Computational Difficulty

Scheduling and Circuit Complexity

by Keisuke Tanaka

A dissertation submitted to
School of Information Science
Japan Advanced Institute of Science and Technology
in partial fulfillment of the requirements
for the degree of
Doctor of Information Science
Graduate Program in Information Science

Written under the direction of
Professor Milan Vlach

January 14, 1997

Abstract of the Dissertation

Computational Difficulty
Scheduling and Circuit Complexity

by Keisuke Tanaka

Dissertation Director: Professor Milan Vlach

The classification of problems according to their computational difficulty requires establishing both upper and lower bounds on the amount of computational resources necessary to solve them. In this thesis, we are investigating computational difficulty by studying the complexity of scheduling problems and boolean circuits. In particular, we are focusing on single machine scheduling problems with new types of due dates and release dates, and negation-limited circuits, where the number of negations available is restricted.

In 1986, Hall proposed a new type of due dates called generalized due dates. First, we investigate the problems of minimizing the maximum absolute lateness and range of lateness under generalized due dates. In contrast to the traditional due date cases, we show that these problems are NP-hard in the strong sense. Furthermore, we present simple approximation algorithms with non-trivial performance guarantee for these problems. Second, we are concerned with scheduling with both traditional and generalized due dates, and show that a polynomial time algorithm exists for the problem of minimizing the maximum of maximum latenesses induced by traditional and generalized due dates. In addition to scheduling involving generalized due dates, we also consider scheduling with a new type of release dates which are related to the traditional release dates in a similar way as the generalized due dates to the traditional ones. In 1992, Ishii, Tada, and Masuda proposed another new type of due dates called fuzzy due dates. We improve the algorithms for basic scheduling problems with fuzzy due dates.

The difficulty of a problem can be expressed as the number of gates in a circuit with the minimum number of gates, which computes the problem. We consider the complexity of negation-limited inverters, giving lower bounds as well as new upper bounds on the size and depth of the inverters. This suggests improved upper bounds for slice functions as well as a tighter relationship between negation-limited and general circuit complexity. We also show lower bounds for various symmetric functions. Finally, we establish relationships between the number of negations available and circuit sizes.

Acknowledgements

I wish to thank everyone who has been helped me during these five years of my studies at JAIST.

I am very grateful to my supervisor, Milan Vlach for his guidance. I cannot imagine how my studies at JAIST would have been without him. The discussion with him has always stimulated my interest and curiosity. He has been a tireless source of insights and inspiration. I thank him for his generous contribution to this work.

I wish to thank Tetsuro Nishino for his help. The work on this thesis has been done in part also under his supervision, while he was at JAIST. I am greatly indebted to him for devoting his time to help me at various stages in my studies. I thank him for his contribution to this work.

I would like to thank Masayuki Kimura for accepting me in his laboratory. He has supported me and enabled me to work smoothly. I wish to thank Hiroshi Shimodaira and Mitsuru Nakai for their support. I have benefited greatly from them.

I wish to thank Kunihiko Hiraishi and Hiroakira Ono for serving on my committee and providing valuable suggestions on this thesis.

I am grateful to Yoji Kajitani and Osamu Watanabe for serving on my advisors for peripheral works. I also thank Osamu Watanabe for handling my frequent visits to Tokyo Institute of Technology.

I would like to thank Robert Beals, Masanori Okada, and Shao-Chin Sung for helpful discussion and their contribution to this work.

I with to thank all my teachers for their help. I am grateful to Magnús M. Halldórsson for teaching me various things including approximation algorithms. I also thank him for his encouragement. I would like to thank Jaikumar Radhakrishnan for sharing with me his knowledge of circuit complexity. I thank Tomohiko Uematsu, Yoshihito Toyama, Barun Chandra, Mineo Kaneko and Kunihiro Fujiyoshi for teaching me algebraic and combinatorial arguments.

I am grateful to Tetsuo Asano, Seinosuke Toda, Tatsuie Tsukiji, Kazuyuki Amano, Akira Maruoka, Akihiro Nozaki, Shigeki Iwata, Hiroaki Ishii, and Gilbert Young for sharing their knowledge of computational geometry, circuit complexity, fuzzy scheduling, and generalized due date scheduling.

I would like to thank James Wong. I have had an enjoyable time in San Francisco with him. I thank him for the helpful discussion on generalized due date scheduling. I wish to thank Toshihiko Yamakami. I have stayed fruitfully at NTT Telecommunication Networks Laboratories in Yokosuka. His fertile imagination has benefited me.

I with to thank Takuya Katayama for his support and mentoring. I am grateful to Yoshimichi Watanabe for his encouragement. Without him, I would not have been at

JAIST for my graduate work.

I would like to thank my fellow graduate students, Takashi Mihara, Takahito Aoto, and Hiroyuki Shirasu for explaining to me their works and for their comments on parts of this thesis.

I would like to thank Hiroshi Iida, Takumi Hayashi, Moriyasu Komase, Keeni Konad, and the members in Kimura–Shimodaira Laboratory and Nishino Laboratory for providing me the environment to pursue my studies. I thank Akira Sano, Tomohiko Morioka, and Yasuhito Suzuki. I have enjoyed the life in JAIST with them.

I am grateful to Chinatsu Kita and the members in the administration office for their assistance.

I would like to thank JAIST and PFU for providing financial support for my studies.

Finally, I wish to thank my family, Yoshihiko, Chifumi, Shinichi, and Takako Tanaka for their support.

Contents

Chapter 1

Introduction

We know from our experience that some computational problems are easy to solve and that some are difficult to solve. When we are confronted with a new problem, a natural question to ask is whether it is solvable *efficiently*, i.e., whether it has a *polynomial time algorithm*. A polynomial time algorithm is defined to be one whose time complexity function can be bounded by some polynomial function of the input length.

If the answer is obviously "yes", then we can concentrate our efforts on trying to find as efficient a polynomial time algorithm as possible.

If the answer seems to be "no", then it is not likely that we will try to find a polynomial time algorithm. Certainly, we can try to find an *exponential time algorithm*, whose time complexity function cannot be bounded by any polynomial function of the input length. However, we know that the difference between polynomial and exponential time algorithms has particular significance when we consider large problem instances. Thus, we can put a second natural question whether the problem is intractable, i.e., whether it cannot be solved by any polynomial time algorithm.

At the present time, we have a tool of *NP-hardness* in hand, with the help of which we can replace the second question with that whether the problem is NP-hard. If the answer is "yes", we will have another question whether NP-hard problems are really intractable. Certainly, the question whether the problem is NP-hard is meaningful and most researchers believe that NP-hard problems cannot be solved by polynomial time algorithms. However, at present, "NP-hard" does not mean *non-polynomially hard*. This is known as the famous P versus NP question.

A natural way of establishing an upper bound is to construct an algorithm to solve the problem under investigation. To establish non-trivial lower bounds are usually more difficult, because it must concern all possible algorithms.

In this thesis, we are studying computational difficulty, by investigating both upper and lower bounds. In particular, we are focusing on the complexity of scheduling problems and boolean circuits.

Scheduling is concerned with an optimal allocation of scarce resources to activities over time. It has been studied extensively because of its practical importance. We are especially concerned with single machine scheduling with new types of due dates and release dates.

Owing to close relationship between Turing machines and boolean circuits, a strong lower bound on the size of circuits computing a problem implies a strong lower bound on

the time required to solve it.

Circuit complexity is supposed to derive strong lower bounds in the area of the computation theory, because it is simple and has good mathematical property. To understand circuit complexity is one of the basic and important issues in theoretical computer science. We are especially concerned with and negation-limited circuits, where the number of negations available is restricted.

Computational models When we consider the difficulty of computational problems, we somehow assume some reasonable model of computation. In the areas of algorithms and computation, there are models which reflect well our natural notion of computation, namely, *Turing machines* and *boolean circuits*. They have been commonly used for many years to investigate the difficulty of problems. This thesis is also based on these computational models.

In this thesis, we construct algorithms for scheduling problems. We do not intend to make these algorithms in order to run them on Turing machines, however, there are always Turing machines behind the algorithms.

Optimization and decision problems A computational problem can be viewed as a function f that maps each input x in some given domain to an output $f(x)$ in some given range. In this thesis, we consider scheduling problems. When we refer to scheduling problems, they are supposed to be *optimization problems*, where for input x, the output $f(x)$ is the smallest value in a range of feasible values with x. However, it is sometimes convenient to consider *decision problems*, where the output of $f(x)$ ranges "yes" and "no". For example, a problem to ask how large value of optimality criterion an optimal schedule gives with a given instance is a optimization problem, while a problem to ask whether there is a schedule with some value of optimality criterion with a given instance is a decision problem.

Turing machines When we study the time required to solve problems, a deterministic Turing machine is known to be reasonable. A nondeterministic Turing machine makes an important notion of NP-completeness in the theory of computation. A Turing machine was proposed by Alan Turing [102], and it can formalize the notion of effectiveness. This model was invented quite a long ago in the history of the theory of computation, but is still the most common model. Here, we mention the two important variants of a Turing machine.

Deterministic Turing machines A *deterministic Turing machine* consists of a *finite control* and a finite collection of *tapes* each with a *head* to read and write (see Figure 1.1). The finite control is a finite collection of states. A tape is an infinite list of cells each containing a symbol. Initially, all tapes have blanks except for the first, which contains the input string. Once started, the machine goes from state to state, reading the symbols under the heads, writing new ones, and moving the heads. The exact action taken is determined by the current state, the symbols read, and the transition function of the machine. This continues until a designated halt state is entered. The machine indicates its output by the halting condition of the tapes.

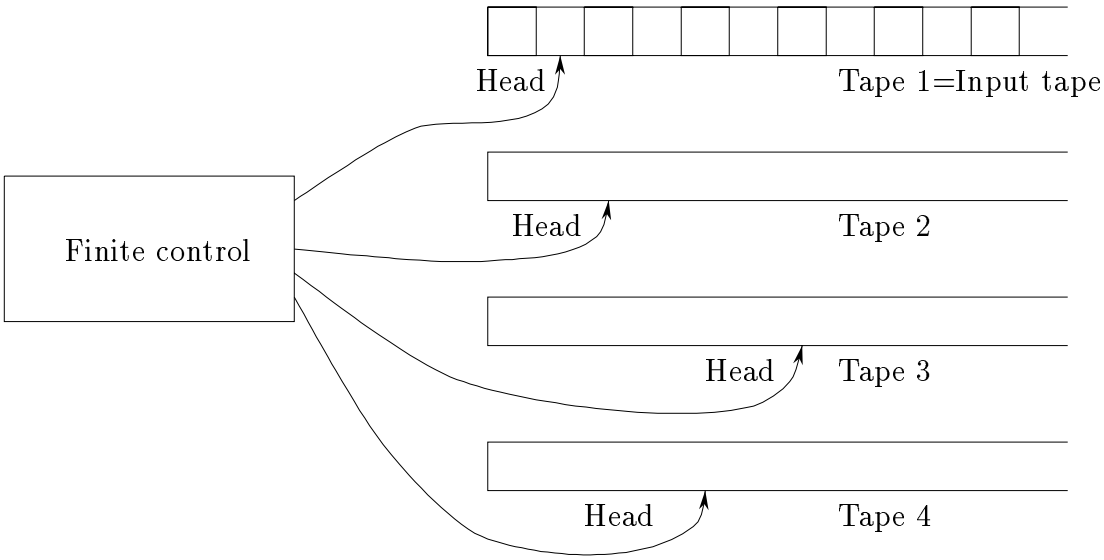

Figure 1.1: Turing machine.

The deterministic Turing machine model provides the basis for defining the difficulty of computational problems. We now define the Turing machine computations within a time bound T and a space bound S. Let f be a decision problem, and let T and S be functions from N to N, where N denotes the set of natural numbers. Problem f is computable in deterministic time $T(n)$ and space $S(n)$, if, for all inputs of size n, f can be computed by a deterministic Turing machine which halts after at most $T(n)$ steps and scans at most $S(n)$ cells on any tape, respectively.

Nondeterministic Turing machines In a *nondeterministic Turing machine*, the transition function is *multivalued*. This implies that there may be several computations on a given input and several output values.

We also define nondeterministic Turing machine computations within a time bound T and a space bound S. Let f be a decision problem and, let T and S be functions from N to N. Problem f is computable in nondeterministic time $T(n)$ and space $S(n)$, if there is a nondeterministic Turing machine such that for any input of size n whose output is "yes", some sequence of moves terminates and returns "yes" in at most $T(n)$ steps and scans at most $S(n)$ cells on any tape, respectively.

Incidentally, the effect of one step of a Turing machine is localized around the heads and the finite control. Clearly, this local property helps the simulations of Turing machines by circuits, as you can see in Chapter 7.

Complexity classes We now define the time and space *complexity classes*. The decision problems computable in deterministic time $O(T(n))$ and space $O(S(n))$ comprise TIME($T(n)$) and SPACE($S(n)$), respectively. Analogously one has the classes NTIME($T(n)$) and NSPACE($S(n)$) for nondeterministic computation. Here we mention

two important classes of the complexity of Turing machines:

$$\text{P} = \bigcup_k \text{TIME}(n^k),$$
$$\text{NP} = \bigcup_k \text{NTIME}(n^k).$$

P corresponds to the class of deterministic polynomial time computable decision problems. NP can be regarded as the class of polynomial time *verifiable* problems.

NP-completeness We know that, when we consider the algorithms on Turing machines, it is sufficient to consider those in any standard programming language. Actually, we write the algorithms for scheduling problems in this thesis, in order not to depend particular machines.

Roughly speaking, NP-complete problems are hardest problems in NP in the following sense. If one of the NP-complete problems is solvable in polynomial time, then so is each problem in NP. For two decision problems Q and R, we say that Q *reduces to* R if there exists a polynomial time computable function f that transforms inputs for Q into those for R such that x is "yes" input for Q if and only if $f(x)$ is a "yes" input for R. A problem is *NP-complete* if it is in NP and every problem in NP reduces to it. An optimization problem is called *NP-hard* if the associated decision problem is NP-complete.

Although we have thus far ignored questions of encoding the inputs, there is one distinction which plays an important role in our discussion. There is an exponential gap between the lengths of unary and binary encodings. Let us consider the *partition* problem, one of so-called *number problems*. Given a positive integer B and a set $A = \{a_1, a_2, \ldots, a_n\}$ of positive integers such that $\sum_{j=1}^{n} a_j = B$, the question is whether there exists a partition of A into two sets A_1 and A_2 such that $\sum_{a_j \in A_i} a_j = B/2$ for $i = 1, 2$. This problem is NP-complete under a binary encoding. On the other hand, it can be solved by dynamic programming in $O(n \sum a_j)$ time, which is polynomial under a unary encoding. Such a method is called a *pseudo-polynomial time* algorithm.

There are number problems, ex. the *3-partition* problem, which are NP-complete even when the numbers are encoded in unary. Such kind of problems are called *strongly NP-complete*.

For further detail on the theory of algorithms and NP-completeness, see e.g. the book by Garey and Johnson [31] and the article by Stockmeyer [86]

In the connection to the current models of computation, we may add a new model proposed recently, quantum computation (see e.g. the paper by Deutsch [21] and the thesis by Mihara [68]).

Outline of this thesis

The remainder of this thesis is divided into ten chapters.

Chapter 2 to 6 deal with single machine scheduling with new types of due dates and release dates.

Scheduling with traditional due dates occupies a large part in the whole scheduling area. Recently, several new types of due dates are proposed. They arise quite naturally in many practical situations. In 1986, Hall proposed a new type of due dates called generalized due dates. After Chapter 2 for providing the formulation of our scheduling problems, we consider the problems with generalized due dates in Chapter 3. Especially,

the problems of minimizing the maximum absolute lateness and range of lateness under generalized due dates are studied. Then, in Chapter 4, we investigate the problems with both traditional and generalized due dates. In particular, we consider the problem to minimize the maximum lateness with both traditional and generalized due dates.

In addition to scheduling involving generalized due dates, in Chapter 5, we consider scheduling with a new type of release dates, which are related to the traditional release dates in a similar way as the generalized due dates to the traditional ones. Particularly, we consider the problems to minimize the sum of completion times with generalized release dates which have constant intervals, which are equivalent to two machine flow shop problems to minimize the sum of completion times where the processing times on the first machine are the same.

In 1992, Ishii, Tada, and Masuda proposed another new type of due dates called fuzzy due dates. Chapter 6 deals with basic problems involving fuzzy due dates.

The difficulty of a problem can be expressed as the number of gates in a circuit with the minimum number of gates, which computes the problem.

Despite the importance of lower bounds on the circuit complexity of explicit problems, the best bounds known are only linear. However, good lower bounds are known for the complexity of monotone circuits, where negations are forbidden. This motivates the study on the complexity of negation-limited circuits, where the number of negations available is restricted.

In Chapter 7 to 10, we devote ourself to study negation-limited circuit complexity of boolean functions. Chapter 7 is intended to give a brief review for the negation-limited circuit complexity. In Chapter 8, we consider the complexity of negation-limited inverters. We present lower bounds as well as new upper bounds on the size and depth of the inverters. Chapter 9 describes the negation-limited circuit complexity for symmetric functions, and gives lower bounds on the size and depth. Chapter 10 describes relationships between the number of negations available in circuits and the size of the circuits.

Finally, Chapter 11 gives concluding remarks for this entire thesis.

This thesis has been written under the supervision of Professor Milan Vlach. The work on this thesis has been done in part also under the supervision of Professor Tetsuro Nishino, while he was at Japan Advanced Institute of Science and Technology. Parts of this thesis have been published and announced [73, 93, 92, 12, 97, 96, 95, 91, 11, 99, 94, 98].

Part I

Single machine scheduling with new types of due dates and release dates

Chapter 2

Preliminaries

Scheduling is concerned with an optimal allocation of scarce resources to activities over time. It has been studied extensively because of practical importance. Scheduling with traditional due dates occupies a large part in the whole scheduling field. Recently, several new types of due dates have been proposed. They arise quite naturally in many practical situations. We are investigating scheduling with some of these new types of due dates, and giving polynomial approximation and exact optimization algorithms as well as NP-hardness results. We are also investigating scheduling with a new type of release times.

In this chapter, we describe the formulation of our scheduling problems. First, in Section 2.1, we provide a basic formulation for scheduling problems. In the following three sections, we describe new elements for our scheduling problems, i.e., generalized due dates in Section 2.2, generalized release dates in Section 2.3, and fuzzy due dates in Section 2.3.

2.1 Scheduling problems

We are only concerned with the *single machine scheduling*, in which the machine is continuously available from the beginning and can process only one job at a time. Single machine scheduling problems have been researched extensively ever since the seminal work by Jackson [49] and Smith [83].

We have n independent *jobs* $J_1, J_2, \ldots, J_n$ to be processed on the machine. Each job J_j has a single *operation* on the machine, and its *processing time* p_j which is required for completion of processing, and is independent of schedules. Job J_j becomes available for processing on its *release date* r_j. If no release dates are specified, all jobs are available from the beginning. Each job J_j has also a (traditional) *due date* d_j and a *weight* w_j that may be used in defining a given optimality criterion. *Preemption* (*job splitting*) is not allowed, i.e., the processing of any operation may not be interrupted and resumed at a later time.

A *schedule* is an allocation of one time interval on the machine to each job. A schedule is *feasible* if no two time intervals on the machine overlap, and if it meets a number of specific requirements concerning the machine environment and the job characteristics. A schedule is *optimal* if it minimizes a given optimality criterion.

We consider only *deterministic* scheduling problems. All information that defines problem instances is known with certainty in advance. Deterministic scheduling is in-

cluded in the area of combinatorial optimization. Certainly, some of the techniques for combinatorial optimization can be used for scheduling problems. It is an obvious extension to assume that some of the problem instances are subject to random fluctuations. This induces the area of *stochastic* machine scheduling.

Because of the huge variety of machine scheduling problems, several authors proposed various schemes for classifying scheduling problems. We shall follow that of Graham, Lawler, Lenstra, and Rinnooy Kan [37]. It consists of three fields $\alpha|\beta|\gamma$ for specifying problems. We only explain here the notations which are dealt with in this thesis.

The first field α indicates the machine environment. If $\alpha = 1$, there is only a single machine. If $\alpha = F2$, we have a flow shop with two machines.

The second field β is used mainly for the job characteristics. For example, *pmtn* means that preemption is allowed, *prec* means that a non-empty *precedence relation* between jobs is specified, and r_j means that non-identical release dates are given. Occasionally, this field contains additional characteristics such as $p_j = 1$ for unit processing requirement and *nmit* for the requirement that no machine idle time is allowed before the completion of the last job.

If β is empty, then the default assumptions apply. This means that preemption of jobs is not permitted, that no precedence relation is put, that all jobs are available at time zero, that there are no deadlines, and that the processing requirements are arbitrary positive integers.

The third field γ refers to the optimality criterion. It is usually based on simple quantities (computed for each job J_j) such as,

- the *completion time* C_j,
- the *lateness* $L_j = C_j - d_j$,
- the *tardiness* $T_j = \max\{0, C_i - d_i\}$,
- the *unit penalty* $U_j = 0$ if $C_j \leq d_j$, $U_j = 1$ otherwise.

The optimality criteria commonly used are

- the *maximum lateness* $L_{\max} = \max L_j$,
- the *sum of completion times* $\sum C_j$,
- the *sum of tardiness* $\sum T_j$,
- the *number of late jobs* $\sum U_j$.

Traditionally, there has been a enormous number of papers concerning the minimization of a single optimality criterion which is regular, i.e., nondecreasing in each of the job completion times. In this thesis, in addition to such minimization, we also consider the problems of minimizing non-regular criteria, which involve, e.g., the range of lateness of jobs. The scheduling problems with non-regular criteria are relatively unexplored.

For further detail on sequencing and scheduling, see e.g. the survey by Lawler, Lenstra, Rinnooy Kan, and Shmoys [55].

2.2 Generalized due dates

In 1986, Hall [40] introduced a new type of due dates called *generalized due dates*. Each of generalized due dates is not given for a particular job, but they are given for a set of jobs. It does not matter which job is completed by each generalized due date. The first generalized due date indicates the time by which at least one of the jobs should be completed, the second one gives the time by which at least two of the jobs should be completed, and so on.

This new class of scheduling problems is useful in a flexible manufacturing system (FMS) environment, of the kind described by Stecke and Solberg [85]. The aspect of flexibility, discussed by Browne, Dubois, Rathmill, Sethi, and Stecke [18], arises in reductions in machine setup costs and efficient change of component mix, making the order of job arrival less important, a situation which lends itself to the generalized due date model. These also include public utility planning, survey design, and robot scheduling. Automated inspection of work is also a good application for scheduling with generalized due dates.

The following example can be found in the thesis by Yan [110] on the burn-in test of integrated circuits (IC) in an IC company. To find the yield of a batch of wafers just processed and diced, burn-in tests are performed to ten chips on the first date, and twenty more to be tested on the second date, and another twenty to be tested on the third date. In this case, the chances of chips being tested are totally random. Therefore, we obtain three different generalized due dates.

To illustrate the difference between the traditional and Hall's model, we consider lateness of jobs in a schedule. In the traditional model, each job J_i has a non-negative number d_i as its traditional due date. In Hall's model, a set of all jobs has a non-decreasing sequence

$$\delta_1 \leq \delta_2 \leq \cdots \leq \delta_n$$

of non-negative numbers as its generalized due dates. In both models, for each $1 \leq i \leq n$, each schedule S determines uniquely the job $J_{S(i)}$ in the ith position of schedule S, that is, an order (sequence)

$$(S(1), S(2), \ldots, S(n))$$

in which the jobs are processed on the machine, and the completion time $C_i(S)$ of job J_i in schedule S.

The lateness $L_{S(i)}(S)$ of the job $J_{S(i)}$ in the ith position of schedule S under the traditional model is given by

$$L_{S(i)}(S) = C_{S(i)}(S) - d_{S(i)},$$

whereas the lateness $L^H_{S(i)}(S)$ of the same job $J_{S(i)}$ under Hall's model is given by

$$L^H_{S(i)}(S) = C_{S(i)}(S) - \delta_i.$$

For example, if

$$\begin{array}{lll} p_1 = 3, & p_2 = 2, & p_3 = 5, \\ d_1 = 4, & d_2 = 7, & d_3 = 10, \\ \delta_1 = 4, & \delta_2 = 7, & \delta_3 = 10, \end{array}$$

then, for the permutation schedule given by the sequence (J_1, J_3, J_2), we have

$$\begin{array}{lll} L_1 = -1, & L_2 = 3, & L_3 = -2, \\ L_1^H = -1, & L_2^H = 0, & L_3^H = 1. \end{array}$$

2.2.1 Complexity for the traditional and generalized due date models

Since introduction of generalized due dates by Hall [40], there have been several papers dealing with scheduling problems involving this type of due dates (see e.g. the papers by Sriskandarajah [84], Hall, Sethi, and Sriskandarajah [41], Wong, Yan, and Young [109], and Young, Wong, Yiu, and Yan [111]).

We can expect a variety of changes in results concerning the generalized due date counterparts of the traditional scheduling problems. Table 2.1 shows that problems involving generalized due dates may be easier, harder, or equally difficult as their traditional counterparts. This table suggests that the min-max problems tend to be harder and the min-sum problems tend to be easier for the problems involving generalized due dates. (see also Table 1 in the paper by Hall, Sethi, and Sriskandarajah [41] for non-single machine scheduling problems.)

2.2.2 Useful sequencing rules

The following sequencing rules are useful for many problems. We also use them later.

Shortest processing time (SPT) rule: sequence all jobs in non-decreasing order of processing times

$$p_{S(1)} \leq p_{S(2)} \leq \cdots \leq p_{S(n)}.$$

Longest processing time (LPT) rule: sequence the jobs in non-increasing order of processing times

$$p_{S(1)} \geq p_{S(2)} \geq \cdots \geq p_{S(n)}.$$

Earliest due date (EDD) rule: sequence the jobs in non-decreasing order of (traditional) due dates

$$d_{S(1)} \leq d_{S(2)} \leq \cdots \leq d_{S(n)}.$$

Minimum slack time (MST) rule: sequence the jobs in non-decreasing order of slack times

$$d_{S(1)} - p_{S(1)} \leq d_{S(2)} - p_{S(2)} \leq \cdots \leq d_{S(n)} - p_{S(n)}.$$

2.2.3 Sequence-dependent due dates

We see that generalized due dates are independent of jobs. However, they depend in a special way on the sequence in which the jobs are scheduled. Here, we propose more generalized sequence-dependent due dates, which are arbitrary functions of the positions of jobs, called *sequence-dependent due dates*.

Problem (notation for traditional model)	Generalized due date model	Traditional due date model
$1\|\|L_{\max}$	Polynomially solvable [40]	Polynomially solvable [49]
$1\|r_j\|L_{\max}$	NP-hard [41]	NP-hard [59]
$1\|pmtn, r_j\|L_{\max}$	Polynomially solvable [41]	Polynomially solvable [62, 9]
$1\|prec\|L_{\max}$	NP-hard even if $prec = chain$ [84]	Polynomially solvable [53]
$1\|C_{S(j)} \leq \delta_j\|\sum w_j C_j$	NP-hard [41]	NP-hard [59]
$1\|\|\sum T_j$	Polynomially solvable [40]	NP-hard [22]
$1\|r_j\|\sum T_j$	NP-hard [40]	NP-hard [59]
$1\|prec, p_j = 1\|\sum T_j$	Polynomially solvable [41]	NP-hard [56]
$1\|\|\sum w_j T_j$	NP-hard [84]	NP-hard [54]
$1\|prec\|\sum T_j$	NP-hard even if $prec = chain$ [84]	NP-hard [56, 57]
$1\|pmtn, prec\|\sum T_j$	NP-hard even if $prec = chain$ [84]	NP-hard [22]
$1\|p_j = 1\|\sum w_j T_j$	Polynomially solvable [109]	Polynomially solvable [56]
$1\|prec, p_j = 1\|\sum w_j T_j$	NP-hard even if $prec = chain$ [109]	NP-hard even if $w_j = 1$ and $prec = chain$ [56, 57, 58, 60]
$1\|r_j, p_j = 1\|\sum T_j$	Polynomially solvable [111]	Polynomially solvable even for the weighted case [88]
$1\|\|\sum U_j$	Polynomially solvable [40]	Polynomially solvable [69]
$1\|\|\sum w_j U_j$	NP-hard [40]	NP-hard [50]
$1\|r_j\|\sum U_j$	NP-hard [41]	NP-hard [59]
$1\|prec, p_j = 1\|\sum U_j$	Polynomially solvable [41]	NP-hard [57]
$1\|prec\|\sum U_j$	NP-hard even if $prec = chain$ [84]	NP-hard [56, 57]
$1\|pmtn, prec\|\sum U_j$	NP-hard even if $prec = chain$ [84]	Open
$1\|p_j = 1\|\sum w_j U_j$	Polynomially solvable [109]	Polynomially solvable [56]
$1\|prec, p_j = 1\|\sum w_j U_j$	NP-hard even if $prec = chain$ [109]	NP-hard even if $w_j = 1$ and $prec = chain$ [57]
$1\|r_j, p_j = 1\|\sum U_j$	Polynomially solvable [111]	Polynomially solvable even for the weighted case [88]

Table 2.1: Complexity status of single machine scheduling problems with generalized due dates.

This might be useful and natural, when we see that there is also a new model of the processing times of jobs which are dependent on their starting time, which is introduce by Berger [44] (see also [108, 35, 34]). This model is motivated by military applications in which a job consists of destroying an aerial threat and its processing time decreases with time as the threat gets closer.

From this point of view, we obtain the traditional concept and Hall's concept as two special cases of sequence-dependent due dates $D_1(S), D_2(S), \ldots, D_n(S)$. If we take constant (sequence-independent) functions

$$D_i(S) = d_i \quad \text{for } 1 \leq i \leq n,$$

then we obtain the traditional concept. If we take (non-constant) functions

$$D_i(S) = \delta_{S^{-1}(i)} \quad \text{for } 1 \leq i \leq n,$$

then we obtain Hall's concept.

2.3 Generalized release dates

We propose a new type of release dates, called *generalized release dates*. They are related to the traditional release dates in similar way as the generalized due dates to the traditional ones.

Each of generalized release dates is not given for a particular job, but they are given for a set of jobs. It does not matter which job becomes available for processing by each generalized release date. The first generalized release date indicates the time at which one job becomes available for processing, the second one gives the time at which another job becomes available, and so on.

This new class of scheduling problems is useful, e.g., in the following situation. Suppose that we get material at some points of time, which can be turned into three different kinds of goods. This induces generalized release dates. The times needed to make different kinds of goods can be different. This induces processing times. Agnetis, Macchiaroli, Pacciarelli, and Rossi [1] also described the practical situations where this model is appropriate.

To illustrate the difference between the traditional and generalized release date model, we consider optimal schedules for the maximum lateness problems of jobs with release dates. In the traditional model, each job J_i has a non-negative number r_i as its traditional release date. In the generalized release date model, a set of all jobs has a non-decreasing sequence

$$\rho_1 \leq \rho_2 \leq \cdots \leq \rho_n$$

of non-negative numbers as its generalized release dates.

For example, if

$$\begin{array}{lll} p_1 = 3, & p_2 = 5, & p_3 = 2, \\ d_1 = 4, & d_2 = 10, & d_3 = 7, \\ r_1 = 0, & r_2 = 2, & r_3 = 3, \end{array}$$

then, the optimal permutation schedule can be given by the sequence (J_1, J_2, J_3), which gives

$$L_1 = -1, \quad L_2 = -2, \quad L_3 = 3, \quad L_{\max} = 3.$$

On the other hand, if we replace the traditional release dates with the generalized release dates

$$\rho_1 = 0, \quad \rho_2 = 2, \quad \rho_3 = 7,$$

then, the optimal permutation schedule can be given by the sequence (J_1, J_3, J_2), which gives

$$L_1 = -1, \quad L_2 = -2, \quad L_3 = 0, \quad L_{\max} = 0.$$

2.4 Fuzzy due dates

In 1992, Ishii, Tada, and Masuda [48] have proposed to replace the ordinary due dates by *fuzzy due dates*. Most papers on scheduling problems assume that due dates are given numbers. A degree of satisfaction is associated with each job. It is determined according to its completion time and denoted with a certain membership function of a fuzzy set defined on nonnegative real numbers. If its completion time is before or on time with respect to a ordinary due date, we are satisfied completely. If late, the degree of satisfaction decreases, i.e., the degree of dissatisfaction increases.

In the simplest form of Ishii–Tada–Masuda's model, each ordinary due date d_i of job J_i is replaced by a fuzzy due date whose membership function μ_i is defined by

$$\mu_i(t) = \begin{cases} 1 & \text{if } t \leq d_i, \\ 1 - (t - d_i)/e_i & \text{if } d_i < t \leq d_i + e_i, \\ 0 & \text{if } d_i + e_i < t, \end{cases}$$

where e_i is a given positive number. Obviously, the ordinary due date d_i can be interpreted as the limit case for e_i converging to zero, that is, as

$$\mu_i(t) = \begin{cases} 1 & \text{if } t \leq d_i, \\ 0 & \text{if } d_i < t. \end{cases}$$

Scheduling problems with fuzzy due dates arise naturally in real life. For instance, consider a manufacturing company satisfying demand in a remote place. Goods should be produced in time for the departure of a regular flight. If late, an express delivery can recover the delay, but its extra cost reduces the satisfaction of the company. Another example can be found in a recent paper by Han, Ishii, and Fujii [43].

Other fuzzy versions of scheduling specification, e.g., precedence constraint can be found in the theses by Tada [87] and Han [42], and the paper by Ishii and Tada [47].

Outline of this part

The remainder of this part is divided into four chapters. First, we consider the problems with generalized due dates in Chapter 3. Especially, the problems of minimizing the maximum absolute lateness and range of lateness under generalized due dates are studied. Then, in Chapter 4, we investigate the problems with both traditional and generalized due dates. In particular, we consider the problem to minimize the maximum lateness with both traditional and generalized due dates.

In addition to scheduling involving generalized due dates, in Chapter 5, we consider scheduling with generalized release dates. Particularly, we consider the problems to minimize the sum of completion times with generalized release dates which have constant intervals. These are equivalent to two machine flow shop problems to minimize the sum of completion times where the processing times on the first machine are the same.

Finally, Chapter 6 deals with basic problems involving fuzzy due dates.

Chapter 3

Scheduling with generalized due dates

In this chapter, we investigate the problems of minimizing the maximum absolute lateness and range of lateness under generalized due dates on a single machine. In contrast to the traditional due date cases, we show that both problems are NP-hard in the strong sense. Furthermore, we present simple approximation algorithms for these problems, and show that they achieve the performance ratios of n for the problem of minimizing the maximum absolute lateness and of $\lceil n/2 \rceil$ for the problem of minimizing the range of lateness, where n is the number of jobs and $\lceil x \rceil$ is the smallest integer no less than x.

