66] R. Jothi, B. Raghavachari, and S. Varadarajan. A 5/4-approximation algorithm for minimum 2-edge-connectivity. *Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms*, pp. 725-734, 2003.

[67] P. Indyk and R. Motwani. Approximate nearest neighbors: Towards removing the curse of dimensionality. In *Proceedings of the 30th Annual ACM Symposium on Theory of Computing*, pp. 604-613, Dallas, TX, May 23-26, 1998. ACM Press, New York, NY.

[68] A.H.G. Rinnooy Kan. *Machine Scheduling Problems: Classification, complexity and computations*, Nijhoff, The Hague, 1976.

[69] W. Kern and W. Nawijn. Scheduling multi-operation jobs with time lags on a single machine. University of Twente, 1993.

[70] S. Khuller. Approximation algorithms for finding highly connected subgraphs. In D. S. Hochbaum, ed., *Approximation Algorithms for $\mathcal{NP}$-Hard Problems*, pp. 236-265, 1996.

[71] S. Khuller, B. Raghavachari and N. Young. Approximating the minimum equivalent digraph. *Proceedings of the of the 5th Annual ACM-SIAM Symposium on Discrete Algorithms*, pp.177-186, 1994.

[72] S. Khuller, B. Raghavachari and N. Young. On strongly connected digraphs with bounded cycle length. UMIACS-TR-94-10/CS-TR-3212, 1994.

[73] S. Khuller and U. Vishkin. Biconnectivity approximations and graph carvings. *Journal of the ACM*, 41(2):214-235, March 1994.

[74] G. Kortsarz and Z. Nutov. Approximation algorithm for k-node connected subgraphs via critical graphs. *Annual ACM Symposium on Theory of Computing* 2004, pp. 138-145.

[75] M.A. Langston. Interstage transportation planning in the deterministic flow-shop environment. *Operations Research*, 35(4), pp. 556-564, 1987.

[76] E.L.Lawler, J.K. Lenstra, A.H.G. Rinnooy Kan, and D.B. Shmoys. Sequencing and scheduling: Algorithms and complexity. In S.C. Graves, A.H.G. Rinnooy Kan, and P.H. Zipkin (eds.), *Logistics of Production and Inventory, Handbooks in Operations Research and Management Science 4*, North-Holland, Amsterdam, 445-522, 1993.

[77] E. L. Lawler, J. K. Lenstra, A. H. G. Rinnooy Kan, and D. B. Shmoys, editors. *The Traveling Salesman Problem.* John Wiley & Sons, New York, NY, 1985.

[78] C.-Y. Lee and G.L. Vairaktarakis. Minimizing makespan in hybrid flowshops. *Operations Research Letters*, 16, pp. 149-158, 1994.

[79] J. Y-T. Leung and H. Zhao. Minimizing Mean Flowtime and Makespan on Master-Slave Systems. To appear in *Journal of Parallel and Distributed Computing.*

[80] J. Y-T. Leung and H. Zhao. Minimizing Total Completion Time in Master-Slave Systems. *Manuscript.*

[81] C. Levcopoulos and A. Lingas. There are planar graphs almost as good as the complete graphs and almost as cheap as minimum spanning trees. *Algorithmica*, 8:251-256, 1992.

[82] C. Levcopoulos, G. Narasimhan, and M. H. M. Smid. Efficient algorithms for constructing fault-tolerant geometric spanners. *Annual ACM Symposium on Theory of Computing* 1998, pp. 186-195.

[83] C. Levcopoulos, G. Narasimhan, and M. H. M. Smid. Improved algorithms for constructing fault-tolerant spanners. *Algorithmica*, 32(1):144-156, 2002.

[84] X. Li, W. P. Wan and C. Yi. Robust deployment and fault tolerant topology control for wireless ad hoc networks. *Journal of Wireless Communications and Mobile Computing*, 4(1), pp. 109-125, 2004.

[85] R. Lipton and R. Tarjan. Applications of a planar separator theorem. *SIAM Journal on Computing*, 9(3):615-627, 1980.

[86] T. Lukovszki. New results on fault tolerant geometric spanners. In *Proc. of the 6th Workshop on Algorithms and Data Structures*, volume 1663 of *LNCS*, pp. 193-204, 1999.

[87] T. Lukovszki. *New Results on Geometric Spanners and Their Applications*. PhD thesis, University of Paderborn, 1999.

[88] K. Mehlhorn and S. Näher. Dynamic fractional cascading. *Algorithmica*, 5:215-241, 1990.

[89] G. L. Miller. Finding small simple cycle separators for 2-connected planar graphs. *Journal of Computer and System Sciences*, 32:265-279, 1986.

[90] J. S. B. Mitchell. Guillotine subdivisions approximate polygonal subdivisions: A simple polynomial-time approximation scheme for geometric TSP, k-MST, and related problems. *SIAM Journal on Computing*, 28(4):1298-1309, August 1999.

[91] C. L. Monma and D. F. Shallcross. Methods for designing communications networks with certain two-connected survivability constraints. *Operations Research*, 37(4):531-541, July 1989.

[92] D. Peleg. *Distributed Computing: A Locality-sensitive Approach*. Society for Industrial and Applied Mathematics, Philadelphia, PA, 2000.

