Probabilistic Algorithms and Complexity Classes

Ryuhei Uehara

The University of Electro-Communications

March 1998

Probabilistic Algorithms and Complexity Classes

Ryuhei Uehara

A Dissertation Submitted to the Department of Computer Science and Information Mathematics of the University of Electro-Communications in Partial Fulfillment of the Requirements for the Degree of Doctor of Science

March 1998

Probabilistic Algorithms and Complexity Classes

Approved by supervisory committee:

Chairperson: Professor Takumi Kasai

Member: Professor Shigeki Iwata

Member: Professor Kiyoshi Hashimoto

Member: Professor Takashi Yokomori

Member: Professor Tetsuro Nishino

Copyright
by
Ryuhei Uehara
1998

Abstract

The main theme in this dissertation is a computation on a *probabilistic Turing machine* (PTM), which is a Turing machine with distinguished states called coin-tossing states. In general, it is assumed that a PTM has a fair coin. This is led by that any fixed biased coin can simulate another fixed biased coin in a constant expected time. We first consider the computation which will always terminates in *bounded* time, as opposed to *expected* time, and show that the coin with probability $\frac{1}{2}$ or $\frac{1}{3}$ is more expressive coin than the others. The complexity classes PP, BPP and RP are usually defined via PTMs with a perfect random source. Vazirani and Vazirani proved that RP does not change even if the PTMs only have access to an *arbitrary* semi-random source. Later, Vazirani proved that BPP does not change, either. We show that PP collapses to BPP if the PTMs only have access to an *arbitrary* semi-random source. We also show that RP changes to NP while both BPP and PP change to PSPACE if the PTMs only have access to a *specific* semi-random source. Next problem is to identify a Boolean function f out of a given set of Boolean functions F by asking for the value of f at adaptively chosen inputs. We consider the cases where F consists of functions obtained from one function g on n inputs by replacing k inputs by given constants, and g is one of the functions disjunction, parity and threshold. For many of the problems we get almost optimal bounds. The *maximal path set* (MPS) problem is to find, given an undirected graph $G = (V, E)$, a maximal subset F of E such that the subgraph induced by F is a forest in which each connected component is a path. We design two parallel algorithms for the problem. The first algorithm is faster and more efficient than the previously known algorithm. The second one is faster, runs on a weaker computation model, and is more efficient for input graphs of bounded degree. We then use the results to design an NC approximation algorithm for a variation of the shortest superstring problem. We also consider the vertex counterpart of the MPS problem. The NP-completeness of several problems are shown in the last chapter. The first problem is the generalized popular puzzle known as Hi-Q. The second problem is the vertex cover problem on a cubic, planar, and 3-connected graph of girth greater than 3. Problems on a triangulation of the sphere are the last.

Acknowledgement

I would like to express my sincere thanks to Professor Takumi Kasai, my supervisor, who has granted me the chance to start my research, and has given me innumerable advice and unrelenting encouragements through the whole work.

I am also deeply indebted to the other members of the doctoral committee, Professor Shigeki Iwata, Professor Kiyoshi Hashimoto, Professor Takashi Yokomori, and Associate Processor Tetsuro Nishino. Their helpful suggestions, comments, and encouragement were essential for the preparation of this dissertation. I would like to note in particular Professor Shigeki Iwata, who gave me the chance to achieve my first result when I was an undergraduate student.

Along the way I had substantial help from my colleagues and friends. I would like to note in particular Professor Xin He of State University of New York at Buffalo, Professor Ingo Wegener of University Dortmund, Associate Professor Kensei Tsuchida of Toyo University, and Associate Professor Zhi-Zhong Chen of Tokyo Denki University, who have supported me and have achieved the results contributing for this dissertation. I would like to note in particular Dr. Zhi-Zhong Chen, who has given me innumerable advice and essential discussions on my researches. I wishes to thank Associate Professor Seinosuke Toda of Nihon University, and Associate Professor Osamu Watanabe of Tokyo Institute of Technology, for their helpful advice during the course of this work. I thank Associate Professor Misao Nagayama of Tokyo Woman's Christian University, Assistant Professor Sei'ichi Tani of Tokai University, and Dr. Koichi Yamazaki of the University of Electro-Communications for providing me with much help and advice. I am grateful to staff of Center for Information Science in Tokyo Woman's Christian University for their maintenance of the environment and resources which made this effort possible.

Contents

1 Introduction **1**

2 Preliminaries **7**
2.1 Graphs, digraphs, and trees 7
2.2 Models of computation 8
2.3 Model of an efficient computation 11
2.4 Complexity classes with a semi-random source 12
2.5 Computation trees with a semi-random source 13
2.6 Complexity classes based on one Boolean function 15

3 Efficient Simulations by a Biased Coin **17**
3.1 Sets efficiently simulated 17
3.2 Relations between biased coins and a fair n-sided die 18

4 Complexity Classes Characterized by Semi-Random Sources **21**
4.1 Probability of a worst tree 21
4.2 The class δ-PP 22
4.3 The class $^{\exists}\delta$-RP 23
4.4 The classes $^{\exists}\delta$-BPP and $^{\exists}\delta$-PP 23
4.5 The class ARP 24
4.6 Remarks 25
4.6.1 Characterization of a worst tree 25
4.6.2 Complexity classes with irrationally δ-random sources 26

5 Optimal Attribute-Efficient Learning of Disjunction, Parity, and Threshold Functions **29**
5.1 Basic results 29
5.2 Disjunctions 29
5.3 Parity functions 31
5.4 Threshold functions 35

6 Fast NC and RNC Algorithms for Finding a Maximal Set of Paths **39**
6.1 Parallel algorithms for the MPS problem 39
6.1.1 Description of the RNC algorithm 39
6.1.2 Complexity analysis 40
6.1.3 Extension to digraphs 42
6.1.4 The NC algorithm 44
6.2 An application to shortest superstrings with flipping 45

6.3 Parallel algorithms for the MLF problem 49
6.3.1 Reducing the MLF problem to a simpler one 49
6.3.2 Description of the RNC algorithm 50
6.3.3 Complexity Analysis 52
6.3.4 The NC algorithm 54
6.4 Remarks 54

7 NP-completeness of the problems on a simple structure **57**
7.1 NP-completeness of Hi-Q 57
7.2 NP-complete problems on a 3-connected cubic planar graph 61
7.3 NP-complete problems on a triangulation of the sphere 63

Chapter 1

Introduction

The existence of a random coin has been extensively assumed for applications such as randomizing algorithms, cryptographic protocols, and stochastic simulation experiments. The available sources of randomness are sometimes physical devices, such as Zener diodes, and Geiger counters. The sample mean of the devices as an estimate of the bias of a random coin is always rational. Thus, in this dissertation, *bias* is a rational number between 0 and $\frac{1}{2}$ unless otherwise specified. (We note that in order to determine the irrationality of the mean of a random variable is investigated by Cover [Cov73].) A *probabilistic Turing machine* (PTM) is a Turing machine with distinguished states called coin-tossing states [Gil77]. In general, it is assumed that a PTM has a fair coin. This is led by the following results: Santos showed that there is no important difference between a PTM with a fair coin and a PTM with a biased coin for the language recognition [San71, Theorem 5.1]. Moreover, any biased coin can simulate another biased coin in a constant expected time. This follows from that any biased coin can simulate a fair coin in a constant expected time [Neu51, HS70, SW84], and vice versa [KY76].

The interchangeability of the simulation among coins is measured only by the *expected* time of computation. However, it is necessary to terminate in bounded time on some applications such as realtime operations, and synchronous machines. We first consider the computation which will always terminates in *bounded* time, as opposed to *expected* time in Chapter 3. Feldman *et al.* compared these models and they noticed that there is a conceptual difference between these two kinds of measures [FIN$^+$93]. We show that the conceptual difference breaks the interchangeability of the coins. We say that a PTM *efficiently* computes when it always terminates in bounded time. We consider a PTM with a biased coin which efficiently simulates biased coins. A *coin of bias* p, denoted by $\mathcal{C}_p$, is a coin such that an outcome of the coin-tossing is "$\mathcal{H}$" with probability p, and "$\mathcal{T}$" with probability $1-p$. Let $\mathcal{S}_p$ be the set of biases efficiently simulated by a PTM with $\mathcal{C}_p$. The first result in this dissertation is the following: (i) The set $\mathcal{S}_{\frac{1}{2}}$ is equal to the set of the biases in finite figures on the binary system, (ii) the set $\mathcal{S}_{\frac{1}{3}}$ is equal to the set of the biases in finite figures on the ternary system, and (iii) the set $\mathcal{S}_{\frac{b}{a}}$ in reduced form with $a > 3$, is properly contained in the set of the biases in finite figures on the a-ary system. In this sense, $\mathcal{C}_{\frac{1}{2}}$ and $\mathcal{C}_{\frac{1}{3}}$ are more expressive coins than the others. Then, oppositely, how many least coins which efficiently simulate the set of the biases in finite figures on the a-ary system when $a > 3$? We show that the answer to this question is only *two*. The results in this chapter can be found in [Ueh95b].

In Chapter 4, we deal with a *semi-random source*, introduced by Satha and Vazirani, as an extremely general model of an imperfect source of randomness [SV84, SV86]. In a semi-random source, the previous bits generated by the source can condition the next bit in an arbitrarily

way. Accordingly, the next bit is generated by the flip of a coin whose bias is fixed by an adversary who has complete knowledge of the history of the process. The adversary is limited to choosing a bias in $[\delta, 1-\delta]$ with some positive number $0 \leq \delta \leq \frac{1}{2}$. We call a semi-random source with parameter δ a *δ-random source*. The complexity classes PP, BPP and RP are usually defined via PTMs that have access to a perfect random source as in [Gil77]. A natural question is this: Do these classes change if the PTMs only have access to an arbitrary semi-random source rather than a perfect random source? This question was first considered by Vazirani and Vazirani, who defined the class SR_p, corresponding to RP, and showed that SR_p = RP [VV85]. Later, Vazirani similarly defined the class $SBPP$, corresponding to BPP, and proved that $SBPP$ = BPP [Vaz86]. In the same manner, we introduce the class δ-PP, corresponding to PP, as follows: A language L is in δ-PP if there exists a PTM M such that (1) for $x \in L$, M accepts with probability greater than $\frac{1}{2}$ for all δ-random sources and (2) for $x \notin L$, M accepts with probability less than $\frac{1}{2}$ for all δ-random sources. We prove that δ-PP = BPP for $0 < \delta < \frac{1}{2}$. Compared with that SR_p = RP and $SBPP$ = BPP, this result is very interesting. The results for the characterization can be found in [Ueh97].

Recall that in the definition of δ-PP, the PTM M is required to work for *every* δ-random source. This also motivated us to introduce the *dual* classes $^{\exists}\delta$-RP, $^{\exists}\delta$-BPP, and $^{\exists}\delta$-PP. More precisely, we define the class $^{\exists}\delta$-PP as follows: A language L is in $^{\exists}\delta$-PP if there exists a PTM M such that (1) for $x \in L$, M accepts with probability greater than $\frac{1}{2}$ for *at least one* δ-random source and (2) for $x \notin L$, M accepts with probability less than $\frac{1}{2}$ for *all* δ-random sources. The classes $^{\exists}\delta$-BPP and $^{\exists}\delta$-RP are defined similarly. We prove that $^{\exists}\delta$-RP = NP, and $^{\exists}\delta$-BPP = $^{\exists}\delta$-PP = PSPACE for $0 < \delta < \frac{1}{2}$. These results give new characterizations of NP and PSPACE using semi-random sources.

A stochastic Turing machine (STM) is a Turing machine that can perform nondeterministic and probabilistic moves and alternate between both types. The class ABPP characterized by a polynomial time bounded STM with bounded error probability was investigated as games against nature by Papadimitriou [Pap83], Arthur-Merlin games by Babai [Bab85], and interactive proof systems with public coins [GMR85, GS86]. The class APP characterized by a polynomial time bounded STM with unbounded error probability was also investigated by Papadimitriou [Pap94]. It is well known that the classes ABPP and APP is equal to PSPACE [Sha90]. We define the class ARP using the same manner as the class RP on a polynomial time bounded STM. Comparing with the classes RP, BPP, and PP, the class ARP is naturally defined on a STM. In addition to its correspondence to the classes on a PTM, there is another reason for us to be interested in the class ARP. Condon and Ladner investigated the complexity classes defined on the interactive proof systems with some restrictions on provers [CL95]. Their results contain the complexity classes defined by a STM with a restriction on space resources or/and a restriction on the number of random bits, and characterize the complexity classes MA and NP. Liśkiewicz and Reischuk also investigated the complexity classes characterized by a STM with a restriction on space resources or/and a restriction on the number of steps [LR97]. In this sense, the class ARP can be regard as a complexity class characterized by a STM with a restriction on the acceptance probability, or the class ARP can be recognized by an interactive proof system with a restrictions on the acceptance probability of its prover. Using the characterization of NP with a semi-random source, we show that ARP = NP. This result give a new characterization of NP using a stochastic Turing machine or an interactive proof system with a restriction on the acceptance probability. The results for the characterization of the computational classes using semi-random sources can be found in [Ueh95a].

We next consider a variety of learning problem, attribute-efficient learning of Boolean concepts (or Boolean functions) in Chapter 5. This class of problems belongs to the class of exact concept

learning problems where only membership queries are allowed and is a subclass of the problems introduced by Angluin [Ang88]. A Boolean concept class F over $\{0,1\}^n$ is a set of Boolean concepts $f \subseteq \{0,1\}^n$. Since we identify f with its characteristic function $\chi_f : \{0,1\}^n \to \{0,1\}$, we treat the concept f as a Boolean function. The learner tries to find out which function $f \in F$ is chosen by a teacher, an adversary, or by chance. For this reason the learner may ask queries $a \in \{0,1\}^n$ and the teacher answers with the correct value $f(a)$. The queries can be chosen adaptively, i.e., the i-th query may depend on the answers to the first $i-1$ queries. After having asked $a_1, \ldots, a_i$, a concept g is consistent with f if $f(a_1) = g(a_1), \ldots, f(a_i) = g(a_i)$. The learner has finished the job if a unique concept $f \in F$ is consistent with the given answers. Then f has been "learned". All queries have the same cost. The cost of a learning strategy for $f \in F$ is the number of queries to learn f. The cost of a learning strategy for the concept class F is the maximum of the cost to learn f for all $f \in F$ with the considered learning strategy. The task is to design learning strategies whose cost is close to known lower bounds for the learning problem. Since we ask binary queries, the simplest lower bound is the information theoretic lower bound $\lceil \log |F| \rceil$. For many of our problems this lower bound is almost optimal. Sometimes we prove better lower bounds by adversary arguments.

In the following we describe the concept classes we are interested in. These complexity classes typically are much smaller than the class of all Boolean functions. E.g., Angluin, Hellerstein, and Karpinski [AHK93] have investigated the class of functions representable by read-once formulas. We consider classes F based on one Boolean function g (which not necessarily has to belong to F). For some sets $S \subseteq \{1, \ldots, n\}$ (typically all sets or all sets of cardinality k) there exists one function $g_S \in F$ that is a subfunction of g where each x_i with $i \notin S$, say nonessential variable, is replaced by some given constant a_i. The identification of the unknown function in F is equivalent to the identification of the set of essential attributes (sometimes also called relevant variables). The situation described above is known as attribute-efficient learning. Littlestone [Lit88] obtains a lot of results, among them optimal results for disjunction in the mistake-bound model and Bshouty and Hellerstein [BH96] study also different models of learning.

We obtain more precise results for the attribute-efficient learning of special classes of functions. We consider the classes based on one of the following Boolean functions; disjunction, parity, and threshold. They are the most important basic gates in circuit theory and, therefore, the most fundamental functions. This fact motivates us to consider these special cases of attribute-efficient learning. In applications, we obtain the considered problem in the following case; if we know that some wires into a gate are destroyed and we like to identify which wires are destroyed by testing the gates on chosen inputs. This application motivates also the investigation of large k, e. g., $k = n - O(1)$. Interestingly, the complexity of the problem is completely different depending on the kind of the function when k is large.

Our results have been influenced by more general problem classes. Concept learning with membership queries is a special problem of the class of problems which have been considered as so-called "search problems" by Picard [Pic65] and Ahlswede and Wegener [AW87]. A complete learning strategy can be described as a binary decision tree where the inner nodes describe the queries and the outgoing 0-edge (resp. 1-edge) leads to the next query if the last query has been answered by 0 (resp. 1). At the leaves we find the function which is the chosen concept. The cost of the learning strategy is equal to the depth of the corresponding binary decision tree. Hence, lower bound techniques can be adopted from the theory on binary decision trees (see [Weg87]). A preliminary version of the research of attribute-efficient learning appears as an extended abstract in [UTW97].

We next turn to consider parallel complexity of maximality problems in Chapter 6. Since Karp and Wigderson showed that the maximal independent set (MIS) problem is in the class

NC [KW85], much work has been devoted to the study of parallel complexity of maximality problems. A typical maximality problem on graphs is to find either a maximal vertex-induced subgraph (MVIS) or a maximal edge-induced subgraph (MEIS) that satisfies a specified graph property. A common feature of the MVIS or MEIS problems is that they can be solved by simple greedy algorithms but are very difficult from the parallel point of view. So far, only a few MVIS or MEIS problems have been shown to be in NC or RNC [And85, Che91, Che96, Che95b, DGL91, IS86, KR91, PV93, SM95]. Parallel complexity of many natural MVIS or MEIS problems remains unknown.

The *maximal path set* (MPS) problem is to find, given an undirected graph $G = (V, E)$, a maximal subset F of E such that the subgraph induced by F is a forest in which each connected component is a path. The MPS problem is equivalent to the MEIS problem associated with the property "acyclic and maximum-degree ≤ 2". In [Che96], Chen introduced this problem and showed that parallel algorithms for it can be used to design parallel approximation algorithms for the famous shortest superstring problem. It is worth mentioning that the latter problem has been extensively studied due to its important applications in DNA sequencing and data compression [BJL+91, CGPR94, KPS94, TY93]. In [Che96], Chen presented an NC algorithm and an RNC algorithm for the MPS problem. The former runs in $O(\log^3 n)$ time with $O(n+m)$ processors on a CRCW PRAM and the latter runs in $O(\log^2 n)$ expected time with $O(n+m)$ processors on a CRCW PRAM. In this dissertation, we present two faster parallel algorithms for the problem. Our RNC algorithm runs in $O(\log n)$ expected time with $O(n+m)$ processors on a CRCW PRAM. This algorithm is faster and more efficient than Chen's RNC algorithm. Our NC algorithm for the MPS problem is obtained by *carefully* derandomizing the RNC algorithm, and it runs in $O(\log^2 n)$ time with $O(\Delta^2(n+m)/\log n)$ processors on an EREW PRAM, where Δ is the maximum degree of the input graph. Compared with Chen's NC algorithm, this algorithm is faster, runs on a weaker computation model, and is more efficient for input graphs of bounded degree. An immediate consequence of the results is that the parallel approximation algorithms for the shortest superstring problem given in [Che96] can be made faster. These parallel algorithms for the MPS problem was investigated in [UCH96, UCH97].

In [JLD92], Jiang *et al.* introduced an interesting variation of SSP. Let $S = \{s_1, \cdots, s_n\}$ be a set of n strings over an alphabet Σ. A *superstring-with-flipping* of S is a string s over Σ such that for each i, at least one of s_i and its reverse is a substring of s. The *amount of compression* achieved by a superstring-with-flipping s of S is $|S| - |s|$, where $|s|$ is the length of s and $|S|$ is the total length of the strings in S. Define SSP_F to be the following problem: Given a set S of strings, find a shortest superstring-with-flipping of S. Like SSP, SSP_F is NP-hard [JLD92] and it is of interest to design approximation algorithms for SSP_F. In [JLD92], Jiang *et al.* presented a polynomial-time approximation algorithm for SSP_F that produces a superstring-with-flipping whose length is at most 3 optimal. They also pointed out that there is a greedy algorithm for SSP_F that produces a superstring-with-flipping by which the amount of compression achieved is at least $\frac{1}{2}$ optimal [JLD92]. At present, no better sequential approximation algorithms for SSP_F are known. Also, no parallel approximation algorithm for SSP_F has been given previously. Here, using the ideas in our parallel algorithms for the MPS problem, we give an NC approximation algorithm for SSP_F that produces a superstring-with-flipping by which the amount of compression achieved is at least $\frac{1}{3+\epsilon}$ optimal for any $\epsilon > 0$. This NC approximation algorithm was presented in [UCH96].