3.1 Introduction

3.1.1 Background and motivation

Most research in scheduling involving generalized due dates has been concerned with establishing the complexity status of the problems whose traditional counterparts have regular objective functions [40, 84, 41, 109, 111]. Little is known about the problems whose traditional counterparts have non-regular objective functions, and about approximation algorithms for the problems involving generalized due dates.

In what follows, we are concerned with single machine problems involving non-regular objective functions with generalized due dates. Our two main objective functions are the maximum absolute lateness, that is, the maximum of the absolute values of the maximum and minimum lateness, and the range of lateness, that is, the difference between the maximum and minimum lateness.

This scheduling problem is important in real life whenever it is desirable to give equal treatment to all jobs (customers), i.e., the lateness of all jobs should be as equal as possible. In particular, the criteria is natural, as mentioned in [33], in scheduling a sequence of experiments that depend on predetermined external events (such as the position of the sum) and in scheduling manufacturing process steps to coordinate with deliveries (seeking to achieve something similar to just-in-time inventory control, in cases where delivery times are less adjustable than process steps). The maximum absolute lateness problem with traditional due dates has been considered in [33]. The range of lateness problem with traditional due dates has also been considered in [38, 101, 61, 45].

The objective functions we are interested in are defined as follows.

Traditional due date model	Generalized due date model
$L_{\max}(S) = \max_{1 \le i \le n} L_i(S)$	$L^H_{\max}(S) = \max_{1 \le i \le n} L^H_i(S)$
$L_{\min}(S) = \min_{1 \le i \le n} L_i(S)$	$L^H_{\min}(S) = \min_{1 \le i \le n} L^H_i(S)$
$\Delta L(S) = L_{\max}(S) - L_{\min}(S)$	$\Delta L^H(S) = L^H_{\max}(S) - L^H_{\min}(S)$
$L_{\mathrm{abs}}(S) = \max_{1 \le i \le n} \lvert L_i(S) \rvert$	$L^H_{\mathrm{abs}}(S) = \max_{1 \le i \le n} \lvert L^H_i(S) \rvert$

3.1.2 Main results

Main results can be summarized as follows. First, we show that the problems of minimizing the maximum absolute lateness and the range of lateness are NP-hard in the strong sense, both with and without allowing for machine idle time. Second, for all of these problems, we give simple efficient approximation algorithms based on the first-fit strategy.

We show that they achieve the performance ratios of n for the problem of minimizing the maximum absolute lateness and of $\lceil n/2 \rceil$ for the problem of minimizing the range of lateness, where n is the number of jobs and $\lceil x \rceil$ is the smallest integer no less than x.

3.2 Complexity

In this section, we begin with surveying the maximum and minimum lateness problems. For these problems, simple sequencing rules give optimal schedules. Then, we investigate the complexity of the problems of minimizing the maximum absolute lateness and minimizing the range of lateness both under and without the requirement that machine idle time is forbidden.

Proposition 3.1 *Each permutation schedule generated by the*

- *EDD rule is optimal for the* $1||L_{\max}$ *problem,*
- *MST rule is optimal for the* $1|nmit| - L_{\min}$ *problem,*
- *SPT rule is optimal for the* $1||L^H_{\max}$ *problem,*
- *LPT rule is optimal for the* $1|nmit| - L^H_{\min}$ *problem.*

Proof. All these results can easily be proved by a straightforward adjacent pairwise interchange argument. ∎

The first case is also known as Jackson's rule [49]. The second case, a mirror image of Jackson's rule, was already presented in [8, 20].

Now, we investigate the problem of minimizing the maximum absolute lateness. Garey, Tarjan, and Wilfong [33] proved the following proposition.

Proposition 3.2 *The problems*

$$1||L_{\mathrm{abs}} \quad \textit{and} \quad 1|nmit|L_{\mathrm{abs}}$$

can be solved in polynomial time.

In contrast to Proposition 3.2, we have the following result concerning the generalized due date model.

Theorem 3.3 *The problems*

$$1||L_{\max}^H \quad \textit{and} \quad 1|nmit|L_{\max}^H$$

are NP-hard in the strong sense.

Proof. First, we consider the problem $1|nmit|L_{\max}^H$. We will prove its strong NP-hardness by a polynomial reduction from 3-partition problem, which is known to be NP-hard in the strong sense [31].

Suppose we have an instance of 3-partition problem, i.e., suppose we have a positive integer B and a family $A = \{a_1, a_2, \ldots, a_{3n}\}$ of positive integers such that $\sum_{j=1}^{3n} a_j = nB$ and $B/4 < a_j < B/2$ for $1 \leq j \leq 3n$. For this instance, we construct the following instance of the problem $1|nmit|L_{\max}^H$ with $4n$ jobs $J_1, J_2, \ldots, J_{4n}$:

- the processing times $p_1, p_2, \ldots, p_{4n}$ are given by

$$\begin{aligned} p_i &= a_i \qquad \text{for } 1 \leq i \leq 3n, \\ p_i &= B+1 \quad \text{for } 3n+1 \leq i \leq 4n, \end{aligned}$$

- generalized due dates $\delta_1, \delta_2, \ldots, \delta_{4n}$ are given by

$$\delta_i = (2B+1)\lceil i/4 \rceil - B/2 \quad \text{for } 1 \leq i \leq 4n.$$

For easy reference, we call jobs $J_1, J_2, \ldots, J_{3n}$ a-type and $J_{3n+1}, J_{3n+2}, \ldots, J_{4n}$ b-type.

We now show that this instance has a schedule with the maximum absolute lateness $B/2$ iff A has a desired partition. Suppose that A has a partition $A = A_1 \cup A_2 \cup \cdots \cup A_n$ such that $\sum_{a_j \in A_i} a_j = B$ for $1 \leq i \leq n$. Without loss of generality, we assume that $A_i = \{a_{3i-2}, a_{3i-1}, a_{3i}\}$ for $1 \leq i \leq n$. It is easy to check that the permutation schedule

$$(J_{3n+1}, J_1, J_2, J_3, J_{3n+2}, J_4, J_5, J_6, \ldots, J_{4n}, J_{3n-2}, J_{3n-1}, J_{3n})$$

is feasible and gives the maximum absolute lateness $B/2$.

On the other hand, suppose that there exists a schedule with the maximum absolute lateness $B/2$. Notice that all feasible schedules give the maximum lateness no less than $B/2$ and the minimum lateness no greater than $-B/2$, thus, the maximum absolute lateness no less than $B/2$.

Now, let us show that, in the schedule under consideration, a b-type job is scheduled in the $(4i-3)$rd position from $(2B+1)(i-1)$ to $(2B+1)i-B$ for each $1 \leq i \leq n$. Indeed, suppose that an a-type job is scheduled in the $(4i-3)$rd position for some $i > 1$. The lateness of this job must be at least $-B/2$. Since the processing time of the job is less than $B/2$, the lateness of the job in the $(4i-4)$th position must be greater than $B/2$, which contradicts our assumption. The fact that no a-type job is scheduled in the first position follows from the inequalities

$$p_k - \delta_1 = a_k - \delta_1 < -B/2 \quad \text{for } 1 \leq k \leq 3n.$$

A b-type job in the first position must be scheduled from the beginning because of the constraint $nmit$. Suppose that a b-type job in the $(4i-3)$rd position is not scheduled from $(2B+1)(i-1)$ to $(2B+1)i-B$ for some $i>1$. The lateness of this job must be at least $-B/2$. Since the processing time of the job is $B+1$, the lateness of the job in the $(4i-4)$th position must be greater than $B/2$, which contradicts our assumption.

The positions of b-type jobs divide the whole time interval where the machine runs into n time intervals of the same length B, in which only a-type jobs are scheduled. This implies A has a partition $A=A_1\cup A_2\cup\cdots\cup A_n$ such that $\sum_{a_j\in A_i}a_j=B$ for $1\le i\le n$, where the jobs from the $(4i-2)$nd position to the $(4i)$th correspond to the elements in A_i for each $1\le i\le n$.

Next, we consider the problem $1||L^H_{\rm abs}$. Notice that if the machine idle time is inserted, the maximum lateness of $B/2$ cannot be attained, neither can the maximum absolute lateness. So, even if we allow the machine to be idle, no optimal schedule has the machine idle time. This enable us to apply a similar argument as for the $nmit$ version of the problem. ∎

Next, we investigate the problems which are related to the problems of minimizing the range of lateness.

Proposition 3.4 *The problems*

$$\begin{array}{l} 1|L_{\rm min}\ge m|L_{\rm max},\\ 1|L_{\rm min}\ge m,nmit|L_{\rm max},\\ 1|L_{\rm max}\le m|-L_{\rm min},\quad \textit{and}\\ 1|L_{\rm max}\le m,nmit|-L_{\rm min}\end{array}$$

can be solved in polynomial time.

Proof. Efficient algorithms for these problems can be derived by modifying EDD and MST sequencings presented by Hoogeveen [45] and Liao and Huang [61]. ∎

The modifications used for Proposition 3.4 no longer guarantee the optimality in the generalized due date versions. We show that all these problems are NP-hard in the strong sense.

Theorem 3.5 *The problems*

$$\begin{array}{l} 1|L^H_{\rm min}\ge m|L^H_{\rm max},\\ 1|L^H_{\rm min}\ge m,nmit|L^H_{\rm max},\\ 1|L^H_{\rm max}\le m|-L^H_{\rm min},\quad \textit{and}\\ 1|L^H_{\rm max}\le m,nmit|-L^H_{\rm min}\end{array}$$

are NP-hard in the strong sense.

Proof. We only consider the problem $1|L^H_{\rm min}\ge m|L^H_{\rm max}$. A similar argument applies to the remaining problems.

For any number $x\ge 0$, the problem to decide whether there exists a feasible schedule which attains $L^H_{\rm abs}\le x$ is equivalent to that to decide whether there exists a feasible schedule which attains $L^H_{\rm max}\le x$ with the constraint $L^H_{\rm min}\ge -x$.

Thus, the maximum absolute lateness problem can be regarded as the special version of the problem $1|L^H_{\min} \geq m|L^H_{\max}$. This implies that the problem $1|L^H_{\min} \geq m|L^H_{\max}$ is as hard as the problem $1||L^H_{\rm abs}$. ∎

Finally, we investigate the complexity of the problems of minimizing the range of lateness. Based on Proposition 3.4, Hoogeveen [45] proved the following proposition.

Proposition 3.6 *The problems*

$$1||\Delta L \quad \textit{and} \quad 1|nmit|\Delta L$$

can be solved in polynomial time.

On the other hand, we have the following two theorems concerning the generalized due date model.

Theorem 3.7 *The problem* $1|nmit|\Delta L^H$ *is NP-hard in the strong sense.*

Proof. We transform 3-partition problem to the range of lateness problem with the constraint *nmit* by constructing the same lateness problem as in the proof of Theorem 3.3.

Notice that all feasible schedules give the maximum lateness no less than $B/2$ and the minimum lateness no greater than $-B/2$, thus, the range of lateness no less than B. By a similar argument as in the proof of Theorem 3.3, we can show that the problem above has a schedule with the range of lateness equal to B (the maximum lateness $B/2$ and the minimum lateness $-B/2$) iff A has a desired partition. ∎

Theorem 3.8 *The problem* $1||\Delta L^H$ *is NP-hard in the strong sense.*

Proof. We transform 3-partition problem to the range of lateness problem by constructing the same lateness problem as in the proof of Theorem 3.3.

We now show that the problem above has a schedule with the range of lateness equal to B iff A has a desired partition. Suppose that A has a partition. It is easy to check that the following schedule has the range of lateness equal to B: the order of the jobs is the same as in the proof of Theorem 3.3 and there is no machine idle time in the schedule.

On the other hand, suppose that there exists a schedule whose range of lateness is B. Let C be the sum of the intervals where the machine is idle. Then, all feasible schedules give the maximum lateness no less than $B/2 + C$, and the minimum lateness no greater than $-B/2 + C$, thus, the range of lateness no less than B.

We show that the sum of the processing times of the jobs in the $(4i-2)$nd position to the $(4i)$th is exactly B for all $1 \leq i \leq n$. Suppose the sum of the processing times of the jobs in the $(4i-2)$nd position to the $(4i)$th is greater than B for some $1 \leq i \leq n$. Then, the difference between the completion time of the job in the $(4i-3)$rd position and that in the $(4i)$th is greater than B, which prevents us from attaining the range of lateness equal to B.

Now, suppose the sum of the processing times of the jobs in the $(4i-2)$nd position to the $(4i)$th is less than B for some $1 \leq i \leq n$. Then, the sum of the processing times of the jobs in the $(4j-2)$nd position to the $(4j)$th is greater than B for some $1 \leq j \leq n$, $j \neq i$, which we have shown to be impossible. Therefore, the sum of the processing times

of the jobs in the $(4i-2)$nd position to the $(4i)$th is exactly B for all $1 \leq i \leq n$. It follows that there are only a-type jobs in the $(4i-2)$nd position to the $(4i)$th. Consequently, A has a partition $A = A_1 \cup A_2 \cup \cdots \cup A_n$ such that $\sum_{a_j \in A_i} a_j = B$ for $1 \leq i \leq n$, where the jobs from the $(4i-2)$nd position to the $(4i)$th correspond to the elements in A_i for each $1 \leq i \leq n$. ∎

3.3 Approximation algorithms

In this section, we present two simple approximation algorithms for the problems $1||L^H_{\max}$, $1|nmit|L^H_{\max}$, $1||\Delta L^H$, and $1|nmit|\Delta L^H$. The algorithms are based on the first-fit strategy.

First, we introduce Algorithm A which works for the problems of minimizing $L^H_{\max}$ and minimizing ΔL^H without allowing for machine idle time. The algorithm returns the resulting schedule A as a permutation, i.e., A returns the index $A(i)$ of the job in the ith position for each i, $1 \leq i \leq n$.

Algorithm A$(p_1, p_2, \ldots, p_n, \delta_1, \delta_2, \ldots, \delta_n)$
  $\delta_0 \leftarrow 0$
  for $i = 1$ to n do
    $a_i \leftarrow \delta_i - \delta_{i-1}$
  Sort all a_i's
  Sort all p_i's
  $I \leftarrow \{1, 2, \ldots, n\}$
  $J \leftarrow \{1, 2, \ldots, n\}$
  while $I \neq \emptyset$ do
    Choose i such that $a_i = \min_{k \in I} a_k$
    Choose j such that $p_j = \min_{k \in J} p_k$
    $A(i) \leftarrow j$
    $I \leftarrow I \setminus \{i\}$
    $J \leftarrow J \setminus \{j\}$
  output(A)
end

The time complexity of Algorithm A is $O(n \log n)$, if we use a fast sorting scheme. First, we prove the following lemma which plays an important role in the proofs of establishing the performance guarantees.

Lemma 3.9 *For each schedule S, we have*

$$\max_i \{p_{A(i)} - a_i\} \leq \max_i \{p_{S(i)} - a_i\}$$

and

$$\min_i \{p_{A(i)} - a_i\} \geq \min_i \{p_{S(i)} - a_i\}.$$

Proof. We only verify the validity of the first inequality. The proof of the second one is analogous. Without loss of generality, we assume that $p_1 \leq p_2 \leq \cdots \leq p_n$. The proof is

by contradiction. Suppose that there exists a schedule S such that $p_{A(j)} - a_j > p_{S(k)} - a_k$, where j and k are such that $p_{A(j)} - a_j = \max_i\{p_{A(i)} - a_i\}$ and $p_{S(k)} - a_k = \max_i\{p_{S(i)} - a_i\}$.

Since $p_{A(j)} - a_j > p_{S(k)} - a_k \geq p_{S(j)} - a_j$, we have $p_{A(j)} > p_{S(j)}$, consequently, $A(j) > S(j)$. There are at most $(A(j) - 2)$ i's such that $i \neq j$ and $p_{S(i)} < p_{A(j)}$. But, there are at least $(A(j) - 1)$ i's such that $i \neq j$ and $a_i \leq a_j$. Therefore, there exists i, $i \neq j$ such that $a_i \leq a_j$ and $p_{S(i)} \geq p_{A(j)}$. Hence,

$$\begin{aligned} p_{S(k)} - a_k &\geq p_{S(i)} - a_i \\ &= p_{S(i)} - a_j + a_j - a_i \\ &\geq p_{A(j)} - a_j + a_j - a_i \\ &\geq p_{A(j)} - a_j. \end{aligned}$$

This contradicts the assumption. ∎

Now, we analyze the performance of the algorithm for the problem $1|nmit|L^H_{\rm abs}$. Let OPT be an optimal schedule for this problem. We obtain the following bound on the performance of Algorithm A.

Theorem 3.10

$$L^H_{\rm abs}(A) \leq n \times L^H_{\rm abs}(OPT).$$

Proof. Easy calculation shows that

$$\begin{aligned} L^H_{\rm abs}(OPT) &= \max\{L^H_{\rm max}(OPT), -L^H_{\rm min}(OPT)\} \\ &\geq \frac{1}{2} \times (L^H_{\rm max}(OPT) - L^H_{\rm min}(OPT)) \\ &\geq \frac{1}{2} \times \max_i |L^H_{OPT(i)} - L^H_{OPT(i-1)}| \\ &= \frac{1}{2} \times \max_i |C_{OPT(i)} - \delta_i - (C_{OPT(i-1)} - \delta_{i-1})| \\ &= \frac{1}{2} \times \max_i |p_{OPT(i)} - a_i|. \end{aligned}$$

By Lemma 3.9, we have

$$\max_i |p_{OPT(i)} - a_i| \geq \max_i |p_{A(i)} - a_i|.$$

Therefore,

$$L^H_{\rm abs}(OPT) \geq \frac{1}{2} \times \max_i |p_{A(i)} - a_i|.$$

Let $I = \{1, 2, \ldots, n\}$, and let J and K be the set of the indices such that $p_{A(i)} \geq a_i$ for all $i \in J$ and $p_{A(i)} < a_i$ for all $i \in K$, respectively. From the definition of $L^H_{\rm abs}$ and a_i, it follows that

$$\begin{aligned} L^H_{\rm abs}(A) &= \max\{L^H_{\rm max}(A), -L^H_{\rm min}(A)\} \\ &\leq \max\{\sum_{i \in J}(p_{A(i)} - a_i), -\sum_{i \in K}(p_{A(i)} - a_i)\}. \end{aligned}$$

First, we assume that $|J| = n/2$. Then we have

$$\begin{aligned} L^H_{\rm abs}(A) &\leq \max\{|J| \times \max_{i\in J}\{p_{A(i)} - a_i\}, -|K| \times \min_{i\in K}\{p_{A(i)} - a_i\}\} \\ &\leq \max\{\frac{n}{2} \times \max_i\{p_{A(i)} - a_i\}, -\frac{n}{2} \times \min_i\{p_{A(i)} - a_i\}\} \\ &\leq \frac{n}{2} \times \max_i |p_{A(i)} - a_i|. \end{aligned}$$

Since we have shown that

$$\frac{1}{2} \times \max_i |p_{A(i)} - a_i| \leq L^H_{\rm abs}(OPT),$$

we conclude that

$$L^H_{\rm abs}(A) \leq n \times L^H_{\rm abs}(OPT).$$

Next, we assume that $|J| \neq n/2$. If $|J| \leq (n-1)/2$, then we have

$$\begin{aligned} L^H_{\rm abs}(A) &\leq \max\{\sum_{i\in J}(p_{A(i)} - a_i), -\sum_{i\in I\setminus J}(p_{A(i)} - a_i)\} \\ &= \max\{\sum_{i\in J}(p_{A(i)} - a_i), \sum_{i\in J}(p_{A(i)} - a_i) - \sum_{i\in I}(p_{A(i)} - a_i)\} \\ &\leq \sum_{i\in J}(p_{A(i)} - a_i) + |\sum_{i\in I}(p_{A(i)} - a_i)| \\ &\leq |J| \times \max_{i\in J}\{p_{A(i)} - a_i\} + |C_{A(n)} - \delta_n| \\ &\leq \frac{n-1}{2} \times \max_i |p_{A(i)} - a_i| + |L^H_{A(n)}|. \end{aligned}$$

If $|J| \geq (n+1)/2$, then we have

$$\begin{aligned} L^H_{\rm abs}(A) &= \max\{\sum_{i\in I\setminus K}(p_{A(i)} - a_i), -\sum_{i\in K}(p_{A(i)} - a_i)\} \\ &= \max\{\sum_{i\in I}(p_{A(i)} - a_i) - \sum_{i\in K}(p_{A(i)} - a_i), -\sum_{i\in K}(p_{A(i)} - a_i)\} \\ &\leq |\sum_{i\in I}(p_{A(i)} - a_i)| - \sum_{i\in K}(p_{A(i)} - a_i) \\ &\leq |C_{A(n)} - \delta_n| - |K| \times \min_{i\in K}\{p_{A(i)} - a_i\} \\ &\leq |L^H_{A(n)}| + \frac{n-1}{2} \times \max_i |p_{A(i)} - a_i|. \end{aligned}$$

Since

$$|L^H_{A(n)}| = |L^H_{OPT(n)}| \leq L^H_{\rm abs}(OPT),$$

in both cases, we obtain

$$L^H_{\rm abs}(A) \leq L^H_{\rm abs}(OPT) + \frac{n-1}{2} \times \max_i |p_{A(i)} - a_i|.$$

Combining it again with the inequality

$$\frac{1}{2} \times \max_i |p_{A(i)} - a_i| \leq L^H_{\rm abs}(OPT),$$

we again conclude that

$$L^H_{\text{abs}}(A) \leq n \times L^H_{\text{abs}}(OPT).$$

∎

Theorem 3.10 provides the performance ratio n between the optimal value of L^H_{abs} and the value induced by a schedule found by Algorithm A. The following theorem says that this ratio cannot be improved.

Theorem 3.11 *There exists an instance satisfying*

$$L^H_{\text{abs}}(A) = n \times L^H_{\text{abs}}(OPT).$$

Proof. Consider an instance of the absolute lateness problem with n jobs $J_1, J_2, \ldots, J_n$:

- the processing times $p_1, p_2, \ldots, p_n$ are given by

$$\begin{aligned} &p_1 = 4, \\ &p_i = 5 \quad \text{for } 2 \leq i \leq \lceil n/2 \rceil, \\ &p_i = 1 \quad \text{for } \lceil n/2 \rceil + 1 \leq i \leq n, \end{aligned}$$

- generalized due dates $\delta_1, \delta_2, \ldots, \delta_n$ are given by

$$\delta_i = 3i \quad \text{for } 1 \leq i \leq n.$$

From the instance above, it follows that $a_i = 3$ for all $1 \leq i \leq n$. So, Algorithm A has the possibility to give the schedules

$$S_1 = (J_1, J_2, \ldots, J_n)$$

and

$$S_2 = (J_{\lceil n/2 \rceil + 1}, J_{\lceil n/2 \rceil + 2}, \ldots, J_n, J_1, J_2, \ldots, J_{\lceil n/2 \rceil}),$$

while an optimal schedule is

$$(J_1, J_{\lceil n/2 \rceil + 1}, J_2, J_{\lceil n/2 \rceil + 2}, \ldots, J_{\lceil n/2 \rceil}, J_n).$$

Thus, the maximum absolute lateness of S_1 is n if n is odd, and that of S_2 is n if n is even, while that of the optimal schedule is one. ∎

Next, we analyze the performance of the algorithm for the problem $1|nmit|\Delta L^H$. Let OPT be an optimal schedule for this problem. We obtain the following bound on the performance of Algorithm A.

Theorem 3.12

$$\Delta L^H(A) \leq \lceil \frac{n}{2} \rceil \times \Delta L^H(OPT).$$

Proof. By Lemma 3.9, we have

$$\begin{aligned}
\Delta L^H(OPT) &= L^H_{\max}(OPT) - L^H_{\min}(OPT) \\
&\geq \max_i |L^H_{OPT(i)} - L^H_{OPT(i-1)}| \\
&= \max_i |C_{OPT(i)} - \delta_i - (C_{OPT(i-1)} - \delta_{i-1})| \\
&= \max_i |p_{OPT(i)} - a_i| \\
&\geq \max_i |p_{A(i)} - a_i|.
\end{aligned}$$

Let j and k be arbitrary positions in A at which $L^H_{\max}(A)$ and $L^H_{\min}(A)$ are achieved, respectively. If $j = k$, then $\Delta L^H(A) = 0$ and the inequality is satisfied. Let us assume that $j > k$. The proof for the case $j < k$ is analogous. From the definition of ΔL^H and a_i's, it follows that

$$\begin{aligned}
\Delta L^H(A) &= L^H_{\max}(A) - L^H_{\min}(A) \\
&= C_{A(j)} - \delta_j - (C_{A(k)} - \delta_k) \\
&\leq \sum_{i=1}^{j}(p_{A(i)} - a_i) - \sum_{i=1}^{k}(p_{A(i)} - a_i).
\end{aligned}$$

Now, we further assume that $j - k \leq \lceil n/2 \rceil$. Then we have

$$\begin{aligned}
\Delta L^H(A) &\leq \sum_{i=k+1}^{j} (p_{A(i)} - a_i) \\
&\leq (j-k) \times \max_i |p_{A(i)} - a_i| \\
&\leq \lceil \frac{n}{2} \rceil \times \max_i |p_{A(i)} - a_i|.
\end{aligned}$$

Next, we assume that $j - k \geq \lceil n/2 \rceil + 1$. Then we have

$$\begin{aligned}
\Delta L^H(A) &\leq (\sum_{i=1}^{n}(p_{A(i)} - a_i) - \sum_{i=j+1}^{n} (p_{A(i)} - a_i)) - \sum_{i=1}^{k}(p_{A(i)} - a_i) \\
&\leq C_{A(n)} - \delta_n + (n - j + k) \times \max_i |p_{A(i)} - a_i| \\
&\leq L^H_{A(n)} + (\lfloor \frac{n}{2} \rfloor - 1) \times \max_i |p_{A(i)} - a_i| \\
&\leq L^H_{A(n)} + (\lceil \frac{n}{2} \rceil - 1) \times \max_i |p_{A(i)} - a_i|,
\end{aligned}$$

where $\lfloor x \rfloor$ is the largest integer no greater than x. To conclude the proof, it suffices to observe that

$$L^H_{A(n)} = L^H_{OPT(n)} \leq \Delta L^H(OPT).$$

∎

Theorem 3.12 provides the performance ratio $\lceil n/2 \rceil$ between the optimal value of ΔL^H and the value induced by a schedule found by Algorithm A. The following theorem says that this ratio cannot be improved.

Theorem 3.13 *There exists an instance satisfying*

$$\Delta L^H(A) = \lceil \frac{n}{2} \rceil \times \Delta L^H(OPT).$$

Proof. Consider an instance of the range of lateness problem with n jobs $J_1, J_2, \ldots, J_n$:

- the processing times $p_1, p_2, \ldots, p_n$ are given by

$$\begin{aligned} p_i &= 1 \quad \text{for } 1 \le i \le \lceil n/2 \rceil, \\ p_i &= 3 \quad \text{for } \lceil n/2 \rceil + 1 \le i \le n, \end{aligned}$$

- generalized due dates $\delta_1, \delta_2, \ldots, \delta_n$ are given by

$$\delta_i = 2i \quad \text{for } 1 \le i \le n.$$

From the instance above, it follows that $a_i = 2$ for all $1 \le i \le n$. So, Algorithm A has the possibility to give the schedule

$$S = (J_1, J_2, \ldots, J_n),$$

while an optimal schedule is

$$(J_1, J_{\lceil n/2 \rceil + 1}, J_2, J_{\lceil n/2 \rceil + 2}, \ldots, J_{\lceil n/2 \rceil - 1}, J_n, J_{\lceil n/2 \rceil}),$$

if n is odd, and

$$(J_1, J_{\lceil n/2 \rceil + 1}, J_2, J_{\lceil n/2 \rceil + 2}, \ldots, J_{\lceil n/2 \rceil}, J_n),$$

if n is even. Thus, the range of lateness of S is $\lceil n/2 \rceil$, while that of the optimal schedule is one. ∎

Next, we present an approximation algorithm for the problems $1||L^H_{\text{abs}}$ and $1||\Delta L^H$. To describe a schedule S', which may involve inserted idle times, we use an ordered pair $S' = (S_1, S_2)$, where S_1 is a permutation of jobs and S_2 is a function whose value $S_2(i)$ gives the idle time inserted between jobs $J_{S_1(i-1)}$ and $J_{S_1(i)}$.

```
Algorithm A'(p_1, p_2, ..., p_n, δ_1, δ_2, ..., δ_n)
    A_1 ← Algorithm A(p_1, p_2, ..., p_n, δ_1, δ_2, ..., δ_n).
    c_0 ← 0
    for i = 1 to n do
        c_i ← c_{i-1} + p_{A_1(i)}
        if c_i < δ_i then
            A_2(i) ← δ_i − c_i
            c_i ← δ_i
        else
            A_2(i) ← 0
  output((A_1, A_2))
end
```

The time complexity of Algorithm A′ is $O(n \log n)$, if we use a fast sorting scheme. First, we introduce the following lemma which plays an important role in the proofs of establishing the performance guarantees. This lemma is an immediate consequence of Lemma 3.9.

Lemma 3.14 *For each schedule $S' = (S_1, S_2)$, and for each number b, we have*

$$\max_i\{p_{A_1(i)} - a_i + b\} \leq \max_i\{p_{S_1(i)} - a_i + b\}$$

and

$$\min_i\{p_{A_1(i)} - a_i + b\} \geq \min_i\{p_{S_1(i)} - a_i + b\}.$$

Now, we analyze the performance of the algorithm for the problem $1||L^H_{\rm abs}$. Let $OPT' = (OPT_1, OPT_2)$ be an optimal schedule for this problem. We obtain the following bound on the performance of Algorithm A$'$.

Theorem 3.15

$$L^H_{\rm abs}(A') \leq n \times L^H_{\rm abs}(OPT').$$

Proof. From Lemma 3.14, we have

$$\begin{aligned}
L^H_{\rm abs}(OPT') &= \max\{L^H_{\rm max}(OPT'), -L^H_{\rm min}(OPT')\} \\
&\geq \frac{1}{2} \times (L^H_{\rm max}(OPT') - L^H_{\rm min}(OPT')) \\
&\geq \frac{1}{2} \times \max_i |L^H_{OPT_1(i)} - L^H_{OPT_1(i-1)}| \\
&\geq \frac{1}{2} \times \max_i |C_{OPT_1(i)} - \delta_i - (C_{OPT_1(i-1)} - \delta_{i-1})| \\
&= \frac{1}{2} \times \max_i |p_{OPT_1(i)} + OPT_2(i) - a_i| \\
&\geq \frac{1}{2} \times \max_i |p_{A_1(i)} - a_i + OPT_2(i)|.
\end{aligned}$$

Let $I = \{1, 2, \ldots, n\}$, and let J and K be the set of the indices such that $p_{A(i)} \geq a_i$ for all $i \in J$ and $p_{A(i)} < a_i$ for all $i \in K$, respectively. From the definition of $L^H_{\rm abs}$ and a_i, it follows that

$$\begin{aligned}
L^H_{\rm abs}(A') &= \max\{L^H_{\rm max}(A'), -L^H_{\rm min}(A')\} \\
&\leq \max\{\sum_{i\in J}(p_{A_1(i)} - a_i), 0\} \\
&\leq \sum_{i\in J}(p_{A_1(i)} - a_i).
\end{aligned}$$

First, we assume that $|J| \leq n/2$. Then we have

$$\begin{aligned}
L^H_{\rm abs}(A') &\leq |J| \times \max_{i\in J}\{p_{A_1(i)} - a_i\} \\
&\leq \frac{n}{2} \times \max_i\{p_{A_1(i)} - a_i\} \\
&\leq \frac{n}{2} \times \max_i\{p_{A_1(i)} - a_i + OPT_2(i)\} \\
&\leq \frac{n}{2} \times \max_i |p_{A_1(i)} - a_i + OPT_2(i)|.
\end{aligned}$$

Next, we assume that $|J| \geq (n+1)/2$. We have

$$\begin{aligned} L^H_{OPT_1(n)} &= \sum_i (p_{OPT_1(i)} - a_i + OPT_2(i)) \\ &= \sum_i (p_{A_1(i)} - a_i + OPT_2(i)) \\ &\geq \sum_{i \in J} (p_{A_1(i)} - a_i) + \sum_{i \in K} (p_{A_1(i)} - a_i + OPT_2(i)). \end{aligned}$$

From this inequality, it follows that

$$\begin{aligned} L^H_{\mathrm{abs}}(A') &\leq L^H_{OPT_1(n)} - \sum_{i \in K} (p_{A_1(i)} - a_i + OPT_2(i)) \\ &\leq L^H_{OPT_1(n)} - |K| \times \min_{i \in K} \{p_{A_1(i)} - a_i + OPT_2(i)\} \\ &\leq L^H_{OPT_1(n)} - \frac{n-1}{2} \times \min_i \{p_{A_1(i)} - a_i + OPT_2(i)\} \\ &\leq L^H_{OPT_1(n)} + \frac{n-1}{2} \times \max_i |p_{A_1(i)} - a_i + OPT_2(i)|. \end{aligned}$$

To conclude the proof, it is sufficient to observe that

$$L^H_{OPT_1(n)} \leq L^H_{\mathrm{abs}}(OPT').$$

∎

Theorem 3.15 provides the performance ratio n. The following theorem says that this ratio cannot be improved.

Theorem 3.16 *There exists an instance satisfying*

$$L^H_{\mathrm{abs}}(A') = n \times L^H_{\mathrm{abs}}(OPT').$$

Proof. Identical to that of Theorem 3.11. ∎

Next, we analyze the performance of the algorithm for the problem $1||\Delta L^H$. Now, let $OPT' = (OPT_1, OPT_2)$ be an optimal schedule for this problem. We obtain the following bound on the performance of Algorithm A$'$.