[93] D. Peleg and A. Schäffer. Graph spanners. *Journal of Graph Theory*, 13:99-116, 1989.

[94] M. Penn and H. Shasha-Krupnik. Improved approximation algorithms for weighted 2- and 3-vertex connectivity augmentation problems. *Joural of Algorithms*, 22:187-196, 1997.

[95] C. Phillips, C. Stein and J. Wein. Minimizing average completion time in the presence of release dates. *Mathematical Programming*, 82:199-223, 1998.

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

[97] H. J. Prömel and A. Steger. *The Steiner Tree Problem. A Tour Through Graphs, Algorithms and Complexity*. Vieweg Verlag, Wiesbaden, 2002.

[98] M. Queyranne. Structure of a simple scheduing polyhedron. *Mathematical Programming*, 58:263-285, 1993.

[99] M. Queyranne. Personal communication, 1995.

[100] R. Rajaraman. Topology control and routing in ad hoc networks: A survey. *SIGACT News*, 33:60-73, June 2002.

[101] S. B. Rao and W. D. Smith. Approximating geometrical graphs via "spanners" and "banyans." In *Proceedings of the 30th Annual ACM Symposium on Theory of Computing*, pp. 540-550, Dallas, TX, May 23-26, 1998. ACM Press, New York, NY.

[102] G. Robins and J. S. Salowe. Low-degree minimum spanning trees. *Discrete & Computational Geometry*, 14:151-166, 1995. A preliminary version (entitled "On the maximum degree of minimum spanning trees") appeared in *Proceedings of the 10th Annual ACM Symposium on Computational Geometry*, pp. 250-258, Stony Brook, NY, June 6-8, 1994. ACM Press, New York, NY.

[103] J. Ruppert and R. Seidel. Approximating the d-dimensional complete Euclidean graph. In *Proceedings of the of the 3rd Canadian Conference on Computational Geometry*, pp. 207-210, 1991.

[104] S. Sahni. Scheduling master-slave multiprocessor systems. *IEEE Transactions on Computers*, 45(10), 1195-1199, 1996.

[105] S. Sahni and G. Vairaktarakis. The master-slave scheduling model. In J. Y-T. Leung (Ed): *Handbook of Scheduling: Algorithms, Models, and Performance Analysis*, CRC Press, Boca Raton, FL, 2004.

[106] S. Sahni and G. Vairaktarakis. The master-slave paradigm in parallel computer and industrial settings. *Journal of Global Optimization*, 9, 357-377, 1996.

[107] L. Schrage. A proof of the optimality of the shortest remaining processing time discipline. *Operations Research*, 16:687-690, 1968.

[108] A. S. Schulz. Scheduling to minimize total weighted completion time: Performance guarantees of LP-based heuristics and lower bounds. In *Proceedings of the 5th Integer Programming and Combinatorial Optimaization (IPCO)*, pp. 301-315, 1996.

[109] A.S. Schulz and M. Skutella. Scheduling-LPs bear probabilities: Randomized approximations for min-sum criteria, In *Proceedings of the Fifth Annual European Symposium on Algorithms*, pp. 416 429, 1997.

[110] M. Smid. Closest-point problems in computational geometry. In J.-R. Sack and J. Urrutia, editors, *Handbook of Computational Geometry*, chapter 20, pp. 877-935. Elsevier Science B.V., 1997.

[111] D. Smith. A new proof of the optimality of the shortest remaining processing time discipline. *Operations Research*, 26(1):197-199, 1976.

[112] C. Sriskandarajah and S.P. Sethi. Scheduling algorithms for flexible flowshops: worst and average case performance. *European Journal of Operational Research*, 43, pp. 143-160, 1989.

[113] M. Stoer. *Design of Survivable Networks*, volume 1531 of *Lecture Notes in Mathematics*. Springer-Verlag, Berlin, 1992.

[114] K. Talwar Bypassing the embedding: approximation schemes and compact representations for low dimensional metrics. In *Proceedings of the 36th Annual ACM Symposium on Theory of Computing*, pp. 281-290, 2004.

[115] G. Vairaktarakis. Analysis of algorithms for master-slave system. *IIE Transactions*, 29, 11, 939-949, 1997.

[116] S. Vempala and A. Vetta. Factor 4/3-approximations for minimum 2-connected subgraphs. In *Proceedings of the 3rd Workshop APPROX, LNCS*, 1913:262-273, 2000.

[117] D. P. Willimason, M. X. Goemans, M. Mihail, and V. V. Vazirani. A primal-dual approximation algorithm for generalized Steiner network problem. *Combinatorica*, 15:435-454, 1995.

[118] P. Winter. Steiner problem in networks: A survey. *Networks*, 17:129-167, 1987.

[119] L.A.Wolsey. Mixed integer programming formulations for production planning and scheduling problems. Invited talk at the *12th International Symposium on Mathematical Programming*, MIT, Cambridge, 1985.

[120] A. C. Yao. On Constructing minimum spanning trees in k-demensional spaces and related problems. *SIAM Journal on Computing*, 11:721-736, 1982.

[121] W. Yu, H. Hoogeveen and J.K. Lenstra. Minimizing makespan in a two-machine flowshop with delays and unit-time operations is NP-hard. *Journal of Scheduling*, 7(5), 333 - 348, 2004.