We also consider the *vertex* counterpart of the MPS problem, which is the MVIS problem associated with the property "acyclic and maximum-degree ≤ 2". More precisely, we consider the following problem: Given an undirected graph $G = (V, E)$, find a maximal subset U of V such that the subgraph induced by U is a forest in which each connected component is a path. We call this problem the *maximal linear forest* (MLF) problem. For several special cases of the

MVIS problem associated with the property "acyclic", which is equivalent to the problem of computing a *maximal forest*, NC algorithms are given in [CH95]. However, parallel complexity of this problem still remains open. In this dissertation, we show that a related problem, namely, the MLF problem is in NC. This result can be found in [UC97]. In addition to its correspondence to the problem of computing a maximal forest, there is another reason for us to be interested in parallel complexity of the MLF problem. The MPS problem can be efficiently reduced to the MLF problem as follows: To solve the MPS problem for a given graph G, it suffices to solve the MLF problem for the edge graph of G. Thus, efficient parallel algorithms for the MLF problem can be used to design parallel approximation algorithms for the shortest superstring problem, too. Surprisingly, designing an NC algorithm for the MLF problem is a little more difficult than for the MPS problem even though they look very similar. Our RNC algorithm for the MLF problem runs in $O(\log n)$ expected time using n^2 processors on a CRCW PRAM. Our NC algorithm for the MLF problem is obtained by derandomizing the RNC algorithm. It runs in $O(\log^2 n)$ time using n^4 processors on an EREW PRAM.

Our RNC algorithm for the MPS problem has a similar structure to that of Israeli and Itai's RNC algorithm (I&I algorithm, for short) for the maximal matching problem [II86]. Our RNC algorithm for the MLF problem also has a similar structure to that of Luby's RNC algorithm for the MIS problem [Lub86]. All of these algorithms proceeds in stages. In each stage, the main jobs of the I&I algorithm and our algorithm for the MPS problem are to compute a random matching M in a certain subgraph of G and to delete (from G) some edges incident to the vertices matched by M. On the other hand, in each stage, the main jobs of Luby's algorithm and our algorithm for the MLF problem are to compute a certain independent set I in a certain (augmented) subgraph of G and to delete (from G) the vertices in I and (part of) their neighbors. In both the I&I algorithm and Luby's algorithm, the expected number of edges deleted in each stage is a constant fraction of the number of edges in G [II86, Lub86]. However, both our algorithms do not have this property. Instead, we define a *potential function* and prove that in each stage, the value of the potential function decreases by a constant fraction *on average*. This is the key for us to show that the algorithms run very fast.

The NP-completeness of several problems are shown in the last chapter of this dissertation. To show the NP-completeness of the problems on a simple structure means from two points of view. First, it is easy to consider the approximation/probabilistic algorithm for the problems on a simple structure. Secondly, since the reduction to the other problems is easy to consider, it is useful to analyze the more complex problems.

The first NP-complete problem is the generalized popular puzzle known as Hi-Q [UI90]. Complexity of problems concerning various generalized puzzles and games have been studied [ET76, Sch78, SC79]. Most of these generalizations were, however, artificial rather than natural, i.e., they were games on propositional formulas, sets, or graphs, or sometimes played three dimensionally. Complexity results on generalized popular two-person games such as Chess, Checkers, Go, Shogi, Gomoku, and Othello have been presented [Rei80, FL81, Rob83, Rob84, AKI87, IK94]. These generalization is natural in the sense that the board is extended to the size $n \times n$ with many pieces. Few results, however, have been presented concerning popular puzzles, which are naturally generalized. We consider a puzzle commonly known as Hi-Q. The objective of the puzzle is to make the board to a position with just one token on the goal. Thus a sequence of jumps which leads the board consisting of no tokens but on the goal will be an answer for the puzzle. The puzzle is generalized in such a way that an initial position is given on the extended board of size $n \times n$, and a goal is also given on which only one token will finally be placed. We show that the problem to determine whether there is an answer for a given generalized Hi-Q is NP-complete. The NP-hardness can be obtained by reducing from a variation of the

Hamiltonian cycle problem.

The second NP-complete problem is the vertex cover (VC) problem (or the maximum independent set problem) on a cubic, planar, and 3-connected graph of girth greater than 3. It is well known that finding a VC, or a maximum independent set of a given graph is NP-complete even for a cubic planar graph [GJ79]. We add the 3-connectedness and the restriction on the girth to the previously known result. The result simplifies the proof of NP-completeness of several well-known computational geometry problems involving 3-dimensional convex polyhedra investigated in [DG95], and also simplifies the proof of NP-completeness of computing the asteroidal number of a graph investigated in [KKM96]. Using the fact for the 3-connectedness of the cubic planar graph, we also have two NP-complete problems on a triangulation of the sphere, which is widely investigated and has many applications [PS90].

The rest of the dissertation is organized as follows. In Chapter 2 we introduce some notations and state basic results. In Chapter 3, the interchangeability of the biased coins on the efficient computation model is shown. In Chapter 4, we consider the complexity classes characterized by a semi-random sources. In Chapter 5, the algorithms for attribute-efficient learning of the special classes are designed. In Chapter 6, the parallel algorithms for the MPS and MLF problem are designed. The NP-completeness of several problems are shown in Chapter 7.

Chapter 2

Preliminaries

2.1 Graphs, digraphs, and trees

Let G be an undirected graph. The vertex set and edge set of G are denoted by $V(G)$ and $E(G)$, respectively. The *neighborhood* of a vertex v in G, denoted $N_G(v)$, is the set of vertices in G adjacent to v; $d_G(v) = |N_G(v)|$ is the *degree* of v in G. Vertices of degree 0 are called *isolated vertices*. For $U \subseteq V$, let $N_G(U) = \cup_{u \in U} N_G(u)$, and $G[U]$ be the graph (U, F), where $F = \{\{u, v\} \mid u, v \in U \text{ and } \{u, v\} \in E\}$. A subset M of E is a *matching* in G if no two edges in M have a common endpoint. A matching is *maximal* if it is not properly included in any other matching. We use $V(M)$ to denote the set of all vertices v such that v is an endpoint of some edge in a matching M. By a *path*, we always mean a simple path. Note that a single vertex is considered as a path (of length 0) with one endpoint. The *girth* of a graph G is the length of a shortest cycle in G. A set F of edges in G is called a *path set* if $(V(G), F)$ is a forest in which each connected component is a path. Intuitively speaking, if F is a path set, then $(V(G), F)$ is a collection of vertex-disjoint paths. A *maximal path set* (MPS) in G is a path set that is not properly contained in another path set. The MPS problem is to find, given G, an MPS in G. A subset U of V is called a *linear forest set* (LFS) if $G[U]$ is a forest in which each connected component is a path. Intuitively speaking, if U is an LFS, then $G[U]$ is a collection of vertex-disjoint paths. A *maximal linear forest set* (MLFS) in G is an LFS that is not properly contained in another LFS. The *maximal linear forest* (MLF) problem is to find, given a graph G, an MLFS in G. Throughout this dissertation, unless stated otherwise, G always denotes the input (undirected) graph, Δ denotes the maximum degree of G, and n and m denote the numbers of vertices and edges in G, respectively.

Now we turn to the definitions of *digraphs*. Let D be a digraph. The vertex set and arc set of D are denoted by $V(D)$ and $A(D)$, respectively. The *tail* and *head* of an arc (u, v) are u and v, respectively. For a subset M of $A(D)$, we use $V(M)$ to denote the set of all vertices v such that v is the tail or head of some arc in M. The *underlying graph* of D is the undirected graph $(V(D), E)$, where E consists of those edges $\{u, v\}$ with $(u, v) \in A(D)$ or $(v, u) \in A(D)$. The *indegree* (or *outdegree*) of a vertex u in D is the number of arcs with head (respectively, tail) u in D and is denoted by $d_D^-(u)$ (respectively, $d_D^+(u)$). The *total degree* of a vertex u is $d_D^-(u) + d_D^+(u)$ and is denoted by $d_D(u)$. Hereafter, a path in D always means a simple directed path. A set B of arcs in D is called a *directed path set* (DPS) if $(V(D), B)$ is an acyclic digraph in which the indegree and outdegree of each vertex are both at most 1. A *maximal directed path set* (MDPS) in D is a DPS that is not properly contained in another DPS. The MDPS problem is to find, given D, an MDPS in G. Throughout this subsection, D always denotes the input digraph, and n and m denote the numbers of vertices and arcs in D, respectively.

A *tree* is an acyclic connected graph. A *rooted tree* is a tree in which we identify a distinguished vertex r which we call *root*. We define the *depth* of a vertex; vertices at distance i from the root have depth i; the root itself has depth 0. A *leaf* of a tree is a vertex of the tree of degree 1. An *internal node* of a tree is a vertex being not a leaf. We call a digraph a *rooted directed tree* if its underlying graph is a tree, and there are directed paths from its root to every other vertex. Let (u, v) is an arc in a directed tree such that the depth of u is less than the depth of v. Then, u is called the *parent* of v, and v is called a *son* of v. If there is a directed path from a vertex u to a vertex v in a directed tree, then u is called an *ancestor* of v, and then v is called a *descendant* of u. The subgraph of a directed tree induced by a vertex v and all its descendants is called the *subtree rooted at* v. An *ordered tree* is a directed tree in which the set of sons of each vertex is ordered. A *binary tree* is an ordered tree in which no vertex has more than two sons. One of the sons is called the *left son*, while the other is called the *right son*. The subtree rooted at the left and right sons of v is called the *left* and *right subtrees*, respectively.

2.2 Models of computation

We assume a standard Turing machine model (see, e.g., [HU79] for further details). A *Turing machine* (TM) is denoted

$$M = (Q, \Sigma, \Gamma, \delta, q_0, B, F),$$

where

Q is the finite set of *states*,
Γ is the finite set of allowable *tape symbols*,
B ($B \in \Sigma$) is the *blank*,
Σ ($\Sigma \subseteq \Gamma - \{B\}$) is the set of *input symbols*,
δ is the *transition function*, a partial function from $Q \times \Gamma$ to $Q \times \Gamma \times \{L, R\}$,
q_0 in Q is the *start state*,
F ($F \subseteq Q$) is the set of *final states*.

We denote an *instantaneous description* (ID) of the Turing machine M by $\alpha_1 q \alpha_2$, where q is the current state of M (in Q), and $\alpha_1\alpha_2$ is the string in Γ^* that is the contents of the tape. We define a *move* of M as follows. Let $X_1X_2 \cdots X_{i-1}qX_i \cdots X_n$ be an ID. Suppose $\delta(q, X_i) = (p, Y, L)$. Then we write

$$X_1X_2 \cdots X_{i-1}qX_i \cdots X_n \vdash_M X_1X_2 \cdots X_{i-2}pX_{i-1}YX_{i+1} \cdots X_n.$$

Alternatively, suppose $\delta(q, X_i) = (p, Y, R)$. Then we write

$$X_1X_2 \cdots X_{i-1}qX_i \cdots X_n \vdash_M X_1X_2 \cdots X_{i-2}X_{i-1}YpX_{i+1} \cdots X_n.$$

If two ID's are related by $\vdash_M$, we say that the second results from the first by one move. If one ID results from another by some finite number of moves, they are related by the symbol $\vdash^*_M$. We drop the subscript M when no confusion results.

We sometimes call a TM defined above a *deterministic Turing machine* (DTM) in comparison with an nondeterministic Turing machine defined later.

Definition 2.1 A DTM $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ *accepts* an input x in Σ^* if $q_0x \vdash^* \alpha_1 p \alpha_2$ for some $p \in F$, and $\alpha_1, \alpha_2 \in \Gamma^*$. The language accepted by M is the set of strings in Σ^* accepted by M, and denoted by $L(M)$.

A *nondeterministic Turing machine* (NTM) is denoted

$$M = (Q, \Sigma, \Gamma, \delta, q_0, B, F),$$

where each element is as in the deterministic case, with the exception that the transition function is defined by

$$\delta : Q \times \Gamma \to \{Q \times \Gamma \times \{L, R\}\}^2.$$

Remark that we standardize, without loss of generality, the NTM that has exactly two choices (called *0-choice* and *1-choice*, respectively) per step.

The *instantaneous description* (ID) of the NTM M is defined as the same as in the deterministic case. The *move* of the NTM M is defined as follows. Let $X_1X_2 \cdots X_{i-1}qX_i \cdots X_n$ be an ID. When $(p, Y, L) \in \delta(q, X_i)$, we write

$$X_1X_2 \cdots X_{i-1}qX_i \cdots X_n \vdash_M X_1X_2 \cdots X_{i-2}pX_{i-1}YX_{i+1} \cdots X_n.$$

Alternatively, when $(p, Y, R) \in \delta(q, X_i)$, we write

$$X_1X_2 \cdots X_{i-1}qX_i \cdots X_n \vdash_M X_1X_2 \cdots X_{i-2}X_{i-1}YpX_{i+1} \cdots X_n.$$

The notation $\vdash_M^*$ is defined as the same way as in the deterministic case.

Definition 2.2 An NTM $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ *accepts* an input x in Σ^* if $q_0x \vdash^* \alpha_1p\alpha_2$ for some $p \in F$, and $\alpha_1, \alpha_2 \in \Gamma^*$. The language accepted by M is the set of strings in Σ^* accepted by M, and denoted by $L(M)$.

We sometimes call states of an NTM *nondeterministic states*.

Definition 2.3 The class of the languages recognized by a DTM in a polynomial time of the input length is denoted by P. On the other hand, the class of languages recognized by an NTM in a polynomial time of the input length is denoted by NP. We denote by PSPACE the class of the languages recognized by a DTM in a polynomial space of the input length.

Remark that the class of the languages recognized by an NTM in a polynomial space is equal to PSPACE. However, it is very famous open problem to determine whether $\mathsf{P} = \mathsf{NP}$ or not (see, e.g., [HU79, BDG88, Joh90] for further details).

A *probabilistic Turing machine* (PTM) is a Turing machine with distinguished states called probabilistic states defined by Gill [Gil77]. It is defined as follows; a PTM is denoted

$$M = (Q, \Sigma, \Gamma, \delta, q_0, B, F),$$

where each element is as in the nondeterministic case. The *instantaneous description* (ID) and the *move* of the PTM M are defined as the same as in the nondeterministic case. Without loss of generality, we assume that every move ends to the states in F, and each state in F is either *accepting state* or *rejecting state*.

Definition 2.4 Let $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ be a PTM. The *probability function* p_M, a partial function from ID to $[0, 1]$, is defined as follows: $p_M(X) = 1$ if X contains an accepting state; $p_M(X) = 0$ if X contains a rejecting state; and $p_M(X) = \frac{1}{2}(p_M(Y) + p_M(Z))$ where X, Y, and Z are IDs of M such that $X \vdash_M Y$ and $X \vdash_M Z$. We say that M *accepts* x *with probability* $p_M(q_0x)$.

We sometimes call states of a PTM *probabilistic states*.

Definition 2.5 The classes of the languages recognized by a PTM in a polynomial time of the input length are defined as follows:

A language L is in RP if there exists a PTM M such that (1) for $x \in L$, M accepts x with probability greater than $\frac{1}{2}$, and (2) for $x \notin L$, M always rejects.

A language L is in BPP if there exists a PTM M such that (1) for $x \in L$, M accepts x with probability greater than $\frac{3}{4}$, and (2) for $x \notin L$, M accepts x with probability less than $\frac{1}{4}$.

A language L is in PP if there exists a PTM M such that (1) for $x \in L$, M accepts x with probability greater than $\frac{1}{2}$, and (2) for $x \notin L$, M accepts x with probability less than $\frac{1}{2}$.

A *stochastic Turing machine* (STM) M is a Turing machine that has nondeterministic states and probabilistic states. It is denoted as follows; an STM is denoted

$$M = (Q, \Sigma, \Gamma, \delta, q_0, B, F),$$

where each element is as in the nondeterministic case. The *instantaneous description* (ID) and the *move* of the STM M are defined as the same as in the nondeterministic case. We assume that every state in Q is either *probabilistic state* or *nondeterministic state*. Without loss of generality, we assume that every move ends to the states in F, and each state in F is either *accepting state* or *rejecting state*. Moreover, without loss of generality, we assume that an STM starts from a nondeterministic state, alternates between nondeterministic states and probabilistic states, and ends in a probabilistic state.

Definition 2.6 Let $M = (Q, \Sigma, \Gamma, \delta, q_0, B, F)$ be an STM. The *probability function* p_M, a partial function from ID to $[0, 1]$, is defined as follows: $p_M(X) = 1$ if X contains an accepting state; $p_M(X) = 0$ if X contains a rejecting state; $p_M(X) = \max(p_M(Y), p_M(Z))$ where X, Y, and Z are IDs of M such that $X \vdash_M Y$, $X \vdash_M Z$, and X contains a nondeterministic state; and $p_M(X) = \frac{1}{2}(p_M(Y) + p_M(Z))$ where X, Y, and Z are IDs of M such that $X \vdash_M Y$, $X \vdash_M Z$, and X contains a probabilistic state. We say that M *accepts* x *with probability* $p_M(q_0x)$.

The classes ABPP and APP are defined by Papadimitriou [Pap94]:

Definition 2.7 The classes of the languages recognized by an STM in a polynomial time of the input length are defined as follows:

A language L is in ABPP if there exists an STM M such that (1) for $x \in L$, M accepts x with probability greater than $\frac{3}{4}$, and (2) for $x \notin L$, M accepts x with probability less than $\frac{1}{4}$.

A language L is in APP if there exists an STM M such that (1) for $x \in L$, M accepts x with probability greater than $\frac{1}{2}$, and (2) for $x \notin L$, M accepts x with probability less than $\frac{1}{2}$.

Similarly, we define the new class ARP as follows:

Definition 2.8 A language L in ARP is accepted by an STM M in a polynomial time of the input length such that (1) for $x \in L$, M accepts x with probability greater than $\frac{1}{2}$, and (2) for $x \notin L$, M always rejects.

For APP and ABPP, the following theorem is well known:

Theorem 2.9 ABPP = APP = PSPACE.

Outline of proof. Babai introduced "Arthur-Merlin games", and defined the class AM(*poly*) using the games [Bab85]. An Arthur-Merlin game directly corresponds to the computation of an STM; an Arthur's (resp. Merlin's) turn corresponds to a probabilistic (resp. nondeterministic) state. Thus it is easy to see that ABPP = AM(*poly*). On the other hand, Goldwasser, Micali, and Rackoff introduced interactive proof systems and defined the class IP using the systems [GMR85], and Goldwasser and Sipser showed that IP = AM(*poly*)[GS86]. Moreover, Shamir showed that PSPACE = IP [Sha90]. Thus ABPP = AM(*poly*) = IP = PSPACE. It is easy to see that ABPP $\subseteq$ APP $\subseteq$ PSPACE. ∎

Now we turn to the parallel computation model. In this dissertation, we use the *parallel random-access machine* (PRAM) model (see e.g. [KR90]). A PRAM consists of several independent sequential processors, each with its own private memory, communicating with one another through a global memory. In one unit of time, each processor can read one global or local memory location, execute a single RAM operation, and write into one global or local memory location. An *Exclusive-Read Exclusive-Write* (EREW) PRAM is the parallel model where the processors operate synchronously and share a common memory, but no two of them are allowed simultaneous access to a memory cell (whether the access is for reading or for writing in that cell). A *Concurrent-Read Concurrent-Write* (CRCW) PRAM differs from the EREW PRAM in that both simultaneous reading and simultaneous writing to the same cell are allowed; in case of simultaneous writing, the processor with lowest index succeeds.

Definition 2.10 The class of the languages recognized by a PRAM that uses at most $\log^{O(1)} n$ time and $n^{O(1)}$ processors on inputs of size n is denoted by NC. A language L is in RNC if and only if there is a polynomial p and a language L' in NC such that for all input strings x the following properties hold; (1) if $x \in L$, then for over half of the possible strings y with $|y| = p(|x|)$, the string xy is in L'; and (2) if $x \notin L$, then for all of the possible strings y with $|y| = p(|x|)$, the string xy is not in L'.