Theorem 3.17

$$\Delta L^H(A') \leq \lceil \frac{n}{2} \rceil \times \Delta L^H(OPT').$$

Proof. From Lemma 3.14, we have

$$\begin{aligned} \Delta L^H(OPT') &= L^H_{\max}(OPT') - L^H_{\min}(OPT') \\ &\geq \max_i |L^H_{OPT_1(i)} - L^H_{OPT_1(i-1)}| \\ &= \max_i |C_{OPT_1(i)} - \delta_i - (C_{OPT_1(i-1)} - \delta_{i-1})| \\ &\geq \max_i |p_{OPT_1(i)} + OPT_2(i) - a_i| \\ &\geq \max_i |p_{A_1(i)} - a_i + OPT_2(i)|. \end{aligned}$$

Let j be an arbitrary position in A' at which $L^H_{\max}(A')$ is achieved. From the definition of ΔL^H and a_i's, it follows that

$$\begin{aligned}\Delta L^H(A') &= L^H_{\max}(A') - L^H_{\min}(A') \\ &\leq (C_{A_1(j)} - \delta_j) - 0 \\ &\leq \sum_{i=1}^{j}(p_{A_1(i)} - a_i).\end{aligned}$$

First, we assume that $j \leq \lceil n/2 \rceil$. Then we have

$$\begin{aligned}\Delta L^H(A') &\leq j \times \max_i\{p_{A_1(i)} - a_i\} \\ &\leq \lceil \frac{n}{2} \rceil \times \max_i\{p_{A_1(i)} - a_i + OPT_2(i)\} \\ &\leq \lceil \frac{n}{2} \rceil \times \max_i |p_{A_1(i)} - a_i + OPT_2(i)|.\end{aligned}$$

Next, we assume that $j \geq \lceil n/2 \rceil + 1$. We have

$$\begin{aligned}L^H_{OPT_1(n)} &= \sum_{i=1}^{n}(p_{OPT_1(i)} - a_i + OPT_2(i)) \\ &= \sum_{i=1}^{n}(p_{A_1(i)} - a_i + OPT_2(i)) \\ &\geq \sum_{i=1}^{j}(p_{A_1(i)} - a_i) + \sum_{i=j+1}^{n}(p_{A_1(i)} - a_i + OPT_2(i)).\end{aligned}$$

From this inequality, it follows that

$$\begin{aligned}\Delta L^H(A') &\leq L^H_{OPT_1(n)} - \sum_{i=j+1}^{n}(p_{A_1(i)} - a_i + OPT_2(i)) \\ &\leq L^H_{OPT_1(n)} - (n-j) \times \min_i\{p_{A_1(i)} - a_i + OPT_2(i)\} \\ &\leq L^H_{OPT_1(n)} + (\lfloor \frac{n}{2} \rfloor - 1) \times \max_i |p_{A_1(i)} - a_i + OPT_2(i)| \\ &\leq L^H_{OPT_1(n)} + (\lceil \frac{n}{2} \rceil - 1) \times \max_i |p_{A_1(i)} - a_i + OPT_2(i)|.\end{aligned}$$

To conclude the proof, it is sufficient to observe that

$$L^H_{OPT_1(n)} \leq \Delta L^H(OPT').$$

∎

Theorem 3.17 provides the performance ratio $\lceil n/2 \rceil$ between the optimal value of ΔL^H and the value induced by a schedule found by Algorithm A′. The following theorem says that this ratio cannot be improved.

Theorem 3.18 *There exists an instance satisfying*

$$\Delta L^H(A') = \lceil \frac{n}{2} \rceil \times \Delta L^H(OPT').$$

Proof. Identical to that of Theorem 3.13. ∎

Chapter 4

Scheduling with both traditional and generalized due dates

In this chapter, we consider single machine problems involving both traditional and generalized due dates simultaneously. We show that a polynomial time algorithm exists for the problem of minimizing $\max\{L_{\max}, L^H_{\max}\}$, where $L_{\max}$ and $L^H_{\max}$ are the maximum lateness induced by traditional and generalized due dates, respectively. We also show that a simple efficient algorithm exists for the problem of minimizing the maximum lateness induced by due dates which are natural generalization of both traditional and generalized due dates. In addition to the algorithmic results above, we show that the problems of minimizing $\max\{L^H_{\max}, -L_{\min}\}$ and $\max\{L_{\max}, -L^H_{\min}\}$ are NP-hard in the strong sense, where $L_{\min}$ and $L^H_{\min}$ are the minimum lateness induced by traditional and generalized due dates, respectively.

4.1 Introduction

4.1.1 Background and motivation

Since introduction of generalized due dates by Hall [40], there have been several papers dealing with scheduling problems involving this type of due dates [84, 41, 109, 111]. However, to the best of our knowledge, there has been no investigation of problems with both traditional and generalized due dates although they are interesting from both theoretical and practical view points.

In this chapter, we are mainly concerned with the problems minimizing the maximum of the maximum lateness induced by traditional and generalized due dates. We consider two types of such problems. Suppose that all jobs $J_1, J_2, \ldots, J_n$ have their traditional due dates $d_1, d_2, \ldots, d_n$ and that simultaneously generalized due dates $\delta_1, \delta_2, \ldots, \delta_n$ are given. Then, we can calculate the maximum lateness $L_{\max}$ induced by traditional due dates and the maximum lateness $L^H_{\max}$ induced by generalized due dates. One of the simplest functions which combine two criteria is the maximum function. We take as our first main criterion the maximum of $L_{\max}$ and $L^H_{\max}$.

This problem may be useful in many practical situations. For example, a computer dealer may face a situation to install the same computer systems to several companies. The dealer is required to install five systems per month by a computer provider. The

computer provider does not care which systems are sold. This induces generalized due dates. However, the dealer is required to install each system by its due date specified by a company. The times needed to install a system can be different, and the dates at which companies want to start to use a system can also be different. This induces processing times and traditional due dates. The dealer should make schedules which satisfy both requirements given by the provider and companies.

Suppose that all jobs $J_1, J_2, \ldots, J_n$ are partitioned into $k \leq n$ sets $J^1, J^2, \ldots, J^k$ and that each set J^i has its own generalized due dates $\delta^i_1, \ldots, \delta^i_{|J^i|}$. Notice that, if all sets $J^1, J^2, \ldots, J^k$ are singletons (i.e., $k = n$), then we have the traditional due date model, and that, if we do not partition (i.e., $k = 1$), then we have the standard generalized due date model. Thus, we can consider the traditional and generalized due date models as special cases. We take as our second main criterion the maximum lateness induced by such due dates.

This problem arises naturally as follows. A public utility company may face a situation to contract multiple services. For one service, the company is required to contract the service to ten new cities. It is required to include five within three months, a further three within six months, and all ten within one year. For another service, the company is required to contract the service to ten new cities. It is required to include three within two months, a further five within ten months, and all ten within one year. This induces generalized due dates for each set of jobs.

We are also interested in the complexity for the problems of minimizing $\max\{L^H_{\max}, -L_{\min}\}$ and $\max\{L_{\max}, -L^H_{\min}\}$, where $L_{\min}$ and $L^H_{\min}$ are the minimum lateness induced by the traditional and generalized due dates, respectively. We do not know how important these problems are in practice. However, we are concerned with them because of theoretical interests.

4.1.2 Definitions and preliminaries

The objective functions which we are interested in are defined as follows:

$$\max\{L_{\max}(S), L^H_{\max}(S)\}, \tag{4.1}$$

$$\max\{L^H_{\max}(S), -L_{\min}(S)\}, \tag{4.2}$$

$$\max\{L_{\max}(S), -L^H_{\min}(S)\}, \tag{4.3}$$

where

$$\begin{aligned} L_{\max}(S) &= \max_{1\leq i\leq n} L_i(S), \\ L^H_{\max}(S) &= \max_{1\leq i\leq n} L^H_i(S), \\ L_{\min}(S) &= \min_{1\leq i\leq n} L_i(S), \\ L^H_{\min}(S) &= \min_{1\leq i\leq n} L^H_i(S). \end{aligned}$$

In addition, we are also interested in the following objective function. Let all jobs $J_1, J_2, \ldots, J_n$ are partitioned into $k \leq n$ sets $J^1, J^2, \ldots, J^k$. A set of all jobs in J^i has a non-decreasing sequence

$$\delta^i_1 \leq \delta^i_2 \leq \cdots \leq \delta^i_{|J^i|}$$

of non-negative numbers as its generalized due dates for each $1 \leq i \leq k$. Let job $J_{S^i(j)}$ be in the jth position of partial schedule S^i in schedule S for each $1 \leq i \leq k$. Then, the

objective function which we consider is defined as follows:

$$\max_{1 \le i \le k} L^i_{\max}(S), \tag{4.4}$$

where

$$\begin{aligned} L^i_{\max}(S) &= \max_{1 \le j \le |J^i|} L^i_j(S), \\ &= \max_{1 \le j \le |J^i|} \{C_{S^i(j)}(S) - \delta^i_j\}. \end{aligned}$$

4.1.3 Main results

In Section 4.2 we survey the previous results on the complexity for the problems related to our problems. In Section 4.3, first, we give an efficient algorithm for the problem of minimizing $L^H_{\max}$ with the constraint $L_{\max} \le m$, which plays an important role in solving problem (4.1). Then, we show that a polynomial time algorithm exists for problem (4.1). In Section 4.4, we show that problem (4.4) can be solved by a simple efficient algorithm. Finally, in Section 4.5, we give NP-hardness results for problems (4.2) and (4.3).

4.2 Previous works

In this section, we survey the complexity of the problems related to our problems, i.e., the problems of minimizing $L_{\max}$, $L^H_{\max}$, $-L_{\min}$, $-L^H_{\min}$, $\max\{L_{\max}, -L_{\min}\}$, and $\max\{L^H_{\max}, -L^H_{\min}\}$.

For the first four problems, simple sequencing rules give optimal schedules.

Proposition 4.1 *Each permutation schedule generated by the*

- *EDD rule is optimal for the* $1||L_{\max}$ *problem,*
- *MST rule is optimal for the* $1|nmit| - L_{\min}$ *problem,*
- *SPT rule is optimal for the* $1||L^H_{\max}$ *problem,*
- *LPT rule is optimal for the* $1|nmit| - L^H_{\min}$ *problem.*

Proof. All these results can easily be proved by a straightforward adjacent pairwise interchange argument. ∎

The first case is also known as Jackson's rule [49]. The second case, a mirror image of Jackson's rule, was already presented in [8, 20].

Next, we mention the complexity for the problems of minimizing the maximum absolute lateness for both traditional and generalized due dates. Then, we investigate the complexity for problems involving both traditional and generalized due dates.

Let us proceed to the maximum absolute lateness problems, i.e., the problems of minimizing $\max\{L_{\max}, -L_{\min}\}$ and $\max\{L^H_{\max}, -L^H_{\min}\}$. Using the results by Garey, Tarjan, and Wilfong [33], Hoogeveen [45], and Liao and Huang [61], the following proposition concerning the traditional due date version can be proved.

Proposition 4.2 *The problems*

$$1||\max\{L_{\max}, -L_{\min}\} \quad \textit{and} \quad 1|nmit|\max\{L_{\max}, -L_{\min}\}$$

can be solved in polynomial time.

On the other hand, Tanaka and Vlach [99] proved the following proposition concerning the generalized due date version.

Proposition 4.3 *The problems*

$$1||\max\{L^H_{\max}, -L^H_{\min}\} \quad \textit{and} \quad 1|nmit|\max\{L^H_{\max}, -L^H_{\min}\}$$

are NP-hard in the strong sense.

4.3 An algorithm for minimizing $\max\{L_{\max}, L^H_{\max}\}$

In this section, we consider problem (4.1), i.e., the problem of minimizing $\max\{L_{\max}, L^H_{\max}\}$. First, we present an algorithm for minimizing $L^H_{\max}$ with the constraint $L_{\max} \leq m$, which has a relation to problem (4.1). The algorithm is based on ideas of Garey, Tarjan, and Wilfong [33], Hoogeveen [45], and Liao and Huang [61].

The algorithm returns the resulting schedule A as a permutation, i.e., A returns the index $A(i)$ of the job in the ith position for each i, $1 \leq i \leq n$. First, the algorithm finds an EDD sequencing. This sequencing gives the optimal value of $L_{\max}$. Then, the algorithm fixes a longest remaining job from the last position by checking feasibility. When the algorithm reaches the first position, an optimal schedule is found.

```
Algorithm A(p_1, p_2, ..., p_n, d_1, d_2, ..., d_n, m)
  Find an EDD sequencing A
  if L_max(A) > m then
    output("There is no feasible schedule.")
        and exit
  I ← {1, 2, ..., n}
  u ← Σ_{j=1}^{n} p_j
  for k = n down to 2 do
    Find i ∈ I such that
        (1) u − d_i ≤ m, and
        (2) p_i ≥ p_j for all j ∈ I \ {i}
                  such that u − d_j ≤ m
    A(k) ← i
    I ← I \ {i}
    u ← u − p_i
  output(A)
end
```

The time complexity of the algorithm is $O(n \log n)$ if we use a fast sorting scheme and employ a binary search for the Find statement with the table of jobs sorted by their processing times. Now, we show the correctness of Algorithm A.

Lemma 4.4 *If there exists a feasible schedule, then there exists an optimal schedule for the problem minimizing $L^H_{\max}$ with the constraint $L_{\max} \leq m$, and with the property that J_i precedes J_j for each pair J_i and J_j of jobs such that there exist an EDD and an SPT sequences in both of which J_i precedes J_j.*

Proof. Let S be an optimal schedule. If S has the required property described above, then we are done. If not, we construct another optimal schedule as follows. Since S does not have the required property, there is a pair of jobs J_k and J_l such that

- there exist an EDD and an SPT sequences in both of which J_k precedes J_l,
- J_l precedes J_k in S,
- no such pair exists with any job between J_l and J_k.

Let S' be the schedule obtained from S by interchanging J_l and J_k. To prove that S' is also optimal, it suffices to show that

1. $L_{\max}(S') \leq m$,
2. $L^H_l(S') \leq L^H_{\max}(S)$,
3. $L^H_r(S') \leq L^H_{\max}(S)$ for each r such that J_r is scheduled between J_l and J_k.

Proof of 1. We observe that $p_k \leq p_l$. Consequently $C_r(S') \leq C_r(S)$ for all r. Therefore $L_r(S') \leq L_r(S)$ for all $r \neq l$. Moreover $C_l(S') - d_l \leq C_k(S) - d_k$, because $C_l(S') = C_k(S)$ and $d_k \leq d_l$. Thus $L_l(S') \leq L_k(S)$, and we conclude that $L_{\max}(S') \leq L_{\max}(S) \leq m$.

Proof of 2. Obvious because $L^H_l(S') = L^H_k(S)$.

Proof of 3. We again observe that $p_k \leq p_l$. Therefore $L^H_r(S') \leq L^H_r(S)$ for all r.

If S' does not have the required property, we repeat this interchange, and after a finite number of steps, we obtain an optimal schedule having the required property. ∎

Theorem 4.5 *If there exists a feasible schedule, then Algorithm A yields an optimal schedule for the problem of minimizing $L^H_{\max}$ with the constraint $L_{\max} \leq m$.*

Proof. We show that a slightly stronger statement holds, Namely, Algorithm A yields an optimal schedule which satisfies the property described in Lemma 4.4.

First notice that, if an EDD sequencing is feasible, then after the algorithm fixes the job in the last position, an EDD sequencing is again feasible for the updated problem. Therefore the algorithm always delivers a feasible schedule, provided one exists.

Suppose that Algorithm A gives a schedule A which is not optimal. Let OPT be an optimal schedule having the required property. Comparing A and OPT from the last position, we meet the first difference in some position, say kth, of A and OPT. Let J_i and J_j be scheduled in the kth position in A and in OPT, respectively. Algorithm A always choose a longest possible job with respect to the constraint $L_{\max} \leq m$. So, we have $p_i \geq p_j$.

Let OPT' be the schedule where J_i and J_j are interchanged in OPT. In order to prove that A is optimal and feasible, it is sufficient to prove the following three claims:

1. OPT' is feasible with respect to the constraint $L_{\max} \leq m$,

2. OPT' is optimal,

3. OPT' can be transformed into a new schedule OPT'' which is optimal, feasible with respect to the constraint $L_{\max} \leq m$, and equal to A in the last $n-k+1$ positions, and has the required property.

Proof of 1. Since $p_i \geq p_j$, we have $C_k(OPT') \leq C_k(OPT)$. Thus $L_k(OPT') \leq L_k(OPT)$ for all k except i. Since J_i is chosen by the algorithm, we have $L_i(OPT') \leq m$, from which the claim follows.

Proof of 2. Analogous to the proof of 2 and 3 in the proof of Lemma 4.4.

Proof of 3. It is possible that there exists job J_l, scheduled between J_i and J_j in OPT, with $p_l < p_j$ and $d_i \leq d_l \leq d_j$, or with $p_l \leq p_j$ and $d_i \leq d_l < d_j$. If so, then OPT' does not have the required property. However, it follows immediately from the proof of Lemma 4.4 that OPT' can be adjusted to a new schedule OPT'' which is feasible, optimal, and equal to A in the last $n-k+1$ positions, and which has the required property.

The interchange argument as above can be repeated until A and the new schedule OPT'' are the same. This proves that schedule A is optimal and feasible with respect to the constraint $L_{\max} \leq m$, and has the required property. ∎

An efficient algorithm for the problem of minimizing $L^H_{\max}$ with the constraint $L_{\max} \leq m$ enable us to get a polynomial time algorithm for problem (4.1).

Theorem 4.6 *A polynomial time algorithm exists for the problem of minimizing* $\max\{L_{\max}, L^H_{\max}\}$.

Proof. Using Algorithm A, we can get the algorithm for the problem by using a binary search on the possible values with querying whether $L^H_{\max}$ of an optimal schedule under the constraint $L_{\max} \leq m$. The time complexity of this algorithm is $O(n \log n \log P)$, where $P = \max\{L^H_{\max}(EDD) - L^H_{\max}(SPT), L_{\max}(SPT) - L_{\max}(EDD)\}$. ∎

A polynomial time algorithm analogous to Algorithm A can be made for the problem of minimizing $-L^H_{\min}$ with the constraint $L_{\min} \geq m$ and *nmit*. Thus, a polynomial time algorithm exists for the problem of minimizing $\max\{-L_{\min}, -L^H_{\min}\}$ with the constraint *nmit*.

4.4 An algorithm for minimizing $\max L^i_{\max}$

In this section, we consider problem (4.4), i.e., the problem of minimizing $\max_{1 \leq i \leq k} L^i_{\max}$. Recall that all jobs $J_1, J_2, \ldots, J_n$ are partitioned into $k \leq n$ sets $J^1, J^2, \ldots, J^k$ and that for each $1 \leq i \leq k$, all jobs have their own generalized due dates $\delta^i_1, \ldots, \delta^i_{|J^i|}$.

The algorithm returns the resulting schedule B as a permutation similarly as Algorithm A. First, the algorithm finds an SPT ordering for each set S^i. Then, it makes traditional due dates according to SPT orderings. Finally, it finds an EDD ordering on new due dates.

Algorithm B($p^1_1, p^1_2, \ldots, p^k_{m_k}, \delta^1_1, \delta^1_2, \ldots, \delta^k_{m_k}$)
  for $i = 1$ to k do
    Find an SPT ordering SPT^i of all jobs in J^i

```
    for j = 1 to |J^i| do
      d^i_j ← δ^i_{(SPT^i)^{-1}(j)}
    Find a sequence B where all jobs are in
          non-decreasing order of d^i_j
    output(B)
  end
```

The time complexity of the algorithm is

$$\sum_{i=1}^{k} O(m_k \log m_k) + O(n \log n) = O(n \log n).$$

Now, we show the correctness of Algorithm B.

Lemma 4.7 *There exists an optimal schedule where the order of jobs in J^i is SPT for all $1 \leq i \leq k$.*

Proof. Let S be an optimal schedule. If it has the required property, then we are done. If not, we construct another optimal schedule as follows. Since S does not have the required property, there exists a pair of jobs J^i_j and J^i_k such that

- there exist an SPT sequencing in which J^i_j precedes J^i_k,
- J^i_k precedes J^i_j in S.

Let S' be the schedule obtained from S by interchanging J^i_k and J^i_j. Then, S' can be shown to be optimal by observing that the completion time of J^i_k in S' is equal to that of J^i_j in S and that the completion time of each job in S', scheduled between J^i_k and J^i_j, is no greater than that in S. Repeating such interchanges makes a schedule with the required property. ∎

Theorem 4.8 *Algorithm B yields an optimal schedule for the problem of minimizing* $\max_{1 \leq i \leq k} L^i_{\max}$.

Proof. For each $1 \leq i \leq k$, if we fix the order of all jobs in J^i in a schedule, we can regard a due date of each job in J^i as a traditional due date. Thus, we can find an optimal schedule by an EDD sequencing on all jobs. ∎

4.5 NP-hardness

In this section, we consider the complexity for problems (4.2) and (4.3), i.e., the problems of minimizing $\max\{L^H_{\max}, -L_{\min}\}$ and $\max\{L_{\max}, -L^H_{\min}\}$.

Theorem 4.9 *The problems*

$$1||\max\{L^H_{\max}, -L_{\min}\} \quad \textit{and} \quad 1|nmit|\max\{L^H_{\max}, -L_{\min}\}$$

are NP-hard in the strong sense.

Proof. First, we consider the problem $1|nmit|\max\{L^H_{\max}, -L_{\min}\}$. We will prove its strong NP-hardness by a polynomial reduction from 3-partition problem, which is known to be NP-hard in the strong sense [31].

Suppose we have an instance of 3-partition problem, i.e., suppose we have a positive integer B and a family $A = \{a_1, a_2, \ldots, a_{3n}\}$ of positive integers such that $\sum_{j=1}^{3n} a_j = nB$ and $B/4 < a_j < B/2$ for $1 \leq j \leq 3n$.

For this instance, we construct the following instance of the problem $1|nmit|\max\{L^H_{\max}, -L_{\min}\}$ with $4n$ jobs $J_1, J_2, \ldots, J_{4n}$:

- the processing times $p_1, p_2, \ldots, p_{4n}$ are given by

$$\begin{aligned} p_i &= B + a_i \quad \text{for } 1 \leq i \leq 3n, \\ p_i &= B \qquad\quad \text{for } 3n+1 \leq i \leq 4n, \end{aligned}$$

- traditional due dates $d_1, d_2, \ldots, d_{4n}$ are given by

$$\begin{aligned} d_i &= 0 \qquad\quad\ \text{for } 1 \leq i \leq 3n, \\ d_i &= 5B(i-3n) \quad \text{for } 3n+1 \leq i \leq 4n, \end{aligned}$$

- generalized due dates $\delta_1, \delta_2, \ldots, \delta_{4n}$ are given by

$$\delta_i = 5B\lceil i/4 \rceil \quad \text{for } 1 \leq i \leq 4n,$$

  where $\lceil x \rceil$ is the smallest integer no less than x.

We now show that the problem above has a schedule with $L^H_{\max} = -L_{\min} = 0$ iff A has a desired partition. Suppose that A has a partition $A = A_1 \cup A_2 \cup \cdots \cup A_n$ such that $\sum_{a_j \in A_i} a_j = B$ for $1 \leq i \leq n$. Further suppose that $A_i = \{a_{3i-2}, a_{3i-1}, a_{3i}\}$ for $1 \leq i \leq n$. It is easy to check that the schedule

$$(J_1, J_2, J_3, J_{3n+1}, J_4, J_5, J_6, J_{3n+2}, \ldots, J_{3n-2}, J_{3n-1}, J_{3n}, J_{4n})$$

gives $L^H_{\max} = -L_{\min} = 0$.

On the other hand, suppose that there exists a schedule with $L^H_{\max} = -L_{\min} = 0$. Notice that all feasible schedules give $L^H_{\max}$ no less than zero and $L_{\min}$ no greater than zero. This follows from the fact that, in any feasible schedule, the last job must give $L^H_{\max}$ no less than zero since $\delta_{4n} = \sum_{1 \leq i \leq 4n} p_i$, and job J_{4n} must give $L_{\min}$ no greater than zero since $d_{4n} = \sum_{1 \leq i \leq 4n} p_i$.

Let us consider the jobs in the first position to the fourth. We call jobs $J_1, J_2, \ldots, J_{3n}$ a-type and jobs $J_{3n+1}, J_{3n+2}, \ldots, J_{4n}$ b-type. We show that the jobs in the first position to the third are a-type and the job in the fourth position is b-type. First, suppose that there is no b-type job among the first four jobs. Then, the completion time of the job in the fourth position is greater than $5B$. So, the lateness of the job induced by generalized due dates is greater than zero, which disable us from attaining the minimum value zero of $L^H_{\max}$. Next, suppose that there is a b-type job among the first three jobs. Then, the completion time of this job is less than $4B$. So, the lateness of the job induced by traditional due dates is less than $-B < 0$, which disable us from attaining the maximum value zero of $L_{\min}$.

We can apply a similar argument as above to the jobs in the $(4i-3)$rd position to the $(4i)$th for $2 \leq i \leq n$. The positions of b-type jobs divide the whole time interval where the machine runs into n time intervals of the same length B. This implies A has a partition $A = A_1 \cup A_2 \cup \cdots \cup A_n$ such that $\sum_{a_j \in A_i} a_j = B$ for $1 \leq i \leq n$, where the jobs from the $(4i-3)$rd position to the $(4i-1)$st correspond to the elements in A_i for each $1 \leq i \leq n$.

It remains to consider the problem $1||\max\{L^H_{\max}, -L_{\min}\}$. Notice that we cannot attain the minimum value zero of $L^H_{\max}$ if we insert the machine idle time. So, even if we allow the machine to be idle, no optimal schedule has the machine idle time. This enable us to apply a similar argument as for the *nmit* version of the problem. ∎

Theorem 4.10 *The problems*

$$1||\max\{L_{\max}, -L^H_{\min}\} \quad \textit{and} \quad 1|nmit|\max\{L_{\max}, -L^H_{\min}\}$$

are NP-hard in the strong sense.

Proof. Similar to the proof of Theorem 4.9. ∎

4.6 Conclusion

We have considered single machine problems with both traditional and generalized due dates. We have shown that a polynomial time algorithm exists for the problem of minimizing $\max\{L_{\max}, L^H_{\max}\}$ (problem 4.1) and that a simple polynomial time algorithm exists for the problem of minimizing $\max_{1\leq i\leq k} L^i_{\max}$ (problem 4.4). We have also shown that the problems of minimizing $\max\{L^H_{\max}, -L_{\min}\}$ and $\max\{L_{\max}, -L^H_{\min}\}$ (problems 4.2 and 4.3) are NP-hard in the strong sense. From these results above, we derive a complete characterization of the solvability of all problems arising from $\max\{A, B\}$, where A and B are $L_{\max}$, $-L_{\min}$, $L^H_{\max}$, and $-L^H_{\min}$, which is shown in Table 4.1.

A/B	$L_{\max}$	$-L_{\min}$ (under *nmit*)	$L^H_{\max}$	$-L^H_{\min}$ (under *nmit*)
$L_{\max}$	EDD	Poly [33]	Poly [this chapter]	NP-hard [this chapter]
$-L_{\min}$	—	MST	NP-hard [this chapter]	Poly [this chapter]
$L^H_{\max}$	—	—	SPT	NP-hard [99]
$-L^H_{\min}$	—	—	—	LPT

Table 4.1: Complexity status

Chapter 5

Scheduling with generalized release dates

In this chapter, we are concerned with single machine scheduling with generalized release dates. They are related to the traditional release dates in similar way as the generalized due dates to the traditional ones. First, we consider the problem to minimize the sum of completion times with generalized release dates which have constant intervals. The strong NP-hardness is proved for this problem. We also consider the maximum lateness problems with traditional and generalized due dates in the presence of generalized release dates.

5.1 Introduction

First, we are concerned with single machine problems of minimizing the sum of completion times with generalized release dates which have constant intervals. We show that these problems both with and without permitted machine idle time are NP-hard in the strong sense.

These problems are equivalent to a special class of two machine flow shop scheduling problems. In general, two machine flow shop problems can be described as follows. Suppose that we have two machines M_1 and M_2. Each J_j consists of a pair of operations (O_{1j}, O_{2j}). Operation O_{ij} has to be processed on M_i during p_{ij} time units, and the order in which the operations are executed is the same through machines. Without loss of generality, we can assume that each job has to be processed first on machine M_1, then on machine M_2. Each machine can process at most one job at a time, and each job can be processed on only one machine at a time.

Here we extend our notation $nmit$ to multiple machine cases. Now, by $nmit$, we mean that, on each machine, no idle time from the start of the first operation to the completion of the last is allowed, and the processing on each machine starts as early as possible.

It is known that the problems $F2||\sum C_j$ and $F2|nmit|\sum C_j$ are NP-hard in the strong sense [32].

The instance constructed in the proof in [32] allows for various processing times on the first machine. We show that analogous results hold even under the assumption that all processing times are equal to a given positive number.

Furthermore, we show that the maximum lateness problems with generalized release dates are polynomially solvable, both with traditional and with generalized due dates.

5.2 Results

5.2.1 The sum of completion time problems

We consider that the problems to minimize the sum of completion times with generalized release dates which have constant intervals. These problems are equivalent to the two machine flow shop problems to minimize the sum of completion times where the processing times on the first machine are the same, with and without allowing for machine idle time. We consider these flow shop problems.

We assume that all the jobs have the same processing time a on the first machine. We show that these problems are NP-hard in the strong sense under this assumption.

Theorem 5.1 *The problem* $F2|p_{1j} = a|\sum C_j$ *is NP-hard in the strong sense.*

Proof. We use 3-partition problem, which is known to be NP-hard in the strong sense [31]. We transform this 3-partition problem to the flow shop problem. Let $A = \{a_1, a_2, \ldots, a_{3n}\}$ be the set and B a positive number considered on 3-partition problem, such that $\sum_{j=1}^{3n} a_j = nB$ and $B/4 < a_j < B/2$ for $1 \leq j \leq 3n$.

We construct the flow shop problem with four types of jobs. Let all jobs have the same processing times nB^2 on the first machine. Let p_i^j be the processing time of j-type job J_i^j on the second machine for $j \in \{a, b, c, d\}$. The first type, which we call a-type, has $3n$ jobs $J_1^a, J_2^a, \ldots, J_{3n}^a$ whose processing times $p_1^a, p_2^a, \ldots, p_{3n}^a$ on the second machine are

$$p_i^a = B + a_i \quad \text{for } 1 \leq i \leq 3n.$$

The second type, which we call b-type, has $n(nB-4)$ jobs $J_1^b, J_2^b, \ldots, J_{n(nB-4)}^b$ whose processing times $p_1^b, p_2^b, \ldots, p_{n(nB-4)}^b$ on the second machine are

$$p_i^b = B \quad \text{for } 1 \leq i \leq n(nB-4).$$

The third type, which we call c-type, has n jobs $J_1^c, J_2^c, \ldots, J_n^c$ whose processing times $p_1^c, p_2^c, \ldots, p_n^c$ on the second machine are

$$p_i^c = nB^2(nB-1) \quad \text{for } 1 \leq i \leq n.$$

The fourth type, which we call d-type, has n^5B^5 jobs $J_1^d, J_2^d, \ldots, J_{n^5B^5}^d$ whose processing times $p_1^d, p_2^d, \ldots, p_{n^5B^5}^d$ on the second machine are

$$p_i^d = n^5B^5 \quad \text{for } 1 \leq i \leq n^5B^5.$$

We now show that the problem above has a schedule with the sum of completion times less than

$$T = T_1 + T_2,$$

where

$$T_1 = \sum_{j=0}^{n-1}(\sum_{i=0}^{nB-1} (nB^2 + n^2B^3j + nB^2(nB-1) + Bi) + 9B/4),$$

and

$$T_2 = \sum_{j=1}^{n^5B^5} (nB^2 + n^3B^3 + n^5B^5j),$$

iff A has a desired partition. Intuitively, T_2 corresponds to the sum of completion times of d-type jobs and T_1 corresponds to that of the rest of jobs.

Suppose that A has a partition $A = A_1 \cup A_2 \cup \cdots \cup A_n$ such that $\sum_{a_j \in A_i} a_j = B$ for $1 \leq i \leq n$. Further suppose that $A_i = \{a_{3i-2}, a_{3i-1}, a_{3i}\}$ for $1 \leq i \leq n$. An easy calculation shows that the following schedule gives the sum of completion times less than T:

$$(J^c_1, J^b_1, J^b_2, \ldots, J^b_{nB-4}, J^a_1, J^a_2, J^a_3, J^c_2, J^b_{nB-3}, J^b_{nB-2}, \ldots, J^b_{2(nB-4)}, J^a_4, J^a_5, J^a_6, \ldots,$$
$$J^c_n, J^b_{(n-1)(nB-4)+1}, J^b_{(n-1)(nB-4)+2}, \ldots, J^b_{n(nB-4)}, J^a_{3n-2}, J^a_{3n-1}, J^a_{3n}, J^d_1, J^d_2, \ldots, J^d_{n^5B^5}).$$

On the other hand, suppose that there exists a schedule which gives the sum of completion times less than T.

First, we claim that all d-type job must be scheduled in the last n^5B^5 positions from time $nB^2 + n^3B^3$. Suppose that there exists a job scheduled after d-type jobs. Then, the job must be completed after $nB^2 + n^5B^5$ and the sum must be greater than $T_2 + n^5B^5 - nB^2(nB-1) > T_2 + T_1 = T$. Next, suppose that d-type jobs are not scheduled from time $nB^2 + n^3B^3$. Then again, the sum must be greater than $T_2 + n^5B^5 > T_2 + T_1 = T$. Thus, we have the claim.

Notice here that the sum of completion times of all d-type jobs is T_2 and that the sum of processing times of all a-type, b-type, and c-type jobs is n^3B^3. From the claim above, a-type, b-type, and c-type jobs must be scheduled from time nB^2 with no inserted idle time, and we can suppose that there exists a schedule which gives the sum of completion times of these jobs, less than T_1. So, from now on, we restrict our attention to only the schedules for a-type, b-type, and c-type jobs from time nB^2 with no inserted idle time.

Next, we claim that, for any schedule, the sum of completion times of a-type, b-type, and c-type jobs must be greater than

$$T_3 = \sum_{j=0}^{n-1}(\sum_{i=0}^{nB-1} (nB^2 + n^2B^3j + nB^2(nB-1) + Bi) + 7B/4),$$

if we schedule them from time nB^2 with no inserted idle time. In order to claim this, we consider an optimal schedule and show that it must give the sum greater than T_3.

We first consider the job in the first position. It must be c-type. Otherwise, there must be inserted idle time between the jobs in the first and second positions, or we cannot schedule the first job from time nB^2.

We secondly consider the jobs in the second to (nB)th position. The sum of processing times of these jobs must be greater than nB^2. Otherwise, there must be inserted idle time between the jobs in the (nB)th and $(nB+1)$st positions.

If there are c-type jobs in the second to (nB)th position, we can transform a schedule with c-type jobs in these positions into one with no c-type job in these positions without increasing the sum by an interchange argument.

Thus, there are at least three three a-type jobs in the second to (nB)th position. Even if we assign $(nB-4)$ b-type jobs in the second to $(nB-3)$th position and three a-type jobs, whose completion times have the sum no less than $4B$, in the $(nB-2)$th to nBth position, the sum of completion times of the jobs in the first nB positions must be greater than

$$\sum_{i=0}^{nB-1}(nB^2+nB^2(nB-1)+Bi)+7B/4.$$

The sum of processing times of the jobs in the second to $(nB+1)$st position must be greater than $2nB^2$. Otherwise, there must be inserted idle time between the jobs in the $(nB+1)$st and $(nB+2)$nd positions. However, the sum of processing times of any nB jobs of a-type and/or b-type is no greater than $nB^2+nB<2nB^2$. So, the job in the $(nB+1)$st position must be c-type job. Even if we assign a c-type job in the $(nB+1)$st position from time $nB^2+n^2B^3$ and a-type and/or b-type jobs in the $(nB+2)$nd to $(2nB)$th position as above, the sum of completion times of the jobs in the $(nB+1)$st to $(2nB)$th position must be greater than

$$\sum_{i=0}^{nB-1}(nB^2+n^2B^3+nB^2(nB-1)+Bi)+7B/4.$$

We can apply a similar argument for the jobs in the $(nBj+1)$st to $nB(j+1)$st position for each $2\leq j\leq n-1$, and show the sum of completion times of the jobs in the $(nBj+1)$st to $nB(j+1)$st position must be greater than

$$\sum_{i=0}^{nB-1}(nB^2+n^2B^3j+nB^2(nB-1)+Bi)+7B/4.$$

Thus, for any schedule, the sum of completion times of a-type, b-type, and c-type jobs must be greater than T_3.

Recall that we assume there exists a schedule which gives the sum of completion times less than T_1. As we have seen, the job in the first position must be c-type.

We consider the jobs in the second to (nB)th position under this assumption. We show that there exist no c-type jobs in these positions of a schedule which gives the sum of completion times less than T_1. If there exists a c-type job in these positions, we exchange this c-type job with an a-type or b-type job scheduled after the c-type job. If the resulting schedule makes inserted idle time, we further exchange b-type jobs scheduled before the c-type job with a-type jobs scheduled after the c-type job. Finally, we get a schedule which makes no inserted idle time. This schedule improves the sum by at least $nB^2(nB-1)-2nB$, and the new sum is less than $T_1-(nB^2(nB-1)-2nB)<T_3$. This contradicts that any schedule gives the sum greater than T_3.

We can apply a similar argument for the jobs in the $(nBj+1)$st to $nB(j+1)$st position for each $1\leq j\leq n-1$, and show a c-type job is in the $(nBj+1)$st position for each $1\leq j\leq n-1$.

Finally, we consider the completion time of the job in (nBj)th position for each $1\leq j\leq n$. We show that it must be exactly $nB^2+n^2B^3j$. Suppose that the completion time of the job in the (nBj)th position is greater than $nB^2+n^2B^3j$. This causes the delays of completion times of jobs in the $(nBj+1)$ to $(nBj+nB-3)$rd position. Then, the

sum must be greater than $T_3 + nB - 3 > T_1$. This contradicts that there exists a schedule which gives the sum less than T_1.

Thus, it follows that there are three a-type jobs in the $(nBj+1)$st to $(nB(j+1))$th position and the sum of the processing times of these jobs must be $4B$. This implies that A has a desired partition. ∎

In the proof above, we construct an instance with no inserted idle time. Therefore, the following theorem has been proved.

Theorem 5.2 *The problem* $F2|p_{1j} = a, nmit|\sum C_j$ *is NP-hard in the strong sense.*

From Theorem 5.1 and 5.2, we obtain the following corollary.

Corollary 5.3 *The problems*

$$1|\rho_j = cj|\sum C_j \quad \textit{and} \quad 1|\rho_j = cj, nmit|\sum C_j$$

are NP-hard in the strong sense.

5.2.2 The maximum lateness problems

We consider two other basic problems with release dates, namely, the maximum lateness problems with traditional and generalized due dates under generalized release dates. The problems $1|r_j|L_{\max}$ and $1|r_j|L^H_{\max}$ are known to be NP-complete in the strong sense [59, 41].

Theorem 5.4 *The problems*

$$1|\rho_j|L_{\max} \quad \textit{and} \quad 1|\rho_j|L^H_{\max}$$

are polynomially solvable.

Proof. First, we claim that $1|\rho_j|L^H_{\max}$ can be solved by $O(n \log n)$ algorithm using the SPT sequencing rule. The algorithm is as follows. First, we schedule a shortest job from the first generalized release date. Second, we schedule a second shortest job as soon as the process of the first job is completed and the second job is released. We repeat this until we schedule all jobs. We can prove that this algorithm actually gives an optimal schedule by a standard interchange argument.

Next, we claim that $1|\rho_j|L_{\max}$ can be solved by $O(n \log n)$ algorithm using the EDD sequencing rule. The algorithm is as follows. First, we schedule a job with the earliest due date from the first generalized release date. Second, we schedule a job with the second earliest due date as soon as the process of the first job is completed and the second job is released. We repeat this until we schedule all jobs.

We can prove that this algorithm actually gives an optimal schedule by a standard interchange argument combined with the result by Ferris and Vlach [28] on the maximum lateness problem with traditional release dates, which claims that, if the due dates are agreeable with release dates in the sense that $r_i < r_j$ implies $d_i \leq d_j$, then the problem can be solved with an EDD based algorithm ∎

From Theorem 5.4, we can easily show the following theorem, where the traditional counterparts of the problems are polynomially solvable [62, 9, 41, 14].

Theorem 5.5 *The problems*

$$1|pmtn, \rho_j|L_{\max} \quad \textit{and} \quad 1|pmtn, \rho_j|L^H_{\max}$$

are polynomially solvable.

5.3 Summary and open problems

We have studied the algorithms and complexity of the problems with generalized release dates. We have only investigated the problems to minimize the sum of completion times and to minimize the maximum lateness. The following table which summarizes our results and open problems.

Problem (notation for traditional model)	Generalized release date model	Traditional release date model
$1\|r_j\|L_{\max}$	Polynomially solvable (Theorem 5.4)	NP-hard [59]
$1\|r_j\|L_{\max}^H$	Polynomially solvable (Theorem 5.4)	NP-hard [41]
$1\|pmtn, r_j\|L_{\max}$	Polynomially solvable (Theorem 5.5)	Polynomially solvable [62, 9]
$1\|pmtn, r_j\|L_{\max}^H$	Polynomially solvable (Theorem 5.5)	Polynomially solvable [41]
$1\|pmtn, prec, r_j\|L_{\max}$	Open	Polynomially solvable [14, 9]
$1\|r_j\|\sum C_j$	NP-hard even if $\rho_j = cj$ (Theorem 5.1) [32]	NP-hard [59]
$1\|r_j, nmit\|\sum C_j$	NP-hard even if $\rho_j = cj$ (Theorem 5.2) [32]	Open
$1\|pmtn, r_j\|\sum C_j$	Open	Polynomially solvable [8]
$1\|pmtn, r_j\|\sum w_j C_j$	Open	NP-hard [51]
$1\|r_j\|\sum T_j^H$	Open	NP-hard [40]
$1\|r_j, p_j = 1\|\sum w_j T_j$	Open	Polynomially solvable [55]
$1\|r_j, p_j = 1\|\sum T_j^H$	Open	Polynomially solvable [111]
$1\|r_j\|\sum U_j$	Open	NP-hard [59]
$1\|r_j\|\sum U_j^H$	Open	NP-hard [41]
$1\|r_j, p_j = 1\|\sum U_j$	Open	Polynomially solvable even for the weighted case [88]
$1\|r_j, p_j = 1\|\sum U_j^H$	Open	Polynomially solvable [111]

Table 5.1: Complexity status of single machine scheduling problems with generalized release dates.

Chapter 6

Scheduling with fuzzy due dates

This chapter deals with single machine scheduling problems involving fuzzy due dates. The objective is to minimize the maximum dissatisfaction with the completion times of jobs. Several algorithms improving the efficiency of previously known algorithms are presented and analyzed.

6.1 Introduction

6.1.1 Background and previous results

We are concerned with single machine problems involving fuzzy due dates. One of the problems studied in [87] is the problem of scheduling a finite number of jobs on a single machine so as to maximize the minimum degree of satisfaction with completion times. The problem can be formally stated as follows: given n jobs $J_1, J_2, \ldots, J_n$ with processing times $p_1, p_2, \ldots, p_n$ and fuzzy due dates $\mu_1, \mu_2, \ldots, \mu_n$, find a permutation schedule S which maximizes

$$\min_{1 \leq i \leq n} \mu_i(C_i(S)),$$

where $C_i(S)$ denotes the completion time of job J_i in schedule S. For this problem, Tada presents an $O(n^2 \log n)$ algorithm. Furthermore, he considers the weighted version of the problem, i.e., the problem of finding a permutation schedule S which maximizes

$$\min_{1 \leq i \leq n} w_i \mu_i(C_i(S)),$$

where w_i is a given positive number. He shows that the argument for the unweighted problem can be extended to the weighted problem and presents an $O(n^2 \log n)$ algorithm for the weighted problem.

6.1.2 Main results

In this section, we present and analyze several algorithms for the problems mentioned above. First, for the sake of completeness, we briefly recall Tada's algorithm, and introduce a fast modification of his algorithm. Next, we show that the problem can be solved in

$O(n^2)$ time by a straightforward application of Lawler's algorithm for minimizing the maximum of non-decreasing functions of completion times. Then, we introduce two methods based on Lawler's algorithm, and show how they improve Tada's and Lawler's algorithms.

6.2 Tada's algorithm and its improvement

In this section, we briefly describe Tada's approach and show how it can be improved.

Let $\bar{u}$ be the optimal value of the objective function under consideration, i.e.,

$$\bar{u} = \max_{S} \min_{1 \le i \le n} \mu_i(C_i(S)),$$

where the maximum is taken over all permutation schedules. Obviously, $0 \le \bar{u} \le 1$.

Observation 6.1 *If $\bar{u} > 0$, then each optimal schedule S satisfies the following system of inequalities*

$$C_i(S) \le e_i(1 - \bar{u}) + d_i, \quad 1 \le i \le n.$$

Observation 6.2 *If $\bar{u} = 0$, then, for each permutation schedule S, there exists job J_k such that*

$$e_k(1 - u) + d_k < C_k(S), \quad \textit{for each } 0 \le u \le 1.$$

For each $0 \le u \le 1$, we introduce the following deadlines for completion of jobs:

$$D_i(u) = e_i(1 - u) + d_i, \quad \text{for } 1 \le i \le n.$$

According to the previous observations, we consider as feasible only those permutation schedules which satisfy the deadlines $D_i(u)$ for some $0 < u \le 1$, i.e.,

$$C_i(S) \le D_i(u), \quad \text{for } 1 \le i \le n,$$

for some $0 < u \le 1$. If no such a feasible schedule exists, then every permutation schedule is optimal and the maximum of the minimum degree of satisfaction is zero.

Observation 6.3 *For each fixed value of u, a feasible schedule exists if and only if the permutation schedule induced by ordering the jobs in the non-decreasing order of $D_i(u)$ is feasible.*

Observation 6.4 *If I is a subinterval of the unit interval $[0, 1]$ such that there is no $u \in I$ satisfying*

$$u(e_i - e_j) = d_i - d_j + e_i - e_j,$$

then the order of $D_i(u)$ does not change throughout of I.

These observations suggest the following procedure for finding an optimal schedule. First, find all points of intersection of D_i and D_j satisfying $0 < u < 1$, then sort them and employ a binary search for finding the maximum among them for which a feasible schedule exists. If no feasible schedule exists, then every permutation schedule is optimal.

The algorithm returns the resulting schedule B as a permutation, i.e., B returns the index $B(i)$ of the job in the ith position for each $1 \le i \le n$.

```
Tada's algorithm
    Let B be an arbitrary schedule
    Find all 0 < u < 1 such that u = (d_i − d_j + e_i − e_j)/(e_i − e_j)
            for some i, j, j ≠ i
    Let Intersection be a set of such u
    Add 0 and 1 to Intersection
    Sort all items in Intersection
    Choose the initial u for the binary search
    while the maximum u is not found do
        Find schedule S according to the non-decreasing order of D_i(u)
        if C_i(S) ≤ D_i(u) for all 1 ≤ i ≤ n then
            B ← S
            Update u for the binary search
        else
            Update u for the binary search
    output(B)
end
```

In [87], Tada showed that the time complexity of the algorithm is $O(n^2 \log n)$. His algorithm needs to find all intersections and to sort them. The cost for these operations is crucial for the complexity. Actually, to find all $0 < u < 1$ such that $u = (d_i - d_j + e_i - e_j)/(e_i - e_j)$ for some i, j, $j \neq i$, his algorithm calculates $u = (d_i - d_j + e_i - e_j)/(e_i - e_j)$ and checks whether $0 < u < 1$ or not for each i, j, $j \neq i$. It costs $O(n^2)$.

Next, we show that we can calculate all such u in $O(P + n \log n)$ time, where P is the number of intersections of n membership functions. With this method, his algorithm has the complexity $O(P \log P + n \log n \log P)$. The idea of this method is similar to that of Chazelle and Edelsbrunner [19]. They are concerned with the general case of intersecting line segments in the plane. We take advantage of the special property of our functions, which enables us to have a simpler algorithm.

The algorithm returns all such u as a set $Intersection$ of points. That is, $Intersection$ is the set of all points u_0 such that

$$\begin{aligned} &D_i(u) \neq D_j(u) \qquad \text{for } 0 < u < 1,\ u \neq u_0, \\ &D_i(u_0) = D_j(u_0). \end{aligned}$$

In the algorithm, we use two lists $UpperJobList$ and $LowerJobList$. Intuitively, jobs according to $UpperJobList$ are ordered by non-decreasing of $d_i + e_i$'s with sharp-sloped job last, and jobs according to $LowerJobList$ are ordered by non-decreasing of d_i's with sharp-sloped job first.

```
FindIntersection
    Intersection ← ∅
    Make a list of sorted indices {1, 2, ..., n} of jobs
      according to the order <, defined by i < j
      if either d_i + e_i < d_j + e_j, or d_i + e_i = d_j + e_j and d_i < d_j
    Let UpperJobList be such a list
    Make a list of sorted indices {1, 2, ..., n} of jobs
```

```
    according to the order <, defined by i < j
    if either d_i < d_j, or d_i = d_j and e_i < e_j
  Let LowerJobList be such a list
  Choose the last item i in LowerJobList
  while such i is found do
    Choose the item j following i in UpperJobList
    while such j is found do
      Find the point u of intersection such that D_i(u) = D_j(u)
      Add u to Intersection
      Choose the item k following j in UpperJobList
      j ← k
    Choose the item k preceding i in LowerJobList
    Delete i from UpperJobList
    i ← k
  output(Intersection)
end
```

Now we analyze the complexity of the algorithm. The cost for making two lists $UpperJobList$ and $LowerJobList$ of indices is $O(n \log n)$. The costs for choosing the last, next and previous item in these lists are $O(1)$ per operation with suitable data structures, for example, doubly linked lists. For each $i \in LowerJobList$, the number of iteration of the inner while loop is that of intersections of D_i. Thus, the total cost of the algorithm is $O(P + n \log n)$.

The correctness of the algorithm is guaranteed as follows. For each $i \in LowerJobList$, we surely find all points of intersection of D_i and the segments crossing D_i from left bottom to right top. All points of intersection of D_i and the segments crossing D_i from left top to right bottom are already found in earlier phases. For each point of intersection, we cannot find it twice because, once it is found, one segment must be deleted at the end of the phase.

6.3 Lawler's algorithms

In the preceding sections, we consider the problem of maximizing the minimum degree of satisfaction with completion times of jobs. This problem is equivalent to that of minimizing the maximum degree of dissatisfaction. From now on, we consider the problem minimizing the maximum degree of dissatisfaction. First, we present an algorithm whose time complexity is $O(n^2)$ based on Lawler's algorithm [37]. We are concerned with the membership functions defined by

$$f_i(t) = \begin{cases} 0 & \text{if } t \leq d_i, \\ (t - d_i)/e_i & \text{if } d_i < t \leq d_i + e_i, \\ 1 & \text{if } d_i + e_i < t, \end{cases}$$

for each $1 \leq i \leq n$. The algorithm returns the resulting schedule A as a permutation, i.e., A returns the index $A(i)$ of the job in the ith position for each $1 \leq i \leq n$.

```
Lawler's algorithm
    I ← {1, 2, ..., n}
    u ← Σ_{i∈ I} p_i
    for j = n down to 1 do
        Choose i ∈ I such that f_i(u) = min_{j∈ I} f_j(u)
        I ← I \ {i}
        u ← u − p_i
        A(j) ← i
    output(A)
end
```

Even if we employ a straightforward implementation, the time complexity of the algorithm is $O(n^2)$. This improves an $O(n^2 \log n)$ algorithm by Tada [87]. The correctness of the algorithm is immediate from the result in [37].

If we replace membership functions f_i by

$$g_i(t) = w_i f_i(t) \quad \text{for each } 1 \leq i \leq n,$$

then, Lawler's algorithm will be again of complexity $O(n^2)$, which also improves an $O(n^2 \log n)$ algorithm by Tada [87].

6.4 Improved methods based on Lawler's algorithms

We compare two algorithms based on Tada's and Lawler's algorithm. If P is large, Lawler's algorithm certainly is better in time complexity than Tada's. But, if P is fairly small, for example n, then Tada's algorithm is superior to Lawler's. In this section, we introduce an algorithm whose time complexity is $O(P' \log P' + n \log n)$, where P' is the number of all intersections of n membership functions excluding the intersections of f_i and f_j both of which appear in an initial lower envelope. This algorithm improves Tada's algorithm whose complexity is $O(P \log P + n \log n \log P)$. Recall that P is the number of all intersections of n membership functions. So, $P' \leq P$.

In Lawler's algorithm, if we can choose $i \in I$ such that $f_i(u) = \min_{j \in I} f_j(u)$ faster, we can get a faster algorithm. This leads to the idea of utilizing lower envelopes of membership functions. We define an lower envelope f of $f_1, f_2, \ldots, f_n$ by $f(t) = \min_{i \in I} f_i(t)$. For horizontal line segments of $f(t)$, we treat them as below. We regard the right top horizontal segment of $f(t)$ is as a part of the graph which gives the rightmost non-horizontal segment intersecting this horizontal segment. Similarly, we regard the left bottom horizontal segment of $f(t)$ is a part of the graph which gives the leftmost non-horizontal segment intersecting this horizontal segment.

The algorithm consists of two major parts. First, we find an initial lower envelope of n membership functions. Then, we repeat the following with updating the sum u of the completion times of the remaining jobs: choosing the job facing the lower envelope at u and updating the lower envelope.

We also show that a slight modification of the algorithm leads to an $O(Ln + n \log n)$ time algorithm, where L is the number of membership functions not appearing in the

initial lower envelope. This enables us to have a fast algorithm when P' is large, and the complexity does not grow faster than Lawler's algorithm.

Now, we present the first major part of the algorithm, which finds an initial lower envelope of n membership functions. The idea of this algorithm is similar to that of Megiddo [67]. He deals with linear functions while we are concerned with special line segments induced by membership functions. To describe a lower envelope f of membership functions $f_1, f_2, \ldots, f_n$, we use a list $LowerEnvelope$ of sorted pairs (i, j), where i is an index of the job which takes part in the lower envelope and j is the left breaking point of the job in the lower envelope. Let $(i_1, j_1), (i_2, j_2), \ldots, (i_k, j_k)$ be the first, second, ..., and last (kth) item in $LowerEnvelope$, respectively. Then, for each $1 \leq l \leq k$, $f(t) = f_l(t)$ for all $j_l \leq t \leq j_{l+1}$ ($j_{k+1} = \sum_{l=1}^{n} p_l$). The algorithm returns a lower envelope as a list $LowerEnvelope$.

```
FindLowerEnvelope
    LowerEnvelope ← ∅
    Make UpperJobList as in FindIntersection
    Make LowerJobList as in FindIntersection
    Choose the first item i in UpperJobList
    Add (i, 0) to LowerEnvelope
    Choose the item j following i in UpperJobList
    i ← j
    while such i is found do
        Let LowerEnvelope = ((i_1, j_1), (i_2, j_2), ..., (i_k, j_k))
        if f_i(d_{i_1}) ≤ 0 then
            Delete all items from LowerEnvelope
            Add (i, 0) to LowerEnvelope
        else
            Choose the largest l such that f_{i_l}(j_l) < f_i(j_l)
            Find the intersection u such that f_{i_l}(u) = f_i(u)
            Delete (i_{l+1}, j_{l+1}), (i_{l+2}, j_{l+2}), ..., (i_{l+1}, j_{l+1}) from LowerEnvelope
            Add (i, u) to the last in LowerEnvelope
        Choose the item j following i in UpperJobList
        i ← j
    output(LowerEnvelope)
end
```

Now we analyze the complexity of the algorithm. The cost for adding an item to $LowerEnvelope$ and that for deleting an item from $LowerEnvelope$ is $O(1)$ per operation with a reasonable data structure.

In the while loop, the cost for choosing the largest l such that $f_{i_l}(j_l) < f_i(j_l)$ is $O(\log n)$ per operation by using a binary search. The total cost for deleting items from $LowerEnvelope$ is $O(n)$ because, for each i, (i, j) is deleted at most once. Thus, the total cost of the algorithm is $O(n \log n)$.

We check the correctness of the algorithm inductively. Suppose that, after some step, we have a lower envelope $LowerEnvelope = ((i_1, j_1), (i_2, j_2), \ldots, (i_k, j_k))$. Consider the next step in which we add f_i to the envelope. Since either $d_k + e_k < d_i + e_i$, or $d_k + e_k =$

$d_i + e_i$ and $d_k \leq d_i$, we can always add f_i to the last in the envelope. The graph of $f_i(t)$ either intersects the previous lower envelope at a unique point or not at any points.

If we have the latter case, $f_i(t)$ covers the previous lower envelope from below completely. This is done in the case $f_i(d_{i_1}) \leq 0$ in the algorithm. If we have the former case, we divide a new lower envelope into two parts at a unique intersection. Then, only $f_i(t)$ faces the right part and only the previous lower envelope faces the left part. This is done in the case $f_i(d_{i_1}) > 0$ in the algorithm.

We proceed to the second major part of the algorithm. Suppose that we are given $LowerEnvelope$ and u. To choose the job facing the lower envelope at u, we choose (i, j) with the largest j no greater than u in $LowerEnvelope$. A binary search implements this in $O(\log n)$ time.

Here we introduce a modified version of FindIntersection, which we use in the algorithm to update the lower envelope. The algorithm returns all intersections of n membership functions excluding all intersections of f_i and f_j both of which appear in the lower envelope.

The algorithm returns all such intersections as n sets $Intersection_1$, $Intersection_2, \ldots, Intersection_n$, where $Intersection_i$ corresponds to f_i. That is, $Intersection_i$ is a set of pairs of (j, u) of a job and an point of intersection, where $f_i(t)$ intersects $f_j(t)$ at u from below, i.e.,

$$\begin{array}{ll} f_i(t) < f_j(t) & \text{for } d_j < t < u, \\ f_i(u) = f_j(u), & \\ f_i(t) > f_j(t) & \text{for } u < t < d_j + e_j, \end{array}$$

where $f_j(t)$ appeared in the lower envelope is excluded.

```
FindIntersection'(LowerEnvelope)
    Intersection_1, Intersection_2, ..., Intersection_n ← ∅
    Let I be the set of i such that (i, j) ∈ LowerEnvelope
    Make a list of sorted items in I according to the order similar to
      that for UpperJobList in FindIntersection
    Let UpperJobList' be such a list
    Make LowerJobList as in FindIntersection
    Choose the last item i in LowerJobList
    while such i is found do
        Choose the item j following i in UpperJobList
        while such j is found do
            Find the point u of intersection such that f_i(u) = f_j(u)
            Add (j, u) to Intersection_i
            Add (i, u) to Intersection_j
            Choose the item k following j in UpperJobList'
            j ← k
        Choose the item k preceding i in LowerJobList
        Delete i from UpperJobList'
        i ← k
    output(Intersection_1, Intersection_2, , ..., Intersection_n)
end
```

The analysis of the complexity is analogous to that for FindIntersection. The number of iteration of the inner while loop is that of intersections of f_i and f_j such that f_j does not appear in the lower envelope. Thus, the total cost of the algorithm is $O(P'+n\log n)$, where P' is the number of all intersections of n membership functions excluding all intersections of f_i and f_j both of which appear in the lower envelope. Notice that P' is bounded by Ln, where L is the number of f_i which does not appear in the lower envelope. So, the total cost of the algorithm is $O(Ln + n\log n)$ alternatively. The correctness of the algorithm is analogous to that of FindIntersection.

The algorithm updating a lower envelope also needs the following process before its first execution. The process sorts all items in $Intersection_i$ and returns them as a new list $Intersection_i$ for each i.

The algorithm updating a lower envelope also needs a process SortItemsInIntersection before its first execution. For each i, this process sorts all items in $Intersection_i$ according to the order $<$, defined by $(i_1, j_1) < (i_2, j_2)$ if either $j_1 < j_2$, or $j_1 = j_2$ and $d_{i_1} < d_{i_2}$, and returns them as a new list $Intersection_i$.

SortItemsInIntersection($Intersection_1, Intersection_2, \ldots, Intersection_n$)
  for each k do
    Make a list of sorted items (i, j) in $Intersection_k$
      according to the order $<$, defined by $(i_1, j_1) < (i_2, j_2)$
      if either $j_1 < j_2$, or $j_1 = j_2$ and $d_{i_1} < d_{i_2}$
    Let $Intersection_k$ be such a list
  output($Intersection_1, Intersection_2, \ldots, Intersection_n$)
end

The total cost of the algorithm is $O(\sum_{i=1}^{n}(P_i' \log P_i'))$, where P_i' is the number of items in $Intersections_i'$. Thus, the total cost is $O(P' \log P')$.

Now, we present an algorithm to update a lower envelope. The algorithm uses two lists $UpperJobList'$ and $LowerJobList$ to indicate the remaining jobs together with $Intersection_i$'s. Let (i, j) to be deleted from $LowerEnvelope$. The algorithm returns updated $LowerEenvelope$ with updated $UpperJobList'$, $LowerJobList$, and $Intersection_i$'s.

UpdateLowerEnvelope($(i, j), LowerEnvelope, UpperJobList', LowerJobList,$
  $Intersection_1, Intersection_2, \ldots, Intersection_n$)
  Let $LowerEnvelope = ((i_1, j_1), \ldots, (i_{l-1}, j_{l-1}), (i_l, j_l), (i_{l+1}, j_{l+1}), \ldots, (i_k, j_k))$
  Let $(i, j) = (i_l, j_l)$
  Delete i_l from $UpperJobList'$
  Delete i_l from $LowerJobList$
  Delete (i_l, j_l) from $LowerEnvelope$
  for each item $(i, j) \in Intersection_{i_l}$ do
    Delete (i_l, j) from $Intersection_i$
  if $l = 1$ then
    Choose the last item i_0 in $LowerJobList$
    $j_0 \leftarrow 0$
  if $l = k$ then
    Choose the last item i_{k+1} in $UpperJobList'$
  Delete (i_{l+1}, j_{l+1}) from $LowerEnvelope$

```
    if i_{l-1} ≠ i_{l+1} then
        Find the intersection u such that f_{i_{l-1}}(u) = f_{i_{l+1}}(u)
        Choose the smallest item (i, j) such that j_l ≤ j in Intersection_{i_{l-1}}
          according to the order similar to that in SortItemsInIntersection
        if j > u then
            Insert (i_{l+1}, u) into LowerEnvelope
        else
            while i ≠ i_{l+1} do
                Insert (i, j) into LowerEnvelope
                i_{l-1} ← i
                j_l ← j
                Choose the smallest item (i, j) such that j_l ≤ j in Intersection_{i_{l-1}}
                  according to the order similar to that in SortItemsInIntersection
            Insert (i_{l+1}, j) into LowerEnvelope
    output(LowerEnvelope, UpperJobList', LowerJobList,
           Intersection_1, Intersection_2, ..., Intersection_n)
end
```

Now we analyze the complexity of the algorithm. The cost to get the position of (i, j) in the lower envelope is $O(\log n)$ by a binary search. The cost for choose the smallest item (i, j) such that $j_l \leq j$ in $Intersection_{i_{l-1}}$ is $O(\log P')$ per operation with a binary search. Since we know the position of (i, j) in the lower envelope, then the cost for inserting (i, j) into $LowerEnvelope$ is $O(1)$ per operation. Let L' be the number of the segments which is newly inserted into the lower envelope. Then, the number of iteration of the while loop is L'. Thus, the total cost of the algorithm is $O(L' \log P' + \log n)$.

The correctness of the algorithm is guaranteed as follows. First, the algorithm updates the lists for the remaining jobs and intersections.

Then, the algorithm finds the segment $f_{i_{l-1}}$ whose right breaking point j_l is the left breaking point of the deleted segment f_l. If there is no such $f_{i_{l-1}}$, the algorithm takes the last job in $UpperJobList'$ as $f_{i_{l-1}}$, which is the leftmost segment of the new lower envelope, and set $j_l = 0$. The algorithm also finds the segment $f_{i_{l+1}}$ whose left breaking point is the right breaking point of the deleted segment f_l. If there is no such $f_{i_{l+1}}$, the algorithm takes the last job in $UpperJobList'$ as $f_{i_{l+1}}$. In this part, we find the part which needs to be updated in the lower envelope. That is, we find that we need to update the lower envelope from $f_{i_{l-1}}$ to $f_{i_{l+1}}$. Notice that the part which we need to update is connected, because lower envelopes are convex functions.

Next, the algorithm updates the part of the envelope found above. If $f_{i_{l-1}} = f_{i_{l+1}}$, the new lower envelope consists of a unique function $f_{i_{l-1}}$ $(= f_{i_{l+1}})$. Otherwise, the algorithm finds the intersection x of $f_{i_{l-1}}$ and $f_{i_{l+1}}$, and the leftmost intersection y on the right of j_l, of $f_{i_{l-1}}$ and f_i, where f_i is a segment which does not face the initial lower envelope.

If x is not on the right of y, the updated part consists of $f_{i_{l-1}}$ and $f_{i_{l+1}}$, and the left breaking point of $f_{i_{l+1}}$ is x. This is correct, since x is the leftmost intersection on the right of j_l, of $f_{i_{l-1}}$ and f_j, where f_j is a segment which faces the initial lower envelope.

If x is on the right of y, the algorithm begins with $f_{i_{l-1}}$ and y, and traces the intersections along the updated part with updating y until it reaches $f_{i_{l+1}}$. This is correct, since no segment f_j appears in the updated part except $f_{i_{l+1}}$, where f_j is a segment which faces

the initial lower envelope.

We present the entire algorithm of minimizing the maximum dissatisfaction.

```
Our algorithm
    FindLowerEnvelope
    Let LowerEnvelope be such a list
    FindIntersection'(LowerEnvelope)
    Let Intersection_1, Intersection_2, ..., Intersection_n be such sets
    SortItemsInIntersection(Intersection_1, Intersection_2, ..., Intersection_n)
    Let Intersection_1, Intersection_2, ..., Intersection_n be such lists
    Make UpperJobList' as in FindIntersection'
    Make LowerJobList as in FindIntersection
    u ← Σ_{i∈ I} p_i
    for k = n down to 1 do
        Choose (i, j) with the largest j no greater than u in LowerEnvelope
        UpdateLowerEnvelope((i, j), LowerEnvelope, UpperJobList',
          LowerJobList, Intersection_1, Intersection_2, ..., Intersection_n)
        Let LowerEnvelope, UpperJobList', LowerJobList,
          Intersection_1, Intersection_2, ..., Intersection_n be updated lists
        u ← u − p_i
        A'(k) ← i
    output(A')
end
```

Now we analyze the complexity of the algorithm. Recall that the cost of UpdateLowerEnvelope is $O(L' \log P' + \log n)$, where L' is the number of the segments which is newly inserted into a lower envelope. Since each segment is newly inserted into a lower envelope at most once, the total cast of this operation is $O(n \log P' + n \log n)$. Thus, from the analysis of each element of the algorithm, the total cost is $O(P' \log P' + n \log n)$.

Finally, we present a slight modified version of the algorithm above. In the new algorithm, we do not execute SortItemsInIntersection. Owing to this, we can discard the cost $O(P' \log P')$ for sorting intersections. Consequently, in UpdateLowerEnvelope, we use $Intersection_i$'s as sets (not lists), i.e., the items in $Intersection_i$'s are not ordered.

This prevents us from using a binary search to choosing the smallest item (i, j) such that $j_l \leq j$ in $Intersection_{i_{l-1}}$ in UpdateLowerEnvelope. The cost for this operation is changed to $O(P'_{i_{l-1}})$, where $P'_{i_{l-1}}$ is the number of items in $Intersections'_{i_{l-1}}$. Recall that L is the number of membership functions not appeared in the initial lower envelope. Since $P'_{i_{l-1}}$ is bounded by L, the cost for the operation is $O(L)$. Thus, the total cost of UpdateLowerEnvelope is $O(LL' + \log n)$, so, the total cost of the algorithm is $O(Ln + n \log n)$.

Part II

Negation-limited circuit complexity

Chapter 7

Preliminaries

The study of circuit complexity dates back to the pioneering works by Shannon [82]. Among computational models, the circuit model has an especially simple definition, and enables us to apply combinatorial analyses to it.

Despite the importance of lower bounds on the circuit complexity of explicit problems, the best bounds known are only linear. However, good lower bounds are known for the complexity of monotone circuits, where negations are forbidden. This motivates the study on the complexity of negation-limited circuits, where the number of negations available is restricted.

This chapter is intended to give a brief review concerning negation-limited circuit complexity. First, in Section 7.1, we describe background of our study on negation-limited circuit complexity. Then, we provide a basic formulation for negation-limited circuit complexity in Section 7.2.

The time and space complexity of Turing machines currently reflects intuitive notions of the complexity of computation. Section 7.1 describes relationships between the time and space required by a Turing machine and the size and depth required by a circuit. Finally, in Section 7.4, we present several known results on inversion complexity on which negation-limited circuit complexity is based.

7.1 Background and motivation

The theory of monotone boolean circuit complexity has met with considerable success. Good lower bounds for both size and depth of monotone circuits (i.e., circuits *without* NEGATION gates) computing many explicit functions are now known. Razborov [77] obtained a superpolynomial lower bound of size $n^{\Omega(log n)}$ for the monotone circuit complexity of the clique function. Soon after, Andreev [7] proved an exponential lower bound for some class of monotone functions. Alon and Boppana [5] strengthened the combinatorial arguments of Razborov, and proved a lower bound for the clique function of size exponential in $\exp(\Omega((n/\log n)^{1/3}))$ (see also [16, 66]). Recently, Amano and Maruoka [6] further strengthened this argument with a succinct proof, and gave a better bound for the clique function (see also [39]).