We remark that RNC could be defined in terms of probabilistic PRAMs with certain probabilities of obtaining the correct answer (see e.g. [Joh90]).

As the input representation of a graph G, we assume that $V(G) = \{0, 1, \cdots, n-1\}$ and that each vertex has a list of the edges incident to it. Thus, each edge $\{i, j\}$ has two copies — one in the edge list for vertex i and the other in the edge list for vertex j. We assume that each of the two copies has a pointer to the other. The input representation of a digraph D is the same as that of an undirected graph except that each vertex i now has *two* lists; one of the lists consists of all arcs with tail i and the other consists of all arcs with head i.

2.3 Model of an efficient computation

We define an *efficient algorithm* $\mathcal{A}_p$, representing an efficient simulation by a PTM with $\mathcal{C}_p$, by a finite rooted binary directed tree such that;

(i) each internal node, representing a coin-tossing, has two arcs to its sons,
(ii) one of the two arcs above is labeled p, and the other is labeled $1-p$, and
(iii) each leaf is labeled "$\mathcal{H}$" or "$\mathcal{T}$".

The *depth of a leaf* (or the *depth of an internal node*) is the number of the arcs of the directed path from the root to the leaf (or the internal node, respectively). The *depth of an efficient algorithm* is the maximal depth of its leaves. The *probability of a leaf* is defined by the product of all labels on the path from the leaf to the root. An efficient algorithm $\mathcal{A}_p$ *simulates* $\mathcal{C}_q$, if the sum of all probabilities of the leaves of $\mathcal{A}_p$ labeled "$\mathcal{H}$" is equal to q. We write $\mathcal{C}_p \to \mathcal{C}_q$ if there

is an efficient algorithm $\mathcal{A}_p$ simulating $\mathcal{C}_q$, and by $\mathcal{C}_p \not\to \mathcal{C}_q$ otherwise. An efficient algorithm is called a *complete efficient algorithm* if the depths of all leaves are equal to each other. Note that for each efficient algorithm of depth k, there exists a complete efficient algorithm of depth k, simulating the same coin. Let $\mathcal{S}_p$ be the set of biases efficiently simulated by $\mathcal{A}_p$.

To deal with a rational number as a bias, we introduce the notation on the a-ary system. The rational numbers consist of the finite decimals and the recurring decimals. For a finite decimal p on the a-ary system, we denote p by $(0.p_1p_2\cdots p_k)_a$, where $k > 0$, $0 < i \leq k$, $0 \leq p_i < a$ and $p_k \neq 0$, and $p = p_1a^{-1} + p_2a^{-2} + \cdots + p_ia^{-i} + \cdots + p_ka^{-k}$. The number of figures of p on the a-ary system is denoted by $|p|_{(a)}$ (equal to k, in the case above). For a recurring decimal q on the a-ary system, $|q|_{(a)}$ is defined as an infinity. Note that for a rational number p, it depends on a whether $|p|_{(a)}$ is finite or not (e.g. $\frac{1}{2} = (0.1)_2 = (0.11\cdots)_3$).

2.4 Complexity classes with a semi-random source

A *semi-random source* with parameter δ, or simply a *δ-random source*, is defined as follows.

Definition 2.11 [SV84] Let δ be a number such that $0 \leq \delta \leq \frac{1}{2}$. A *semi-random source* with parameter δ outputs bits $X_1X_2\cdots$, such that for all i and for all $x_1, x_2, \cdots$,

$$\delta \leq \Pr[X_i = x_i \mid X_1 = x_1, \cdots, X_{i-1} = x_{i-1}] \leq 1 - \delta.$$

Definition 2.12 [VV85] A language L is in δ-RP if there exists a PTM M such that (1) for $x \in L$, M accepts with probability greater than $\frac{1}{2}$ for every δ-random source, and (2) for $x \notin L$, M always rejects.

Definition 2.13 [Vaz86] A language L is in δ-BPP if there exists a PTM M such that (1) for $x \in L$, M accepts with probability greater than $\frac{3}{4}$ and (2) for $x \notin L$, M accepts with probability less than $\frac{1}{4}$ for every δ-random source.

We note that the class δ-RP is denoted by SR_p in [VV85], and the class δ-BPP is denoted by $SBPP$ in [Vaz86]. To unify the notations, we use δ-RP and δ-BPP following from [Pap94]. Here, we introduce new classes δ-PP, $^{\exists}\delta$-RP, $^{\exists}\delta$-BPP, and $^{\exists}\delta$-PP:

Definition 2.14 A language L is in δ-PP if there exists a PTM M such that (1) for $x \in L$, M accepts with probability greater than $\frac{1}{2}$ and (2) for $x \notin L$, M accepts with probability less than $\frac{1}{2}$ for every δ-random source.

Definition 2.15 A language L is in $^{\exists}\delta$-RP if there exists a PTM M such that (1) for $x \in L$, M accepts with probability greater than $\frac{1}{2}$ for *at least one* δ-random source and (2) for $x \notin L$, M always rejects.

Definition 2.16 A language L is in $^{\exists}\delta$-BPP if there exists a PTM M such that (1) for $x \in L$, M accepts with probability greater than $\frac{3}{4}$ for *at least one* δ-random source and (2) for $x \notin L$, M accepts with probability less than $\frac{1}{4}$ for *every* δ-random source.

Definition 2.17 A language L is in ${}^{\exists}\delta$-PP if there exists a PTM M such that (1) for $x \in L$, M accepts with probability greater than $\frac{1}{2}$ for *at least one* δ-random source and (2) for $x \notin L$, M accepts with probability less than $\frac{1}{2}$ for *every* δ-random source.

The following proposition follows from the definitions immediately:

Proposition 2.18 For every δ with $0 < \delta < \frac{1}{2}$, the following hold.

$$\begin{aligned}
\mathsf{P} &= 0\text{-}\mathsf{RP} \subseteq \delta\text{-}\mathsf{RP} \subseteq \frac{1}{2}\text{-}\mathsf{RP} = \mathsf{RP} = {}^{\exists}\frac{1}{2}\text{-}\mathsf{RP} \subseteq {}^{\exists}\delta\text{-}\mathsf{RP} \subseteq {}^{\exists}0\text{-}\mathsf{RP} = \mathsf{NP}, \\
\mathsf{P} &= 0\text{-}\mathsf{BPP} \subseteq \delta\text{-}\mathsf{BPP} \subseteq \frac{1}{2}\text{-}\mathsf{BPP} = \mathsf{BPP} = {}^{\exists}\frac{1}{2}\text{-}\mathsf{BPP},\ {}^{\exists}0\text{-}\mathsf{BPP} = \mathsf{NP}, \\
\mathsf{P} &= 0\text{-}\mathsf{PP} \subseteq \delta\text{-}\mathsf{PP} \subseteq \frac{1}{2}\text{-}\mathsf{PP} = \mathsf{PP} = {}^{\exists}\frac{1}{2}\text{-}\mathsf{PP}, \text{ and } {}^{\exists}0\text{-}\mathsf{PP} = \mathsf{NP}.
\end{aligned}$$

Proof. Since a $\frac{1}{2}$-random source is a perfect random source, it is easy to see that $\mathsf{RP} = \frac{1}{2}\text{-}\mathsf{RP} = {}^{\exists}\frac{1}{2}\text{-}\mathsf{RP}$, $\mathsf{BPP} = \frac{1}{2}\text{-}\mathsf{BPP} = {}^{\exists}\frac{1}{2}\text{-}\mathsf{BPP}$, and $\mathsf{PP} = \frac{1}{2}\text{-}\mathsf{PP} = {}^{\exists}\frac{1}{2}\text{-}\mathsf{PP}$. For δ and δ' with $0 \leq \delta \leq \delta' \leq \frac{1}{2}$, a δ'-random source is also a δ-random source. A 0-random source generates 1 with probability between 0 and 1. Thus for 0-RP, 0-BPP and 0-PP, since every computation for each input must produce same answer, the algorithm must be deterministic. This implies $0\text{-}\mathsf{RP} = 0\text{-}\mathsf{BPP} = 0\text{-}\mathsf{PP} = \mathsf{P}$. Conversely, for ${}^{\exists}0\text{-}\mathsf{RP}$, ${}^{\exists}0\text{-}\mathsf{BPP}$ and ${}^{\exists}0\text{-}\mathsf{PP}$, since it is sufficient to accept that at least one computation reaches accepting state, the algorithm is nondeterministic. This implies ${}^{\exists}0\text{-}\mathsf{RP} = {}^{\exists}0\text{-}\mathsf{BPP} = {}^{\exists}0\text{-}\mathsf{PP} = \mathsf{NP}$. ∎

Note that for $\delta < \delta'$, ${}^{\exists}\delta'\text{-}\mathsf{BPP} \subseteq {}^{\exists}\delta\text{-}\mathsf{BPP}$ and ${}^{\exists}\delta'\text{-}\mathsf{PP} \subseteq {}^{\exists}\delta\text{-}\mathsf{PP}$ do *not* hold from its definition immediately, whereas ${}^{\exists}\delta'\text{-}\mathsf{RP} \subseteq {}^{\exists}\delta\text{-}\mathsf{RP}$ holds. To see this, let M be a PTM such that $L(M) \in {}^{\exists}\delta'\text{-}\mathsf{BPP}$ and δ and δ' be two numbers with $0 \leq \delta < \delta' \leq \frac{1}{2}$. It likely holds that $L(M) \in {}^{\exists}\delta\text{-}\mathsf{BPP}$ since the input x with $x \in L(M)$ satisfies the condition of the definition of ${}^{\exists}\delta\text{-}\mathsf{BPP}$ since a δ'-random source is also a δ-random source. However, for $x \notin L$, M must accepts with probability less than $\frac{1}{4}$ for *every* δ-random source. Since the set of δ-random sources properly contains the set of δ'-random sources when $\delta < \delta'$, M may not satisfy the condition.

For $0 < \delta \leq \frac{1}{2}$, the following results are known.

Theorem 2.19 [VV85] $\delta\text{-}\mathsf{RP} = \mathsf{RP}$.

Theorem 2.20 [Vaz86] $\delta\text{-}\mathsf{BPP} = \mathsf{BPP}$.

2.5 Computation trees with a semi-random source

The probability that a PTM with a δ-random source accepts, depends not only on the number of the accepting leaves, but also on the distribution of the accepting leaves in the computation tree of the PTM. To consider the distribution, we need several definitions and propositions.

A *δ-tree* is a rooted full binary directed tree whose every leaf is labeled "accept" or "reject". The leaves labeled "accept" and "reject" are respectively called *accepting* and *rejecting leaves*. For a δ-tree T, we denote by $|T|$ the number of accepting leaves of T.

Definition 2.21 A *δ-assignment* F to a δ-tree is a mapping from the set of arcs of the δ-tree to the interval $[\delta, 1-\delta]$ such that the two arcs leaving each internal node are assigned values whose sum is 1.

Definition 2.22 Let T be a δ-tree, and F be a δ-assignment to T. The *probability of a node* of T under F is the product of the values assigned to the arcs on the path from the root to the node by F. The *probability of* T under F, denoted by $\Pr[T|F]$, is the sum of the probabilities of the accepting leaves.

Definition 2.23 For a given δ-tree T and an assignment F to T, F is *a maximum assignment* if $\Pr[T|F] \geq \Pr[T|F']$ for every δ-assignment F' to T.

Proposition 2.24 For every δ-tree T, there exists a maximum assignment F to T such that F only assigns δ or $1-\delta$ to each arc of T.

Proof. For a given δ-tree T, we recursively define an assignment F as follows: First, let v be a node whose sons are leaves. If at least one of its sons is accepting leaf, assign $(1-\delta)$ to the arc between v and the son, and assign δ to the other arc. Otherwise, assign $(1-\delta)$ and δ to the two arcs between v and its two sons in any way. Second, let u be a node whose sons are the roots of two subtrees, say T_0 and T_1, whose all arcs are already assigned. If $\Pr[T_0|F] > \Pr[T_1|F]$, assign $(1-\delta)$ to the arc between u and the root of T_0 and assign δ to the arc between u and the root of T_1. Otherwise, assign $(1-\delta)$ to the arc between u and the root of T_1 and assign δ to the arc between u and the root of T_0. Using a simple induction on the depth of the tree, we find that $\Pr[T|F] \geq \Pr[T|F']$ for every δ-assignment F' to T. ∎

By the proposition, it is sufficient to consider δ-assignments that only assign the values δ and $1-\delta$ to maximize the probability. For a given δ-tree T, we denote by $F_{\max}(T)$ the maximum assignment defined in the proof of the proposition.

Definition 2.25 A δ-tree T_w is *worst* if $\Pr[T_w|F_{\max}(T_w)] \leq \Pr[T|F_{\max}(T)]$ for every δ-tree T with $|T_w| = |T|$. A δ-tree T_b is *best* if $\Pr[T_b|F_{\max}(T_b)] \geq \Pr[T|F_{\max}(T)]$ for every δ-tree T with $|T_b| = |T|$.

Proposition 2.26 Let T_w be a worst tree and T_b be a best tree such that $\Pr[T_w|F_{\max}(T_w)] = \Pr[T_b|F_{\max}(T_b)]$. Then, for every δ-tree T such that $\Pr[T_w|F_{\max}(T_w)] = \Pr[T|F_{\max}(T)]$, $|T_w| \geq |T| \geq |T_b|$.

Proof. There exists a worst tree T' such that $|T'| = |T|$ and $\Pr[T'|F_{\max}(T')] \leq \Pr[T|F_{\max}(T)] = \Pr[T_w|F_{\max}(T_w)]$. Thus, $|T_w| \geq |T'| = |T|$. A symmetric argument works for the best tree. ∎

Proposition 2.27 Let T_b be a best tree of depth d with a δ-random source. Then, the probability of the best tree under a maximum assignment is: $\Pr[T_b|F_{\max}(T_b)] = \sum_{i=0}^{t_0} \binom{d}{i} \delta^i (1-\delta)^{d-i} + t_1 \delta^{t_0+1}(1-\delta)^{d-t_0-1}$, where $|T_b| = \sum_{i=0}^{t_0} \binom{d}{i} + t_1$ with $t_1 \leq \binom{d}{t_0+1}$.

Proof. For each i with $0 \leq i \leq d$, a δ-tree of depth d has $\binom{d}{i}$ leaves corresponding to the probability $\delta^i(1-\delta)^{d-i}$, and $\delta^i(1-\delta)^{d-i} < \delta^{i-1}(1-\delta)^{d-i+1}$ since $\delta < (1-\delta)$. Thus, taking $|T_b|$ leaves which correspond to the largest $|T_b|$ probabilities, and labeling them "accept", we have a best tree. This implies the proposition. ∎

The structure of a best tree is characterized completely by Proposition 2.27. Hereafter, we will only consider the structure of a worst tree. From now on, we explicitly distinguish the two sons of each internal node in a δ-tree as the left and right sons. We denote by T_l and T_r the left and right subtrees of T, respectively. We similarly denote by T_{lr} the right subtree of T_l, and so on. We call the root of T the *parent* of T_l and T_r.

Definition 2.28 Let T be a δ-tree of depth $d \geq 0$. T is a *clustered tree* if
(1) $d = 0$,
(2) $|T_l| = 0$, and T_r is a clustered tree, or
(3) T_l is a clustered tree, and $|T_r| = 2^{d-1}$.

Intuitively, a clustered tree is a tree whose rightmost leaves are all accepting leaves. We fix the assignment defined as follows; for each internal node, assign $(1 - \delta)$ to every arc incident to a right son, and δ to every arc incident to a left son. Simple induction shows that the fixed assignment is the maximum assignment to the clustered trees.

2.6 Complexity classes based on one Boolean function

A *query* is an input $a \in \{0,1\}^n$. In the following we sometimes identify a query a with the query set $A = \{i \mid a_i = 1\}$. A query set leading to the answer 1 is called a *winner*. For each i with $1 \leq i \leq n$, we denote the query set $\{i\}$ of cardinality 1 by E_i.

The classes OR and $\mathsf{OR}(k)$ are based on the disjunction of n variables. The nonessential variables are replaced by the constant 0. For a set $S \subseteq \{1,\ldots,n\}$, we denote the disjunction of all x_i with $i \in S$ by OR_S. Then, the class $\mathsf{OR}(k)$ consists of all OR_S where $S \subseteq \{1,\ldots,n\}$ and $|S| = k$, and the class OR is $\bigcup_{k=0}^{n} \mathsf{OR}(k)$. In other papers the class $\mathsf{OR}(k)$ is called 1-DNF of size k (over n variables). We use the shorter notion $\mathsf{OR}(k)$. In the same way we obtain the classes PAR and $\mathsf{PAR}(k)$ for the parity function which decides whether the number of ones in the input is odd or even. The threshold function on n variables and threshold t decides whether the input contains at least t ones. Again we replace the nonessential variables by 0. Then we consider the classes THR, $\mathsf{THR}(k)$, and $\mathsf{THR}_t(k)$, i. e., we distinguish whether we consider all $S \subseteq \{1,\ldots,n\}$ or all $S \subseteq \{1,\ldots,n\}$ of cardinality k and we distinguish whether the threshold t is unknown or known. In the theory of neural nets and also in learning theory more general threshold functions are considered. For given weights $w_1,\ldots,w_n$ and a threshold t the output of the corresponding threshold function is 1 iff $w_1x_1 + \cdots + w_nx_n \geq t$. We restrict ourselves to the case $w_1 = \cdots = w_n = 1$.

By de Morgan's rules it easily follows that all results for disjunctions also hold for conjunctions, if the nonessential variables for conjunctions are replaced by ones.

Chapter 3

Efficient Simulations by a Biased Coin

3.1 Sets efficiently simulated

The following theorem is the main theorem in this chapter.

Theorem 3.1 (i) The set $\mathcal{S}_{\frac{1}{2}}$ is equal to the set of the biases in finite figures on the binary system,

(ii) the set $\mathcal{S}_{\frac{1}{3}}$ is equal to the set of the biases in finite figures on the ternary system, and

(iii) the set $\mathcal{S}_{\frac{b}{a}}$ in reduced form with $a > 3$, is properly contained in the set of the biases in finite figures on the a-ary system.

We first show that for every integer a greater than 1 and b with $0 < b \leq \frac{a}{2}$, $\mathcal{C}_{\frac{b}{a}} \rightarrow \mathcal{C}_p$ implies that p is a finite decimal on the a-ary system. This is a sufficient condition of Theorem 3.1.

Lemma 3.2 For every $\mathcal{C}_p$ simulated by an efficient algorithm $\mathcal{A}_{\frac{b}{a}}$ of depth k, $|p|_{(a)}$ is bounded by k.

Proof. For an efficient algorithm $\mathcal{A}_{\frac{b}{a}}$ of depth k, the denominator of the probability of a leaf is a^i with $0 < i \leq k$. Since p is the sum of these probabilities, it is a fraction of denominator a^k. This implies the lemma. ∎

Next we show necessary conditions of Theorem 3.1 for each case (i), (ii), and (iii). We now show a technical lemma in order to prove the claim for $a = 3$. In this lemma, we consider a PTM with many coins:

Lemma 3.3 Let a_1 and a_2 be integers with $2 \leq a_1 \leq a_2$. Then for any i with $0 < i < a_1 a_2$, there exists a PTM M with $\mathcal{C}_{\frac{1}{a_1}}, \mathcal{C}_{\frac{1}{a_2}}, \mathcal{C}_{\frac{2}{a_2}}, \cdots, \mathcal{C}_{\frac{(a_2-1)}{a_2}}$ such that M efficiently simulates $\mathcal{C}_{\frac{i}{a_1 a_2}}$.

Proof. We constructs an efficient algorithm which simulates $\mathcal{C}_{\frac{i}{a_1 a_2}}$ by using three of the coins. Without loss of generality, we can assume that $0 < i < \lfloor \frac{a_1 a_2}{2} \rfloor$. Let i_1 and i_2 be integers such that $i_2 = i \bmod (a_1 - 1)$ and $i_1 = \lfloor \frac{i}{a_1 - 1} \rfloor$. We note that $i = i_1(a_1 - 1) + i_2$ holds. Since $(a_1 - 1)(a_2 - 1) + (a_2 - 1) = a_1(a_2 - 1) \geq \frac{a_1 a_2}{2} \geq i$, we can assume that $0 \leq i_1, i_2 \leq a_2 - 1$. For these i_1 and i_2, we construct an efficient algorithm as follows:

1. toss $\mathcal{C}_{\frac{1}{a_1}}$;

2. if outcome is "$\mathcal{H}$" then toss $\mathcal{C}_{\frac{i_2}{a_2}}$ else toss $\mathcal{C}_{\frac{i_1}{a_2}}$;
3. if outcome is "$\mathcal{H}$" then output "$\mathcal{H}$" else output "$\mathcal{T}$".

The efficient algorithm outputs "$\mathcal{H}$" with probability equal to $\frac{1}{a_1}\frac{i_2}{a_2} + \frac{a_1-1}{a_1}\frac{i_1}{a_2} = \frac{i_2}{a_1 a_2} + \frac{i_1(a_1-1)}{a_1 a_2} = \frac{i}{a_1 a_2}$. ∎

Next we show necessary condition of Theorem 3.1(ii).

Lemma 3.4 For every $\mathcal{C}_p$ such that $|p|_{(3)}$ is finite, $\mathcal{C}_{\frac{1}{3}} \to \mathcal{C}_p$ holds.

Proof. Let $|p|_{(3)} = k$ and $p = \frac{c}{3^k}$ where $0 \le c \le 3^k$. We show by induction for k. Since it is clear when $k = 1$, suppose $k > 1$. By letting $a = 3$, and $b = 3^k$, inductive hypothesis satisfies the conditions of Lemma 3.3. Thus Lemma 3.3 implies the lemma. ∎

As same as Lemma 3.4, we can show it for $\mathcal{C}_{\frac{1}{2}}$, which is a necessary condition of Theorem 3.1(i). We note that it has been shown in [KY76] how to construct a tree algorithm with $\mathcal{C}_{\frac{1}{2}}$, which *optimally* simulates any given rational bias. For every rational bias, the expected value of the times of the coin-tossing of the algorithm is at most 2. Although the algorithm does not terminate for some biases, it efficiently simulates any rational bias in finite figures on the binary system. This also gives a necessary condition of Theorem 3.1(i). Finally we show necessary condition of Theorem 3.1(iii).

Lemma 3.5 Let a and b be positive integers such that a and b are relatively prime, $a > 3$, and $0 < b < \frac{a}{2}$. Then there exists a rational number p such that $|p|_{(a)}$ is finite and $\mathcal{C}_{\frac{b}{a}} \not\to \mathcal{C}_p$ holds.

Proof. Since $a > 3$, there exists an integer c such that $0 < c \le \frac{a}{2}$ and $c \ne b$. (Note that a and c may not be relatively prime.) To prove the lemma, assume $\mathcal{C}_{\frac{b}{a}} \to \mathcal{C}_{\frac{c}{a}}$ and derive a contradiction. Then there would exist a complete efficient algorithm $\mathcal{A}_{\frac{b}{a}}$ simulating $\mathcal{C}_{\frac{c}{a}}$. Let k be the depth of $\mathcal{A}_{\frac{b}{a}}$. Then for each integer i with $0 \le i \le k$, it would have $\binom{i}{k}$ leaves of probability equal to $\left(\frac{b}{a}\right)^{k-i}\left(\frac{a-b}{a}\right)^{i} = \frac{(a-b)^i b^{k-i}}{a^k}$. Hence every leaf has a probability being a multiple of $\frac{a-b}{a^k}$ (we say that x is a *multiple* of y if $x = wy$ for some integer w), except one leaf with probability equal to $\frac{b^k}{a^k}$. Let H be the set of the leaves labeled "$\mathcal{H}$", and T be the set of the leaves labeled "$\mathcal{T}$". Then one of H and T only contains leaves of probabilities being multiples of $\frac{a-b}{a^k}$. Thus one of $\frac{c}{a}(= \sum_{j\in H}(\text{probability of } j))$ and $\frac{a-c}{a}(= \sum_{j\in T}(\text{probability of } j))$ is a multiple of $\frac{a-b}{a^k}$. Consequently, one of the following holds for some positive integer d_1 and d_2: $\frac{c}{a} = d_1\frac{a-b}{a^k}$, or $\frac{a-c}{a} = d_2\frac{a-b}{a^k}$. If the former holds, $a^{k-1}c = d_1(a-b)$. If a and b are relatively prime, so are a and $(a-b)$. Hence $c = d_1'(a-b)$ for some positive integer d_1', contradicting $c \le \frac{a}{2} < a-b$. If the latter holds, $a^{k-1}(a-c) = d_2(a-b)$. Since a and $(a-b)$ are relatively prime, $a-c = d_2'(a-b)$, or $c = a - d_2'(a-b)$ for some positive integer d_2'. Since $\frac{a}{2} < a-b$, $d_2' = 1$, consequently, $b = c$, contradicting $c \ne b$. Thus $\mathcal{C}_{\frac{b}{a}} \not\to \mathcal{C}_{\frac{c}{a}}$ holds. ∎

3.2 Relations between biased coins and a fair n-sided die

Feldman *et al.* [FIN+93] investigated a fair n-sided die efficiently simulated by a PTM with a rationally biased coin. A fair n-sided die and a set $\{\mathcal{C}_{\frac{i}{n}} \mid 0 < i < n\}$ of coins are interchangeable for a random algorithm which randomly selects one of two choices with probability of denominator n. (Notice that n-sided die can be regarded as $\mathcal{C}_{\frac{i}{n}}$ for any i by properly relabeling each side.) A PTM with a rationally biased coin can efficiently simulate a fair n-sided die if and only if n is a power of 2 (and then the PTM has $\mathcal{C}_{\frac{1}{2}}$) as shown in [FIN+93]. On the other hand,

Theorem 3.1 also can be stated as follows: A PTM with a rationally biased coin can efficiently simulate $\mathcal{C}_{\frac{i}{n}}$ for every integer i with $0 < i < n$, if and only if n is a power of 2 or a power of 3. In this sense, a set $\{\mathcal{C}_{\frac{i}{n}} \mid 0 < i < n\}$ of coins is more useful than a fair n-sided die when n is a power of 3, since the PTM with $\mathcal{C}_{\frac{1}{3}}$ can efficiently simulate it.

Theorem 3.1 shows that a PTM with $\mathcal{C}_{\frac{b}{a}}$ in reduced form with $a > 3$ can only efficiently simulate the subset of the biases in finite figures on the a-ary system. Then, oppositely, how many least coins which efficiently simulate the set of the biases in finite figures on the a-ary system? By observing Lemma 3.5, it is likely to be equal to $\lfloor \frac{a}{2} \rfloor$, which is given by the set $\{\mathcal{C}_{\frac{1}{a}}, \mathcal{C}_{\frac{2}{a}}, \cdots, \mathcal{C}_{\frac{\lfloor a/2 \rfloor}{a}}\}$. (In the fact, Lemma 3.3 shows its sufficiency.) However, the answer is only *two*, which is $\mathcal{C}_{\frac{1}{a}}$ and $\mathcal{C}_{\frac{1}{2}}$. A PTM with $\mathcal{C}_{\frac{1}{a}}$ and $\mathcal{C}_{\frac{1}{2}}$ can efficiently simulate $\mathcal{C}_{\frac{1}{a^k}}$ for any k by tossing $\mathcal{C}_{\frac{1}{a}}$ k times. This is proved as follows: Theorem 1 in [FIN+93] showed that a fair coin and $\mathcal{C}_{\frac{1}{a^k}}$ can efficiently simulate a^k-sided die. As noticed above, a^k-sided die can be regarded as $\mathcal{C}_{\frac{i}{a^k}}$ for any i with $0 < i < a^k$.

Chapter 4

Complexity Classes Characterized by Semi-Random Sources

4.1 Probability of a worst tree

First, we show two crucial lemmas for a clustered tree.

Lemma 4.1 Let T be a clustered tree of depth d. Let $t_0, t_1, \cdots, t_k$ be the integers such that $|T| = \sum_{i=0}^{k} 2^{t_i}$ with $t_k > \cdots > t_1 > t_0 \geq 0$. Then, $\Pr[T|F_{\max}(T)] = \sum_{i=0}^{k} \delta^i (1-\delta)^{d-t_{k-i}-i}$.

Proof. From a δ-tree of depth d with no accepting leaf, we construct a δ-tree of depth d with $|T|$ accepting leaves using the following steps:

Step k: Let T_k be the rightmost subtree of depth t_k. Change all the labels of the leaves of T_k from "reject" to "accept".

Step i $(i = k-1, k-2, \cdots, 0)$: Let T'_{i+1} be the subtree such that its parent is also the parent of T_{i+1} defined in step $i+1$. Let T_i be the rightmost subtree of depth t_i of T'_{i+1}. (Note that this step works since $t_{i+1} > t_i$.) Change all the labels of the leaves of T_i from "reject" to "accept".

Since each T_i is always taken from rightmost side, we obtain a clustered tree with $|T|$ accepting leaves after the construction. Thus the constructed tree is the same tree as T. The path from the root of T to the root of T_i consists of the path from the root of T to the parent of T_k (whose $(d - t_k - 1)$ arcs are assigned $(1-\delta)$), an arc to the root of T'_k (which is assigned δ), the path to the parent of T_{k-1} (whose $(t_k - t_{k-1} - 1)$ arcs are assigned $(1-\delta)$), an arc to the root of T'_{k-1}, $\cdots$, and the path to the root of T_i. Thus, the probability of the root of T_i $(k \geq i \geq 0)$ is $\delta^i (1-\delta)^{d-t_{k-i}-i}$. Adding them together, we have the lemma. ∎

Lemma 4.2 Let l, r and d be any integers with $0 \leq l \leq r \leq 2^{d-1}$, and T be a clustered tree of depth d with $|T| = l + r$. Let T' be a δ-tree of depth d such that $|T'_l| = l$, $|T'_r| = r$, and both T'_l and T'_r are clustered trees. Then, $\Pr[T'|F_{\max}(T')] \geq \Pr[T|F_{\max}(T)]$.

Proof. We show the lemma by induction on the depth of the tree. Since it is clear when $d \leq 2$, we assume $d > 2$. Here, $\Pr[T'|F_{\max}(T')] = \delta^2 \Pr[T'_{ll}|F_{\max}(T')] + \delta(1-\delta)\Pr[T'_{lr}|F_{\max}(T')] + (1-\delta)\delta\Pr[T'_{rl}|F_{\max}(T')] + (1-\delta)^2 \Pr[T'_{rr}|F_{\max}(T')]$. Thus, the probability of T' does not change even if we exchange T'_{lr} and T'_{rl}. For T'_{ll} and T'_{rl}, four cases arise:
Case 1: $|T'_{ll}| > 0$ and $|T'_{rl}| = 0$. This case is impossible since $0 \leq l \leq r$.

Case 2: $|T'_{ll}| = |T'_{rl}| = 0$. Exchanging T'_{lr} and T'_{rl}, we find that only T'_r has accepting leaves with $|T'_{rl}| = l$ and $|T'_{rr}| = r$. Thus, by the inductive hypothesis, $\Pr[T'_r|F_{\max}(T'_r)] \geq \Pr[T_r|F_{\max}(T_r)]$. Since $|T_l| = |T'_l| = 0$, the lemma holds.
Case 3: $|T'_{ll}| > 0$ and $|T'_{rl}| > 0$. Since T'_l and T'_r are clustered trees, $|T'_{lr}| = |T'_{rr}| = 2^{d-2}$. Thus, exchanging T'_{lr} and T'_{rl}, we find that $|T'_r| = 2^{d-1}$. Moreover, $\Pr[T'_l|F_{\max}(T'_l)] \geq \Pr[T_l|F_{\max}(T_l)]$ by the inductive hypothesis. Since $|T_r| = |T'_r| = 2^{d-1}$, the lemma holds.
Case 4: $|T'_{ll}| = 0$ and $|T'_{rl}| > 0$. Here, T'_{llr}, T'_{lrl} and T'_{rll} are exchangeable each other, and so are T'_{lrr} and T'_{rlr}. For T'_{lrl} and T'_{rll}, four subcases arise:
Subcase 4a: $|T'_{lrl}| = |T'_{rll}| = 0$. The probabilities of the roots of T'_{lrr} and T'_{rll} are $\delta(1-\delta)^2$ and $\delta^2(1-\delta)$, respectively. Thus, since $|T'_{rll}| = 0$, even if we exchange T'_{lrr} and T'_{rll}, the probability of T' does not increase. Thus we have the lemma using the inductive hypothesis for T'_{rl}.
Subcase 4b: $|T'_{lrl}| > 0$ and $|T'_{rll}| = 0$. First, exchange T'_{lrr} and T'_{rlr}. Next, replace T'_{lr} by a clustered tree with $|T'_{lr}|$ accepting leaves. After replacing, if $|T'_{lrl}| = 0$, this case is reduced to Subcase 4a; otherwise, $|T'_{lrl}| > 0$, $|T'_{lrr}| = 2^{d-3}$, and $|T'_{rll}| = 0$. In the latter case, exchanging T'_{lrr} and T'_{rll} and further exchanging T'_{lrl} and T'_{lrr}, we see that T' is reduced to a clustered tree. Thus, it is sufficient to show that these exchanges do not increase the probability. Let $p = \Pr[T'_{lrl}|F_{\max}(T')]$ before the exchanges. Then the change of the probability by the exchanges is $-\delta(1-\delta)^2 + \delta^2(1-\delta) - \delta^2(1-\delta)p + \delta(1-\delta)^2 p = \delta(1-\delta)(1-2\delta)(p-1) < 0$, since $0 < \delta < \frac{1}{2}$ and $0 < p < 1$. This implies the lemma.
Subcase 4c: $|T'_{lrl}| = 0$ and $|T'_{rll}| > 0$. Since T'_{lr} and T'_{rl} are exchangeable, this case is reduced to Subcase 4b.
Subcase 4d: $|T'_{lrl}| > 0$ and $|T'_{rll}| > 0$. First, exchange T'_{llr} and T'_{rll}. Since $|T'_{ll}| = 0$ before the exchange, $|T'_{rll}| = 0$ after the exchange. Next, replace T'_l by a clustered tree with $|T'_l|$ accepting leaves. After replacing, if $|T'_{llr}| = 0$, this case is reduced to Subcase 4b; otherwise, $|T'_{llr}| > 0$ and $|T'_{lr}| = 2^{d-2}$. In the latter case, exchanging T'_{lrl} and T'_{rll} and further exchanging T'_{llr} and T'_{lrl}, we get a clustered tree without changing the probability. ∎

Now, we are ready to show the main theorem in this section.

Theorem 4.3 Every clustered tree is worst.

Proof. We show the theorem by induction on the depth of the tree. Since it is clear when $d \leq 1$, we assume that $d > 1$. Let T be a given clustered tree of depth d, and T' be any worst tree of depth d with $|T'| = |T|$. Since T' is worst, it is sufficient to show that $\Pr[T'|F_{\max}(T')] \geq \Pr[T|F_{\max}(T)]$. If either T'_l or T'_r is not worst, we can improve the probability of T' by replacing them. This contradicts that T' is worst. Thus, both T'_l and T'_r are worst. By the inductive hypothesis, we can replace T'_l and T'_r by clustered trees with $|T'_l|$ and $|T'_r|$ accepting leaves, respectively. Thus, by Lemma 4.2, $\Pr[T'|F_{\max}(T')] \geq \Pr[T|F_{\max}(T)]$. ∎

Corollary 4.4 Let T be a worst tree of depth d. Then, $\Pr[T|F_{\max}(T)] = \sum_{i=0}^{k} \delta^i (1-\delta)^{d-t_{k-i}-i}$, where $|T| = \sum_{i=0}^{k} 2^{t_i}$ with $t_k > \cdots > t_1 > t_0 \geq 0$.

4.2 The class δ-PP

Theorem 4.5 δ-PP = BPP for $0 < \delta < \frac{1}{2}$.

Proof. By Theorem 2.20, Definitions 2.13 and 2.14, BPP = δ-BPP $\subseteq$ δ-PP. Thus it is sufficient to show that δ-PP $\subseteq$ BPP. Let M_1 be a PTM with $L(M_1) \in \delta$-PP for some δ, and $p(n)$ be the depth of the computation path of M_1 on input of length n. Let m be a minimum positive integer such that $(1-\delta)(1-\delta^m) \geq \frac{1}{2}$. Since $\lim_{m\to\infty}(1-\delta)(1-\delta^m) = 1-\delta > \frac{1}{2}$, m must be a constant.

Without loss of generality, we assume that $p(n) > m+1$. Here we consider a worst tree T of depth $p(n)$ with $|T| \geq 2^{p(n)-1} - 2^{p(n)-m-1}$. Since $2^{p(n)-1} - 2^{p(n)-m-1} = \sum_{i=1}^{m} 2^{p(n)-i-1}$, using Corollary 4.4, we find that $\Pr[T|F_{\max}(T)] = (1-\delta)^2(\delta^0+\delta^1+\cdots+\delta^{m-1}) = (1-\delta)(1-\delta^m) \geq \frac{1}{2}$.

First, we consider an x_1 of length n with $x_1 \notin L(M_1)$. Since $x_1 \notin L(M_1)$, for its computation tree T_1 and every δ-assignment F to T_1, $\Pr[T_1|F] < \frac{1}{2}$. Consequently, $\Pr[T_1|F_{\max}(T_1)] < \frac{1}{2}$. Thus, Proposition 2.26 shows that $|T_1| < \frac{1}{2} \leq |T| = 2^{p(n)-1} - 2^{p(n)-m-1}$. Hence if M_1 computes with a perfect random source, $\Pr[M_1 \text{ accepts } x_1] < \frac{2^{p(n)-1}-2^{p(n)-m-1}}{2^{p(n)}} = \frac{1}{2} - \frac{1}{2^{m+1}}$. Since m is a constant, by repeating the algorithm enough times, the probability can be improved to $\frac{1}{4}$.

Second, we consider an x_2 of length n with $x_2 \in L(M_1)$. For the computation tree T_2 of x_2, we denote by $\bar{T}_2$ the computation tree constructed by replacing each accepting (rejecting) leaf by a rejecting (resp. accepting) leaf. Then, $\Pr[T_2|F] > \frac{1}{2}$ for every δ-assignment F to T_2, consequently, $\Pr[\bar{T}_2|F_{\max}(\bar{T}_2)] < \frac{1}{2}$. Thus, using the same argument above, we have $|\bar{T}_2| < 2^{p(n)-1} - 2^{p(n)-m-1}$, or $|T_2| > 2^{p(n)} - |\bar{T}_2| = 2^{p(n)-1} + 2^{p(n)-m-1}$. Hence if M_1 computes with a perfect random source, $\Pr[M_1 \text{ accepts } x_2] > \frac{1}{2} + \frac{1}{2^{m+1}}$. This completes the proof. ∎

4.3 The class $^{\exists}\delta$-RP

Theorem 4.6 $^{\exists}\delta\text{-RP} = \mathsf{NP}$ for $0 < \delta < \frac{1}{2}$.

Proof. From Proposition 2.18, it is sufficient to show that $\mathsf{NP} \subseteq {}^{\exists}\delta\text{-RP}$. Let M_2 be an NTM with $L(M_2) \in \mathsf{NP}$, and $p(n)$ be the length of M_2's computation on input of length n. Let $q(n) = \lceil -\frac{\log(8(p(n)+1))}{\log(2\sqrt{\delta(1-\delta)})} \rceil$. Since $\log(2\sqrt{\delta(1-\delta)}) < 0$ for $0 < \delta < \frac{1}{2}$, $q(n) > 0$. We construct a PTM M_2' which simulates M_2, with a specific δ-random source. M_2' simulates M_2 straightforwardly if M_2 is not in a nondeterministic state. Otherwise, M_2' simulates M_2 as follows: When M_2 nondeterministically selects 0-choice (resp. 1-choice), M_2' assigns $(1-\delta)$ to the probability that the outcome of a coin tossing is 0 (resp. 1); tosses a coin $q(n)$ times; and selects i-choice, where i is the majority of the outcomes.