However, the theory of general boolean circuits, *with* NEGATION gates, is much less well understood, although Shannon [82] proved an $\Omega(2^n/n)$ lower bound on the size of circuits for *almost all* boolean functions. Up to now, only linear lower bounds with small

constants have been obtained. The largest bound of this sort is $3n$, proved by Blum [15] for circuits on basis of all two-input boolean functions, and $4n$, proved by Zwick [112] for circuits on basis any two-input boolean functions except equivalence and exclusive-or.

Furthermore, exponential gaps between monotone and general circuit complexity are known, both for circuit size [100] and circuit depth [76]. The effect of NEGATION gates on circuit complexity remains to a large extent a mystery.

This motivates the study of *negation-limited* circuit complexity: what is the effect on circuit complexity of restricting the number of negations available? Markov [65] gives an explicit formula (see below) for the maximum number r of NEGATION gates *required* to compute a system of boolean functions F, without regard to circuit complexity. The maximum value of r for a circuit with n inputs is $\lceil \log(n+1) \rceil$, the number of bits in the binary representation of n (all logarithms in this part are base two). We shall denote this number by $b(n)$.

Fischer [29] shows that restricting the number of negations in a circuit to $b(n)$ entails only a polynomial blowup in circuit size. This is in sharp contrast to the situation for monotone circuits. É. Tardos [100] has shown that there is an exponential gap between the general complexity and the monotone complexity of some monotone functions. In related work, Santha and Wilson [78] have studied the negation-limited complexity of constant depth circuits, obtaining both upper and lower bounds for the number of NEGATION gates required by circuits of a given depth. Recently, Tanaka and Nishino [91] have found an alternative to Fischer's construction; they decrease the size of the circuits at the expense of increasing the depth.

For some class of monotone functions, the gap between monotone and general circuit complexity is polynomial. Berkowitz [13] introduced a class of this kind, which is called slice functions. For any slice function, a lower bound of size $\omega(n \log^2 n)$ of monotone circuit complexity implies a superlinear lower bound on general circuit complexity [104]. Dunne [23] generalized the concept of slice functions, and introduced a special type of replacement rule called pseudo-complementation. Furthermore, Dunne [24] also showed that some specific slice functions are NP-complete. Thus, a superpolynomial lower bound on the monotone circuit complexity of this kind of slice function implies that P $\neq$ NP. But up to date, the methods proving strong lower bounds on the monotone complexity of boolean functions could not be used to prove strong lower bounds of slice functions (see also [105, 106, 26, 27]).

7.2 Definitions

A *boolean circuit* (or *network*) is a suitably labeled directed acyclic graph. The nodes with in-degree zero are called *inputs*, and are labeled with a variable x_i or with a constant 0 or 1. The nodes with in-degree $k > 0$ are called *gates*, and are labeled with k-input boolean functions. Unless otherwise specified, we restrict these functions to boolean functions AND, OR and NEGATION.

We refer to the in-degree of a node as its *fan-in* and its out-degree as its *fan-out*. The fan-out of a circuit is the maximal fan-out of any node. The fan-in of a circuit is the maximal fan-in of any node. Unless otherwise specified, we restrict the fan-in of a circuit to two. One of the nodes is designated the *output* node. Each gate in a circuit computes

a function by applying the function labeling it to the functions computed by the nodes feeding it. A circuit computes a boolean function in this way.

The *size* of a circuit is the number of gates, and the *depth* of a circuit is the length of a longest path from an input to an output. Let F be a collection of boolean functions $f_1, \ldots, f_m$ defined on $\{0,1\}^n$. We denote by $C(F)$ or $C(f_1, \ldots, f_m)$ the *circuit complexity* of F, i.e., the size of the smallest circuit of AND, OR, and NEGATION gates with inputs

$$x_1, \ldots, x_n$$

and outputs

$$f_1(x_1, \ldots, x_n), \ldots, f_m(x_1, \ldots, x_n).$$

We call a circuit with no more than r NEGATION gates an r-circuit. We denote by $C^r(F)$ or $C^r(f_1, \ldots, f_m)$ the size of the smallest r-circuit computing F. If the system of functions cannot be computed with only r NEGATION gates, then $C^r(F)$ is undefined. Similarly, we denote by $D^r(F)$ the minimum *depth* of an r-circuit computing F.

The *inverter* is the collection I_n of functions $f_1, \ldots, f_n$, where for all $1 \leq i \leq n$, $f_i(x_1, \ldots, x_n) = \neg x_i$. For $0 \leq k \leq n$, the (k,n)-*inverter* is the collection I_n^k of functions $f_1, \ldots, f_n$, where for all $1 \leq i \leq n$,

$$f_i(x_1, \ldots, x_n) = \begin{cases} 0 & \text{if } \sum_{j=1}^n x_j < k \\ \neg x_i & \text{if } \sum_{j=1}^n x_j = k \\ 1 & \text{if } \sum_{j=1}^n x_j > k. \end{cases}$$

A circuit with no NEGATION gates is *monotone*. A *monotone function* f is a boolean function which satisfies $f(x_1, \ldots, x_n) \leq f(y_1, \ldots, y_n)$ whenever $x_i \leq y_i$ for all i. Monotone functions are exactly those computed by monotone circuits. Observe that the (k,n)-inverter is a system of monotone functions. Berkowitz [13] shows that the monotone complexity of the (k,n)-inverter is polynomial, with a construction of size $O(n^2 \log n)$ and depth $O(\log n)$. Valiant [104] constructs monotone (k,n)-inverters of size $O(n \log^2 n)$ and depth $O(\log^2 n)$.

We define the kth *slice* function f_k of f by

$$f_k(x_1, \ldots, x_n) = \begin{cases} 0 & \text{if } \sum_{i=1}^n x_i < k \\ f(x_1, \ldots, x_n) & \text{if } \sum_{i=1}^n x_i = k \\ 1 & \text{if } \sum_{i=1}^n x_i > k. \end{cases}$$

We denote by $T_k^n(x_1, \ldots, x_n)$, the kth *threshold* function which returns one iff $\sum_{i=1}^n x_i \geq k$.

For more detail on circuit complexity, see e.g. the survey by Boppana and Sipser [16], and the books by Dunne [25] and Wegener [107].

7.3 Simulations of circuits by Turing machines

The time and space complexity of Turing machines currently reflects intuitive notions of the complexity of computation. In this section, we discuss the relationships between the time and space required by a Turing machine and the size and depth required by a circuit.

In order to make close relation between these models, *oblivious* Turing machines were introduced. A Turing machine is oblivious if, for fixed n, the positions of all heads at each step are the same for all inputs of size n.

The Turing machine time and space complexity $T(n)$ and $S(n)$ can bound the complexity of boolean circuits. Savage [79] showed that the circuit complexity is at most $O(T(n)^2)$. Pippenger and Fischer [75] reduced this bound to $O(T(n))$ for oblivious Turing machines and to $O(T(n)\log T(n))$ for unrestricted Turing machines. These results were extended by Schnorr [81]. The following three propositions are due to Pippenger and Fischer [75].

Let f_n be the boolean function obtained by restricting f to inputs of size n.

Proposition 7.1 *Let an oblivious Turing machine compute f in time $T(n)$. Then,*

$$C(f_n) = O(T(n)).$$

Proposition 7.2 *Let a Turing machine M compute f in time $T(n)$. Suppose in addition M has the property that the steps at which inputs are read or outputs produced depend only on the length n of the input. Then we can find a two tape oblivious Turing machine for f which runs in time*

$$O(T(n)\log T(n)).$$

Proposition 7.3 *Let a Turing machine compute f and run within time $T(n) > n$ for all inputs of length n. Then,*

$$C(f_n) = O(T(n)\log T(n)).$$

We present a proof for the simulation of a Turing machine by a circuit, based on the lecture notes by Zwick [113]. The proof is for a weaker bound of $O(T^2(n))$, or actually $O(T(n)S(n))$, and given for the case in which the Turing machine has a single tape. It is easy to generalize it to the case with multiple tapes.

Proof. Let M be a quintuple $M = (\Sigma, Q, q_s, q_t, \delta)$, where

- Σ is a finite alphabet,
- Q is a finite set of states,
- $q_s \in Q$ is the initial state,
- $q_t \in Q$ is the accepting state,
- $\delta : \Sigma \times Q \to \Sigma \times Q \times \{L, R\}$ is the transition function.

Each state in Q can be represented by $log|Q|$ bits, and each letter of Σ by $log|\Sigma|$ bits. Using this representation, δ can be seen as a boolean function of a small number of bits, and can be implemented by a circuit of small size, as shown in Figure 7.1, which is labeled δ.

The circuit δ is a basic building block in a circuit that simulates M. Let $x_1, x_2, \ldots, x_n$ be binary sequences representing the first n binary digits is the representation of the input written on the tape of M, before it starts executing. The circuit simulating M will receive the variables x_i as input and compute the function calculated by M. It will have $T(n)$ different layers, one for each time unit $0 \leq t \leq T(n)$. The binary values at each layer of the circuit encode a *configuration* of the machine (its sate, tape content and the location of the head). It is clearly sufficient to include in such a configuration, the content of the

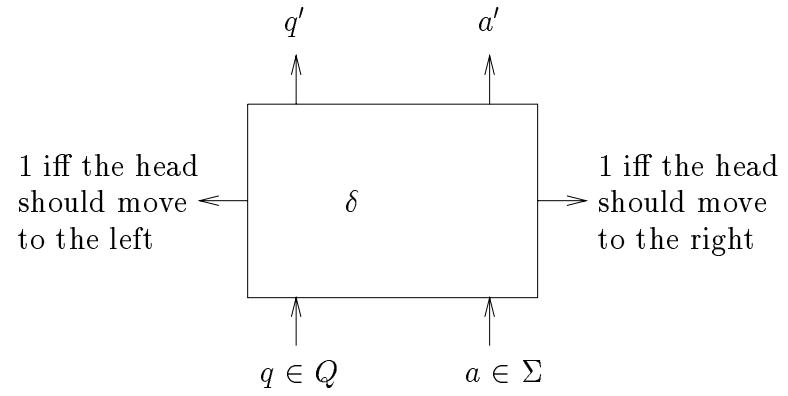

Figure 7.1: The circuit δ.

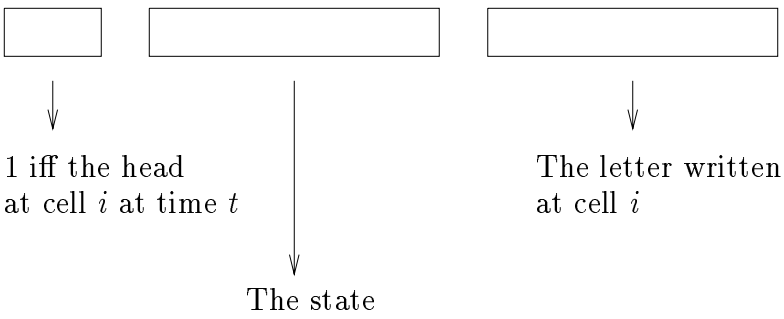

Figure 7.2: The binary array.

tape cells $-T(n) \leq i \leq T(n)$ only. For each point $0 \leq t \leq T(n)$ of time, for every cell $-T(n) \leq i \leq T(n)$ of the tape, we keep a binary array, shown in Figure 7.2.

The circuit computes these arrays, according to the following layers:

First layer For $t = 0$, these arrays are composed of the variables $x_1, x_2, \ldots, x_n$, and constants.

Intermediate layers Given the arrays for time t, the arrays for time $t+1$ are constructed in parallel, using γ circuits, similar to δ (see Figure 7.3).

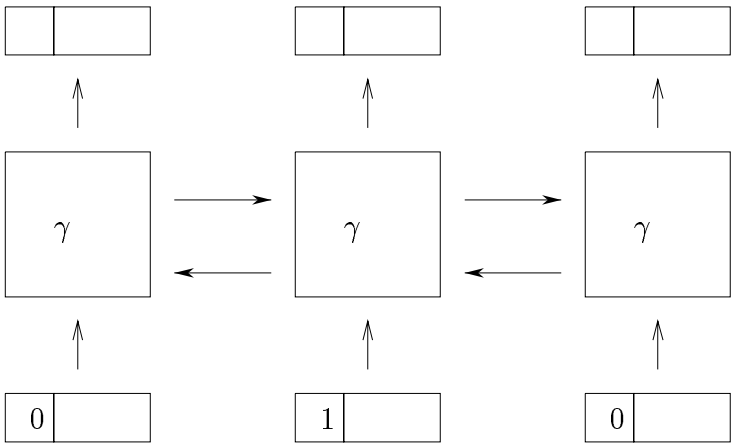

Figure 7.3: A circuit to simulate M.

Each γ circuit gets one of the arrays of time t as input, and creates, at its output, the corresponding array of time $t+1$. If the head is in the cell, for which the γ circuit is *responsible*, the array is changed according to δ, and the appropriate neighboring γ circuit is notified of the head movement. This neighbor then updates its array accordingly. All other γ circuits output their input arrays without change.

A difference between the original transition function of M and the circuit γ, is that once M reaches q_1, the accepting state, it halts. On the other hand, the circuit cannot stop, so when a γ circuit gets as input an array, in which the encoded state is q_1, the array is left unchanged, and the head is not moved.

Last layer The last layer is the construction is a circuit, that checks whether the state, encoded at the array where the head bit is 1, is q_1.

The size of the circuit, thus constructed, is $O(T^2(n))$. The depth is $O(T(n))$, which is the same as M's time complexity. ∎

The following proposition, proved by Borodin [17], gives a relation between the depth of circuits and the space of nondeterministic Turing machines.

Proposition 7.4 *Let a nondeterministic Turing Machine compute f using space $S(n) \geq \log n$, then*

$$D(f_n) = O(S(n)^2).$$

From Proposition 7.3 and 7.4, large enough lower bounds on size and depth of circuit complexity can provide superlinear lower bounds on time and space of Turing machine complexity. The converse of this is false. This is because Turing machines are a *uniform* model of computation, while boolean circuits a *non-uniform* model. Incidentally, all decision problems may be solved by networks but it is well known that some decision problems are not computable with Turing machines.

For more detail on the simulations of Turing machines by circuits, see e.g. the lecture notes by Fischer [30] and Zwick [113], and the books by Dunne [25] and Wegener [107].

7.4 Inversion complexity

In this section, we describe the theorem proved by Markov in 1958 [65]. A theorem of Markov precisely determines the number of NEGATION gates necessary and sufficient to compute a system of boolean functions.

Let F be a system of n-input boolean functions $f_1 \ldots, f_m$. Let $I(F)$ be the minimum number of negations in any circuits which computes the set F of boolean functions. $I(F)$ is called the *inversion complexity* of F.

A *chain* C in the boolean lattice $\{0,1\}^n$ is an increasing sequence $a^1 < \ldots < a^k \in \{0,1\}^n$. The *decrease* of F on C is the number of $i \leq k$ such that for some j, $f_j(a^{i-1}) > f_j(a^i)$. We define $d(F)$ to be the maximum decrease of F on any chain C. Markov has proved that $b(d(F))$ NEGATION gates are necessary and sufficient to compute any system F of functions $f_1, \ldots, f_m$, where $b(d(F)) = \lceil \log(d(F)+1) \rceil$, the number of bits in the binary representation of $d(F)$.

Theorem 7.5 (Markov)

$$I(F) = \lceil \log(d(F)+1) \rceil,$$

Corollary 7.6 (Markov) *Let F be a system of n-input boolean functions $f_1 \ldots, f_m$, where $m \geq 2$, then,*

$$\max_F I(F) = \lceil \log(n+1) \rceil,$$

Let f be a n-input boolean function, then,

$$\max_f I(f) = \lfloor \log(n+1) \rfloor.$$

Markov's theorem says nothing about the circuit size, when we achieve the minimal numbers of negations.

Nakamura, Tokura and Kasami [71] and Fischer [30] gave algorithms for finding a circuit computing F using $\lceil \log(d(F)+1) \rceil$ negations. A proof that $I(F) \geq \lceil \log(d(F)+1) \rceil$ also appears in [30].

In this section, using slice functions, we give another proof that $\lceil \log(n+1) \rceil$ NEGATION gates are sufficient to compute any system F of functions.

Proof. We construct a circuit T having $\lceil \log(n+1) \rceil$ NEGATION gates which computes F. For simplicity, we construct a circuit T for the case when $|F| = 1$ i.e. F is singleton, and $n+1$ is a power of two. Let $F = \{f\}$.

If we are given the $n+1$ slice functions, $f_0, \ldots, f_n$ of f, and the complements of n threshold functions, $\neg T_1^n, \ldots, \neg T_n^n$, we can compute f by using the following relation:

$$f = \bigvee_{k=0}^{n} [\neg T_{k+1}^n \wedge f_k].$$

Notice that $\neg T_{n+1}^n$ is 1 (a constant function).

Since any slice function is monotone, Each f_k can be realized by a monotone circuit. So, it is suffice to construct a circuit including $\lceil \log(n+1) \rceil$ NEGATION gates which computes $\{\neg T_1^n, \ldots, \neg T_n^n\}$. This can be performed by combining a monotone sorting network for n variable and a circuit M_n (see Figure 7.4).

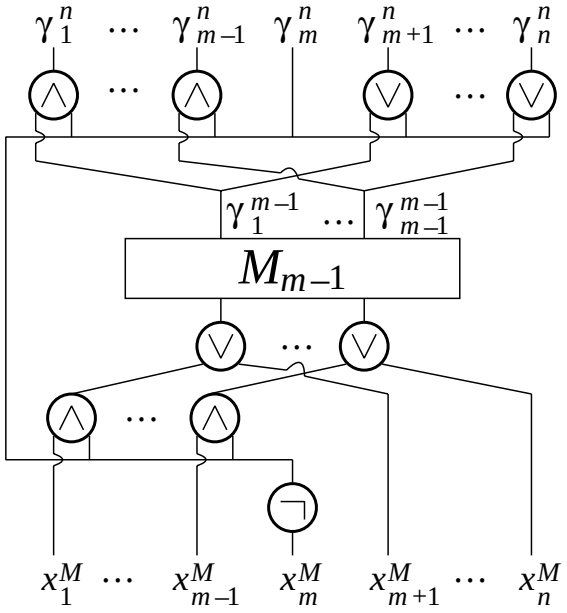

Figure 7.4: A network M_n.

Let $x_1^S, \ldots, x_n^S$ be the inputs and $y_1^S, \ldots, y_n^S$ the outputs of the monotone sorting network. First, we identify the inputs of the monotone sorting network with the input of T as follows:

$$x_i^S = x_i \quad \text{for each } 1 \leq i \leq n.$$

Let $x_1^M, \ldots, x_n^M$ be the inputs and $y_1^M, \ldots, y_n^M$ the outputs of M_n. Next, we identify each input of M_n with the outputs of the sorting network as follows:

$$x_i^M = y_i^S \quad \text{for each } 1 \leq i \leq n.$$

We use the circuit M_n due to Fischer [29], inductively defined as shown in Figure 7.4. Let $n = 2^l - 1$ and $m = 2^{l-1}$. In Figure 7.4, γ_i^n $(1 \leq i \leq n)$ corresponds to y_i^M, and γ_j^{m-1} $(1 \leq j \leq m-1)$ denotes an output of a subcircuit M_{m-1}.

Then, from [29], we have

$$y_i^M = \neg T_i^n(X_n) \quad \text{for each } 1 \leq i \leq n,$$

and M_n includes only $\log(n+1)$ NEGATION gates. The resulting circuit is shown in Figure 7.5.

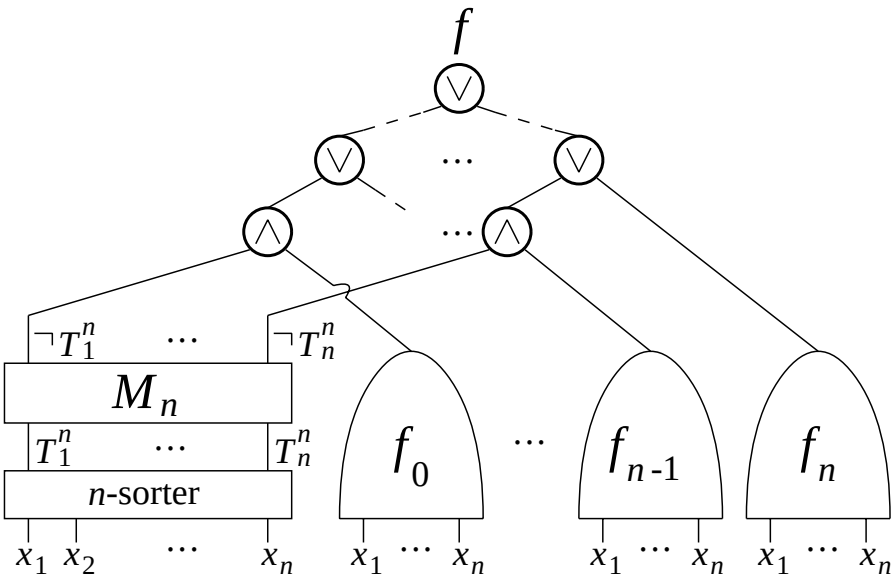

Figure 7.5: The network computing f using its slices.

In the case when F is not singleton, the circuit computing $\{\neg T_1^n, \ldots, \neg T_n^n\}$ can be commonly used to compute all functions in F. So, the construction above can easily be applied to the case when $|F| \geq 2$. This completes the proof. ∎

For yet another proof for the number of negations necessary to compute parity functions, see the paper by Nishino and Radhakrishnan [72].

Note that Gilbert [36] considered inversion complexity in 1954, and that the syntheses of circuits with minimal numbers of negations are included in the survey by Fischer [29] and the early papers by Ibaraki and Muroga [46] and Nakamura, Tokura, and Kasami [71]. In the connection to the relationship with negation and computation, see e.g. [103], in which he showed that negation is exponentially powerful for computing arithmetic functions. For further detail on negation-limited circuit complexity, see also e.g. the lecture notes and survey by Fischer [30, 29] and the theses by Tanaka [89, 90].

Outline of this part

The remainder of this part is divided into three chapters. First, in Chapter 8, we consider the complexity of negation-limited inverters. Then, Chapter 9 describes the negation-limited circuit complexity for symmetric functions. Finally, in Chapter 10, we investigate relationships between the number of negations available in circuits and the size of the circuits.

Chapter 8

The complexity of negation-limited inverters

A circuit with inputs $x_1, \ldots, x_n$ and outputs $\neg x_1, \ldots, \neg x_n$ is called an *inverter*. In this chapter, we mainly study negation-limited inverters, in which we use only $\lceil \log_2(n+1) \rceil$ NEGATION gates.

Fischer has constructed negation-limited inverters of size $O(n^2 \log^2 n)$ and depth $O(\log n)$. Recently, Tanaka and Nishino have reduced the circuit size to $O(n \log^2 n)$ at the expense of increasing the depth to $\log^2 n$. We construct negation-limited inverters of size $O(n \log n)$, with depth only $O(\log n)$, and we conjecture that this is optimal. We also improve a technique of Valiant for constructing monotone circuits for slice functions (introduced by Berkowitz).

Next, we introduce some lower bound techniques for negation-limited circuits. We provide a $5n + 3\log(n+1) - c$ lower bound for the size of a negation-limited inverter. In addition, we show that for two different restricted classes of circuit, negation-limited inverters require superlinear size.

8.1 Introduction

8.1.1 Background

Markov [65] gives an explicit formula (see below) for the maximum number r of NEGATION gates *required* to compute a system F of boolean functions, without regard to circuit complexity. The maximum value of r for a circuit with n inputs is $\lceil \log(n+1) \rceil$, the number of bits in the binary representation of n (all logarithms in this chapter are base two). We shall denote this number by $b(n)$. Fischer [29, 30] shows that restricting the number of negations in a circuit to $b(n)$ entails only a polynomial blowup in circuit size. This is in sharp contrast to the situation for monotone circuits. É. Tardos [100] has shown that there is an exponential gap between the general complexity and the monotone complexity of some monotone functions. In related work, Santha and Wilson [78] have studied the negation-limited complexity of constant depth circuits, obtaining both upper and lower bounds for the number of NEGATION gates required by circuits of a given depth.

Tanaka and Nishino [91] have found an alternative to Fischer's construction; they decrease the size of the circuits at the expense of increasing the depth. We further reduce the size, while simultaneously reducing the depth to that of Fischer. We also exhibit a relationship between negation-limited complexity of the inverter (see below) and the monotone complexity of certain types of monotone functions.

8.1.2 Definitions and preliminaries

The *inverter* is the collection I_n of functions $f_1, \ldots, f_n$, where for all $1 \leq i \leq n$, $f_i(x_1, \ldots, x_n) = \neg x_i$. For $0 \leq k \leq n$, the (k, n)-*inverter* is the collection I_n^k of functions $f_1, \ldots, f_n$, where for all $1 \leq i \leq n$,

$$f_i(x_1, \ldots, x_n) = \begin{cases} 0 & \text{if } \sum_{j=1}^n x_j < k \\ \neg x_i & \text{if } \sum_{j=1}^n x_j = k \\ 1 & \text{if } \sum_{j=1}^n x_j > k. \end{cases}$$

Let F be a system of n-input boolean functions $f_1, \ldots, f_m$. A *chain* C in the boolean lattice $\{0,1\}^n$ is an increasing sequence $a^1 < \ldots < a^k \in \{0,1\}^n$. The *decrease* of F on C is the number of $i \leq k$ such that for some j, $f_j(a^{i-1}) > f_j(a^i)$. We define $d(F)$ to be the maximum decrease of F on any chain C. Note that $d(F) \leq n$, and this is attained for $F = I_n$. Markov [65] has shown that $b(d(F))$ NEGATION gates are necessary and sufficient to compute any system $f_1, \ldots, f_m$ of functions. Thus, $C^r(F)$ is always defined for $r \geq b(n)$. We call $C^{b(d(F))}(F)$ the *negation-limited* complexity of the system of functions F.

8.1.3 Main results

We give improved upper bounds and lower bounds on the negation-limited complexity of the inverter. We show:

Theorem 8.1 *The negation-limited complexity of the inverter I_n is $O(n \log n)$. In fact, I_n may be computed by negation-limited circuits of size $O(n \log n)$ and depth $O(\log n)$.*

This in turn yields upper bounds on $C^{b(n)}(F)$ for an arbitrary system of functions F, via the standard technique of using DeMorgan's laws to push all negations in a circuit to the inputs:

Corollary 8.2 *For any system F of n-input boolean functions,*

$$C^{b(n)}(F) \leq 2C(F) + O(n \log n).$$

Currently no superlinear lower bound for the general circuit complexity of an explicit boolean function is known. The above Theorem and Corollary imply that if we can show an $\omega(n \log n)$ lower bound on $C^{b(n)}(f)$ for some explicit boolean function f, we also obtain an $\omega(n \log n)$ lower bound on the general circuit complexity of f.

Markov [65] constructs inverters using monotone functions and $b(n)$ negations, but he does not consider the complexity of the monotone functions that he uses. Akers [4] (v. [70]) gives the first explicit construction of a negation-limited inverter. His circuit uses

$b(n)$ NEGATION gates and positive weight threshold gates, and has size $O(n)$ and depth $O(\log n)$. For the remainder of this chapter, we restrict our attention to circuits of AND, OR, and NEGATION gates. Fischer [29] gives such a circuit for I_n with size $O(n^2 \log^2 n)$ and depth $O(\log^2 n)$, using only $b(n)$ NEGATION gates. Sorting networks play a key role in Fischer's construction (and in all subsequent constructions). Using the sorting network of Ajtai, Komlós, and Szemerédi [2, 3] (v. [74]). Fischer's construction reduces the size to $O(n^2 \log n)$ and the depth to $O(\log n)$. Tanaka and Nishino [91] have investigated the negation-limited complexity of the inverter, giving an upper bound of $O(n \log^2 n)$, using a construction of depth $\theta(\log^2 n)$.

Our circuit, with size $O(n \log n)$ and depth $O(\log n)$, improves on previous negation-limited circuits for I_n by a factor of at least $(\log n)$ in size. Note that the only previous construction with logarithmic depth had super-quadratic size.

A circuit with no NEGATION gates is *monotone*. A *monotone function* f is a boolean function which satisfies $f(x_1, \ldots, x_n) \leq f(y_1, \ldots, y_n)$ whenever $x_i \leq y_i$ for all i. Monotone functions are exactly those computed by monotone circuits. Observe that the (k, n)-inverter is a system of monotone functions. Berkowitz [13] shows that the monotone complexity of the (k, n)-inverter is polynomial, with a construction of size $O(n^2 \log n)$ and depth $O(\log n)$. Valiant [104] constructs monotone (k, n)-inverters of size $O(n \log^2 n)$ and depth $O(\log^2 n)$. We show:

Theorem 8.3 *The monotone complexity of the (k, n)-inverter is $O(n \log n)$. In fact, I_n^k has monotone circuits of size $O(n \log n)$ and depth $O(\log n)$.*

We also show (v. Theorem 8.10) that there is a close relationship between $C^0(I_n^k)$ and $C^{b(n)}(I_n)$. (For more background on monotone circuits, we refer to Section 4 of Boppana and Sipser's article [16].)

We also obtain several lower bounds. For arbitrary negation-limited inverters, we obtain:

Theorem 8.4 *Let n be one less than a power of 2. Then any negation-limited circuit for I_n has size $\geq 5n + 3\log(n+1) - c$ and depth $\geq 4\log(n+1) - c$.*

In addition, we show that for two different restricted classes of circuits, negation-limited inverters require size $\Omega(n \log n)$.

The remainder of this chapter is organized as follows. In Section 8.2, we first precisely describe the Tanaka–Nishino inverter, which previously gives the best upper bound. Then, we describe elements of previous constructions of negation-limited inverters which are common to ours. In Section 8.3, we give the heart of our construction of negation-limited inverters. In Section 8.4, we show the relationship between negation-limited inverters and monotone (k, n)-inverters. Section 8.5 gives the main technical lemma on functions computed at the NEGATION gates of negation-limited inverters, and proves Theorem 8.4. In Section 8.6, we prove superlinear bounds for various restricted classes of negation-limited inverters. We conclude in Section 8.7 by mentioning some open problems.

8.2 The Fischer and Tanaka–Nishino inverters

8.2.1 Description of the Tanaka–Nishino inverter

We found an alternative to Fischer's construction in 1994, referred as the Tanaka–Nishino inverter, reducing the size at the expense of the depth. Later in this chapter, we further reduce the size, while simultaneously reducing the depth to that of Fischer. Here, we precisely describe the construction of the Tanaka-Nishino inverter, which might be still interesting and is referred later.

Proposition 8.5

$$C^{b(n)}(I_n) = O(n \log^2 n).$$

Proof. We construct an inverter whose size is $O(n \log^2 n)$. Let M_n be a circuit which takes $T_1^n(X_n), \ldots, T_n^n(X_n)$ as the inputs, and outputs $\neg T_1^n(X_n), \ldots, \neg T_n^n(X_n)$. Our circuit consists of the following three components: a monotone sorting network on n valuable, a circuit M_n (see Figure 8.1), and a monotone $(n, 2n)$-inverter.

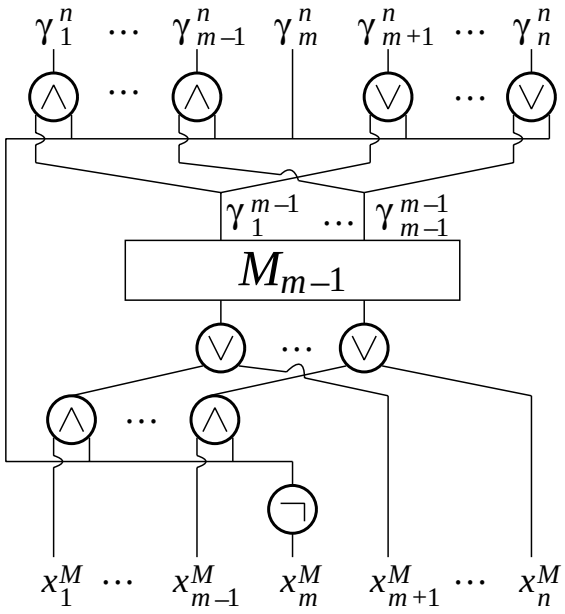

Figure 8.1: A network M_n.

Let $x_1^S, \ldots, x_n^S$ be the inputs and $y_1^S, \ldots, y_n^S$ the outputs of the monotone sorting network. First, we identify the inputs of the monotone sorting network with the input of the inverter as follows:

$$x_i^S = x_i \quad \text{for each } 1 \leq i \leq n.$$

Let $x_1^M, \ldots, x_n^M$ be the inputs and $y_1^M, \ldots, y_n^M$ the outputs of M_n. Next, we identify each input of M_n with the outputs of the sorting network as follows:

$$x_i^M = y_i^S \quad \text{for each } 1 \leq i \leq n.$$

We use the circuit M_n due to Fischer [29], inductively defined as shown in Figure 8.1. Let $n = 2^l - 1$ and $m = 2^{l-1}$. In Figure 8.1, γ_i^n $(1 \leq i \leq n)$ corresponds to y_i^M, and γ_j^{m-1} $(1 \leq j \leq m-1)$ denotes an output of a sub-network M_{m-1}.

Then, from [29], we have

$$y_i^M = \neg T_i^n(X_n) \quad \text{for each } 1 \leq i \leq n,$$

and M_n includes only $\log(n+1)$ NEGATION gates.

Let $x_1^I, \ldots, x_{2n}^I$ be the inputs and $y_1^I, \ldots, y_{2n}^I$ outputs of the monotone $(n, 2n)$-inverter. Finally, we identify the inputs x_i^I of the monotone $(n, 2n)$-inverter with the inputs of the inverter and the outputs of M_n as follows:

$$x_i^I = \begin{cases} x_i & \text{for each } 1 \leq i \leq n, \\ y_{i-n}^M & \text{for each } n+1 \leq i \leq 2n. \end{cases}$$

The inverter constructed as above is shown in Figure 8.2.

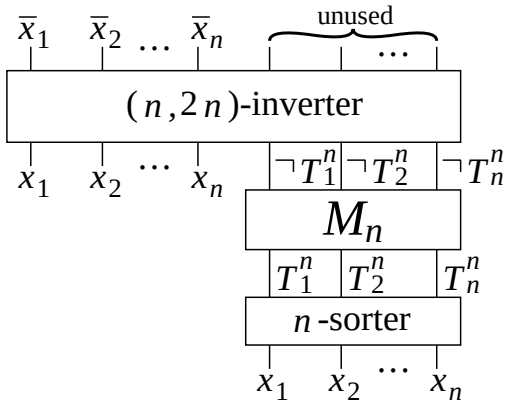

Figure 8.2: The inverter.

Let $M_n(X_n)$ denote the output of M_n when M_n is given X_n as the inputs. We denote by $\#_1(X)$ the number of ones in X, and by $\#_0(X)$ the number of zeros in X. Then, it follows that

$$\#_1(M_n(X_n)) = \#_0(X_n).$$

It is obvious that $\#_0(X_n) = n - \#_1(X_n)$, so $\#_1(M_n(X_n)) = n - \#_1(X_n)$. Thus, we have

$$\begin{aligned} \#_1(x_1^I, \ldots, x_{2n}^I) &= \#_1(x_1^I, \ldots, x_n^I) + \#_1(x_{n+1}^I, \ldots, x_{2n}^I) \\ &= \#_1(X_n) + \#_1(y_1^M, \ldots, y_n^M) \\ &= \#_1(X_n) + \#_1(M_n(X_n)) \\ &= \#_1(X_n) + (n - \#_1(X_n)) \\ &= n. \end{aligned}$$

Hence, $y_i^I = \overline{x_i^I}$ for all $1 \leq i \leq 2n$, and in particular, $y_i^I = \overline{x_i}$ for all $1 \leq i \leq n$ (see Figure 8.2).

It is known that the size of a monotone sorting network is $O(n \log n)$ [2]. Notice here that we can get a much smaller constant factor, if we use the Batcher network [10] whose size is $O(n \log^2 n)$ in our construction. This is enough to establish the bound stated in the theorem. The size of a monotone (k, n)-inverter is $O(n \log^2 n)$ for any k [104]. Finally, the size $S(n)$ of the circuit M_n satisfies the following relation:

$$S(n) = 2n - 1 + S(\frac{n-1}{2}).$$

Thus, by an easy induction on n, we have

$$S(n) \leq 4n.$$

Hence, $C^{b(n)}(M_n) = O(n)$. This completes the proof of Proposition 8.5. ∎

Notice that our inverter has depth $O(\log^2 n)$ while Fischer's inverter with a sorting network by Ajtai, Komlós, and Szemerédi has depth $O(\log n)$.

8.2.2 Common elements of the inverters

The circuits of Fischer [29] and Tanaka–Nishino [91] have several elements in common with each other, and with ours as well. The most important of these is a sorting network.

A *comparator* is a circuit element with two inputs and two outputs. The inputs are taken from some ordered set U. The first output of the comparator is the maximum of the two inputs, while the second output is the minimum of the two inputs. A sorting network is a circuit, with n inputs and n outputs, composed entirely of comparators. For any n-tuple $x_1, \ldots, x_n$ of inputs from U, the n outputs $y_1, \ldots, y_n$ of the sorting network are a permutation of the x_i and are in decreasing order. Ajtai, Komlós, and Szemerédi [2] (v. [74]) have constructed sorting networks using $O(n \log n)$ comparators, organized in $O(\log n)$ levels of $\lfloor n/2 \rfloor$ comparators each. Note that if the set U is $\{0, 1\}$, then a comparator can be constructed without negations. Indeed, for $x, y \in \{0, 1\}$, $\min(x, y) = x \wedge y$ and $\max(x, y) = x \vee y$. Thus, the Ajtai–Komlós–Szemerédi result yields a monotone circuit of size $O(n \log n)$ and depth $O(\log n)$ for sorting n bits.

Both Fischer and Tanaka–Nishino first sort the n input bits $x_1, \ldots, x_n$. The outputs $y_1, \ldots, y_n$ of the sorting network are fed into a subcircuit M_n (see Figure 8.1,due to Fischer [29]) with the following properties:

1. M_n has n binary inputs $y_1, \ldots, y_n$ and n outputs $z_1, \ldots, z_n$.
2. M_n has size $O(n)$, depth $O(\log n)$, and uses $b(n)$ NEGATION gates.
3. If $y_1 \geq y_2 \geq \ldots \geq y_n$ then $z_i = \neg y_i$ for $1 \leq i \leq n$.

The negations of the x_i may now be calculated monotonically from the x_i, the y_i, and the z_i. Fischer does this using $O(n)$ sorting networks in parallel, resulting in a circuit of size $O(n^2 \log n)$ and depth $O(\log n)$ if the Ajtai–Komlós–Szemerédi sorting network is used. Tanaka and Nishino, on the other hand, use the monotone $(n, 2n)$-inverter of Valiant [104] which inverts $2n$ inputs provided that exactly n of them equal 1 (as is the case with $x_1, \ldots, x_n, z_1, \ldots, z_n$). Valiant's circuit is monotone, with size $O(n \log^2 n)$ and depth $O(\log^2 n)$.

Note that in both the Fischer and the Tanaka–Nishino constructions, the circuit is arranged in 3 phases: the sorting network, the subcircuit M_n, and the top phase which computes the final answer. (We view circuits as having their inputs at the bottom and outputs at the top, so the final phase of a circuit with several phases is the top phase.) The bottom two phases require only size $O(n \log n)$ and depth $O(\log n)$ (in fact, the second phase has linear size). Also, both the Fischer and the Tanaka–Nishino constructions require no more than the x_i, the y_i, and the z_i as inputs to the top phase. We show that by allowing the top phase access to intermediate results from the sorting network (first phase), the top phase may be computed by a size $O(n \log n)$, depth $O(\log n)$ circuit. Our top phase may be thought of as an "upside-down" sorting network. We describe this in the next section.

8.3 Description of the inverter

Our negation-limited circuit for I_n, like its predecessors, begins with a sorting network and Fischer's M_n circuit. We present a new approach to the third phase, which achieves considerable savings in complexity over previous constructions by means of a novel application of the Ajtai–Komlós–Szemerédi result.

Before going into the details of the circuit, we describe briefly the intuition. The differences between the three constructions ([29, 91] and ours) seem to stem from how one thinks of the outputs of the M_n subcircuit. Of course, in all three constructions, the outputs of M_n are computing exactly the same system of n functions. Nevertheless, these outputs are treated completely differently by the three approaches. In Fischer's original negation-limited inverter [29], the inputs and outputs of M_n are thought of as the threshold functions (of $x_1, \ldots, x_n$) and their negations, respectively. Fischer uses these bits to select the output of an appropriate subcircuit, according to the number k of ones in the input. There are $n+1$ such subcircuits, one for each $0 \leq k \leq n$. Tanaka and Nishino [91] avoid having to consider $(n+1)$ different cases in parallel by noticing that the x_i, together with the outputs of M_n, are a string of $2n$ bits containing exactly n ones (such a string can be inverted monotonically using Valiant's monotone $(n, 2n)$-inverter [104]). In their construction, the outputs of M_n are thought of as *balancing* the inputs $x_1, \ldots, x_n$. We take a different view. We view the outputs of M_n as being a *permutation* of the negations of the x_i. It is the task of the third phase to *rearrange* these bits into their proper position. This rearrangement essentially undoes the sorting performed by the first phase, one level at a time.

We now present the details. Let ℓ be the depth of the sorting network used in the first phase. For each level in the sorting network, the third phase has a corresponding level, composed of a monotone, linear size, constant depth circuit. The ordering of the levels, however, is reversed between the first phase and the third phase. That is, we number the levels in the sorting network from the bottom (numbered 1) to top (numbered ℓ), and we number the levels in the third phase from 1 at the top to ℓ at the bottom. When numbered in this way, level i of the sorting network corresponds to level i of the third phase. We will use S_i to denote level i of the sorting network, and T_i to denote level i of the top phase.

For $1 \leq i \leq \ell$, let $x_1^i, \ldots, x_n^i$ denote the inputs to S_i, and let $x_1^{\ell+1}, \ldots, x_n^{\ell+1}$ denote the

outputs of S_ℓ. The subcircuit T_i receives $2n$ input bits: the n input bits $x_1^i, \ldots, x_n^i$ of S_i, and the negations $\neg x_1^{i+1}, \ldots, \neg x_n^{i+1}$ of the n output bits of S_i. For $1 \leq j \leq n$, $\neg x_j^{i+1}$ is computed one level below by T_{i+1}, or by M_n in the case $i = \ell$. Using only linear size, constant depth, monotone circuitry, T_i outputs the negations $\neg x_1^i, \ldots, \neg x_n^i$ of the n input bits to S_i. If $i > 1$, these are passed up to the next level, T_{i-1}. Note that at the top level, T_1 outputs the negations of the inputs to S_1, i.e., T_1 outputs $\neg x_1, \ldots, \neg x_n$. The overall structure of the circuit is shown in Figure 8.3. Note that, for readability, the wires connecting the outputs of S_{i-1} to the inputs of T_i are not shown.

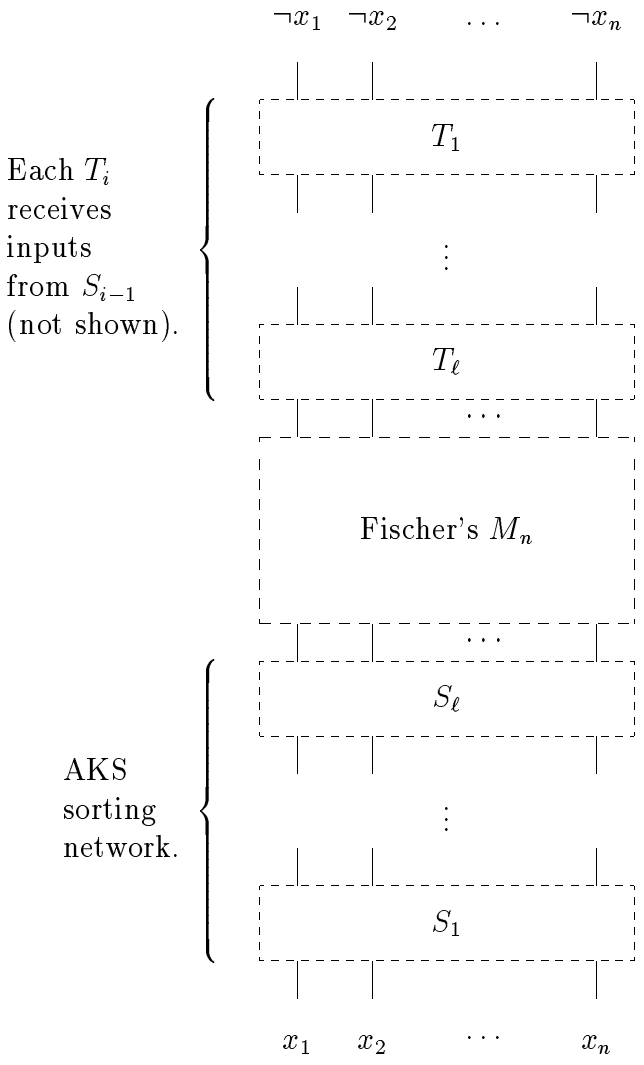

Figure 8.3: Overall structure of the circuit.

To complete the description of the circuit for I_n, we need to describe a monotone, linear size, constant depth circuit for T_i. Since S_i is simply $\lfloor n/2 \rfloor$ comparators operating in parallel, it suffices to give a (constant size and depth) monotone circuit for each comparator. This we do in the following Lemma:

Lemma 8.6 *Let x and y be boolean variables, and let $u = \neg \max(x, y)$ and $v = \neg \min(x, y)$. Then $\neg x$ and $\neg y$ may be computed monotonically from x, y, u, and v.*

Proof. A quick check shows that $\neg x = u \vee (y \wedge v)$ and $\neg y = u \vee (x \wedge v)$. ∎

Suppose that S_i has a comparator with inputs $x^i_{j_1}, x^i_{j_2}$ and outputs $x^{i+1}_{k_1}, x^{i+1}_{k_2}$. Then T_i will have a corresponding circuit element with inputs

$$\neg x^{i+1}_{k_1}, \neg x^{i+1}_{k_2}, x^i_{j_1}, x^i_{j_2}$$

and outputs

$$\neg x^i_{j_1}, \neg x^i_{j_2}.$$

By the above Lemma, this can be done monotonically. This is illustrated in Figure 8.4. As S_i is a linear size, depth 1 circuit of comparators, we can construct T_i to be a monotone, linear size, depth 2 circuit.

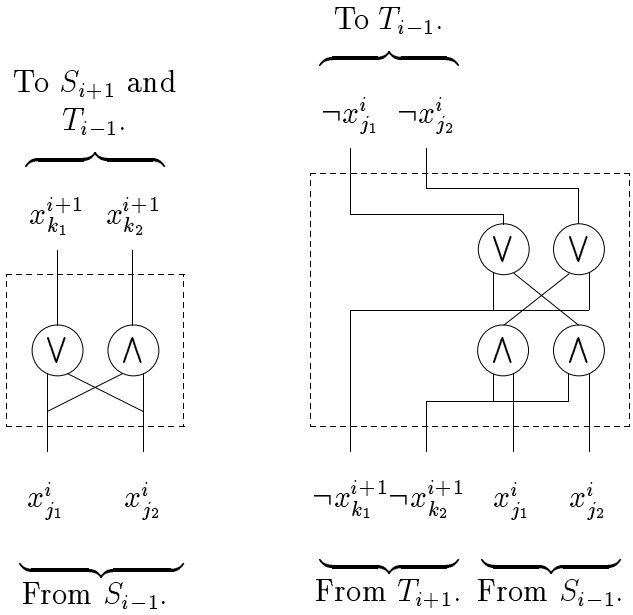

Figure 8.4: Corresponding circuit elements of S_i (left) and T_i (right).

We are now ready to prove Theorem 8.1:

Proof. We describe the circuit. As in Fischer [29] and Tanaka–Nishino [91], the input variables $x_1, \ldots, x_n$ are the inputs of a sorting network with ℓ levels of $O(n)$ gates each. By Ajtai–Komlós–Szemerédi [2], we may assume $\ell = O(\log n)$. Still following Fischer and Tanaka–Nishino, the outputs $y_1, \ldots, y_n$ of the sorting network are the inputs to Fischer's M_n circuit, which has size $O(n)$ and depth $O(\log n)$ and uses $b(n)$ NEGATION gates. Since the inputs to M_n are sorted, the outputs $z_1, \ldots, z_n$ satisfy $z_i = \neg y_i$ for $i = 1, \ldots, n$.

From the z_i, and the inputs to the various levels of the sorting network, the subcircuits $T_\ell, \ldots, T_1$ described above compute the negations of the inputs to successively lower levels in the sorting network. It is clear by induction on $(\ell - i)$ that T_i outputs the negations

$\neg x_1^i, \ldots, \neg x_n^i$ of the inputs to S_i. At the top of our circuit, T_1 computes the negations $\neg x_1, \ldots, \neg x_n$ of the input variables.

The circuit has depth $3\ell + O(\log n) = O(\log n)$. As each level of the circuit has linear size, the total number of gates used is $O(n \log n)$. The only NEGATION gates used are the $b(n)$ NEGATION gates used by M_n, as desired. ∎

8.4 Monotone (k, n)-inverters

Berkowitz [13] introduces the notion of a *slice function*. The boolean function $f : \{0,1\}^n \to \{0,1\}$ is a kth slice if

$$f(x_1, \ldots, x_n) = \begin{cases} 0 & \text{if } \sum_{i=1}^n x_i < k \\ 1 & \text{if } \sum_{i=1}^n x_i > k. \end{cases}$$

For input strings containing exactly k ones, the behavior of f is not constrained. Any slice function is monotone. Note that an alternative, nonconstructive proof of Lemma 8.6 may be obtained by observing that, of the four bits x, y, u, v, exactly two are equal to one, so any function of x, y, u, v may be extended to a (monotone) slice function. The importance of monotone (k, n)-inverters is underscored by the following:

Proposition 8.7 *For any system F of n-input kth slice functions,*

$$C^0(F) \leq 2C(F) + C^0(I_n^k).$$

The following allows us to focus on the behavior of our circuit on inputs with exactly k ones:

Proposition 8.8 *I_n^k is the unique system of monotone functions which agrees with I_n on all inputs with exactly k ones.*

Proof. Suppose that $f_1, \ldots, f_n$ are a system of monotone functions which agree with I_n on all inputs with exactly k ones. Suppose that the number of ones in $x_1, \ldots, x_n$ is strictly greater than k. Let i be any integer between 1 and n. We must show that $f_i(x_1, \ldots, x_n) = 1$. By the monotonicity of f_i, it suffices to exhibit $y_1, \ldots, y_n$ such that $f_i(y_1, \ldots, y_n) = 1$ and for all $1 \leq j \leq n$, $y_j \leq x_j$. For this, we assign the y_j in such a way that exactly k of them are one, $y_i = 0$, and each $y_j \leq x_j$ (we obtain the y_j by changing some of the x_j from one to zero, including x_i if it is nonzero, until exactly k bits are one). The proof for the case that fewer than k of the x_j are one is analogous. ∎

Berkowitz [13] constructs monotone (k, n)-inverters of size $O(n^2 \log n)$ and depth $O(\log n)$. Valiant [104] gives monotone circuits of size $O(n \log^2 n)$ and depth $O(\log^2 n)$ for the (k, n)-inverter (see also Wegener [105, 106]). Our techniques can be used to improve on Valiant's results by a factor of $(\log n)$ in size and depth. We present the proof of Theorem 8.3:

Proof. We use our negation-limited circuit for I_n, replacing the M_n subcircuit by a circuit which outputs k zeroes followed by $(n-k)$ ones. Note that the only negations in our circuit for I_n occur in the M_n subcircuit. Also, if there are exactly k ones among the inputs to our

negation-limited inverter, then the M_n subcircuit will output k zeroes followed by $(n-k)$ ones. Therefore, the new circuit will agree with I_n (i.e., it will compute the negations of its input bits) whenever there are exactly k ones in the input. By Proposition 8.8, we are done. ∎

Since the inputs to M_n are sorted, the functions computed by the NEGATION gates depend only on the number of ones in the input. Therefore, in the above construction, instead of replacing the outputs of M_n by constants, we could have replaced the outputs of the NEGATION gates by constants. One might ask whether it is always possible to obtain a monotone (k,n)-inverter from a negation-limited inverter by replacing the negation gates with constants. We shall see (perhaps surprisingly) that something close to this is true. Before formalizing this, we quote the following Lemma of Tanaka–Nishino (v. [91, Lemma 4.2]):

Lemma 8.9 (Tanaka–Nishino) *Suppose $n+1$ is a power of 2, and let f be the function computed by some NEGATION gate of a negation-limited circuit for I_n. Then $f(x_1,\ldots,x_n)$ depends only on the number of i such that $x_i = 1$.*

We will prove a stronger version of this Lemma later (v. Lemma 8.12).

Now we can make precise the relationship between negation-limited inverters and monotone (k,n)-inverters:

Theorem 8.10 *Let n be a positive integer. Then $C^0(I_n^k) \leq C^{b(n')}(I_{n'})$, for some $n' < 2n$. In fact, a monotone (k,n)-inverter can be constructed with the same size and depth as a negation-limited circuit for $I_{n'}$.*

Proof. Let $n' = 2^r - 1$, where 2^r is the smallest power of 2 greater than n. Note that 2^r is at most $2n$. Let Δ be any negation-limited circuit for $I_{n'}$. By Lemma 8.9, if the number of ones in the input is constant, then the NEGATION gates in Δ compute constant functions. Replacing these gates by appropriate (according to k) constants yields a monotone (k,n')-inverter. To obtain a monotone (k,n)-inverter, we simply fix the first $(n'-n)$ input bits to 0, and ignore the first $(n'-n)$ output bits. This yields a monotone circuit Δ' computing I_n^k with the same size and depth as Δ. ∎

In fact, *all* known constructions of monotone (k,n)-inverters may be seen as negation-limited inverters with the NEGATION gates replaced by constants. This modification applied to Fischer's circuit for I_n (using the AKS sorting network [2]) yields Berkowitz' (k,n)-inverter. There is no previously published negation-limited inverter corresponding to Valiant's (k,n)-inverter, so we describe one here. First we give a sketch of Valiant's construction.

Valiant's circuit is very similar to ours (with the M_n removed), except that instead of using comparators, *merging networks* are used. A merging network takes as inputs two sorted lists u and v, and outputs the list $w = \mathrm{merge}(u,v)$ obtained by sorting the concatenated list uv. Batcher [10] shows that two lists of m elements can be merged by a network of $O(m \log m)$ comparators. Thus, if u and v are binary strings of length m, they can be merged by a monotone circuit of size $O(m \log m)$ (this circuit has depth $O(\log m)$). If u is a binary string, we denote by $\neg u$ the string obtained from u by replacing each symbol by its negation, and we denote by $\mathrm{reverse}(u)$ the string obtained from u by

reversing the order of the symbols. Note that if u is in sorted (i.e., decreasing) order, then so is reverse($\neg u$).

The bottom half of Valiant's circuit is a sorting network, built out of $\log n$ layers of merging networks (the ith layer merges pairs of consecutive substrings of length 2^i). The top half is analogous to our top phase, with the following taking the place of Lemma 8.6:

Lemma 8.11 (Valiant) *Suppose that the sorted binary strings u of length s and v of length t, together with the string $w =$* reverse($\neg$merge(u, v)) *are given. Then* reverse($\neg u$) *and* reverse($\neg v$) *may be calculated by merging networks.*

Proof. By symmetry, we only need to consider reverse($\neg u$). This is easily seen to be the middle s bits of merge(v, w). ∎

(Note that the case $s = t = 1$ of Valiant's Lemma provides yet another proof of Lemma 8.6.)

Since k is fixed, the negations of the output bits of the top layer of the bottom half are constants. (Actually, this top layer of the bottom half is superfluous for the monotone (k, n) inversion problem, and Valiant does without it. We leave it in for simplicity, and because it is necessary if we are to obtain a negation-limited inverter.) Successively higher layers of the top half calculate the reverses of the negations of inputs to merging networks in successively lower layers of the bottom half. At the top layer of the top half, the negations of the input bits are calculated (reversing a string of length one has no effect).

To obtain a negation-limited inverter from Valiant's circuit, we simply feed the output of the bottom half to Fischer's M_n circuit, which computes the negation of the output of the top layer of the bottom half. This is then given as input to the top half. (The circuit constructed in this way is about half the size of that of Tanaka and Nishino [91], since we modify Valiant's (k, n)-inverter instead of using Valiant's $(n, 2n)$ inverter.) Therefore, Valiant's techniques for constructing monotone (k, n)-inverters can also be used to construct negation-limited inverters. It is curious that this is true of all known (k, n)-inverters, since no converse of Theorem 8.10 is known to hold.

8.5 Lower bounds for the inverter

In this section, we shall prove a main technical lemma on functions computed at NEGATION gates in negation-limited inverters. These techniques have recently been generalized to obtain results for negation-limited circuits computing symmetric functions [11].

For $x \in \{0,1\}^n$, let $|x|$ denote the number of ones in x. Note that in all known constructions of negation-limited inverters, the negation gates compute the negations of the bits of $|x|$. We shall see that this must always hold:

Lemma 8.12 *Let $n = 2^r - 1$. Consider any r-circuit Δ which computes I_n. Label the NEGATION gates $N_1, \ldots, N_r$ in such a way that the input to N_i does not depend on the outputs of any of the negation gates $N_{i+1}, \ldots, N_r$ (such a labeling exists since the circuit is a DAG). Let z_i and y_i be the functions computed at the input and output of N_i. Then $z_1 z_2 \ldots z_r$ is the binary representation of $|x|$.*

Proof. We proceed by induction on r. The Lemma is trivially true for $r = 0$ (i.e., the circuit is monotone). We now suppose that the Lemma holds for $r - 1$ NEGATION gates.

Let $a \in \{0,1\}^n$. We wish first to show that $z_1(a)$ is the first (i.e., leftmost) bit of $|a|$. We consider the case that $z_1(a) = 1$ (the proof for the case $z_1(a) = 0$ is analogous), and we wish to show that $|a| \geq (n+1)/2$. Note that simultaneously permuting the inputs and output of Δ according to the same permutation yields another circuit for I_n, so we may assume that a is of the form $1^k 0^{n-k}$. Also, I_{n-k} is obtained from I_n fixing the first k bits to 1 and ignoring the first k output bits. Since $z_1(a) = 1$, and z_1 is monotone, I_{n-k} is computed by the circuit Δ_k obtained by fixing the first k inputs of Δ to 1 and replacing N_1 by the constant 0. Note that Δ_k is an $(r-1)$-circuit. Therefore, by Markov's Theorem, $d(I_{n-k}) \leq 2^{r-1} - 1 = (n-1)/2$. However, it is clear that $d(I_{n-k}) = n - k$. Therefore

$$|a| = k = n - d(I_{n-k}) \geq n - (n-1)/2 = (n+1)/2,$$

as desired.

Now it remains to show that the functions $z_2, \ldots, z_r$ are as specified. Again, we restrict our attention to the case $z_1 = 1$, since the case $z_1 = 0$ is handled similarly. Let $a \in \{0,1\}^n$ such that $z_1(a) = 1$. As above, we assume that a is a string of the form $1^\ell 0^{n-\ell}$. Let $k = (n-1)/2 \leq \ell$. We apply the $r-1$ case of the Lemma to the $(n-k)$-input/output inverter obtained from Δ by fixing the first k inputs to 1 and replacing N_1 by the constant 0. The proof is completed by observing that for all $x \in \{0,1\}^{n-k}$, the binary representations of $|x|$ and $|1^k x|$ agree in their $(r-1)$ least significant bits. ∎

In the sequel, we assume that $z_1, \ldots, z_r$ are as in the Lemma. Let N_i be the NEGATION gate corresponding to z_i. We remark that we did not assume that there exists a path from N_{i-1} to N_i for all $2 \leq i \leq r$. (By path we mean a sequence of gates $G_1, \ldots, G_p$ which satisfies the following conditions: $G_1 = N_i$, $G_p = N_{i+1}$, and for any q $(1 \leq q \leq p-1)$, an output of G_q is an input of G_{q+1}.) However, the existence of such paths follows at once from the Lemma: If for some i, there did not exist a path from N_{i-1} to N_i, then we could interchange the labels of N_{i-1} and N_i. This is impossible, since by the Lemma the labeling of a negation gate is determined by what function is computed at the gate.

Lemma 8.13 *For any i, $1 \leq i \leq r-1$, in any path from N_i to N_{i+1}, there exists at least one AND gate and one OR gate.*

Proof. The proof is by contradiction. By Lemma 8.12, all four possible values for z_i and z_{i+1} are attained for some input $x \in \{0,1\}^n$. Note that the NEGATION gates are labeled in such a way that no path from N_i to N_{i+1} contains a NEGATION gate other than N_i and N_{i+1}. If the nodes along a path from N_i to N_{i+1} consisted entirely of AND (respectively OR) gates, then the possibility $z_i = 1, z_{i+1} = 1$ (respectively $z_i = 0, z_{i+1} = 0$) would be ruled out. Therefore, all such paths must include both an AND gate and an OR gate. ∎

Let L_j $(1 \leq j \leq n)$ be a gate computing $\neg x_j$ in Δ.

Lemma 8.14 *For all $1 \leq i \leq r$ and $1 \leq j \leq n$, there exists a path from N_i to L_j which does not include any NEGATION gate except for N_i.*

Proof. First, we fix all the outputs of the NEGATION gates $N_1, \ldots, N_r$ in Δ to 0. Then, we obtain an (n,n)-inverter C from Δ. In this case, $C(a) = (0, \ldots, 0)$ for all $a \in \{0,1\}^n$. Let us fix all the inputs of the circuit C to 1. Next, for some $1 \leq i \leq r$, we change the output of N_i to 1. Then, for some $1 \leq k \leq n-1$, we obtain a (k,n)-inverter by the proof

of Theorem 8.10. Since we fixed all the inputs of C to 1, each of the outputs of $L_1, \ldots, L_n$ in C changes from 0 to 1. Thus, for all $1 \leq j \leq n$, there exists a path from N_i to L_j, such that the output of every gate in the path changes from 0 to 1. Since we fixed all the outputs of the NEGATION gates except for N_i to 0, we can conclude that there is no other NEGATION gate in the path. ∎

Lemma 8.15 *For all j, $1 \leq j \leq n$, on any path from N_r to L_j, there exist at least one AND gate and one OR gate.*

Proof. Note that at least one such path exists by the previous Lemma. Now we wish to show that any such path contains an AND gate and an OR gate. As above, we note that by Lemma 8.12, each of the four possibilities for the values of z_i, L_j are attained for some input $x \in \{0,1\}^n$. Also, there is a path from N_r to L_j which contains no other NEGATION gate. Therefore, as in the previous Lemma, we see that any path from N_r to L must include at least one AND gate and at least one OR gate (all other possibilities lead to contradictions). ∎

Theorem 8.16 *Let $n = 2^r - 1$. Then $C^r(I_n) \geq 5n + 3\log(n+1) - c$.*

Proof. By Lemma 8.12, z_1 is the majority function, computed by a monotone subcircuit, so the number of gates required to compute z_1 is at least $C^0(T^n_{(n+1)/2})$. Since it is known that $C^0(T^n_{n/2}) \geq 4n - c$ where c is a constant (see [63]), we have $C^0(T^n_{(n+1)/2}) \geq 4n - c$.

From Lemma 8.13, the length of any path from N_1 to N_r is at least $3r - 2$. Note that the gates on such a path are distinct from the gates in the subcircuit computing z_1.

Furthermore, from Lemma 8.14, for all $1 \leq j \leq n$, there exists a path from N_r to L_j. So, for all $1 \leq j \leq n$, L_j differs from any gate in any path from N_1 to N_r. And the functions computed at $L_1, \ldots, L_n$ are mutually distinct non-monotone functions (i.e., $\neg x_1, \ldots, \neg x_n$). Hence, we have

$$\begin{aligned} C^r(I_n) &\geq C^0(T^n_{(n+1)/2}) + (3r-2) + n \\ &\geq (4n - c') + 3r - 2 + n \\ &\geq 5n + 3\log(n+1) - c \end{aligned}$$

as desired. ∎

Theorem 8.17 *Let $n = 2^r - 1$. Then $D^r(I_n) \geq 4\log(n+1) - c$.*

Proof. From Lemma 8.12, the number of gates used to compute z_1 is at least $C^m(T^n_{(n+1)/2})$. These gates are located below N_1, that is, there exists a path from each of these gates to N_1.

From Lemma 8.13, the length of any path from N_1 to N_r is at least $3r-2$. Furthermore, from Lemma 8.15, for all $1 \leq j \leq n$, the length of any path from N_r to L_j is at least 3. Then, we obtain

$$\begin{aligned} D^r(I_n) &\geq \lceil \log(C^m(T^n_{(n+1)/2}) + 1) \rceil + (3r-2) + 3 - 1 \\ &\geq \lceil \log(4n - c') \rceil + 3r \\ &\geq \log(n+1) - c'' + 3r \\ &\geq 4\log(n+1) - c \end{aligned}$$

as desired. ∎

8.6 Superlinear lower bounds for particular inverters

First, we consider so-called *synchronous circuits* which are circuits with the property that all paths from the inputs to a given gate have the same length (see [80]).

Theorem 8.18 *Let $n = 2^r + 1$. Then any synchronous negation-limited circuit for I_n has at least $4n\log(n+1) - cn$ gates.*

Proof. From Theorem 8.17, for each $1 \leq i \leq n$, the length of some (and therefore any) path from an input to L_i is at least $4\log(n+1) - c$. In a synchronous circuit computing I_n, the outputs of the gates at level m ($2 \leq m \leq 4\log(n+1) - c$) are computed using the outputs of the gates at level $m-1$.

On the other hand, I_n can compute 2^n different values, i.e., all binary vectors of length n. Hence, the number of gates at level m is at least $\log 2^n = n$. This completes the proof. ∎

Next, we show another instance where the size of the negation-limited inverter has a superlinear lower bound. For that purpose, we introduce the following restriction.

> **Restriction A**: Any gate G in the inverter which satisfies the following two conditions computes some symmetric function:
>
> **A1** there exists a path from one of $N_1, \ldots, N_{r-1}$ to a gate G in Δ.
>
> **A2** there exists a path from G to N_r in Δ.

Theorem 8.19 *Let $n = 2^r - 1$. Then any negation-limited inverter Δ which satisfies Restriction A has $\Omega(n \log n)$ gates.*

Proof. Label the NEGATION gates of Δ as in Lemma 8.12. Let z_i be the function which is computed at the input of N_i. By Lemma 8.12, for $x \in \{0,1\}^n$ the string $z_1(x)z_2(x)\ldots z_r(x)$ is the binary representation of $|x|$. This is an increasing function which attains all possible values in the range $0, \ldots, n$.

Let $1 \leq k \leq n$ be arbitrary. We wish to show that there exists a monotone subcircuit of Δ computing T_n^k. Let w be such that $w01^{r-i}$ is the binary representation of $k-1$, where w is a binary string $w_1w_2\ldots w_i$ of length i for some $0 \leq i < r$. Then the binary representation of k is $w10^{r-i}$.

Let Δ' be a circuit obtained from Δ by replacing N_j with the constant $\neg w_j$ for each $1 \leq j \leq i$. For a gate G in Δ, let G' denote the corresponding gate in Δ'. Let g and g' denote the functions computed at gates G and G', respectively. Let $\Omega \subseteq \{0,1\}^n$ denote the set of all binary strings with exactly $k-1$ or k ones. We have:

1. For any gate G in Δ, and for any $x \in \Omega$, $g(x) = g'(x)$.
2. If for all $j > i$, there is no path from N_j to G in Δ, then g' is computed by a monotone subcircuit of Δ'.
3. If g is computed by a monotone subcircuit of Δ, then $g = g'$.

Let G be the gate computing z_{i+1}. Note that by (2), g' is monotone, and by (1), $g'(x) = g(x) = T_n^q(x)$ for $x \in \Omega$. Since T_n^k is the unique monotone function which agrees with T_n^k on Ω, we have $g'(x) = T_n^k(x)$ for all $x \in \{0,1\}^n$.

Recall that we are trying to prove that there is a monotone subcircuit of Δ computing T_n^k (from which the theorem follows). So far we have shown the existence of a gate G in Δ such that $g' = T_n^k$. It now suffices to show that for any such gate, either g is computed by a monotone subcircuit of Δ (so, by (3), $g = g'$ and we are done) or there is a gate lower down in the circuit with the same property.

Suppose that g is not computed by a monotone subcircuit of Δ. Then there is a path from some negation gate N_j to G in Δ. This is illustrated in Figure 8.5.

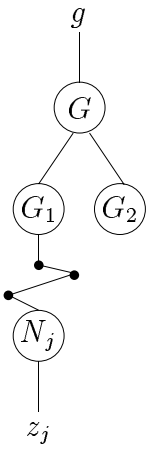

Figure 8.5: A gate G such that G' computes T_n^k. One of G_1, G_2 shares this property.

By Restriction A, we know that g is a symmetric function. Let G_1 and G_2 be the gates whose outputs are the inputs to G, labeled such that a path from N_j to G passes through G_1. Therefore g_1 computes a symmetric function. Since g_1' is a monotone function which agrees with g_1 on Ω, either $g_1' = T_n^k$ or g_1' is constant on Ω. In the latter case, since g' is not constant on Ω, we have that either g_1' is identically 1 on Ω and G is an AND gate, or g_1' is identically 0 on Ω and G is an OR gate. So g' and g_2' agree on Ω, and $g_2' = T_n^k$. In either case we have found a gate G_ℓ below G in Δ with $g_\ell' = T_n^k$ as desired. This can be continued until a monotone subcircuit of Δ is found which computes T_n^k.