It is clear that M_2' simulates M_2 in time polynomial in n, and M_2' rejects x for $x \notin L$. We consider the probability that M_2' accepts x for $x \in L$. M_2' fails to simulate one nondeterministic choice of M_2 with probability equal to $\sum_{i=0}^{\frac{1}{2}q(n)} \binom{q(n)}{i} \delta^{q(n)-i}(1-\delta)^i$. Since $0 < \frac{1}{2} < (1-\delta) < 1$, we have $\sum_{i=0}^{\frac{1}{2}q(n)} \binom{q(n)}{i} \delta^{q(n)-i}(1-\delta)^i \leq 2^{q(n)\log(2\sqrt{\delta(1-\delta)})} \leq 2^{-\log(8(p(n)+1))} = \frac{1}{8(p(n)+1)}$ using the lemma due to Luts [Lut90] that can be proved using Chernoff bounds [Che52]. Thus the probability that M_2' successes in simulating at most $p(n)$ nondeterministic choices of M_2 is greater than or equal to $\left(1 - \sum_{i=0}^{\frac{1}{2}q(n)} \binom{q(n)}{i} \delta^{q(n)-i}(1-\delta)^i\right)^{p(n)} \geq \left(1 - \frac{1}{8(p(n)+1)}\right)^{p(n)}$. Here, $e^{-p} < \left(1 - \frac{p}{n+1}\right)^n$ holds for $0 < p < 1$ and any positive integer n. Thus, M_2' accepts x for $x \in L$ with probability greater than or equal to $\left(1 - \frac{1}{8(p(n)+1)}\right)^{p(n)} > e^{-\frac{1}{8}} > \frac{1}{2}$. Consequently, $L \in {}^{\exists}\delta\text{-RP}$. ∎

Intuitively, Theorem 4.6 shows that the PTM using polynomial number of semi-random bits generated by a specific δ-random source can simulate polynomial number of nondeterministic choices with high probability.

4.4 The classes $^{\exists}\delta$-BPP and $^{\exists}\delta$-PP

Theorem 4.7 $^{\exists}\delta\text{-BPP} = {}^{\exists}\delta\text{-PP} = \mathsf{PSPACE}$ for $0 < \delta < \frac{1}{2}$.

Proof. It is sufficient to show that $\mathsf{ABPP} \subseteq {}^{\exists}\delta\text{-}\mathsf{BPP}$ for $0 < \delta < \frac{1}{2}$ from Theorem 2.9 together with the fact that ${}^{\exists}\delta\text{-}\mathsf{BPP} \subseteq {}^{\exists}\delta\text{-}\mathsf{PP} \subseteq \mathsf{PSPACE}$. (We remark that ${}^{\exists}\delta\text{-}\mathsf{PP} \subseteq \mathsf{PSPACE}$ is easy to show since we assume that δ is a *rational* number. We will consider the case with an *irrational* bias in subsection 4.6.2.) Let M_3 be a STM such that $L(M_3) \in \mathsf{ABPP}$, and $2p(n)$ be the length of the computation of M_3 on input x of length n. Without loss of generality, we can assume that M_3 accepts with probability greater than $\frac{63}{64}$ for $x \in L$, and less than $\frac{1}{64}$ for $x \notin L$. On the computation of M_3, a *state pair* contains a nondeterministic state and the next two probabilistic states. That is, a computation of M_3 contains $p(n)$ state pairs.

Here, the PTM M_2' constructed in the proof of Theorem 4.6 simulates nondeterministic choices with a specific δ-random source. On the other hand, the PTM N in the proof of [Pap94, Theorem 11.4] simulates probabilistic choices with an arbitrary δ-random source. Combining M_2' and N, we construct a PTM M_3' which simulates M_3 with a specific δ-random source. Let $q(n) = \lceil -\frac{\log(8(p(n)+1))}{\log(2\sqrt{\delta(1-\delta)})} \rceil$, and $r(n) = \lceil \frac{3\log p(n)+6}{2\delta-2\delta^2} \rceil$. A sequence of $r(n)$ bits will be called *block*. The $2^{r(n)}$ possible blocks are denoted by the corresponding binary integers $0, 1, \cdots, 2^{r(n)}-1$. If $\kappa = (\kappa_1, \kappa_2, \cdots, \kappa_{r(n)})$ and $\lambda = (\lambda_1, \lambda_2, \cdots, \lambda_{r(n)})$ are blocks, then their *inner product* is defined $\kappa \cdot \lambda = \sum_{i=1}^{r(n)} \kappa_i \lambda_i \pmod 2$. Notice that the inner product of two blocks is a bit. M_3' simulates $2^{r(n)}$ M_3s in parallel to simulate probabilistic choices. The jth state pair of ith simulation of an M_3 is performed by the following:

(Simulation of a nondeterministic choice:)

(1) when M_3 nondeterministically selects m-choice ($m = 0$ or 1), assign $(1-\delta)$ to the probability that the outcome of a coin tossing is m; and
(2) select k-choice, where k is the majority of the outcomes of $q(n)$ coin tossing.

(Simulation of one of two probabilistic choices:)

(3) generate β_j with $r(n)$ bits generated by a δ-random source; and
(4) select $h_{(i,j)}$-choice, where $h_{(i,j)} = \beta_j \cdot i$;

At the end of the simulation, M_3' accepts if at least half of the $2^{r(n)}$ simulations accepts, or rejects otherwise.

When $x \in L$, as in the proof of Theorem 4.6, M_3' successes in simulating $p(n)$ nondeterministic choices with probability greater than $e^{-1/8}$. In the case, using the technique in the proof of Theorem 11.4 in [Pap94], we can show that the majority of M_3s rejects with probability less than $\frac{1}{8}$. Thus M_3' accepts with probability greater than $e^{-1/8} \cdot \frac{7}{8} > \frac{3}{4}$. On the other hand, when $x \notin L$, using the technique in the proof of Theorem 11.4 in [Pap94], we can show that the majority of M_3s accepts with probability less than $\frac{1}{8}$ for every nondeterministic choices. Consequently, M_3' rejects with probability greater than $\frac{3}{4}$. Thus $L \in {}^{\exists}\delta\text{-}\mathsf{BPP}$. ∎

In the proof, notice that we can't replace a simulation of a probabilistic choice by a fair coin tossing. Although fair coin tossing works for $x \in L$, it does not work for $x \notin L$, since a probabilistic choice must be simulated by *every* δ-random source.

We summarize the results in sections 4.2 through 4.4 in Figure 4.1.

4.5 The class ARP

In the previous section, we have indirectly shown that ${}^{\exists}\delta\text{-}\mathsf{BPP} \subseteq \mathsf{PSPACE}(= \mathsf{ABPP})$. Moreover, in our model, it is sufficient to consider the δ-random sources that only assign one of the probabilities δ and $1-\delta$ from Proposition 2.24. Thus, we can directly show that ${}^{\exists}\delta\text{-}\mathsf{BPP} \subseteq \mathsf{ABPP}$ and ${}^{\exists}\delta\text{-}\mathsf{PP} \subseteq \mathsf{APP}$ using the gadget in Figure 4.2. Although the gadget in the figure only works for the δ-random sources with $\delta = \frac{1}{4}$, Knuth and Yao showed that for an arbitrarily rational

$$
\begin{array}{ccccccccccc}
\mathsf{P} & = & 0\text{-}\mathsf{RP} & = & 0\text{-}\mathsf{BPP} & = & 0\text{-}\mathsf{PP} & & \\
 & & \bigcap\vert & & \bigcap\vert & & \bigcap\vert & & \\
 & & \delta\text{-}\mathsf{RP} & \subseteq & \delta\text{-}\mathsf{BPP} & = & \delta\text{-}\mathsf{PP} & & \\
 & & \| & & \| & & \bigcap\vert & & \\
 & & \frac{1}{2}\text{-}\mathsf{RP} & & \frac{1}{2}\text{-}\mathsf{BPP} & & \frac{1}{2}\text{-}\mathsf{PP} & & \\
 & & \| & & \| & & \| & & \\
 & & \mathsf{RP} & \subseteq & \mathsf{BPP} & \subseteq & \mathsf{PP} & & \\
 & & \| & & \| & & \| & & \\
 & & {}^{\exists}\frac{1}{2}\text{-}\mathsf{RP} & & {}^{\exists}\frac{1}{2}\text{-}\mathsf{BPP} & & {}^{\exists}\frac{1}{2}\text{-}\mathsf{PP} & & \\
 & & \bigcap\vert & & & & & & \\
 & & {}^{\exists}\delta\text{-}\mathsf{RP} & \subseteq & {}^{\exists}\delta\text{-}\mathsf{BPP} & = & {}^{\exists}\delta\text{-}\mathsf{PP} & = & \mathsf{PSPACE} \\
 & & \| & & & & & & \\
\mathsf{NP} & = & {}^{\exists}0\text{-}\mathsf{RP} & = & {}^{\exists}0\text{-}\mathsf{BPP} & = & {}^{\exists}0\text{-}\mathsf{PP} & &
\end{array}
$$

Figure 4.1: Summary of the results ($0 < \delta < \frac{1}{2}$)

number δ, we can construct such a gadget [KY76]. From this observation, we can show the following theorem.

Theorem 4.8 $\mathsf{ARP} = \mathsf{NP}$.

Proof. It is easy to see that $\mathsf{ARP} \subseteq \mathsf{NP}$ holds. Using the gadget shown in Figure 4.2 together with Theorem 4.6, we can show that $\mathsf{NP} = {}^{\exists}\frac{1}{4}\text{-}\mathsf{RP} \subseteq \mathsf{ARP}$. This establishes the theorem. ∎

4.6 Remarks

4.6.1 Characterization of a worst tree

The structure of a clustered tree plays an important role as the structure of a worst tree in the sections 4.1 and 4.2. Then, do clustered trees completely characterize worst trees? The answer is no.

Definition 4.9 Let T be a δ-tree of depth $d \geq 0$. T is a *scattered tree* if

(1) $d = 0$, or
(2) $0 \leq |T_r| - |T_l| \leq 1$, and both T_r and T_l are scattered trees.

Intuitively, a scattered tree is a tree whose all subtrees of the same depth have the same number of accepting leaves. Although the structure of a scattered tree is opposite to the structure of a clustered tree, the following corollary holds.

Corollary 4.10 Every scattered tree is worst.

Proof. By Corollary 4.4, it is sufficient to show that for any scattered tree T of depth d, $\Pr[T|F_{\max}(T)] = \sum_{i=0}^{k} \delta^i (1-\delta)^{d-t_{k-i}-i}$, where $t_0, t_1, \cdots, t_k$ are the integers such that $|T| = \sum_{i=0}^{k} 2^{t_i}$ with $t_k > \cdots > t_1 > t_0 \geq 0$. An induction on the depth of the tree shows the fact. ∎

Moreover, combining the definitions of clustered and scattered trees, we can define a *chimera tree* as follows:

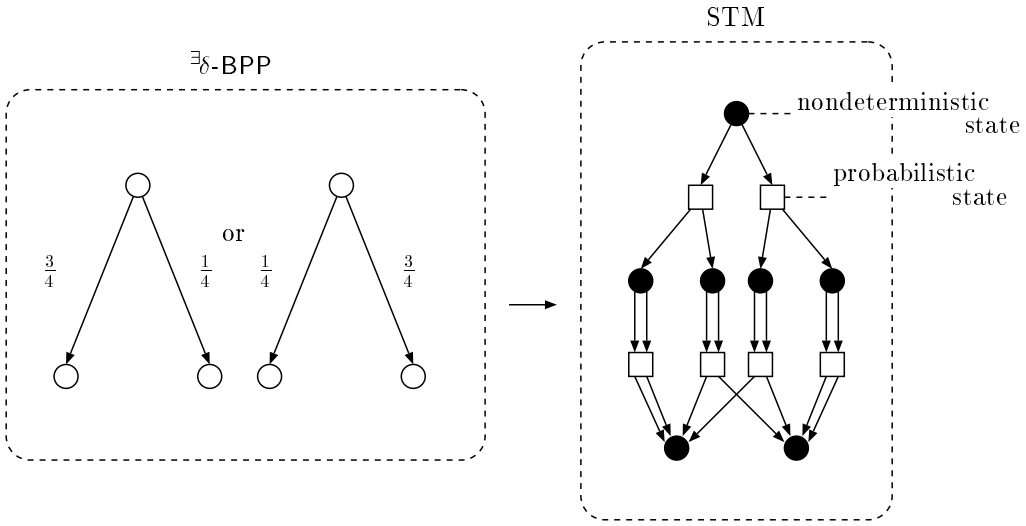


Figure 4.2: gadget for δ-random sources with $\delta = \frac{1}{4}$

Definition 4.11 Let T be a δ-tree of depth $d \geq 0$. Then T is a *chimera tree* if
(1) $d = 0$,
(2) $|T_l| = 0$, and T_r is a chimera tree,
(3) T_l is a chimera tree and $|T_r| = 2^{d-1}$, or
(4) $0 \leq |T_r| - |T_l| \leq 1$ and both T_l and T_r are chimera trees.

By a simple induction on the depth of the tree together with Theorem 4.3, Corollaries 4.4 and 4.10, it is not hard to see that a chimera tree is worst also. Moreover, as shown in the proof of Lemma 4.2, a δ-tree has many subtrees that are exchangeable each other. Exchanging these subtrees of a chimera tree, we get many variations of a worst tree. Although the author conjectures that every worst tree is a chimera tree or its variation, the complete characterization of a worst tree remains an open problem.

4.6.2 Complexity classes with irrationally δ-random sources

In the sections above, we only consider a δ-random source with rational δ. All results also can be shown for a δ-random source with irrational δ except Theorem 4.7. That is, $^{\exists}\delta$-PP $\subseteq$ PSPACE is not obvious when δ is an *irrational* number. How can we simulate a PTM with an irrational δ-random source by a DTM with polynomial space? In this subsection, we solve this problem. We first show the following lemma to deal with an *arbitrary* δ.

Lemma 4.12 Let L be a language with $L \in {}^{\exists}\delta$-PP for some δ. Then there exists a number δ' such that $L \in {}^{\exists}\delta'$-PP and δ' can be represented in polynomial space for the input length.

Proof. Let L be a language with $L \in {}^{\exists}\delta$-PP for some δ, and M_4 be a PTM such that $L(M_4) = L$. Let p be the length of a computation of M_4. (We write only p, depending on the input length, for short.) Let $d = \frac{\delta^p}{2^{p+1}p(1-\delta)^{p-1}}$. Since d can be represented in polynomial space for the input length, there exists some δ' with $|\delta' - \delta| < d$ such that δ' can be represented in polynomial space for the input length. Without loss of generality, we assume that $\delta' > \delta$. We simulate

M_4 in polynomial space by substituting δ' for δ. Then the error of the probability of a leaf by substituting is equal to $\delta^i(1-\delta)^{p-i} - \delta'^i(1-\delta')^{p-i}$ for some i with $0 \leq i \leq p$. If the sum of every error of the probability of the leaf is less than half of the probability of any leaf, $L(M_4)$ is also in ${}^{\exists}\delta'$-PP. Here, the minimum probability of a leaf is equal to δ^p. On the other hand, an error of the probability of a leaf is at most $\max\{\delta'^p - \delta^p, (1-\delta)^p - (1-\delta')^p\} = (1-\delta)^p - (1-\delta')^p$ since $\delta' > \delta$. Thus, since M_4 has 2^p leaves, the sum of every error is at most

$$2^p \ |\delta^i(1-\delta)^{p-i} - \delta'^i(1-\delta')^{p-i}| < 2^p \left((1-\delta)^p - (1-\delta')^p\right).$$

Using Taylor series, we get

$$2^p \left((1-\delta)^p - (1-\delta')^p\right) < 2^p pd(1-\delta)^{p-1} = \frac{\delta^p}{2}.$$

Hence, the sum of every error of the probability of any computation tree is less than a half of the probability of any leaf. This implies the lemma. ∎

Lemma 4.13 For an *arbitrary* δ with $0 < \delta < \frac{1}{2}$, ${}^{\exists}\delta$-PP $\subseteq$ PSPACE.

Proof. Let L be a language with $L \in {}^{\exists}\delta$-PP for some δ. Let M_5 be a PTM, such that $L = L(M_5)$. Let δ' be a number satisfying the conditions in the proof of Lemma 4.12. We construct an NTM M_5', which accepts L as follows;

1: nondeterministically compute δ';
2: simulate all computations of M_5, and counts up its probability by using δ' instead of δ; and
3: accept if the probability is greater than $\frac{1}{2}$; or reject otherwise.

Clearly, M_5' uses at most polynomial space for the input length, and $L = L(M_5')$. Thus $L \in$ PSPACE. ∎

Chapter 5

Optimal Attribute-Efficient Learning of Disjunction, Parity, and Threshold Functions

5.1 Basic results

Since the queries have two possible answers, $\lceil \log |F| \rceil$ queries are necessary for the class F. For the classes OR and PAR this leads to the lower bound n which is optimal, since these classes can be learned by asking $E_1, E_2, \ldots, E_n$. Theorem 5.1 contains some other information theoretic lower bounds.

Theorem 5.1 The following lower bounds hold:

(i) $\lceil \log \binom{n}{k} \rceil$ for $\mathsf{OR}(k)$, $\mathsf{PAR}(k)$, and $\mathsf{THR}_t(k)$, if $k \geq t > 0$. This bound is larger than $k \log \frac{n}{k}$ and $(n-k) \log \left(\frac{n}{n-k} \right)$.

(ii) $\lceil \log \left(\binom{n}{k} \cdot k + 2 \right) \rceil$ for $\mathsf{THR}(k)$.

(iii) $n - 1 + \lceil \log(n+1) \rceil$ for THR.

Proof. The classes $\mathsf{OR}(k)$, $\mathsf{PAR}(k)$, and $\mathsf{THR}_t(k)$ contain $\binom{n}{k}$ functions. The estimations of $\log \binom{n}{k}$ are standard calculations. For $\mathsf{THR}(k)$ we have the possible threshold values $t \in \{1, \ldots, k\}$ each leading to $\binom{n}{k}$ functions. For different t we obtain different functions. Moreover, we obtain for $t = 0$ the constant function 1 and for $t > k$ the constant function 0. Thus for the class THR we have

$$\sum_{1 \leq k \leq n} k \binom{n}{k} + 2$$

different functions. The number of different functions equals $n2^{n-1} + 2$ and $\lceil \log(n2^{n-1} + 2) \rceil = \left\lceil \log 2^{n-1} \left(n + 2^{-(n-2)} \right) \right\rceil = n - 1 + \lceil \log(n+1) \rceil$. ∎

5.2 Disjunctions

We present two learning algorithms for the class $\mathsf{OR}(k)$. The second one needs less queries but the first one does not use the knowledge of k. In Section 5.4 we investigate the learning of

$\mathsf{THR}(k)$. Since the threshold value t is unknown, we obtain the subproblem of learning $\mathsf{OR}(l)$ and $\mathsf{AND}(m)$ for unknown l and m. Then we apply the algorithm presented in the proof of Theorem 5.2.

Theorem 5.2 There exists a learning algorithm which learns $\mathsf{OR}(k)$ with $2k\,(\log \frac{n}{k} + 2)$ queries without knowing k (if $k = 0$, the algorithm needs one query).

Proof. We start with the query $\{1, \ldots, n\}$. Each winner (a query with answer 1) is split into two sets of almost the same size and these sets are used as query sets. Without knowing k the algorithm does not produce more than k winners in each phase. The number of queries in the i-th phase can be bounded by $\min\{2^{i-1}, 2k\}$ and after $\lceil \log n \rceil + 1$ phases all query sets have size 1. Hence, the total number of queries is bounded by

$$\begin{aligned} 2^0 + \cdots + 2^{\lceil \log k \rceil} + (\lceil \log n \rceil - \lceil \log k \rceil)2k &\leq 2 \cdot 2^{\lceil \log k \rceil} - 2k\lceil \log k \rceil + 2k(\log n + 1) \\ &\leq 2k - 2k \log k + 2k(\log n + 1) \\ &= 2k \left(\log \frac{n}{k} + 2 \right). \end{aligned}$$

∎

Theorem 5.3 The class $\mathsf{OR}(k)$ can be learned with at most $\min\{n - 1, k\lceil \log \frac{n}{k} \rceil + 2k - 2\}$ queries.

Proof. The upper bound $n-1$ follows from the strategy to ask $E_1, ..., E_{n-1}$. For the other upper bound we partition the set of n elements to k sets of size at most $\lceil \frac{n}{k} \rceil$. We ask these k sets as queries. We assume that a set S is a winner. Then, we can find an essential element x by binary search with at most $\lceil \log \lceil \frac{n}{k} \rceil \rceil = \lceil \log \frac{n}{k} \rceil$ queries. Then we test whether $S - \{x\}$ is still a winner and continue in this way. For each of the k essential elements the binary search can be done with at most $\lceil \log \frac{n}{k} \rceil$ queries. Moreover, we ask k queries in the beginning. For each essential element x we have found we ask an additional query (the query $S - \{x\}$ above). We can save this additional query for the last two essential elements. After having found the last essential element we are done. If we have found the last but one essential element, we know whether there is another set which is a winner. Thus, we can determine whether the corresponding set $S - \{x\}$ is a winner without making a query. ∎

The second upper bound beats the trivial upper bound $n - 1$ only if $k = n/4$ or $k \leq n/5$. If $k = o(n)$, the upper bound in Theorem 5.3 is $k \log \frac{n}{k} + o(k \log \frac{n}{k})$ while the lower bound is larger than $k \log \frac{n}{k}$. Hence, our algorithm is almost optimal. Moreover, the upper bound of $\mathsf{OR}(k)$ given by the algorithm in Theorem 5.3 is asymptotically optimal if $0 \leq k < \frac{n}{2}$. However, the lower bound given in Theorem 5.1 is much smaller than the upper bound for large k. We prove a better lower bound, and show that the upper bound of $\mathsf{OR}(k)$ given by the algorithm in Theorem 5.3 is optimal if $\frac{n}{2} \leq k \leq n - 1$.

Theorem 5.4 If $\frac{n}{2} \leq k \leq n - 1$, $n - 1$ queries are necessary for $\mathsf{OR}(k)$.

Proof. The claim is proved with an adversary argument. An adversary answers the queries of the learner consistently and tries to force the learner to ask many queries. It is our aim to describe an adversary which can force the learner to ask $n - 1$ queries. In order to simplify the analysis the adversary gives sometimes more information to the learner. If the learner nevertheless has to ask $n - 1$ queries, we are done.

First we consider the class $\mathsf{OR}(n-1)$. Query sets of size larger than 1 are useless, since the answer 1 is known in advance. Hence, only E_is are used as queries and the adversary may answer the first $n-2$ queries by 1. Then the learner still has to ask a query.

We restrict ourselves w. l. o. g. to strategies which do not include identified elements (elements which are known to be essential or nonessential) into further query sets. The adversary strategy is the following. If $k = n-1$, we follow the strategy described above. Otherwise, the adversary does the following. As long as the query sets are disjoint and at most $n-k-1$ query sets of size 1 are asked, the adversary answers 0 for the query sets of size 1 and answers 1 for larger query sets. The answers 1 are allowed, since $k \geq \frac{n}{2}$. There are two cases to consider. In the first case the learner has asked $n-k-1$ query sets of size 1. Then the learner has obtained the knowledge of $n-k-1$ nonessential elements by $n-k-1$ queries. The answers to these queries do not include more information. Hence, the learner is faced with the problem $\mathsf{OR}(k)$ on $k+1$ variables. As we have shown, the learner needs k additional queries for this purpose and we are done.

In the second case the learner asks a query A which is not disjoint from a previous query B. By our assumptions $|B| \geq 2$. The adversary answers the query by 1. Furthermore, the adversary gives the following information for free. He chooses an element $i \in A \cap B$ and an element $j \in B - \{i\}$. Then he tells the learner that i is essential and j is nonessential. This is consistent with the previous answers. Now the answers to the queries A and B do not contain further information. The learner has obtained the knowledge of one essential element and one nonessential element by 2 queries. Hence, the learner is faced with the problem $\mathsf{OR}(k-1)$ on $n-2$ variables. For this problem we prove with the same strategy that $n-3$ queries are necessary. ∎

5.3 Parity functions

For small k a small number of queries is sufficient to learn $\mathsf{OR}(k)$ while the maximum number of $n-1$ queries is necessary to learn $\mathsf{OR}(n-k)$. For parity functions we prove that the complexity of learning $\mathsf{PAR}(k)$ is equal to the complexity of learning $\mathsf{PAR}(n-k)$. What is the reason for this difference?

Parity (or mod 2 denoted by $\oplus$) is like disjunction a binary, associative and commutative operation with 0 as neutral element. But $(\{0,1\}, \oplus)$ is an additive group while the equation $1 \vee x = 0$ has no solution.

Proposition 5.5 For given n, the complexity of learning $\mathsf{PAR}(k)$ is equal to the complexity of learning $\mathsf{PAR}(n-k)$.

Proof. Let g_1 be the parity function on k of the n variables and g_2 be the parity function on the $n-k$ other variables. This defines a one-to-one mapping between the classes $\mathsf{PAR}(k)$ and $\mathsf{PAR}(n-k)$. For a query set S let $a_1(S)$ and $a_2(S)$ be the answers given for g_1 and g_2 resp. Then $a_1(S) \oplus a_2(S) \equiv |S| \mod 2$. Hence, we can compute $a_1(S)$ from $a_2(S)$ and vice versa. A strategy for $\mathsf{PAR}(k)$ which identifies g_1 identifies with the corresponding answers g_2, if it is used for $\mathsf{PAR}(n-k)$. ∎

Because of Proposition 5.5 we assume without loss of generality that $k \leq \frac{n}{2}$. For the OR-classes, one essential variable can be identified with binary search using $\lceil \log n \rceil$ queries. For the PAR-classes the same holds, if the number of essential elements is odd. If we partition the set of elements into two parts, one has an odd number of essential elements and the other one an even number. With one query we can find out which is the "odd part" and can continue

with this set. After having found one essential element, we have an even number of unknown essential elements and are in the difficult situation. We cannot determine whether a subset contains essential elements when the answer to the query is zero. If we find a set S with an odd number of unknown essential elements, then the complement of S with respect to the unidentified elements also contains an odd number of essential elements. Then we can apply the binary search technique to both sets. We get good upper bounds (see Theorem 5.1) for $k \in \{1, 2, 3\}$.

Proposition 5.6 (i) $\mathsf{PAR}(1)$ can be learned with $\lceil \log n \rceil$ queries.

(ii) $\mathsf{PAR}(2)$ can be learned with $3 \lceil \log n \rceil - 2$ queries.

(iii) $\mathsf{PAR}(3)$ can be learned with $4 \lceil \log n \rceil - 3$ queries.

Proof. The first result follows from the binary search technique. For the second claim we use a so-called separating system $S_0, \ldots, S_{\lceil \log n \rceil - 1}$. That means, for $i \neq j$ there is a set S_m where $i \in A_m$ and $j \notin A_m$ or vice versa. For this purpose let A_m consist of the numbers j whose m-th bit in the binary representation equals 1. First we query the sets of the separating system. If i and j are the essential elements, the corresponding set A_m leads to the answer 1. Then we may start binary searches on A_m and its complement, sets of at most $\lceil \frac{n}{2} \rceil$ elements. The third result is obvious; we find the first essential element with binary search, and apply the second result (even for a set of $n - 1$ elements). We save one query, since the first query can be chosen as a set belonging to the separating system in the second round. ∎

How can we find for larger k a set with an odd number of essential objects? For this purpose we may use (n, k, r, α)-splitters defined as follows (see Naor, Schulman, and Srinivasan [NSS95]):

Definition 5.7 An (n, k, r, α)-*splitter* of size s is a sequence $P_1, \ldots, P_s$ of partitions of $\{1, \ldots, n\}$ into r sets such that for each set S of cardinality k at least a fraction of $\alpha > 0$ partitions put the elements of S into r different sets.

Lemma 5.8 An (n, k, r, α)-splitter of size s implies that $\mathsf{PAR}(k)$ can be learned with $rs + k \lceil \log n \rceil$ queries.

Proof. With rs queries of the sets in $P_1, \ldots, P_s$ we find one partition where k subsets give the answer 1. With binary search on these sets we find the k essential elements. ∎

We review the applications of splitters in learning theory. Friedman constructs splitters for constant k leading to $O(\log n)$ learning algorithms for $\mathsf{PAR}(k)$ and constant k [Fri84]. Ragde and Widgerson [RW91] (based on [Meh82]) and Håstad, Wegener, Wurm, and Yi [HWWY94] use splitters for the construction of Boolean circuits. The deterministic construction of splitters by Bshouty and Hellerstein [BH96] is a rediscovery of the construction in [HWWY94]. These splitters lead by Lemma 5.8 to the existence of learning algorithms for $\mathsf{PAR}(k)$ with $O(k^3 \log n)$ queries and the efficient construction of learning algorithms with $O(k^4 \log n)$ queries [Hof96].

Using special properties of the parity function we improve this upper bound. We concentrate our discussion on the minimal number of necessary queries and not on the efficient construction of the learning algorithms.

For the class of parity functions we may find the essential elements sequentially. We do not have to partition the essential elements in classes of size 1. For the identification of one essential element it is sufficient to find a set with an odd number of essential elements.

First we consider randomized learning algorithms with zero-error. Either we can guarantee that the algorithm always gives the right answer and that the expected number of queries is small or we guarantee an upper bound on the number of queries and a small upper bound on the probability that the algorithm answers "don't know". The reader not familiar with randomized algorithms may look at [MR95] for an introduction. The use of randomized algorithms in the context of query complexity has been proved in [SW86] and [HW91] for randomized decision trees.

Obviously, one half of the queries leads to the answer 1. Asking randomized queries we get in expected time 2 a query set with answer 1. Then, we find two essential elements with binary search. This leads to an error-free randomized algorithm with an expected number of $k\,(\lceil \log n \rceil + 1)$ queries. In order to meet the lower bound we should decrease the size of the query sets.

Theorem 5.9 For $\mathsf{PAR}(k)$ there is a randomized error-free learning algorithm with an expected number of $k \log \frac{n}{k} + O(k)$ queries. If the number of queries is bounded by $O(k \log \frac{n}{k})$, the probability of not having identified the unknown function can be bounded by a function tending with respect to $k \log \frac{n}{k}$ exponentially fast to 0.

Proof. The randomized algorithm repeats the following steps until all essential elements have be found: Without loss of generality, we assume that the algorithm has k' unidentified essential elements in a set of size n' with $0 < k' < n'$. First, the algorithm chooses at random and independently r elements, where $r = \left\lceil \frac{n'}{k'} \right\rceil$. The choices are made with replacement, i.e., the algorithm may choose an element more than once. Next the algorithm collects the chosen elements into a query set (of size at most r), and makes a query. Then, the query set contains exactly one essential element with probability at least

$$\binom{r}{1} \frac{k'}{n'} \left(1 - \frac{k'}{n'}\right)^{r-1} = \left\lceil \frac{n'}{k'} \right\rceil \frac{k'}{n'} \left(1 - \frac{k'}{n'}\right)^{\left\lceil \frac{n'}{k'} \right\rceil - 1} \geq \left(1 - \frac{k'}{n'}\right)^{\frac{n'}{k'}} > \frac{1}{e},$$

where e is the Eulerian constant $2.718\cdots$. Hence, the probability of success to find a set containing an odd number of essential elements is bounded below by $\frac{1}{3}$. For a successful query we perform the binary search only on the small query set and not on the large complement. For the binary searches we are done in any case with at most

$$\begin{aligned} \sum_{1 \leq m \leq k} \left\lceil \log \frac{n-k+m}{m} \right\rceil &\leq \sum_{1 \leq m \leq k} \log \frac{n}{m} + O(k) = k \log n - \log(k!) + O(k) \\ &= k \log n - (k \log k - O(k)) + O(k) = k \log \frac{n}{k} + O(k) \end{aligned}$$

queries. The expected number of queries to find the next set with an odd number of essential elements is bounded by 3 leading to an additional expected number of $3k$ queries.

For the second claim we get smaller upper bounds than for general error-free randomized algorithms. Only a small part of our algorithm is randomized. Furthermore, we have independent trials each with a probability of success at least $\frac{1}{3}$. We need k successes and may perform $O(k \log \frac{n}{k})$ trials. Now the probability of less then k successes can be estimated with Chernoff's bounds. ∎

Now we look for deterministic learning algorithms. The main idea is to "recycle" query sets, i. e. to try to use them more than once. A random query set S contains approximately half of the essential elements. The set is good, if it contains an odd number of essential elements. If

the number of essential elements in S is even, we try other sets. If we find the first essential element i contained in S, we know without any further query that $S - \{i\}$ contains an odd number of essential elements. These ideas are made precise in the next proposition and lead to a derandomization of the algorithm.

Proposition 5.10 The class $\mathsf{PAR}(k)$ can be learned with $\lceil \log \binom{n}{k} \rceil + k \lceil \log n \rceil + 2$ queries.

Proof. If $k = \Omega(n)$, the result is obvious, since $n-1$ queries of E_is suffice. If $k \leq \frac{n}{3}$,

$$\binom{n}{k-1} \leq \frac{1}{2}\binom{n}{k}$$

and

$$\binom{n}{k} + \binom{n}{k-1} + \binom{n}{k-2} + \cdots + \binom{n}{1} \leq 2\binom{n}{k}.$$

We choose $m = \lceil \log \binom{n}{k} \rceil + 2$ random sets $R_1, \ldots, R_m$. Let $S \subseteq \{1, \ldots, n\}$ be one set where $1 \leq |S| \leq k$. The probability that $S \cap R_j$ has odd cardinality equals $\frac{1}{2}$.

The probability that all random sets are bad for S, i. e. $S \cap R_1$, ..., $S \cap R_m$ all have even cardinality, equals $\frac{1}{4}\binom{n}{k}^{-1}$. Therefore, the probability that for at least one of the less than $2\binom{n}{k}$ sets S all random sets are bad is bounded above by $\frac{1}{2}$. Hence, there exist m query sets $T_1, \ldots, T_m$ such that for each $S \subseteq \{1, \ldots, n\}$ with $1 \leq |S| \leq k$ there exists a set T_j such that $|S \cap T_j|$ is odd.

The sets $T_1, \ldots, T_m$ are used as query sets. Let S be the unknown set of essential elements. Then $|S \cap T_{i(1)}|$ is odd for some $i(1)$ and we find an essential element $j(1)$ with binary search on $T_{i(1)}$. Let $S_1 := S - \{j(1)\}$. Then $|S_1 \cap T_{i(2)}|$ is odd for some $i(2)$ and we find an essential element $j(2)$ with binary search on $T_{i(2)} - \{j(1)\}$ and so on. Here we have used the fact that we have implicitly knowledge about the query set $T_i - \{j(1)\}$ if we have asked the query T_i and know that $j(1)$ is essential.

Altogether we have asked m queries $T_1, \ldots, T_m$ and need $k \lceil \log n \rceil$ further queries for the binary searches. ∎

The term $\lceil \log \binom{n}{k} \rceil$ is equal to the information theoretic lower bound. In order to obtain an asymptotically optimal algorithm the term $k \log n$ should be replaced by $O\left(k \log \frac{n}{k}\right)$. If $k = O\left(n^{1-\varepsilon}\right)$, nothing has to be done. Hence, we have to consider functions like $k = \frac{n}{\log^2 n}$ or $k = \frac{n}{\log \log n}$.

Theorem 5.11 The class $\mathsf{PAR}(k)$ can be learned with $O\left(k \log \frac{n}{k}\right)$ queries.

Proof. If $k \leq n/2$ and $k = \Omega(n)$, the proposed upper bound is $O(n)$ and we are done by using the trivial strategy asking $E_1, \ldots, E_{n-1}$. Hence, we can assume that $k = \frac{n}{h(n)}$ for some function $h(n)$ tending to ∞. The first idea is to replace the random sets in the proof of Proposition 5.10 by random sets of size $\frac{n}{k} = h(n)$. For sets S with $k' = o(k)$ elements the intersection of S and such a random set is empty with large probability. Therefore, we work with random sets of size $\left(\frac{n}{k}\right)^2 = h(n)^2$. The cost for each binary search is increased by a factor 2 only. For sets S where $|S| \geq \frac{n}{h(n)^2}$ the probability of success for a random set is at least $\frac{1}{3}$. Hence, with $O\left(\log \binom{n}{k}\right)$ sets we can identify all but $\frac{n}{h(n)^2}$ essential elements.

In the second phase we work with sets of size $h(n)^4$ and can identify all but $\frac{n}{h(n)^4}$ essential elements. This process is continued until all essential elements can be identified.

The number of query sets besides the queries in the binary searches is bounded by

$$\begin{aligned} & O\left(\log\binom{n}{n/h(n)} + \log\binom{n}{n/h(n)^2} + \log\binom{n}{n/h(n)^4} + \cdots\right) \\ = \; & O\left(\frac{n}{h(n)}\log h(n) + \frac{n}{h(n)^2}2\log h(n) + \frac{n}{h(n)^4}4\log h(n) + \cdots\right) \\ = \; & O\left(k\log\frac{n}{k}\right). \end{aligned}$$

We still have to estimate the number of queries during the binary searches. In the first phase we find not more than $\frac{n}{h(n)}$ essential elements each with a cost of $2\log h(n)$. In the second phase we find not more than $\frac{n}{h(n)^2}$ essential elements each with a cost of $4\log h(n)$ and so on. Hence, the same estimation as before shows that $O\left(k\log\frac{n}{k}\right)$ queries suffice. ∎

The upper bound of Theorem 5.11 is asymptotically optimal for $k \leq n/2$, since it meets the lower bound of Theorem 5.1. Applying Proposition 5.5 we have proved that the complexity of learning the concept classes $\mathsf{PAR}(k)$ is only by a constant factor larger than the simple information theoretic lower bound.

Remark. If one is interested in *applying* learning algorithms, one has to notice that in Proposition 5.10 and Theorem 5.11 only the existence of learning algorithms with the proposed bounds has been proved. For most applications the simple randomized error-free learning algorithm described in the proof of Theorem 5.9 is good enough. The algorithms proving all the other bounds in this paper are explicitly described and can be applied without further considerations.

5.4 Threshold functions

Theorem 5.12 The class THR can be learned with $n - 1 + \lceil\log(n+2)\rceil$ queries.

Proof. In the first phase we only use query sets of type $\{1,\ldots,m\}$. Since threshold functions are monotone, we can find with binary search and $\lceil\log(n+2)\rceil$ queries the minimal $s \in \{0,\ldots,n\}$ such that the answer to the query $\{1,\ldots,s\}$ is 1 or we find out that such an s does not exist. In the last case we are done, since the function we look for is the constant 0. If $s = 0$, the function is the constant 1. If $s \in \{1,\ldots,n\}$, we know that s is an essential element. The query $\{1,\ldots,s\}-\{j\}$ decides for $j \in \{1,\ldots,s-1\}$ whether j is essential and the query $\{1,\ldots,s-1,j\}$ does the same for $j \in \{s+1,\ldots,n\}$. These are $n-1$ additional queries. ∎

Theorem 5.13 Let $0 \leq t \leq k \leq n$. The class $\mathsf{THR}_t(k)$ can be learned with at most $(k-1)\log\frac{n-1}{k-1} + 3k - 3 + \lceil\log(n-k+1)\rceil$ queries.

Proof. The first phase is the same as in the proof of Theorem 5.12. Afterwards we know the parameters n, k, and t. Moreover, if we are not done, we know an essential element s such that the query $\{1,\ldots,s\}$ leads to the answer 1 and the query $\{1,\ldots,s-1\}$ leads to the answer 0. Hence, we know that $\{1,\ldots,s-1\}$ contains exactly $t-1$ essential elements and $\{s+1,\ldots,n\}$ contains the remaining $k-t$ essential elements. Here, we know that $t \leq s \leq n-k+t$. Thus, the step finding s requires at most $\lceil\log(n-k+1)\rceil$ queries.

We use an algorithm for $\mathsf{OR}(k-t)$ on the set $\{s+1,\ldots,n\}$ and add to each query the set $\{1,\ldots,s-1\}$. This algorithm identifies the $k-t$ essential elements in $\{s+1,\ldots,n\}$. Then we use an algorithm for $\mathsf{AND}(t-1)$ on the set $\{1,\ldots,s-1\}$ and add to each query the essential element s. This algorithm identifies the $t-1$ essential elements in $\{1,\ldots,s-1\}$.