Since k above was arbitrary, all threshold functions $\{T_n^1, T_n^2, \ldots, T_n^n\}$ are computed by monotone subcircuits of Δ. Thus, from [52], Δ has size $\Omega(n \log n)$. ∎

Notice that, if we do not have the condition A2, certain symmetric functions must be computed at the output gates of Δ_n. But this is impossible. Hence, the condition A2 is necessary although it is not used in the proof of Theorem 8.19.

Since the inverter we construct satisfies Restriction A and has size $O(n \log n)$, this Theorem is tight to within a constant factor.

8.7 Conclusion

In this chapter, we have presented new upper and lower bounds on the size of negation-limited inverters. We have also investigated the negation-limited circuit complexity of some boolean functions, and have shown relationships between combinational complexity and negation-limited complexity of boolean functions.

It should be investigated whether the methods used in this chapter to derive lower bounds on the negation-limited complexity of the inverter can be applied to derive similar bounds on the negation-limited complexity of one-output boolean functions. We have made some progress in this regard for symmetric functions [11, 93].

G. Turán posed the following question: is the size of any $c \log n$ depth inverter using $c \log n$ NEGATION gates superlinear? Though Turán's question remains open, we hope that our work represents a step towards its resolution.

Chapter 9

Negation-limited circuit complexity for symmetric functions

In this chapter, we investigate the complexity of negation-limited circuits which compute symmetric functions. First, we shall prove a main technical lemma on functions computed at NEGATION gates in negation-limited circuits computing symmetric functions. Using this lemma, we show a number of lower bounds on the size and depth of negation-limited circuits computing several symmetric functions such as PARITY_n, $\neg\text{PARITY}_n$, MOD_n^k and others. For example, a $4n + 3\log_2(n+1) - c$ lower bound is given on the size of circuits computing PARITY_n using $\lceil \log_2(n+1) - 1 \rceil$ of NEGATION gates. Furthermore, we show nonlinear lower bounds on the size of certain kinds of negation-limited circuits computing symmetric functions.

9.1 Introduction

We continue the lower bound work of Tanaka and Nishino from [91], obtaining several bounds for symmetric functions. For a negation-limited circuit computing the parity function of n bits, we obtain a $4n + 3b(n) - c$ size lower bound and a $4b(n) - c$ depth lower bound.

Definitions and preliminaries

Let $f = (f_1, \ldots, f_m)$ be a system of n-input boolean functions. A *chain* C in the boolean lattice $\{0,1\}^n$ is an increasing sequence $a^1 < \ldots < a^k \in \{0,1\}^n$. The *decrease* of f on C is the number of $i \leq k$ such that for some j, $f_j(a^{i-1}) > f_j(a^i)$. We define $d(f)$ to be the maximum decrease of f on any chain C. Notice that $d(f) \leq n$ (this is attained for some systems f). Markov [65] has shown that $b(d(F))$ NEGATION gates are necessary and sufficient to compute any system of boolean functions, where $b(x)\lceil\log(x+1)\rceil$, the number of bits in the binary representation of x (all logarithms in this chapter are base two). Thus, $C^r(f)$ is always defined for $r \geq b(n)$. We call $C^{b(d(f))}(f)$ the *negation-limited* complexity of a system f of boolean functions.

Main results

For symmetric boolean functions, we obtain several lower bounds. These depend on a technical result, Lemma 9.2, which determines the functions computed by the NEGATION gates in a negation-limited circuit for a symmetric function. This gives us a great deal of knowledge about the structure of *any* negation-limited circuit for a symmetric function, and allows lower bounds for monotone circuits to be used. For PARITY function, we obtain the following:

Theorem 9.1 *Let f be the parity function of n bits, where $n+1$ is a power of two. Then, a negation-limited circuit for f must have size at least $4n+3\log(n+1)-O(1)$, and depth at least $4\log(n+1)-O(1)$.*

We obtain similar results for MOD functions, and for the complement of PARITY. We also obtain an $\Omega(n \log n)$ lower bound for a restricted type of negation-limited PARITY circuit (specifically those circuits satisfying Restriction A, defined in Section 9.4). More generally, we show that any negation-limited circuit satisfying Restriction A must have monotone subcircuits which compute a collection of threshold functions. Once again, we are able to reduce lower bound results to the monotone case.

The remainder of this chapter is organized as follows. Section 9.2 gives the main technical lemma on functions computed at the NEGATION gates of negation-limited circuits computing symmetric functions. In Section 9.3, we obtain the lower bounds for various symmetric functions. Section 9.4 describes the lower bounds for a restricted type of negation-limited circuits. We conclude in Section 9.5 with a brief discussion of open problems.

9.2 A technical lemma on functions computed at NEGATION gates

In this section, we shall prove a main technical lemma on functions computed at NEGATION gates in negation-limited circuits computing symmetric functions. This lemma is analogous to a result of Tanaka and Nishino [91] on negation-limited inverters (an inverter is an n-input n-output circuit which outputs the negations of its inputs).

Let f be an n-input symmetric boolean function. Suppose that $d(f) = m$, where $m = 2^r - 1$ for some integer r. For $x \in \{0,1\}^n$, let $d_f(x)$ be the maximum decrease of f on a chain $a^1, \ldots, a^k$ with $a^k = x$ (note that this number depends only on the number of ones in x). We show that the NEGATION gates in any r-circuit for f must compute the negations of the bits of the binary representation of d_f:

Lemma 9.2 *Let f be an n-input symmetric function with decrease $d(f) = m$, where $m = 2^r - 1$. Consider any r-circuit Δ which computes f. Label the NEGATION gates $N_1, \ldots, N_r$ in such a way that the input to N_i does not depend on the outputs of any of the negation gates $N_{i+1}, \ldots, N_r$ (such a labeling exists since the circuit is a directed acyclic graph). Let z_i be the function computed at the input of N_i. Then $z_1 z_2 \ldots z_r$ is the binary representation of d_f*

Proof. We proceed by induction on r. The Lemma is trivially true for $r = 0$ (i.e., the circuit is monotone). We now suppose that the Lemma holds for $r-1$ NEGATION gates.

Let $a \in \{0,1\}^n$. We wish first to show that $z_1(a)$ is the first bit of $d_f(a)$. We consider the case that $z_1(a) = 1$ (the proof for the case $z_1(a) = 0$ is analogous), and we wish to show that $d_f(a) \geq (m+1)/2$. Since the circuit computes the same function regardless of the order of the input bits, we may assume that a is of the form $1^k 0^{n-k}$. Let f_k denote the $n-k$ input symmetric function obtained from f by fixing the first k bits to 1. Since $z_1(a) = 1$, and z_1 is monotone, f_k is computed by the circuit Δ_k obtained by fixing the first k inputs of Δ to 1 and replacing N_1 by the constant 0. Note that Δ_k is an $(r-1)$-circuit. Therefore, by Markov's Theorem, $d(f_k) \leq 2^{r-1} - 1 = (m-1)/2$. However, by considering a maximal chain in $\{0,1\}^n$ containing a, we see that $d(f) = d_f(a) + d(f_k)$. Therefore

$$d_f(a) = d(f) - d(f_k) \geq m - (m-1)/2 = (m+1)/2,$$

as desired.

Now it remains to show that the functions $z_2, \ldots, z_r$ are as specified. Again, we restrict our attention to the case $z_1 = 1$, since the case $z_1 = 0$ is handled similarly. Let $a \in \{0,1\}^n$ such that $z_1(a) = 1$. As above, we assume that a is a string of the form $1^\ell 0^{n-\ell}$. Let k be the least integer such that $z_1(1^k 0^{n-k}) = 1$, so $d_f(1^k 0^{n-k}) = 2^{r-1}$. We then apply the $r-1$ case of the Lemma to the function f_k and the circuit Δ_k, and note that for all $x \in \{0,1\}^{n-k}$, the binary representations of $d_{f_k}(x)$ and $d_f(1^k x)$ agree in their $(r-1)$ least significant bits. ∎

We now apply the main Lemma to various families of symmetric boolean functions. Let PARITY_n denote the parity function of n bit strings, i.e., PARITY_n returns one iff the number of ones in the input is odd. Note that $d(\text{PARITY}_n) = \lfloor n/2 \rfloor$, and $d_{\text{PARITY}_n}(x) = \lfloor k/2 \rfloor$ if there are exactly k ones in x. We consider the case that $\lfloor n/2 \rfloor = 2^r - 1$ for some integer r.

Corollary 9.3 *Let n be such that $\lfloor n/2 \rfloor = 2^r - 1$ for some integer r. Let Δ be an r-circuit computing* PARITY_n*. Label the NEGATION gates $N_1, \ldots, N_r$ in such a way that the input to N_i does not depend on the outputs of any of the negation gates $N_{i+1}, \ldots, N_r$ (such a labeling exists since the circuit is a directed acyclic graph). Let z_i be the function computed at the input of N_i. Then $z_1 z_2 \ldots z_r$ is the binary representation of $\lfloor k/2 \rfloor$, where k is the number of ones in the input.*

Similar corollaries can be obtained for the negation of PARITY, and for MOD functions.

9.3 The negation-limited circuit complexity of the parity function

We begin with a statement of the known upper bounds for the size and depth of negation-limited parity circuits [29, 78, 91].

Proposition 9.4 *Let $n \leq 2^{r+1} - 1$. Then*

1. $C^r(\text{PARITY}_n) = O(n \log n)$.

2. $D^r(\text{PARITY}_n) = O(\log n)$.

Proof. By using the Ajtai–Komlós–Szemerédi sorting network [2], it suffices to give an r-circuit of size $O(n \log n)$ and depth $O(\log n)$ for the parity of n sorted bits. Fischer [29] gives an $r+1$-circuit which, for sorted inputs, computes the negations of the bits in the binary representation of the number of ones in the input. The input to the top NEGATION gate in this circuit is PARITY, so by removing the top NEGATION, we obtain an r-circuit for PARITY. Fischer's circuit has linear size and logarithmic depth, so the total size of our circuit is $O(n \log n)$ and the total depth is $O(\log n)$. ∎

We now focus our attention on lower bounds. We consider the case that $n = 2^{r+1} - 1$. Let Δ be an r-circuit computing the parity of n input bits.

In the sequel, we assume that $z_1, \ldots, z_r$ are the functions and $N_1, \ldots, N_r$ are the NEGATION gates satisfying the conditions in Corollary 9.3.

Lemma 9.5 *For any i, $1 \leq i \leq r-1$, in any path from N_i to N_{i+1}, there is at least one AND gate and one OR gate.*

Proof. The proof is by contradiction. By Corollary 9.3, all four possible values for z_i and z_{i+1} are attained for some input $x \in \{0,1\}^n$. Note that the NEGATION gates are labeled in such a way that no path from N_i to N_{i+1} contains no other NEGATION gate. If the nodes along a path from N_i to N_{i+1} consisted entirely of AND (respectively OR) gates, then the possibility $z_i = 1, z_{i+1} = 1$ (respectively $z_i = 0, z_{i+1} = 0$) would be ruled out. Therefore, all such paths must include both an AND gate and an OR gate. ∎

Let L be the output gate of Δ. Since the output of N_r is necessary to compute PARITY_n, there must be a path from N_r to L.

Lemma 9.6 *In any path from N_r to L, there is at least one AND gate and one OR gate.*

Proof. As in the proof of the previous Lemma, we note that by Corollary 9.3, each of the four possibilities for the values of z_r, PARITY are attained for some input $x \in \{0,1\}^n$. Also, no path from N_r to L contains no other NEGATION gate. Therefore, as in the previous Lemma, we see that any path from N_r to L must include at least one AND gate and at least one OR gate (all other possibilities lead to contradictions). ∎

Theorem 9.7 *Let $n = 2^{r+1} - 1$, then*

$$C^r(\text{PARITY}_n) \geq 4n + 3\log(n+1) - c.$$

Proof. By Corollary 9.3, the subcircuit which computes z_1 is a monotone circuit which computes the majority function of the n input bits. It is known [63] that such a circuit must have size $\geq 4n - c$. By Lemma 9.5 and 9.6, the length of any path from N_1 to L is at least $3r$. No gate on this path is used to compute z_1. Hence, we have at least $4n + 3r - c$ gates as desired. ∎

We now turn our attention to arbitrary symmetric functions, obtaining lower bounds for the negation-limited complexity in terms of the decrease. By T_n^ℓ we denote the ℓth threshold function of n bits, which returns one iff at least ℓ of the input bits are one.

Theorem 9.8 *Let f be an n-input symmetric function with decrease m, where $m = 2^r - 1$. Let ℓ be such that for $x \in \{0,1\}^n$, $d_f(x) \geq 2^{r-1}$ iff x has at least ℓ ones. Then $C^r(f) \geq C^0(T_n^\ell) + 3\log(m+1) - c$.*

Proof. Identical to that of Theorem 9.7. ∎

We apply this to the MOD_n^k function, which outputs one iff the number of ones in the input (of n bits) is a multiple of k. Note that $\neg\mathrm{PARITY}_n = \mathrm{MOD}_n^2$, and that the decrease of MOD_n^k is $1 + \lfloor (n-1)/k \rfloor$. We consider those n such that $1 + \lfloor (n-1)/k \rfloor = 2^r - 1$ for some integer r. For such n, the parameter ℓ specified in Theorem 9.8 is given by

$$\ell = k(2^{r-1} - 1) + 1.$$

An elementary calculation shows that

$$(n + 1 - k)/2 \leq \ell \leq (n+1)/2.$$

Therefore, $C^0(T_n^\ell)$ is at least as big as the monotone complexity of the majority function of $n - i$ bits, for some $0 \leq i \leq k$. We obtain:

Corollary 9.9 *Let n and k be such that $\lfloor (n-1)/k \rfloor + 1 = 2^r - 1$ for some integer r. Then $C^r(\mathrm{MOD}_n^k) \geq 4(n-k) + 3r - c$.*

We now consider circuit depth.

Theorem 9.10 *Let $n = 2^{r+1} - 1$, then*

$$D^r(\mathrm{PARITY}_n) \geq 4\log(n+1) - c.$$

Proof. Recall that the subcircuit computing z_1 is a monotone majority circuit, which must have size at least $C^0(T_n^{(n+1)/2}) \geq 4n - c$. Notice that there must be a path from an arbitrary gate in this subcircuit to N_1, so depth of N_1 is at least $\log n$. Recall that the length of any path from N_1 to L is at least $3r$. Thus, we have total depth of at least $4\log(n+1) - c$. ∎

In the case of general symmetric functions, the same techniques yield:

Theorem 9.11 *Let f be an n-input symmetric function with decrease m, where $m = 2^r - 1$. Let ℓ be such that for $x \in \{0,1\}^n$, $d_f(x) \geq 2^{r-1}$ iff x has at least ℓ ones. Then $D^r(f) \geq \log n + 3\log(m+1) - c$.*

Corollary 9.12 *Let n and k be such that $\lfloor (n-1)/k \rfloor + 1 = 2^r - 1$ for some integer r. Then $D^r(\mathrm{MOD}_n^k) \geq \log(n-k) + 3r - c$.*

9.4 Lower bounds on a restricted type of negation-limited parity circuits

We give a setting in which the size of parity circuits has a superlinear lower bound. For this purpose, we introduce the following restriction.

Restriction A: If there is a path from one of $N_1, \ldots, N_r$ to a gate G in Δ, then G computes some symmetric function.

(Here we view the circuit as a directed graph, where the inputs to a gate correspond to in-edges and the outputs from a gate are out-edges.) For the PARITY function, we obtain the following:

Theorem 9.13 *Let $n = 2^{r+1} - 1$. Let Δ be an r-circuit for* PARITY$_n$ *which satisfies Restriction A. Then the size of Δ is $\Omega(n \log n)$.*

This is an immediate consequence of Theorem 9.14 below, together with known lower bounds on the monotone complexity of sorting n bits (cf. [29]). Note that the circuit in Proposition 9.4 satisfies Restriction A, so Theorem 9.13 is within a constant factor of optimal.

In general, for symmetric boolean functions we obtain:

Theorem 9.14 *Let f be an n-input symmetric function with decrease m, where $m = 2^r - 1$. Let Δ be an r-circuit for f satisfying Restriction A. Let $a^0, \ldots, a^n$ be a maximal chain in $\{0,1\}^n$ (so a^i is a binary string with i ones and $n-i$ zeros). Then the size of Δ is at least*

$$C^0(T_n^{q_1}, \ldots, T_n^{q_k}),$$

where $\{q_1, \ldots, q_k\} = \{q | 1 \leq q \leq n, f(a^{q-1}) \neq f(a^q)\}$.

Proof. Let $z_1, \ldots, z_r$ be the functions and $N_1, \ldots, N_r$ the NEGATION gates satisfying the conditions in Lemma 9.2. Let z_{r+1} denote the function f. By Lemma 9.2, for $x \in \{0,1\}^n$ the string $z_1 z_2 \ldots z_{r+1}$ is the binary representation of $2d_f(x) + f(x)$, which we shall denote by $h(x)$. This is an increasing function which attains all possible values in the range $1, \ldots, 2m$ (the values 0 and $2m+1$ may or may not be reached).

Suppose that $f(a^{q-1}) \neq f(a^q)$. We wish to show that for all such q, there is a monotone subcircuit of Δ computing T_n^q. Note that we must have $h(a^{q-1}) + 1 = h(a^q)$. Let w be such that $w01^{r-i}$ is the binary representation of $h(a^{q-1})$, where w is a binary string $w_1 w_2 \ldots w_i$ of length i. Then the binary representation of $h(a^q)$ is $w10^{r-i}$.

Let Δ' be a circuit obtained from Δ by replacing N_j with the constant $\neg w_j$ for each $1 \leq j \leq i$. For a gate G in Δ, let G' denote the corresponding gate in Δ'. Let g and g' denote the functions computed at gates G and G', respectively. Let $\Omega \subseteq \{0,1\}^n$ denote the set of all binary strings with exactly $q-1$ or q ones. We have:

1. For any gate G in Δ, and for any $x \in \Omega$, $g(x) = g'(x)$.

2. If for all $j > i$, there is no path from N_j to G in Δ, then g' is computed by a monotone subcircuit of Δ'.

3. If g is computed by a monotone subcircuit of Δ, then $g = g'$.

Let G be the gate computing z_{i+1}. Note that by (2), g' is monotone, and by (1), $g'(x) = g(x) = T_n^q(x)$ for $x \in \Omega$. Since T_n^q is the unique monotone function which agrees with T_n^q on Ω, we have $g'(x) = T_n^q(x)$ for all $x \in \{0,1\}^n$.

Recall that we are trying to prove that there is a monotone subcircuit of Δ computing T_n^q (from which the Theorem follows). So far we have shown the existence of a gate G in Δ such that $g' = T_n^q$. It now suffices to show that for any such gate, either g is computed by a monotone subcircuit of Δ (so, by (3), $g = g'$ and we are done) or there is a gate lower down in the circuit with the same property.

Suppose that g is not computed by a monotone subcircuit of Δ. Then there is a path from some negation gate N_j to G in Δ. This is illustrated in Figure 9.1.

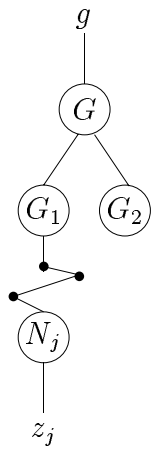

Figure 9.1: A gate G such that G' computes T_n^q. One of G_1, G_2 shares this property.

By Restriction A, we know that g is a symmetric function. Let G_1 and G_2 be the gates whose outputs are the inputs to G, labeled such that a path from N_j to G passes through G_1. Therefore g_1 computes a symmetric function. Since g_1' is a monotone function which agrees with g_1 on Ω, either $g_1' = T_n^q$ or g_1' is constant on Ω. In the latter case, since g' is not constant on Ω, we have that either g_1' is identically 1 on Ω and G is an AND gate, or g_1' is identically 0 on Ω and G is an OR gate. So g' and g_2' agree on Ω, and $g_2' = T_n^q$. In either case we have found a gate G_ℓ below G in Δ with $g_\ell' = T_n^q$ as desired. ∎

9.5 Concluding remarks and open problems

There remains a factor of $\log n$ between the known upper and lower bounds. Also, our lower bound techniques do not seem to allow any slack in the number of negations. For instance, it it not clear if it is possible to extend Theorem 9.7 to the case that $r + 1$ negation gates are allowed. On the other hand, it is not clear how to compute PARITY_n with a circuit of size $o(n \log n)$ that uses $O(\log n)$ negations. A straightforward divide-and-conquer approach combining Proposition 9.4 with linear size unrestricted PARITY circuits yields the following:

Proposition 9.15 *Let a be a function of n such that $a = \omega(1)$ and $a = n^{o(1)}$. Then PARITY_n has circuits of size $O(n \log a)$ using $O(n \log a/a)$ NEGATION gates.*

We ask whether a better tradeoff between circuit size and number of negations is possible for PARITY. In particular, does restricting the number of negations to $O(\log n)$ in a PARITY circuit necessitate superlinear size?

In this chapter we have further developed the lower bound techniques introduced by Tanaka and Nishino [91]. These methods seem very powerful, as they demonstrate a link between monotone complexity (where some good lower bound results are known) and negation-limited complexity. We hope that these methods will someday lead to a better understanding of general circuit complexity.

Chapter 10

Relationships between the number of negations available and circuit sizes

In this chapter, we consider the relationship between the number of NEGATION gates available and the size of circuits.

In particular, we show that there is an optimal circuit computing some single-output function with $\log_2(n+1)$ NEGATION gates, from which the removal of *two* NEGATION gate *must* cause an exponential growth. We also show that there is an optimal circuit computing some two-output function with $\log_2(n+1)$ NEGATION gates, from which the removal of *one* NEGATION gate *must* cause an exponential growth. Both partially answer an open problem presented by Fischer [30].

In addition, we consider the relationship between the size of circuits computing clique functions and the number of NEGATION gates in the circuits, and show that we can eliminate a constant number of NEGATION gates from circuits computing clique functions without exponentially increasing the size of the circuit.

10.1 Introduction

The result of Fischer [29] shows that there is only a small gap between the size of a combinational circuit and that of a circuit which includes only $b(n)$ NEGATION gates for *any* boolean function, where $b(n) = \lceil \log(n+1) \rceil$, the number of bits in the binary representation of n (all logarithms in this chapter are base two).

Proposition 10.1 *For any boolean function f,*

$$C^{b(n)}(f) \leq 2C(f) + O(n^2 \log n^2).$$

From Proposition 10.1, we can say nothing general about the size of the circuit, when we delete even one NEGATION gate from the circuit with $b(n)$ NEGATION gates.

Tardos [100] pointed out that there is a polynomial time computable function F_n whose monotone circuit complexity is exponential. This function is so-called Lovász ϑ function introduced by Lovász [64] to study the Shannon-capacity of graphs, and is known to be polynomial time computable. Tardos showed the following proposition.

Proposition 10.2

$$C^0(F_n) = 2^{\Omega(n^{1/6-o(1)})}.$$

From the proposition above, the following proposition is shown,

Proposition 10.3 *Let $n+1$ be a power of two, then there exists an integer t $(0 \leq t \leq \log(n+1)-1)$ such that*

$$\frac{C^t(F_n)}{C^{t+1}(F_n)} = \exp(\Omega(n^{1/6-o(1)})).$$

Proof. By observing that there is an exponential gap between the sizes of a monotone circuit and a circuit with $\log(n+1)$ NEGATION gates computing F_n while the difference of the number of NEGATION gates is only $\log(n+1)$. ∎

Notice that in Proposition 10.3, we cannot determine the value of the integer t. Actually, t can range over $\log(n+1)$ different values. In other words, we have the possibility to increase exponentially the size of a circuit computing F_n, when we delete only one NEGATION from the circuit with $b(n)$ NEGATION gates.

In this chapter, first, we construct a single-output boolean function H_n, where t can range over only two different values.

Theorem 10.4 *Let $n+1$ be a power of two, then there exists an integer t $(\log(n+1)-2 \leq t \leq \log(n+1)-1)$ such that*

$$\frac{C^t(H_n)}{C^{t+1}(H_n)} = \exp(\Omega(n^{1/6-o(1)})).$$

We also construct another function K_n, which has two outputs, where t can be uniquely determined.

Theorem 10.5 *Let $n+1$ be a power of two, then*

$$\frac{C^{\log(n+1)-1}(K_n)}{C^{\log(n+1)}(K_n)} = \exp(\Omega(n^{1/6-o(1)})).$$

From Theorem 10.5, we know that there is an optimal circuit with $\log(n+1)$ NEGATION gates, from which the removal of one NEGATION gate *must* cause an exponential growth.

The *k-clique* function $\mathrm{CLIQUE}_{k,n}$ has $n(n-1)/2$ variables each for them indicates an edge in graph of n vertices, and returns one iff a given graph contains a clique (i.e. complete subgraph) on some vertices. Notice that clique functions are monotone. We next show the following theorem for clique functions.

Theorem 10.6 *Let n be the number of vertices in graph, and let $N = n(n-1)/2$. Then, for any k $(k \geq 0)$ and sufficiently large n, the $\lceil n\sqrt{1-\frac{1}{2^{k+1}}}\rceil$-clique function f has the following property:*

$$C^{b(N)-k}(f) \leq \frac{N+1}{2^{k-1}}C(f) + O(N^2),$$

Theorem 10.6 says we can eliminate a constant number of NEGATION gates from circuits computing clique functions without exponentially increasing the size of

10.2 An exponential growth with the removal of two negations

Let $f = (f_1, \ldots, f_m)$ be a system of n-input boolean functions. A *chain* C in the boolean lattice $\{0,1\}^n$ is an increasing sequence $a^1 < \ldots < a^k \in \{0,1\}^n$. The *decrease* of f on C is the number of $i \leq k$ such that for some j, $f_j(a^{i-1}) > f_j(a^i)$. We define $d(f)$ to be the maximum decrease of f on any chain C. Notice that $d(f) \leq n$ (this is attained for some systems f). Markov [65] has shown that $b(d(F))$ NEGATION gates are necessary and sufficient to compute any system of boolean functions, where $b(x) = \lceil \log(x+1) \rceil$, the number of bits in the binary representation of x. Thus, $C^r(f)$ is always defined for $r \geq b(n)$.

Let f be a system of n-input symmetric boolean functions. Suppose that $d(f) = m$, where $m = 2^r - 1$ for some integer r. For $x \in \{0,1\}^n$, let $d_f(x)$ be the maximum decrease of f on a chain $a^1, \ldots, a^k$ with $a^k = x$ (note that this number depends only on the number of ones in x). Tanaka, Nishino, and Beals [93, 11] show that the NEGATION gates in any r-circuit for f must compute the negations of the bits of the binary representation of d_f:

Proposition 10.7 *Let f be a system of n-input symmetric functions with decrease $d(f) = m$, where $m = 2^r - 1$. Consider any r-circuit Δ which computes f. Label the NEGATION gates $N_1, \ldots, N_r$ in such a way that the input to N_i does not depend on the outputs of any of the negation gates $N_{i+1}, \ldots, N_r$ (such a labeling exists since the circuit is a directed acyclic graph). Let z_i be the function computed at the input of N_i. Then $z_1 z_2 \ldots z_r$ is the binary representation of d_f*

We are now ready to present the proof of Theorem 10.4.

Recall that F_n is a function in Proposition 10.2. We define an n-input function H_n as follows:

$$H_n(w_1, \ldots, w_{m-1}, x_1, \ldots, x_{m+2}) = w_1 \oplus \cdots \oplus w_{m-1} \oplus F_{m+2}(x_1, \ldots, x_{m+2}),$$

where $n+1$ is a power of two and $m = (n-1)/2$.

Since F_{m+2} is monotone, the maximum decrease $d(H_n)$ is $(m-1)/2 = (n-3)/4$. So, $\log(n+1) - 2$ NEGATION gates are necessary and sufficient to compute H_n by the theorem of Markov [65]. From [11], the following lemma is easily shown.

Lemma 10.8

$$C^{\log(n+1)}(H_n) \leq 2C(H_n) + O(n \log n).$$

Since F_{m+2} is known to be polynomial time computable, $C(H_n)$ is bounded by some polynomial in n, so is $C^{\log(n+1)}(H_n)$.

Proposition 10.7 can be extended by replacing each input x_i with an arbitrary monotone function $f_i(X_i)$, where $X_1, \ldots, X_n$ are disjoint sets of the input variables, i.e., $X_i \cap X_j = \emptyset$ for each i, j $(1 \leq i, j \leq n, i \neq j)$, and f_i is a *non-constant* monotone function over X_i for each i $(1 \leq i \leq n)$. This can be done, since the values of $f_1(X_1), \ldots, f_n(X_n)$ can be changed independently and we can regard $f_1(X_1), \ldots, f_n(X_n)$ as n independent variables. If we apply the argument above to $(\log(n+1) - 2)$-circuit computing H_n, we can obtain the following lemma.

Lemma 10.9 *Let $r = \log(n+1)-2$. Consider any r-circuit Δ which computes H_n. Label the NEGATION gates $N_1, \ldots, N_r$ in such a way that the input to N_i does not depend on the outputs of any of the negation gates $N_{i+1}, \ldots, N_r$ (such a labeling exists since the circuit is a directed acyclic graph). Let z_i be the function computed at the input of N_i. Then $z_1 z_2 \ldots z_r$ is the binary representation of $\lfloor k/2 \rfloor$, where k is the number of ones in $\{w_1, \ldots, w_{m-1}, F_{m+2}\}$.*

Proof. Since $\{w_1, \ldots, w_{m-1}\} \cap \{x_1, \ldots, x_{m+2}\} = \emptyset$, the values of F_{m+2} can be changed independently of $w_1, \ldots, w_{m-1}$. F_{m+2} is a *non-constant* monotone function over $\{x_1, \ldots, x_{m+2}\}$. Thus, we can regard the value of $F_{m+2}(x_1, \ldots, x_{m+2})$ as a new variable w_m which is independent of $w_1, \ldots, w_{m-1}$. Δ includes only $r = \log(n+1) - 2$ NEGATION gates, which are necessary and sufficient to compute the m-input parity function This completes the proof of Lemma 10.9. ∎

Lemma 10.10

$$C^{\log(n+1)-2}(H_n) = \exp(\Omega(n^{1/6-o(1)})).$$

Proof. Consider the sub-circuit computing z_1 in a $(\log(n+1))$-circuit computing H_n. This sub-circuit is monotone and computes the majority function over $\{w_1, \ldots, w_{m-1}, F_{m+2}\}$ from Lemma 10.9. By fixing $(m-1)/2$ variables in $\{w_1, \ldots, w_{m-1}\}$ to ones and fixing the other variables in $\{w_1, \ldots, w_{m-1}\}$ to zeros, we can get the function $F_{m+2}(x_1, \ldots, x_{m+2})$ at the output of the sub-circuit. Combining this with Proposition 10.2, we have

$$\begin{aligned} C^{\log(n+1)-2}(H_n) &\geq C^0(F_{m+2}(x_1, \ldots, x_{m+2})) \\ &= 2^{\Omega(m^{1/6-o(1)})} \\ &= 2^{\Omega(n^{1/6-o(1)})}. \end{aligned}$$

∎

From Lemma 10.8 and 10.10, we obtain Theorem 10.4.

10.3 An exponential growth with the removal of one negation

We present the proof of Theorem 10.5.

Recall that F_n is a function in Proposition 10.2, and that H_n is an n-input function defined as follows:

$$H_n(w_1, \ldots, w_{m-1}, x_1, \ldots, x_{m+2}) = w_1 \oplus \cdots \oplus w_{m-1} \oplus F_{m+2}(x_1, \ldots, x_{m+2}),$$

where $n+1$ is a power of two and $m = (n-1)/2$. We now define an n-input two-output function K_n as $K_n = (H_n, \overline{H_n})$.

Since F_{m+2} is monotone, the maximum decrease $d(K_n)$ is $m = (n-1)/2$. So, $\log(n+1) - 1$ NEGATION gates are necessary and sufficient to compute K_n by the theorem of Markov [65]. From [11], the following lemma is easily shown.

Lemma 10.11

$$C^{\log(n+1)}(K_n) \leq 2C(K_n) + O(n \log n).$$

Since F_{m+2} is known to be polynomial time computable, $C(K_n)$ is bounded by some polynomial in n, so is $C^{\log(n+1)}(K_n)$. If we apply the extending argument for Proposition 10.7 to $(\log(n+1)-1)$-circuit computing K_n, we can obtain the following lemma.

Lemma 10.12 *Let $r = \log(n+1)-1$. Consider any r-circuit Δ which computes K_n. Label the NEGATION gates $N_1, \ldots, N_r$ in such a way that the input to N_i does not depend on the outputs of any of the negation gates $N_{i+1}, \ldots, N_r$ (such a labeling exists since the circuit is a directed acyclic graph). Let z_i be the function computed at the input of N_i. Then $z_1 z_2 \ldots z_r$ is the binary representation of the number of ones in $\{w_1, \ldots, w_{m-1}, F_{m+2}\}$.*

Proof. Since $\{w_1, \ldots, w_{m-1}\} \cap \{x_1, \ldots, x_{m+2}\} = \emptyset$, the values of F_{m+2} can be changed independently of $w_1, \ldots, w_{m-1}$. F_{m+2} is a *non-constant* monotone function over $\{x_1, \ldots, x_{m+2}\}$. Thus, we can regard the value of $F_{m+2}(x_1, \ldots, x_{m+2})$ as a new variable w_m which is independent of $w_1, \ldots, w_{m-1}$. Δ includes only $r = \log(n+1)-1$ NEGATION gates, which are necessary and sufficient to compute the m-input parity function and its complement simultaneously. This completes the proof. ∎

Lemma 10.13

$$C^{\log(n+1)-1}(K_n) = \exp(\Omega(n^{1/6-o(1)})).$$

Proof. Consider the sub-circuit computing z_1 in a $(\log(n+1)-1)$-circuit computing K_n. This sub-circuit is monotone and computes the majority function over $\{w_1, \ldots, w_{m-1}, F_{m+2}\}$ from Lemma 10.12. By fixing $(m-1)/2$ variables in $\{w_1, \ldots, w_{m-1}\}$ to ones and fixing the other variables in $\{w_1, \ldots, w_{m-1}\}$ to zeros, we can get the function $F_{m+2}(x_1, \ldots, x_{m+2})$ at the output of the sub-circuit. Combining this with Proposition 10.2, we have

$$\begin{aligned} C^{\log(n+1)-1}(K_n) &\geq C^0(F_{m+2}(x_1, \ldots, x_{m+2})) \\ &= 2^{\Omega(m^{1/6-o(1)})} \\ &= 2^{\Omega(n^{1/6-o(1)})}. \end{aligned}$$

∎

From Lemma 10.11 and 10.13, we obtain Theorem 10.5.