In Section 5.1 we have seen that the complexity of learning AND-classes is the same as for the corresponding OR-classes. By Theorem 5.3 the number of queries of the second and third phases is bounded by

$$\begin{aligned}
&(k-t)\left\lceil \log \frac{n-s}{k-t} \right\rceil + 2(k-t) + (t-1)\left\lceil \log \frac{s-1}{t-1} \right\rceil + 2(t-1) \\
&\leq \quad (k-t)\log \frac{n-s}{k-t} + (t-1)\log \frac{s-1}{t-1} + 3k - 3 \\
&\leq \quad (k-1)\log \frac{n-1}{k-1} + 3k - 3.
\end{aligned}$$

The last inequality is for $a = \frac{s-1}{n-1}$ and $b = \frac{t-1}{k-1}$ equivalent to

$$-b\log b - (1-b)\log(1-b) \leq -b\log a - (1-b)\log(1-a).$$

This inequality holds, since $0 \leq a,\ b \leq 1$, see, e.g., [AW87, Lemma 5.3, p.20]. ∎

Theorem 5.14 The class $\mathsf{THR}(k)$ can be learned with at most $2(k-1)\log \frac{n-1}{k-1} + 4k - 4 + \lceil \log(n+2) \rceil$ queries.

Proof. The first phase of the algorithm in the proof of Theorem 5.12 does not need the knowledge of t. Hence, we can use this phase also here.

Afterwards we can solve the problem by the application of algorithms for OR (and equivalently AND). But now the number of essential elements is not known, since it depends on the unknown parameter t. Hence, we use the algorithm of Theorem 5.2. Then, the number of queries for $\mathsf{THR}(k)$ is bounded by

$$\lceil \log(n+2) \rceil + 2(k-t)\log \frac{n-s}{k-t} + 4(k-t) + 2(t-1)\log \frac{s-1}{t-1} + 4(t-1).$$

Now we can use the same estimations as in the proof of Theorem 5.13. ∎

We have the following corollary by simple calculations together with the lower bounds in Section 5.1:

Corollary 5.15 The upper bound of THR given in the algorithm in Theorem 5.12 is optimal if $n+1$ is not a power of 2. Otherwise, we miss the lower bound by 1. The upper bound of $\mathsf{THR}_t(k)$ given in the algorithm in Theorem 5.13 is asymptotically optimal within a constant factor if $k \leq \frac{n}{2}$. The upper bound of $\mathsf{THR}(k)$ given in the algorithm in Theorem 5.14 is asymptotically optimal within a constant factor if $k \leq \frac{n}{2}$.

For the classes $\mathsf{THR}_t(k)$ and $\mathsf{THR}(k)$, their upper bounds are asymptotically optimal for $k \leq \frac{n}{2}$. The optimal number of queries does not rely essentially on the threshold value t or even on the knowledge of t. Does this hold also for $k > \frac{n}{2}$? We only answer this question in special cases. We first observe that we get $\mathsf{OR}(k)$ from $\mathsf{THR}_t(k)$ if $t = 1$. We know that for this case we need $n-1$ queries, if $k \geq \frac{n}{2}$. Moreover, the cases $t = 1$ and $t = k$ are equivalent. This observation can be generalized.

Proposition 5.16 For given n, the complexity of learning $\mathsf{THR}_t(k)$ is equal to the complexity of learning $\mathsf{THR}_{k-t+1}(k)$.

Proof. Let S be the set of essential elements for the problem $\mathsf{THR}_t(k)$. A query Q leads to the answer 1 if and only if $|Q \cap S| \geq t$. If S is the set of essential elements for the problem $\mathsf{THR}_{k-t+1}(k)$, the query $Q' := \{1, \ldots, n\} - Q$ leads to the answer 1 if and only if $|Q' \cap S| \geq k-t+1$ which is equivalent to $|Q \cap S| \leq t-1$. Hence, we can simulate the query Q for $\mathsf{THR}_t(k)$ by the query Q' for $\mathsf{THR}_{k-t+1}(k)$ and vice versa. ∎

We are still left with a variety of parameter choices. Because of Proposition 5.16 we may restrict ourselves to the case $t \leq n/2$. If one uses threshold gates in Boolean circuits, the gates become more powerful if t is close to $n/2$. In particular, for $t = 1$ we get the "simple" OR-function and for $t = n/2$ the "complex" MAJORITY-function. Hence, one might believe that threshold functions with small t are easier to learn than threshold functions with t close to $n/2$. But the following theorems prove that this is not true. For the case $k = n-1$ it is very hard to learn the OR-function ($n-1$ queries are necessary) while it is easy to learn the MAJORITY-function ($\lceil \log n \rceil$ queries are sufficient). More precisely the learning complexity decreases, if $t \leq n/2$, with t. In Theorem 5.18 we present an asymptotically optimal learning algorithm (for the lower bound see Theorem 5.1 (i)) to learn the MAJORITY-function. We leave it to the reader to generalize this result to other threshold values.

Theorem 5.17 Let $t \leq \frac{n}{2}$. To learn the class $\mathsf{THR}_t(n-1)$

- $\max\{\lceil \log n \rceil, \left\lceil \frac{n-1}{t} \right\rceil\}$ queries are necessary and
- $\lceil \frac{n}{t} \rceil + \lceil \log t \rceil - 1$ queries are sufficient.

By Proposition 5.16 we obtain similar results for the case $t > \frac{n}{2}$.

Proof. The lower bound $\lceil \log n \rceil$ is the simple information theoretical lower bound. For the lower bound $\left\lceil \frac{n-1}{t} \right\rceil$ a simple adversary argument works. A query set of size $t+1$ or larger is useless, since we know in advance that the answer is 1. A query set of size $t-1$ or smaller is useless, since we know in advance that the answer is 0. Hence, all query sets have size t. An adversary may answer 1 for the first $\left\lceil \frac{n-1}{t} \right\rceil - 1$ queries. Then there are still at least 2 candidates for the nonessential element and another query is necessary.

For the upper bound we construct $\lceil \frac{n}{t} \rceil$ query sets of size t such that each element is contained in at least one set. We look for a query set leading to the answer 0, since this set contains the only nonessential element. At least one of the queries is good for our purposes. Hence, it is sufficient to ask $\lceil \frac{n}{t} \rceil - 1$ queries. Since $t \leq \frac{n}{2}$, we now can use binary search on this set, if we add always as many already identified elements such that the query size becomes t. ∎

Theorem 5.18 The class $\mathsf{THR}^n_{\frac{n}{2}}(n-k)$ can be learned with $O\left(k \log \frac{n}{k}\right)$ queries.

Proof. If $k > \frac{n}{2}$, nothing has to be done and 0 queries are sufficient. If $k \leq \frac{n}{2}$ and $k = \Theta(n)$, the upper bound follows from Theorem 5.12. Hence, we may assume that $k \leq \frac{n}{8}$. Moreover, we assume that k divides n. Otherwise, we may add less than k elements known to be nonessential. Additional nonessential elements have no influence on the answer to queries and can therefore be added. This does not hold for additional essential elements.

We present two algorithms. The first one meets the upper bound $O(k \log \frac{n}{k})$, if $k = \Omega\left(\sqrt{\frac{n}{\log n}}\right)$, and the second one, if $k = O(n^{1-\varepsilon})$ for some $\varepsilon > 0$. Since each k fulfills at least one of the conditions, the two algorithms are sufficient to prove the theorem.

The number of nonessential elements is smaller than the number of essential elements. The main idea is to search for the k nonessential elements. The set $\{1, \ldots, n\}$ of elements is partitioned to $B_1 = \{1, \ldots, \lceil \frac{n}{3} \rceil\}$, $B_2 = \{\lceil \frac{n}{3} \rceil + 1, \ldots, \lceil \frac{2n}{3} \rceil\}$, and $B_3 = \{\lceil \frac{2n}{3} \rceil + 1, \ldots, n\}$. We only

describe how to find the at most k nonessential elements in B_3. The same approach can be used for B_1 and B_2. In order to apply OR-strategies to search for nonessential elements we construct so-called *dummy* sets D_j where $D_1 \supseteq \cdots \supseteq D_m$ for some m and D_j contains exactly $\frac{n}{2} - j$ essential elements. Let S contain $j \leq m$ elements and let S be disjoint from D_1. The query $S \cup D_j$ is answered 0 iff S contains at least one nonessential element. This is equivalent to an OR-query S with respect to nonessential elements. We have to be careful, since the OR-strategy only can work with queries whose size is bounded by m.

Both algorithms determine (as in the proof of Theorem 5.12) the minimal s such that $S^* = \{1, \ldots, s\}$ contains $\frac{n}{2}$ essential elements. Since $\frac{n}{2} \leq s \leq \frac{n}{2} + k$, $\lceil \log(k+1) \rceil$ queries are sufficient for this purpose.

The first algorithm then asks the queries $S^* - \{i\}$, $1 \leq i \leq k + \frac{n}{k}$. The query $S^* - \{i\}$ is answered by 0 iff the element i is essential. Since the number of nonessential elements equals k, we determine at least $\frac{n}{k}$ essential elements $e_1, \ldots, e_{\frac{n}{k}} \in \{1, \ldots, k + \frac{n}{k}\}$. The set $D_j := S^* - \{e_1, \ldots, e_j\}$ contains $\frac{n}{2} - j$ essential elements, $D_j \subseteq D_{j-1}$ and $D_j \subseteq S^* \subseteq B_1 \cup B_2$, since $k \leq \frac{n}{8}$. Now we determine the nonessential elements of B_3 with an OR-strategy starting with a partition of B_3 into $\lceil \frac{k}{3} \rceil$ sets of size bounded by $\frac{n}{k}$. Each query which is a winner, is partitioned to two queries of almost the same size. The number of queries to determine the nonessential elements in B_3 is bounded by $\lceil \log(k+1) \rceil + k + \frac{n}{k}$ for the construction of the dummy sets and $O(k \log \frac{n}{k})$ for the OR-strategy. Since $k = \Omega\left(\sqrt{\frac{n}{\log n}}\right)$, $\frac{n}{k} = O(k \log \frac{n}{k})$ and we are done in this case.

The second algorithm chooses some l where $s < l \leq \lceil \frac{2n}{3} \rceil + 1$ such that the elements $l, \ldots, n$ can be partitioned to $k+1$ sets $R_1, \ldots, R_{k+1}$ of the same size $r \leq \frac{n}{k}$. Since the number of nonessential elements is k, there exists one set R_i consisting of r essential elements. Later the algorithm will determine such an i. We try to find a set $D^* \subseteq S^*$ with exactly $\frac{n}{2} - r$ essential elements. With D^* and R_i it is easy to fulfill our task. Let $R_i = \{e_1, \ldots, e_r\}$, then $D_j := D^* \cup \{e_1, \ldots e_{r-j}\}$, $1 \leq j \leq r$, can be used as dummy sets. The elements of R_i all are essential and we can determine the nonessential elements in the union of all R_h, $h \neq i$, with these dummy sets and an OR-strategy asking queries whose size is bounded by r. This again can be done with $O(k \log \frac{n}{k})$ queries. We still have to describe how we can find the dummy sets.

We consider $C_0 = S^* - \{1, \ldots, r\}, \ldots, C_k = S^* - \{1, \ldots, r+k\}$ as candidates for D^*. We know that C_0 contains at least $\frac{n}{2} - r$ essential elements and that C_k contains at most $\frac{n}{2} - r$ essential elements. Hence, one of the candidates can serve as D^*. Let us assume that we ask the queries $R_1 \cup C_j, \ldots, R_{k+1} \cup C_j$. If all queries are answered by 0, even the union of R_i (with r essential elements) and C_j does not contain $\frac{n}{2}$ essential elements. Hence, C_j contains less than $\frac{n}{2} - r$ essential elements. If at least one query is answered by 1, then C_j contains at least $\frac{n}{2} - r$ essential elements. Hence, with binary search on $j \in \{0, \ldots, k\}$ and altogether $O(k \log k)$ queries we find the largest j such that one of the queries $R_1 \cup C_j, \ldots R_{k+1} \cup C_j$ is answered by 1. This implies that C_j contains at least $\frac{n}{2} - r$ essential elements while C_{j+1} contains less than $\frac{n}{2} - r$ essential elements. Since $C_j \supseteq C_{j+1}$ and $|C_j - C_{j+1}| = 1$, C_j contains exactly $\frac{n}{2} - r$ essential elements and we choose $D^* = C_j$. Moreover, let R_i be one of the sets such that the query $R_i \cup D^*$ has been answered by 1, then all r elements of R_i are essential. Hence, we have found D^* and R_i. The number of additional queries is $O(k \log k) = O(k \log \frac{n}{k})$, since $k = O(n^{1-\varepsilon})$. ∎

Chapter 6

Fast NC and RNC Algorithms for Finding a Maximal Set of Paths

6.1 Parallel algorithms for the MPS problem

In this section, we first present an RNC algorithm for the MPS problem for undirected graphs. We then mention how to modify it for digraphs. At the end of this section, we will also point out how to derandomize them, and apply the resulting algorithms to approximating the shortest superstring problem.

A vertex v in G is *good* if $\sum_{u \in N_G(v)} \frac{1}{d_G(u)} \geq \frac{1}{3}$. An edge in G is *good* if at least one of its endpoints is good. The following two lemmas will be used in our algorithms:

Lemma 6.1 [II86, Koz92] At least half the edges in every graph G are good.

Lemma 6.2 [II86] There is an RNC algorithm which, given a graph $G = (V, E)$, computes a matching M in G such that for each good vertex v in G, the probability that $v \in V(M)$ is no less than a positive constant. It runs in $O(1)$ expected time with $O(n + m)$ processors on a CRCW PRAM.

6.1.1 Description of the RNC algorithm

We first give an intuitive explanation of the algorithm. The algorithm always maintains a set F of edges in G such that the graph (V, F) is a collection of vertex-disjoint paths. Initially, $F = \emptyset$. The algorithm then proceeds in stages. Before each stage gets started, every edge e in G satisfies that $F \cup \{e\}$ is a path set. In each stage, the algorithm uses Lemma 6.2 to compute a matching M between the endpoints of the paths in the graph (V, F). It is clear that the graph $(V, F \cup M)$ is a collection of vertex-disjoint paths and cycles. To break the cycles, for each path P in the graph (V, F), the algorithm randomly chooses one of the two endpoints of P, and deletes from M the edge incident to the unchosen endpoint (if it really exists). Obviously, the resulting M satisfies that each path in the graph $(V, F \cup M)$ contains at most one edge in M. This implies that $F \cup M$ is also a path set. The algorithm then adds the edges in M to F, and deletes from G every edge e such that $F \cup \{e\}$ is not a path set. After this, it proceeds to the next stage if G still contains at least one edge; otherwise, it outputs F and halts.

To implement each stage efficiently, the algorithm maintains an array R for which the following is an *invariant*: For each vertex $i \in V$, $R[i] = i$ if the degree of i in the graph (V, F) is 0, $R[i] = -1$ if the degree of i in the graph (V, F) is 2, and $R[i] = j$ otherwise, where j is the other endpoint

of the path containing i in the graph (V, F). Initially, the algorithm sets $R[i] = i$ for each vertex $i \in V$. Thus, the invariant is true before the first stage gets started.

More precisely, the algorithm performs the following steps in each stage:

1. Use the algorithm in Lemma 6.2 to find a random matching M' in G.
2. For each $i \in V(M')$, select i if $R[i] = i$, and randomly select one of i and $R[i]$ if $i < R[i]$. [*Comment:* $R[i] \neq -1$ for every vertex $i \in V(M')$.]
3. Set M to be the set of those edges $e \in M'$ such that both endpoints of e were selected in step 2.
4. Add the edges in M to F.
5. In parallel, for each edge $\{i_0, i_1\} \in M$, perform the following steps:
   - 5.1. For each $i_j \in \{i_0, i_1\}$, if $R[i_j] \neq i_j$ (or equivalently, the degree of i_j in the graph (V, F) is now 2), then remove all the edges incident to i_j from G, and set $R[i_j] := -1$.
   - 5.2. Remove the edge $\{R[i_0], R[i_1]\}$ from G if it is in G (since the graph $(V, F \cup \{\{R[i_0], R[i_1]\}\})$ contains a cycle).
   - 5.3. Set $R[R[i_0]] := R[i_1]$ and $R[R[i_1]] := R[i_0]$.

Clearly, every edge e such that $F \cup \{e\}$ is not a path set must be deleted either in step 5.1 or 5.2, and the array R is updated in steps 5.1 and 5.3 so that the invariant remains true. By a simple induction on the number of stages the algorithm performs, we can show the correctness of the algorithm.

6.1.2 Complexity analysis

In this subsection, we prove the following theorem:

Theorem 6.3 The RNC algorithm runs in $O(\log n)$ expected time using $O(n + m)$ processors on a CRCW PRAM.

The algorithm uses $O(n + m)$ processors; every vertex and every edge in G has a processor associated with it. Clearly, the initialization step of the algorithm takes $O(1)$ time with $O(n)$ processors on an EREW PRAM. We claim that each stage of the algorithm can be done in $O(1)$ time with $O(n + m)$ processors on a CRCW PRAM. According to Lemma 6.2, step 1 of each stage can be done in $O(1)$ time with $O(n + m)$ processors on a CRCW PRAM. Other steps in each stage use no more resources. This establishes the claim. In the remainder of this subsection, we will show that the expected number of stages performed by the algorithm is $O(\log n)$. This together with the claim implies the theorem.

First, we show the following lemma:

Lemma 6.4 Fix a stage of the algorithm. For each good vertex i in G, the probability that $i \in V(M)$ at the end of the stage is no less than a positive constant. Consequently, at the end of the stage, $\Pr[i \in V(M) \text{ or } j \in V(M)]$ is no less than a positive constant for each good edge $\{i, j\}$ in G.

Proof. By Lemma 6.2, for each good vertex i in G, the probability that $i \in V(M')$ after step 1 of the stage is no less than a positive constant. It is not difficult to see that $\Pr[i \in V(M) \mid i \in V(M')] \geq \frac{1}{4}$ for all vertices i in G. Thus, we have the lemma. ∎

We proceed to the proof of the fact that the expected number of stages performed by the algorithm is $O(\log n)$. To show the fact, we use a potential function argument. For a subgraph $\mathcal{G}$ of the input graph and a path set $\mathcal{F}$, define

$$\Phi(\mathcal{G}, \mathcal{F}) = \sum_{\text{edge } \{i,j\} \text{ in } \mathcal{G}} \phi(\mathcal{G}, \mathcal{F}, \{i, j\}),$$

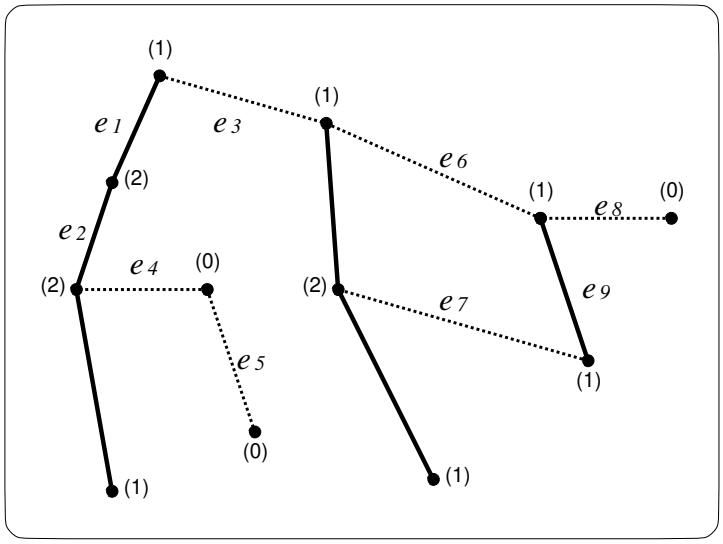


Figure 6.1: Example of the potential function for an undirected graph

where $\phi(\mathcal{G}, \mathcal{F}, \{i, j\}) = (2 - d_{(V,\mathcal{F})}(i))(2 - d_{(V,\mathcal{F})}(j))$.