10.4 The complexity of circuits computing clique functions with a limited number of negations

We define the kth *slice* function f_k of f by

$$f_k(x_1, \ldots, x_n) = \begin{cases} 0 & \text{if } \sum_{i=1}^n x_i < k \\ f(x_1, \ldots, x_n) & \text{if } \sum_{i=1}^n x_i = k \\ 1 & \text{if } \sum_{i=1}^n x_i > k. \end{cases}$$

We denote by $T_k^n(x_1, \ldots, x_n)$, the kth *threshold* function which returns one iff $\sum_{i=1}^n x_i \geq k$.

Now, we present the proof of Theorem 10.6.

Proof. We construct a circuit C computing f. In order to simplify the presentation, without loss of generality, we assume that $N = 2^l - 1$. It is known that the number of variables which can be inverted using $b(N) - k = l - k$ NEGATION gates is $2^{l-k} - 1$. Let $c = 2^{l-k} - 1$. Our circuit C consists of a monotone sorting network on N variables, the circuit M_c (due to Fischer [29]), and monotone circuits which compute the $c+1$ slice functions $f_{N-c}, \ldots, f_N$ of f (see Figure 10.1).

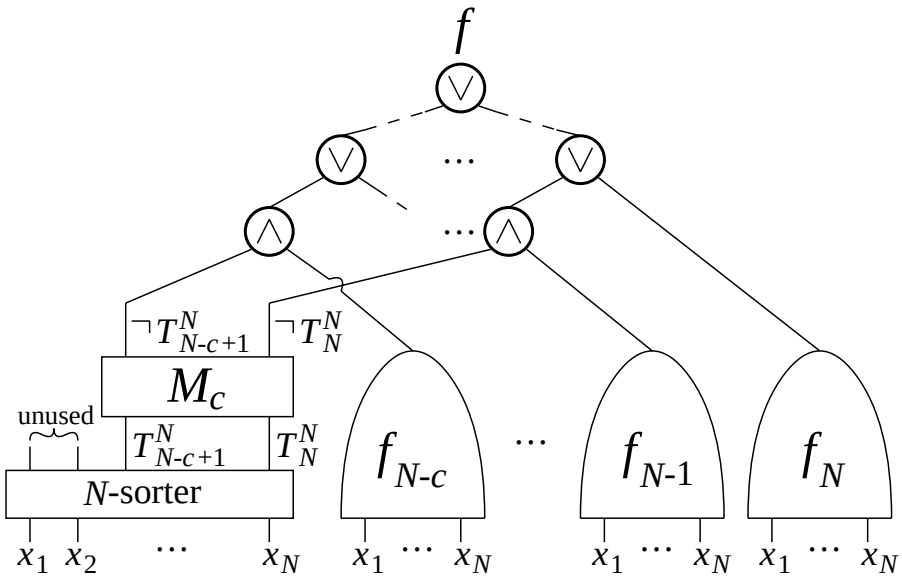

Figure 10.1: A network C for f.

The wires in C are connected as follows. Let $x_1, \ldots, x_N$ be the inputs and $y_1, \ldots, y_N$ the outputs of C, and let $x_1^S, \ldots, x_N^S$ be the inputs and $y_1^S, \ldots, y_N^S$ outputs of the sorting network. First, we identify the inputs of the sorting network with the inputs of C as follows:

$$x_i^S = x_i \quad \text{for } 1 \leq i \leq N.$$

Let $x_1^M, \ldots, x_c^M$ be the inputs and $y_1^M, \ldots, y_c^M$ the outputs of M_c. Next, we identify each input of M_c with an output of the sorting network as follows:

$$x_i^M = y_{N-c+i}^S \quad \text{for } 1 \leq i \leq c.$$

Then, by the construction of M_c, it follows that

$$y_i^M = \neg T_{N-c+i}^N(x_1, \ldots x_N) \quad \text{for } 1 \leq i \leq c.$$

Notice that M_c has only $l - k = \log(n+1) - k$ NEGATION gates.

Incidentally, in order to form an $\lceil n\sqrt{1 - \frac{1}{2^{k+1}}} \rceil$-clique, at least

$$\binom{\lceil n\sqrt{1 - \frac{1}{2^{k+1}}} \rceil}{2} = \left(\frac{1}{2} - \frac{1}{2^{k+2}}\right) n^2 - O(n)$$

edges are necessary. On the other hand, we have

$$\begin{aligned} N - c &= (2^l - 1) - (2^{l-k} - 1) \\ &= (N+1)(1 - \frac{1}{2^k}) \\ &= \left(\frac{1}{2} - \frac{1}{2^{k+1}}\right) n^2 - O(n). \end{aligned}$$

Thus, for sufficiently large n, we have

$$\left(\begin{array}{c} \lceil n\sqrt{1 - \frac{1}{2^{k+1}}} \rceil \\ 2 \end{array} \right) > N - c.$$

Therefore, $\lceil n\sqrt{1 - \frac{1}{2^{k+1}}} \rceil$-clique has more than $N - c$ edges for sufficiently large n. This means $f(x_1, \ldots, x_N) = 0$ for any inputs $\{x_1, \ldots, x_N\}$ such that $\sum_{i=1}^{n} x_i < N - c$. Thus, we can compute f by using the following relation:

$$f = \bigvee_{i=N-c}^{N} [\neg T_{i+1}^N \wedge f_i].$$

Note that $\neg T_{N+1}^N$ always returns one (i.e., is a constant function) (see Figure 10.1).

Finally, let us estimate the size of C. Since it is known that $C^{b(n)}(M_n) = O(n)$ [29], we have $C^{b(n)-k}(M_c) = O(N)$. The size of the sorting network is $O(n \log n)$ [2], and as shown in [106], the monotone complexity of the circuits simultaneously computing slice functions $f_{N-c}, \ldots, f_N$ can be bounded as follows

$$\begin{aligned} C^0(f_{N-c}, \ldots, f_N) &\leq 2(c+1)C(f) + O(N^2) \\ &\leq \frac{N+1}{2^{k-1}} C(f) + (N^2). \end{aligned}$$

This completes the proof of Theorem 10.6. ∎

10.5 Lower bounds for clique functions

The clique function appearing in Theorem 10.6 is known to be NP-complete [31]. In this section, we show exponential lower bounds on the monotone circuit complexity of clique functions.

Before that, we review several definitions. $f(x_1, \ldots, x_n)$ is called a *projection* of $g(y_1, \ldots, y_m)$ if

$$f(x_1, \ldots, x_n) = g(\sigma(y_1), \ldots, \sigma(y_m))$$

for some mapping $\sigma : \{y_1, \ldots, y_m\} \rightarrow \{0, 1, x_1, \ldots, x_n, \overline{x_1}, \ldots, \overline{x_n}\}$. The projection is *monotone* if $\sigma(y_i)$ is not a negated variable for each $1 \leq i \leq m$. It is known that $C(f) \leq C(g)$ if f is a projection of g. $C^0(f) \leq C^0(g)$ if f is a monotone projection of g (see [107]).

The following proposition was shown by Alon and Boppana [5], which improved a superpolynomial lower bounds by Razborov [77].

Proposition 10.14 *For $k \leq n^{1/4}$, the monotone circuit complexity of the function* $\mathrm{CLIQUE}_{k,n}$ *is* $n^{\Omega(\sqrt{k})}$.

Now we show the following theorem.

Theorem 10.15 *For an arbitrary ε, $0 < \varepsilon < 1/2$, the monotone circuit complexity of the function* $\mathrm{CLIQUE}_{(1-\varepsilon)m,m}$ *is* $\Omega(m^{(\varepsilon m)^{1/8}})$.

Proof. Let G be a graph with n vertices. For an arbitrary ε, $0 < \varepsilon < 1/2$, we require a graph H with $\frac{n}{\varepsilon}$ vertices to satisfy the following condition:

$$G \text{ includes } n^{1/4}\text{-clique.} \iff H \text{ includes } \tfrac{1-\varepsilon}{\varepsilon}n\text{-clique (i.e. } (1-\varepsilon)|H|\text{-clique).}$$

Notice that, since $0 < \varepsilon < 1/2$, it follows that $\frac{1-\varepsilon}{\varepsilon}n > n^{1/4}$ for all $n \geq 1$. The construction of H is as follows: H includes $(\frac{1-\varepsilon}{\varepsilon}n - n^{1/4})$-clique H_1, the graph G, and a graph H_2 with $n^{1/4}$ independent vertices, where each vertex of H_1 is connected to all vertices in G (see Figure 10.2).

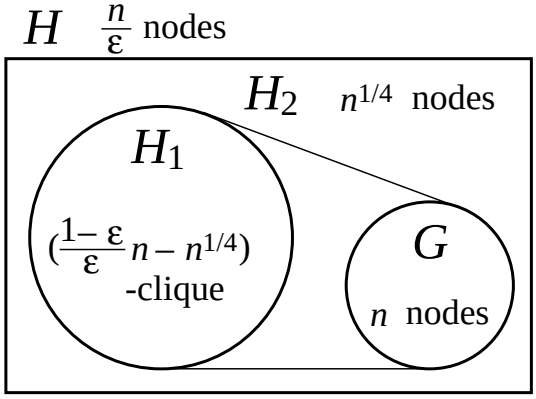

Figure 10.2: A projection of the clique function.

It is easy to see that H satisfies the condition above. We can construct a monotone projection from $\mathrm{CLIQUE}_{(1-\varepsilon)m,m}$ to $\mathrm{CLIQUE}_{n^{1/4},n}$, based on the construction of H. Since $m = \frac{n}{\varepsilon}$, it follows that the monotone complexity of $\mathrm{CLIQUE}_{n^{1/4},n}$) is $(\varepsilon m)^{\Omega(m^{1/8})}$ from Proposition 10.14. ∎

Chapter 11

Conclusion

In this thesis, we have studied computational difficulty. We have constructed several algorithms and circuits for giving upper bounds, and also established NP-hardness of scheduling problems and lower bounds on circuit complexity.

We have investigated single machine scheduling with new types of due dates and release dates. We have dealt with scheduling involving generalized due dates, generalized release dates, and fuzzy due dates.

Concerning single machine scheduling with generalized due dates, the complexity status for the following problems, asked by Young, Wong, Yiu, and Yan [111], remains open.

- $1|pmtn, r_j| \sum T_j^H$
- $1|p_j = 1| \sum w_j T_j^H$
- $1|pmtn, r_j| \sum U_j^H$
- $1|prec, , r_j| \sum U_j^H$
- $1|pmtn, r_j, p_j = 1| \sum U_j^H$
- $1|pmtn, r_j, p_j = 1| \sum w_j U_j^H$

There are also several open problems on multiple machine scheduling with generalized due dates, asked by Hall, Sethi, and Sriskandarajah [41] and Young, Wong, Yiu, and Yan [111].

It might be interesting to extend the generalized due date scheduling model. We have mentioned, in Chapter 2, generalized due dates can be extended to sequence-dependent due dates. We can consider yet other models. One is scheduling with due dates which depend on the starting/completion times of jobs rather than the positions of jobs in schedules. Another is scheduling with due dates which depend on only the preceding/succeeding jobs. This arises analogously from precedence constraints of jobs. We can consider similar generalization as above for generalized release date. Weights are also able to be generalized.

Fuzzy due dates which we have concerned with are determined by membership functions of completion times of jobs, which are nondecreasing. We can allow these functions to decrease. This arises analogously from just-in-time scheduling criteria.

We have also investigated negation-limited circuit complexity of boolean functions.

We have shown a $4n + \Omega(\log n)$ lower bound on the complexity of negation-limited circuits computing the parity function in Chapter 9. As far as we know, this is the largest lower bound on the size of circuits with AND, OR, and NEGATION gates, computing explicit functions with no restriction on circuit depth. However, G. Turán's question still remains open: is the size of any $c \log n$ depth inverter using $c \log n$ NEGATION gates superlinear?

We hope that both sides of investigation will lead us to understand the essence of computational difficulty of problems.

Bibliography

[1] Agnetis, A., Macchiaroli, R., Pacciarelli, D., and Rossi, F. Assigning jobs to time frames on a single machine to minimize total tardiness. *IIE Transactions* (1996). To appear.

[2] Ajtai, M., Komlós, J., and Szemerédi, E. An $O(n \log n)$ sorting network. In *Proceedings of the Fifteenth Annual ACM Symposium on Theory of Computing* (Boston, Massachusetts, 25–27 Apr. 1983), pp. 1–9.

[3] Ajtai, M., Komlós, J., and Szemerédi, E. Sorting in $O(c \log n)$ parallel steps. *Combinatorica 3*, 1 (1983), 1–19.

[4] Akers, Jr., S. B. On maximum inversion with minimum inverters. *IEEE Trans. Comput. c-17*, 2 (Feb. 1972), 134–135.

[5] Alon, N., and Boppana, R. B. The monotone circuit complexity of Boolean functions. *Combinatorica 7*, 1 (1987), 1–22.

[6] Amano, K., and Maruoka, A. Potential of the approximation method. In *37th Annual Symposium on Foundations of Computer Science* (Burlington, Vermont, 14–16 Oct. 1996), IEEE. To appear.

[7] Andreev, A. E. On a method for obtaining lower bounds for the complexity of individual monotone functions. *Soviet Math. Dokl. 31*, 3 (1985), 530–534.

[8] Baker, K. R. *Introduction to Sequencing and Scheduling.* Wiley, New York, 1974.

[9] Baker, K. R., Lawler, E. L., Lenstra, J. K., and Rinnooy Kan, A. H. G. Preemptive scheduling of a single machine to minimize maximum cost subject to release dates and precedence constraints. *Operations Research 31* (1983), 381–386.

[10] Batcher, K. E. Sorting networks and their applications. In *Proceedings of AFIPS Spring Joint Computer Conference, 32* (Montvale, N. J., 1968), AFIPS press, pp. 307–314.

[11] Beals, R., Nishino, T., and Tanaka, K. More on the complexity of negation-limited circuits. In *Proceedings of the Twenty-Seventh Annual ACM Symposium on Theory of Computing* (Las Vegas, Nevada, 29 May–1 June 1995), pp. 585–595.

[12] Beals, R., Nishino, T., and Tanaka, K. On the complexity of negation-limited boolean networks. *SIAM J. Comput.* (1996). To appear.

[13] Berkowitz, S. J. On some relationships between monotone and non-monotone circuit complexity. Tech. rep., Computer Science Department, University of Toronto, 1982.

[14] Blazewicz, J. Scheduling dependent tasks with different arrival times to meet deadlines. In *Modeling and Performance Evaluation of Computer Systems*, E. Gelenbe and H. Beilner, Eds. North-Holland, Amsterdam, 1976, pp. 57–65.

[15] Blum, N. A Boolean function requiring $3n$ network size. *Theoretical Comput. Sci. 28*, 3 (Feb. 1984), 337–345. Note.

[16] Boppana, R. B., and Sipser, M. The complexity of finite functions. In *Handbook of Theoretical Computer Science Volume A—Algorithms and Complexity*, J. van Leeuwen, Ed. Elsevier, MIT Press, 1990, ch. 14, pp. 757–804.

[17] Borodin, A. On relating time and space to size and depth. *SIAM J. Comput. 6*, 4 (Dec. 1977), 733–744.

[18] Browne, J., Dubois, D., Rathmill, K., Sethi, S. P., and Stecke, K. Classification of flexible manufacturing systems. *The FMS Magazine* (Apr. 1984), 114–117.

[19] Chazelle, B., and Edelsbrunner, H. An optimal algorithm for intersecting line segments in the plane. In *29th Annual Symposium on Foundations of Computer Science* (White Plains, New York, 24–26 Oct. 1988), IEEE, pp. 590–600.

[20] Conway, R. W., Maxwell, W. L., and Miller, L. W. *Theory of Scheduling.* Addison-Wesley, Massachusetts, 1967.

[21] Deutsch, D. Quantum theory, the church-turing principle and the universal quantum computer. In *Proc. R. Soc. Lond. A, 400* (1985), pp. 97–117.

[22] Du, J., and Leung, J. Y.-T. Minimizing total tardiness on one machine is NP-hard. *Mathematics of Operations Research 15* (1990), 483–495.

[23] Dunne, P. E. *Techniques for the analysis of monotone Boolean networks.* PhD thesis, University of Warwick, 1984. Theory of Computation Report No. 69.

[24] Dunne, P. E. The complexity of central slice functions. *Theoretical Comput. Sci. 44*, 3 (1986), 247–257.

[25] Dunne, P. E. *The Complexity of Boolean Networks.* Academic Press, London, 1988.

[26] Dunne, P. E. On monotone simulations of nonmonotone networks. *Theoretical Comput. Sci. 66*, 1 (2 Aug. 1989), 15–25.

[27] Dunne, P. E. Ceilings of monotone boolean functions. *Journal of Universal Computer Science 2*, 7 (July 1996), 533–548.

[28] Ferris, M. C., and Vlach, M. Scheduling with earliness and tardiness penalties. *Naval Research Logistics Quarterly 39* (1992), 229–245.

[29] Fischer, M. J. The complexity of negation-limited networks—a brief survey. In *Lecture Notes in Computer Science 33—Automata Theory and Formal Languages, 2nd GI Conference*, H. Brakhage, Ed. Springer-Verlag, 1974, pp. 71–82.

[30] Fischer, M. J. Lectures on network complexity. Tech. Rep. YALEU/DCS/TR-1104, Department of Computer Science, Yale University, June 1974. Revised April 1977, April 1996.

[31] Garey, M. R., and Johnson, D. S. *Computers and Intractability—A Guide to the Theory of NP-Completeness*. Freeman, New York, 1979.

[32] Garey, M. R., Johnson, D. S., and Sethi, R. The complexity of flowshop and jobshop scheduling. *Mathematics of Operations Research 1*, 2 (May 1976), 117–129.

[33] Garey, M. R., Tarjan, R. E., and Wilfong, G. T. One-processor scheduling with symmetric earliness and tardiness penalties. *Mathematics of Operations Research 13*, 2 (May 1988), 330–348.

[34] Gawiejnowicz, S. A note on scheduling on a single processor with speed dependent on a number of executed jobs. *Inf. Process. Lett. 57*, 6 (25 Mar. 1996), 297–300.

[35] Gawiejnowicz, S., and Pankowska, L. Scheduling jobs with varying processing times. *Inf. Process. Lett. 54*, 3 (12 May 1995), 175–178.

[36] Gilbert, E. N. Lattice theoretic properties of frontal switching functions. *J. Mathematics and Physics 33* (1954), 57–67.

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

[38] Gupta, S., and Sen, T. Minimizing the range of lateness on a single machine. *Journal of the Operational Research Society 35*, 9 (1984), 853–857.

[39] Haken, A. Counting bottlenecks to show monotone P $\neq$ NP. In *36th Annual Symposium on Foundations of Computer Science* (Milwaukee, Wisconsin, 23–25 Oct. 1995), IEEE, pp. 36–40.

[40] Hall, N. G. Scheduling problems with generalized due dates. *IIE Transactions 18* (1986), 220–222.

[41] Hall, N. G., Sethi, S. P., and Sriskandarajah, C. On the complexity of generalized due date scheduling problems. *European Journal of Operational Research 51* (1991), 100–109.

[42] Han, S. *Studies on Fuzzy Multiobjective Scheduling*. PhD thesis, Kobe University, Kobe, Japan, Mar. 1994.

[43] Han, S., Ishii, H., and Fujii, S. One machine scheduling problem with fuzzy duedates. *European Journal of Operational Research 79* (1994), 1–12.

[44] Ho, K. I.-J., Leung, J. Y.-T., and Wei, W.-D. Complexity of scheduling tasks with time-dependent execution times. *Inf. Process. Lett. 48* (1993), 315–320.

[45] Hoogeveen, J. A. Minimizing maximum earliness and maximum lateness on a single machine. Tech. Rep. BS-R9001, Centre for Mathematics and Computer Science, Amsterdam, 1990.

[46] Ibaraki, T., and Muroga, S. Synthesis of networks with a minimum number of negative gates. *IEEE Trans. Comput. c-20*, 1 (Jan. 1971), 49–58.

[47] Ishii, H., and Tada, M. Single machine scheduling problem with fuzzy precedence relation. *European Journal of Operational Research 87* (1995), 284–288.

[48] Ishii, H., Tada, M., and Masuda, T. Two scheduling problems with fuzzy due-dates. *Fuzzy Sets and Systems 46* (1992), 339–347.

[49] Jackson, J. R. Scheduling a production line to minimize maximum tardiness. Tech. Rep. 43, Management Science Research Project, University of California, Los Angeles, 1955.

[50] Karp, R. M. Reducibility among combinatorial problems. In *Complexity of Computer Computations*, R. E. Miller and J. W. Thatcher, Eds. Plenum Press, New York, 1972, pp. 85–103.

[51] Labetoulle, J., Lawler, E. L., Lenstra, J. K., and Rinnooy Kan, A. H. G. Preemptive scheduling of uniform machines subject to release dates. *Pulley-blank* (1984), 245–261.

[52] Lamagna, E. A., and Savage, J. E. Combinational complexity of some monotone functions. In *15th Annual Symposium on Switching and Automata Theory* (The University of New Orleans, 14–16 Oct. 1974), IEEE, pp. 140–144.

[53] Lawler, E. L. Optimal sequencing of a single machine subject to precedence constraints. *Management Science 19* (1973), 544–546.

[54] Lawler, E. L. A pseudopolynomial algorithm for sequencing jobs to minimize total tardiness. *Annals of Discrete Mathematics 1* (1977), 331–342.

[55] Lawler, E. L., Lenstra, J. K., Rinnooy Kan, A. H. G., and Shmoys, D. B. Sequencing and scheduling: Algorithms and complexity. In *Handbooks in Operations Research and Management Science, Volume 4—Logistics of Production and Inventory*, S. C. Graves, A. H. G. Rinnooy Kan, and P. H. Zipkin, Eds. North-Holland, Amsterdam, 1993, ch. 9, pp. 445–522.

[56] Lenstra, J. K., and Rinnooy Kan, A. H. G. Complexity of scheduling under precedence constraints. *Operations Research 26* (1978), 22–35.

[57] Lenstra, J. K., and Rinnooy Kan, A. H. G. Complexity results for scheduling chains on a single machine. *European Journal of Operational Research 4* (1980), 270–275.

[58] Lenstra, J. K., and Rinnooy Kan, A. H. G. Two open problems in precedence constrained scheduling. *Annals of Discrete Mathematics 23* (1984), 509–522.

[59] Lenstra, J. K., Rinnooy Kan, A. H. G., and Brucker, P. Complexity of machine scheduling problems. *Annals of Discrete Mathematics 1* (1977), 343–362.

[60] Leung, J. Y.-T., and Young, G. H. Minimizing total tardiness on a single machine with precedence constraints. Tech. Rep. 75083, Computer Science Program, University of Texas at Dallas, Richardson, Texas, 1989.

[61] Liao, C.-J., and Huang, R.-H. An algorithm for minimizing the range of lateness on a single machine. *Journal of the Operational Research Society 42*, 2 (1991), 183–186.

[62] Liu, C. L., and Layland, J. W. Scheduling algorithms for multiprogramming in a hard-real-time environment. *J. ACM 20*, 1 (1973), 46–61.

[63] Long, D. The monotone circuit complexity of threshold functions. University of Oxford, 1986.

[64] Lovász, L. On the Shannon capacity of a graph. *IEEE Transactions on Information Theory 25* (1996), 1–7.

[65] Markov, A. A. On the inversion complexity of a system of functions. *J. ACM 5*, 4 (Oct. 1958), 331–334.

[66] Maruoka, A. Circuit complexity and approximation method. *IEICE Transactions on Information and Systems E75-D* (1992).

[67] Megiddo, N. Combinatorial optimization with rational objective functions. *Mathematics of Operations Research 4*, 4 (Nov. 1979), 414–424.

[68] Mihara, T. *The Complexity of Quantum Computation.* PhD thesis, Japan Advanced Institute of Science and Technology, Ishikawa, Japan, Mar. 1997.

[69] Moore, J. M. An n job, one machine sequencing algorithm for minimizing the number of late jobs. *Management Science 15* (1968), 102–109.

[70] Muroga, S. *Threshold Logic and Its Applications.* Wiley, 1971.

[71] Nakamura, K., Tokura, N., and Kasami, T. Minimal negative gate networks. *IEEE Trans. Comput. c-21*, 1 (Jan. 1972), 5–11.

[72] Nishino, T., and Radhakrishnan, J. On the number of negations needed to compute parity functions. *IEICE Transactions on Information and Systems E78-D*, 1 (Jan. 1995), 90–91.

[73] Nishino, T., and Tanaka, K. On the negation-limited circuit complexity of clique functions. *IEICE Transactions on Information and Systems E78-D*, 1 (Jan. 1995), 86–89.

[74] Pippenger, N. Communication networks. In *Handbook of Theoretical Computer Science Volume A—Algorithms and Complexity*, J. van Leeuwen, Ed. Elsevier, MIT Press, 1990, ch. 15, pp. 805–833.

[75] Pippenger, N., and Fischer, M. J. Relations among complexity measures. *J. ACM 26*, 2 (Apr. 1979), 361–381.

[76] Raz, R., and Wigderson, A. Monotone circuits for matching require linear depth. *J. ACM 39*, 3 (July 1992), 736–744.

[77] Razborov, A. A. Lower bounds for the monotone complexity of some Boolean functions. *Soviet Math. Dokl. 31*, 2 (1985), 354–357.

[78] Santha, M., and Wilson, C. Limiting negations in constant depth circuits. *SIAM J. Comput. 22*, 2 (Apr. 1993), 294–302.

[79] Savage, J. E. Computational work and time on finite machines. *J. ACM 19*, 4 (Oct. 1972), 660–674.

[80] Savage, J. E. *The Complexity of Computing*. Wiley, 1976.

[81] Schnorr, C.-P. The network complexity and the Turing machine complexity of finite functions. *Acta Inf. 7* (1976), 95–107.

[82] Shannon, C. E. The synthesis of two-terminal switching circuits. *Bell System Technical Journal 28*, 1 (1949), 59–98.

[83] Smith, W. E. Various optimizers for single-stage production. *Naval Research Logistics Quarterly 3* (1956), 59–66.

[84] Sriskandarajah, C. A note on the generalized due dates scheduling problems. *Naval Research Logistics Quarterly 37* (1990), 587–597.

[85] Stecke, K. E., and Solberg, J. J. Loading and control policies for a flexible manufacturing system. *International Journal of Production Research 19* (1981), 481–490.

[86] Stockmeyer, L. J. Computational complexity. In *Handbooks in Operations Research and Management Science, Volume 3—Computing*, E. G. Coffman, Jr., J. K. Lenstra, and A. H. G. Rinnooy Kan, Eds. North-Holland, Amsterdam, 1992, ch. 9, pp. 455–517.

[87] Tada, M. *Studies on Fuzzy Combinatorial Optimization*. PhD thesis, Kobe University, Kobe, Japan, Mar. 1994.

[88] Tanaev, V. S., Gordon, V. S., and Shafranskij, Y. M. *Scheduling Theory: One Stage Systems*. Nauka, Moscow, 1984. In Russian.

[89] Tanaka, K. On the complexity of negation-limited circuits. Master's thesis, Japan Advanced Institute of Science and Technology, Ishikawa, Japan, Mar. 1994.

[90] Tanaka, K. *Computational Difficulty—Scheduling and Circuit Complexity.* PhD thesis, Japan Advanced Institute of Science and Technology, Ishikawa, Japan, Mar. 1997.

[91] Tanaka, K., and Nishino, T. On the complexity of negation-limited Boolean networks (preliminary version). In *Proceedings of the Twenty-Sixth Annual ACM Symposium on Theory of Computing* (Montréal, Québec, Canada, 23–25 May 1994), pp. 38–47.

[92] Tanaka, K., and Nishino, T. A relationship between the number of negations and the circuit size. *IEICE Transactions on Information and Systems E79-D*, 9 (Sept. 1996), 1355–1357.

[93] Tanaka, K., Nishino, T., and Beals, R. Negation-limited circuit complexity of symmetric functions. *Inf. Process. Lett. 59*, 5 (9 Sept. 1996), 273–279.

[94] Tanaka, K., and Vlach, M. Improved algorithms for single machine scheduling with fuzzy due dates. In *Proceedings of the Second International Symposium on Operations Research and its Applications* (Guilin, China, Dec. 1996), pp. 260–269.

[95] Tanaka, K., and Vlach, M. Minimizing the maximum absolute lateness and range of lateness under generalized due dates on a single machine. *Annals of Operations Research* (1996). Accepted subject to minor revisions.

[96] Tanaka, K., and Vlach, M. Minimizing the range of lateness on a single machine under generalized due dates. *Information Systems and Operational Research* (1996). Accepted subject to minor revisions.

[97] Tanaka, K., and Vlach, M. Single machine scheduling to minimize the maximum lateness with both specific and generalized due dates. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences* (1996). To appear.

[98] Tanaka, K., and Vlach, M. Single machine scheduling to minimize the maximum lateness with both specific and generalized due dates. In *Proceedings of the Second International Symposium on Operations Research and its Applications* (Guilin, China, Dec. 1996), pp. 250–259.

[99] Tanaka, K., and Vlach, M. Single machine scheduling with generalized due dates. In *Symposium on Combinatorial Optimization* (London, Mar. 1996).

[100] Tardos, É. The gap between monotone and non-monotone circuit complexity is exponential. *Combinatorica 8*, 1 (1988), 141–142.

[101] Tegze, M., and Vlach, M. Improved bounds for the range of lateness on a single machine. *Journal of the Operational Research Society 39*, 7 (1988), 675–680.

[102] Turing, A. M. On computable numbers with an application the Entscheidungsproblem. In *Proc. London Math. Soc. Ser. 2, 42* (1937), pp. 230–265.

[103] Valiant, L. G. Negation can be exponentially powerful. *Theoretical Comput. Sci. 12*, 3 (Nov. 1980), 303–314.

[104] Valiant, L. G. Negation is powerless for Boolean slice functions. *SIAM J. Comput. 15*, 2 (May 1986), 531–535.

[105] Wegener, I. On the complexity of slice functions. *Theoretical Comput. Sci. 38*, 1 (May 1985), 55–68.

[106] Wegener, I. More on the complexity of slice functions. *Theoretical Comput. Sci. 43*, 2–3 (1986), 201–211.

[107] Wegener, I. *The Complexity of Boolean Functions*. Teubner, Wiley, 1987.

[108] Woeginger, G. J. Scheduling with time-dependent execution times. *Inf. Process. Lett. 54*, 3 (12 May 1995), 155–156.

[109] Wong, C. S., Yan, M., and Young, G. H. A note on a single machine generalized due dates scheduling problems. *Journal of Combinatorial Mathematics and Combinatorial Computing* (Dec. 1993). To appear, Revised May 1995.

[110] Yan, M. Q. Generalized due dates scheduling problems on a single machine. Master's thesis, San Francisco State University, San Francisco, Sept. 1993.

[111] Young, G. H., Wong, C. S., Yiu, V. S., and Yan, M. Scheduling tasks with generalized due dates and ready times. In *Proceedings of the Second International Symposium on Operations Research and its Applications* (Guilin, China, Dec. 1996), pp. 209–214.

[112] Zwick, U. A $4n$ lower bound on the combinational complexity of certain symmetric Boolean functions over the basis of unate dyadic Boolean functions. *SIAM J. Comput. 20*, 3 (June 1991), 499–505.

[113] Zwick, U. Boolean circuit complexity. Lecture Notes, Tel-Aviv University, Available at `http://www.math.tau.ac.il/~zwick/scribe-boolean.html`, 1994.

Publications

[1] Nishino, T., and Tanaka, K. On the negation-limited circuit complexity of clique functions. *IEICE Transactions on Information and Systems E78-D*, 1 (Jan. 1995), 86–89.

[2] Tanaka, K., Nishino, T., and Beals, R. Negation-limited circuit complexity of symmetric functions. *Inf. Process. Lett. 59*, 5 (9 Sept. 1996), 273–279.

[3] Tanaka, K., and Nishino, T. A relationship between the number of negations and the circuit size. *IEICE Transactions on Information and Systems E79-D*, 9 (Sept. 1996), 1355–1357.

[4] Beals, R., Nishino, T., and Tanaka, K. On the complexity of negation-limited boolean networks. *SIAM J. Comput.* (1996). To appear.

[5] Tanaka, K., and Vlach, M. Single machine scheduling to minimize the maximum lateness with both specific and generalized due dates. *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences* (1996). To appear.

[6] Tanaka, K., and Vlach, M. Minimizing the range of lateness on a single machine under generalized due dates. *Information Systems and Operational Research* (1996). To appear.

[7] Tanaka, K., and Vlach, M. Minimizing the maximum absolute lateness and range of lateness under generalized due dates on a single machine. *Annals of Operations Research* (1996). To appear.

[8] Tanaka, K., and Nishino, T. On the complexity of negation-limited Boolean networks (preliminary version). In *Proceedings of the Twenty-Sixth Annual ACM Symposium on Theory of Computing* (Montréal, Québec, Canada, 23–25 May 1994), pp. 38–47.

[9] Beals, R., Nishino, T., and Tanaka, K. More on the complexity of negation-limited circuits. In *Proceedings of the Twenty-Seventh Annual ACM Symposium on Theory of Computing* (Las Vegas, Nevada, 29 May–1 June 1995), pp. 585–595.

[10] Tanaka, K., and Vlach, M. Single machine scheduling with generalized due dates. In *Symposium on Combinatorial Optimization* (London, Mar. 1996).

[11] Tanaka, K., and Vlach, M. Single machine scheduling to minimize the maximum lateness with both specific and generalized due dates. In *Proceedings of the Second International Symposium on Operations Research and its Applications* (Guilin, China, Dec. 1996), pp. 250–259.

[12] Tanaka, K., and Vlach, M. Improved algorithms for single machine scheduling with fuzzy due dates. In *Proceedings of the Second International Symposium on Operations Research and its Applications* (Guilin, China, Dec. 1996), pp. 260–269.

Vita

Keisuke Tanaka

1988	Graduated from Kofu Minami High School, Kofu, Yamanashi, Japan.
1988–92	Attended Yamanashi University, Kofu, Yamanashi, Japan.
1992	B.Eng. in Computer Science, Yamanashi University.
1992–97	Graduate work in Information Science, Japan Advanced Institute of Science and Technology, Tatsunokuchi, Ishikawa, Japan.
1994	M.S. in Information Science, Japan Advanced Institute of Science and Technology.
1997	Ph.D. in Information Science, Japan Advanced Institute of Science and Technology.