Example 6.5 Let G be a graph given in Figure 6.1, and F be the set of the thick lines. The number near by each vertex is its degree in the graph (V, F). Then, $\phi(G, F, e_5) = 4$, $\phi(G, F, e_8) = 2$, $\phi(G, F, e_3) = \phi(G, F, e_6) = \phi(G, F, e_9) = 1$, and $\phi(G, F, e_1) = \phi(G, F, e_2) = \phi(G, F, e_4) = \phi(G, F, e_7) = 0$. Thus, $\Phi(G, F) = 9$.

Fix a stage of the algorithm. Let G_b and G_a, respectively, be the graph G before and after the stage. Similarly, let F_b and F_a, respectively, be the path set F before and after the stage. Let $M = F_a - F_b$. Note that M is the matching computed at step 3 of the stage. Clearly, $\Phi(G_b, F_b) \geq \Phi(G_b, F_a) \geq \Phi(G_a, F_a)$. The intuition behind the potential function can be seen from Lemma 6.1, Lemma 6.2, and the following fact:

Fact 6.6 Assume that $e = \{i, j\}$ is an edge in G_b such that $i \in V(M)$ or $j \in V(M)$. Then, $\phi(G_b, F_b, e) - \phi(G_b, F_a, e) \geq 1$.

Proof. Let d_i and d_j be the degrees of i and j in the graph (V, F_b), respectively. According to the values of d_i and d_j, we distinguish four cases as follows:

Case 1: $d_i = d_j = 0$. Then, we have $\phi(G_b, F_b, e) = 4$. If both $i \in V(M)$ and $j \in V(M)$, then $\phi(G_b, F_a, e) = 1$; otherwise, $\phi(G_b, F_a, e) = 2$. Thus, $\phi(G_b, F_b, e) - \phi(G_b, F_a, e) \geq 2$.

Case 2: $d_i = 0$ and $d_j = 1$. Then, we have $\phi(G_b, F_b, e) = 2$. If $j \in V(M)$, then $\phi(G_b, F_a, e) = 0$; otherwise, $i \in V(M)$ and $\phi(G_b, F_a, e) = 1$. Thus, $\phi(G_b, F_b, e) - \phi(G_b, F_a, e) \geq 1$.

Case 3: $d_i = 1$ and $d_j = 0$. This case is similar to Case 2.

Case 4: $d_i = d_j = 1$. Then, we have $\phi(G_b, F_b, e) = 1$ and $\phi(G_b, F_a, e) = 0$. Thus, $\phi(G_b, F_b, e) - \phi(G_b, F_a, e) = 1$. ∎

For a random variable X, let $\mathcal{E}X$ denote the expected value of X, and let $\mathcal{E}(X \mid B)$ denote the expected value of X given that event B occurs.

Lemma 6.7 (Main Lemma.) $\mathcal{E}(\Phi(G_b, F_b) - \Phi(G_a, F_a)) \geq \mathcal{E}(\Phi(G_b, F_b) - \Phi(G_b, F_a)) \geq c \cdot \Phi(G_b, F_b)$ for some constant $c > 0$.

Proof. For each edge $e = \{i, j\}$ in G_b, let $X_e = \phi(G_b, F_b, e)$, $Y_e = \phi(G_b, F_a, e)$, $Z_e = X_e - Y_e$, and B_e be the event that $i \in V(M)$ or $j \in V(M)$. Let the number of edges in G_b be m_b. Clearly, $\Phi(G_b, F_b) \leq 4m_b$.

Fix an edge $e = \{i, j\}$ in G_b. By Fact 6.6, $\mathcal{E}(Z_e \mid B_e) \geq 1$. From this, it follows that $\mathcal{E}Z_e \geq \mathcal{E}(Z_e \mid B_e)\Pr[B_e] \geq \Pr[B_e]$. Thus, if e is good, then by Lemma 6.4, $\mathcal{E}Z_e \geq \Pr[B_e] \geq c'$ for some constant $c' > 0$. Combining this with the fact that G_a is a subgraph of G_b, we now have

$$\begin{aligned}
\mathcal{E}(\Phi(G_b, F_b) - \Phi(G_a, F_a)) &\geq \mathcal{E}(\Phi(G_b, F_b) - \Phi(G_b, F_a)) \\
&= \sum_{\text{edge } e \text{ in } G_b} \mathcal{E}Z_e \geq \sum_{\text{good edge } e \text{ in } G_b} \mathcal{E}Z_e \\
&\geq \sum_{\text{good edge } e \text{ in } G_b} \Pr[B_e] \geq \sum_{\text{good edge } e \text{ in } G_b} c' \geq \frac{c' m_b}{2}.
\end{aligned}$$

The last inequality follows from Lemma 6.1. On the other hand, we have $\Phi(G_b, F_b) \leq 4m_b$. Thus, $\mathcal{E}(\Phi(G_b, F_b) - \Phi(G_a, F_a)) \geq c' m_b/2 \geq \frac{c'}{8}\Phi(G_b, F_b)$. This completes the proof. ∎

Note that $\Phi(G, \emptyset) = 4m$ and that the algorithm halts when $\Phi(G, F) < 1$ (or equivalently, $\Phi(G, F) = 0$). Thus, by Lemma 6.7 above and Theorem 1.3 in [MR95], we immediately have that the expected number of stages performed by the algorithm is at most $\int_1^{4m} \frac{1}{cx} dx = O(\log n)$. This completes the proof of the theorem.

6.1.3 Extension to digraphs

Our algorithm for computing an MDPS in D always maintains a set B of arcs in D such that the digraph (V, B) is a collection of vertex-disjoint paths. Initially, $B = \emptyset$. The algorithm then proceeds in stages. Before each stage gets started, every arc e in G satisfies that $B \cup \{e\}$ is an MPS. In each stage, the algorithm uses Lemma 6.2 to compute a suitable matching M' in the underlying graph of D, and then uses M' to compute a set M of arcs in D such that no two arcs in M have a common endpoint. The remaining task of each stage of the algorithm is essentially the same as that of each stage of the algorithm in the subsection 6.1.1. The algorithm proceeds to the next stage if D still contains at least one arc; otherwise, it outputs B and halts.

To implement each stage efficiently, the algorithm maintains an array R for which the following is an *invariant*: For each vertex $i \in V$, $R[i] = i$ if the total degree of i in the digraph (V, B) is 0, $R[i] = -1$ if the total degree of i in the digraph (V, B) is 2, and $R[i] = j$ otherwise, where j is the other endpoint of the path containing i in the digraph (V, B). Initially, the algorithm sets $R[i] = i$ for each vertex $i \in V$.

More precisely, the algorithm performs the following steps in each stage:

1. Use the algorithm in Lemma 6.2 to find a random matching M' in the underlying graph of D.
2. For each $i \in V(M')$, select i if $R[i] = i$, and randomly select one of i and $R[i]$ if $i < R[i]$. [*Comment:* $R[i] \neq -1$ for every vertex $i \in V(M')$.]
3. Set $M := \{(i, j) \in A(D) \mid \{i, j\} \in M'\}$
   $-\{(i, j) \in A(D) \mid \{i, j\} \in M', (j, i) \in A(D), \text{ and } i > j\}$.
4. Add the edges in M to B.
5. In parallel, for each arc $(i_0, i_1) \in M$, perform the following steps:
   5.1. For each $i_j \in \{i_0, i_1\}$, if $R[i_j] \neq i_j$ (or equivalently, the total degree of i_j in the digraph (V, B) is now 2), then remove all the arcs incident to i_j from D, and set $R[i_j] := -1$.

5.2. Remove the arc $(R[i_1], R[i_0])$ from D if it is in D
(since the digraph $(V, B \cup \{(R[i_1], R[i_0])\})$ contains a cycle).
5.3. Remove all the arcs with tail i_0 or head i_1 from D.
5.4. Set $R[R[i_0]] := R[i_1]$ and $R[R[i_1]] := R[i_0]$.

We say that a vertex i in D is *good* if it is good in G (the underlying graph of D), and say that an arc in D is *good* if its tail or head is good.

Lemma 6.8 At least one third of the arcs in D are good.

Proof. Let E_1 be the set of those edges $\{i, j\}$ in G such that exactly one of (i, j) and (j, i) is in D', and let $E_2 = E(G) - E_1$. Let x_1 (or x_2) be the number of edges $e \in E_1$ (respectively, $e \in E_2$) such that e is good in G. Then, $x_1 + x_2 \geq \frac{|E_1|+|E_2|}{2}$ by Lemma 6.1. Moreover, D has $|E_1| + 2|E_2|$ arcs among which $x_1 + 2x_2$ are good. If $|E_1| \geq |E_2|$, then $x_1 + 2x_2 \geq \frac{|E_1|+|E_2|}{2} = \frac{|E_1|+2|E_2|}{2} - \frac{|E_2|}{2} \geq \frac{|E_1|+2|E_2|}{2} - \frac{|E_1|+2|E_2|}{6} \geq \frac{|E_1|+2|E_2|}{3}$; otherwise, $x_1 + 2x_2 = 2(x_1 + x_2) - x_1 \geq (|E_1| + |E_2|) - |E_1| = |E_2| \geq \frac{|E_1|+2|E_2|}{3}$. This completes the proof. ∎

From Lemma 6.4, it is easy to see the following lemma:

Lemma 6.9 For every stage of the algorithm, the set M computed at step 3 in the stage satisfies that $\Pr[i \in V(M)$ or $j \in V(M)]$ is no less than a positive constant for each good arc (i, j) in D.

To prove that the expected number of stages performed by the algorithm is $O(\log n)$, we need to modify the potential function in Section 6.1.2 as follows: For a subgraph $\mathcal{D}$ of the input digraph and a DPS $\mathcal{B}$, define

$$\Phi'(\mathcal{D}, \mathcal{B}) = \sum_{\text{arc } (i,j) \text{ in } \mathcal{D}} \phi'(\mathcal{D}, \mathcal{B}, (i, j)),$$

where $\phi'(\mathcal{D}, \mathcal{B}, (i, j)) = (1 - d^+_{D[\mathcal{B}]}(i))(1 - d^-_{D[\mathcal{B}]}(j))(2 - d_{D[\mathcal{B}]}(i))(2 - d_{D[\mathcal{B}]}(j))$.

Example 6.10 Let D be a digraph given in Figure 6.2, and B be the set of the bold arcs. The two numbers in the pair near by each vertex are its indegree and outdegree in the digraph (V, B), respectively. The underlying graph of D is the same as the graph G in Figure 6.1, and a_i corresponds to e_i in Figure 6.1. Then, $\phi'(D, B, a_5) = 4$, $\phi'(D, B, a_6) = \phi'(D, B, a_9) = 1$, and $\phi'(D, B, a_1) = \phi'(D, B, a_2) = \phi'(D, B, a_4) = \phi'(D, B, a_7) = \phi'(D, B, a_3) = \phi'(D, B, a_8) = 0$. Thus, $\Phi'(D, B) = 6$.

Using Lemma 6.8 and Lemma 6.9, we can show the following lemma by a similar proof to that of Lemma 6.7:

Lemma 6.11 Fix a stage of the algorithm. Let D_b and D_a, respectively, be the digraph D before and after the stage. Similarly, let B_b and B_a, respectively, be the DPS B before and after the stage. Then, $\mathcal{E}(\Phi'(D_b, B_b) - \Phi'(D_a, B_a)) \geq c' \cdot \Phi'(D_b, B_b)$ for some constant $c' > 0$.

Using the same arguments as in the subsection 6.1.2, we get the following theorem.

Theorem 6.12 An MDPS can be computed in $O(\log n)$ expected time with $O(n+m)$ processors on a CRCW PRAM.

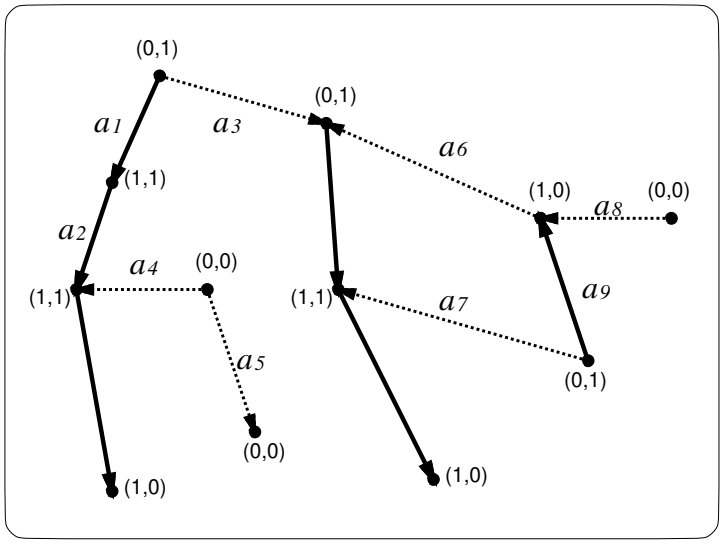


Figure 6.2: Example of the potential function for a directed graph

6.1.4 The NC algorithm

Carefully derandomizing the RNC algorithm in the subsection 6.1.1 with the techniques in [Che95a], we can obtain an NC algorithm for the MPS problem. We describe this result in the following theorem:

Theorem 6.13 An MPS in a given undirected graph can be found in $O(\log^2 n)$ time with $O(\Delta^2(n+m)/\log n)$ processors on an EREW PRAM.

Similarly, we can prove the following theorem:

Theorem 6.14 An MDPS in a given digraph can be found in $O(\log^2 n)$ time with $O(\Delta^2(n+m)/\log n)$ processors on an EREW PRAM.

It is not difficult to modify the proof of Theorem 6.14 to obtain a proof for the following corollary:

Corollary 6.15 Given a digraph D and a DPS F in D, an MDPS B in D with $F \subseteq B$ can be found in $O(\log^2 n)$ time with $O(\Delta^2(n+m)/\log n)$ processors on an EREW PRAM.

An immediate consequence of Corollary 6.15 is that the NC approximation algorithm for the shortest superstring problem given in [Che96] can be made faster. We describe this result in the following theorem.

Theorem 6.16 There is an NC approximation algorithm for the shortest superstring problem with a compression ratio of $\frac{1}{3+\epsilon}$ for any $\epsilon > 0$. It runs in $O(\log^2 n \cdot \log_{1+\epsilon/3}|S|)$ time with $\frac{n^2|S|^2}{\log n}$ processors, where n is the number of the given strings, and $|S|$ is the total length of the given strings.

6.2 An application to shortest superstrings with flipping

For a string s, let s^R denote the reverse of s and $|s|$ denote the length of s. Let s and t be two distinct strings, and let v be the longest string such that $s = uv$ and $t = vw$ for some non-empty strings u and w. $|v|$ is called the *overlap* between s and t and is denoted by $ov(s,t)$. By $s \circ t$, we denote the string uvw.

Let $S = \{s_1, s_2, \cdots, s_n\}$ be a set of strings over some alphabet Σ. Define $S^R = \{s_1^R, ..., s_n^R\}$ and $|S| = \sum_{i=1}^{n} |s_i|$. A *superstring-with-flipping* of S is a string s over Σ such that for each i, at least one of s_i and s_i^R is a substring of s. In the sequel, a superstring-with-flipping is simply called a superstring; this should be distinguished with the usual definition of a superstring in the literature. The *amount of compression* achieved by a superstring s is $|S| - |s|$. Let $opt_{com}(S)$ denote the maximum amount of compression achievable by a superstring of S. W.l.o.g., we assume that the set $S \cup S^R$ is *substring free*, i.e., no string in $S \cup S^R$ is a substring of any other. For simplicity of explanation, we assume that no string in S is a palindrome. At the end of this section, we will point out why this assumption is not essential to our results.

The *overlap graph* of S is a weighted digraph $OG(S) = (S \cup S^R, A, ov)$, where $A = \{(s,t) \mid s, t \in S \cup S^R,\ s \neq t$, and $s \neq t^R\}$ and each arc (s,t) has weight $ov(s,t)$. For a subgraph D of $OG(S)$, the *weight* of D is the total weight of the arcs in D and is denoted by $ov(D)$. The *mate* of a vertex s in $OG(S)$ is s^R. Similarly, the *mate* of an arc $e = (s,t)$ in $OG(S)$ is (t^R, s^R) and is denoted by e^R. Note that e and e^R have the same weight. A (directed) path P in $OG(S)$ is said to be *legal* if for every string $s \in S$, at most one of s and its mate s^R is on P. The *mate* of a legal path P in $OG(S)$ is the path consists of the mates of the arcs on P, and is denoted by P^R. Note that $ov(P) = ov(P^R)$. For a legal path $P = s, t, ..., u$ in $OG(S)$, we call $s \circ t \circ \cdots \circ u$ the *string associated with* P. Note that the string associated with P is a superstring of the strings s, t, $\cdots$, u and has length $(|s| + |t| + \cdots + |u|) - ov(P)$. A legal path P in $OG(S)$ is said to be *Hamiltonian* if for each string s in S, either s or its mate s^R is on P. A *two-path cover* of $OG(S)$ is a subgraph consisting of a Hamiltonian legal path and its mate. We denote by $opt_{cov}(S)$ the weight of a maximum-weight two-path cover of $OG(S)$. Then, we have the following fact immediately:

Fact 6.17 $opt_{cov}(S) = 2 \cdot opt_{com}(S)$. Moreover, the amount of compression achieved by the string associated with a Hamiltonian legal path P in $OG(S)$ is $ov(P)$. Especially, the amount of compression achieved by the string associated with a maximum-weight Hamiltonian legal path is $opt_{com}(S)$.

Recall that a directed path set (DPS) in a digraph $D = (V_D, A_D)$ is a subset B of A_D such that the digraph (V_D, B) is an acyclic digraph in which the indegree and outdegree of each vertex are both at most 1. Consider the following simple algorithm for finding a two-path cover of $OG(S)$ with large weight:

Algorithm GREEDY
Input: $OG(S) = (S \cup S^R, A, ov)$.
1. Initialize B to be the empty set.
2. While the digraph $(S \cup S^R, B)$ is not a two-path cover of $OG(S)$, perform the following: Add to B the maximum-weight arc e and its mate e^R such that $B \cup \{e, e^R\}$ is a DPS in $OG(S)$ but $B \cup \{f, f^R\}$ is not a DPS in $OG(S)$ for all arcs f with $ov(e) < ov(f)$. (Note: We here assume that the arcs of equal weights in $OG(S)$ are linearly ordered.)
3. Output the two-path cover $(S \cup S^R, B)$.

The following fact was implicitly mentioned in [JLD92]:

Fact 6.18 [JLD92] Let P be one of the two paths in the two-path cover output by GREEDY. Then, the amount of compression achieved by the string associated with P is at least $\frac{opt_{com}(S)}{2}$.

Lemma 6.19 Suppose that the weights on the arcs in $OG(S)$ are modified in a manner such that each arc has the same weight as its mate. Let $OG'(S) = (S \cup S^R, A, w)$ be the resulting digraph. Then, the two-path cover $(S \cup S^R, B)$ output by GREEDY on input $OG'(S)$ has weight $\geq \frac{w(C_{max})}{3}$, where C_{max} is a maximum-weight two-path cover of $OG'(S)$.

Proof. The proof uses similar ideas to those in [Tur89] and [CGPR94]. First observe that on input $OG'(S)$, GREEDY selects an arc and its mate during each iteration of the while-loop. Also note that the while-loop is executed exactly $n-1$ times. For $1 \leq i \leq n-1$, let e_i and e_i^R be the two arcs selected at the i-th iteration. For $0 \leq i \leq n-1$, let C_i be a maximum-weight two-path cover of $OG'(S)$ in which all the arcs e_1, e_1^R, e_2, e_2^R, $\cdots$, e_i, e_i^R appear. Clearly, C_0 is a maximum-weight two-path cover of $OG'(S)$ and C_{n-1} is the output of GREEDY on input $OG'(S)$.

We claim that for each $1 \leq i \leq n-1$, $w(C_{i-1}) \leq w(C_i) + 4w(e_i)$. To see the claim, fix an i with $1 \leq i \leq n-1$. Let $e_i = (s,t)$ and P be one of the two Hamiltonian legal paths in C_{i-1}. Then, $e_i^R = (t^R, s^R)$ and P^R is the other Hamiltonian legal path in C_{i-1}. Moreover, one of the following two cases must occur:

Case 1: s and t appear in the same Hamiltonian legal path in C_{i-1}. W.l.o.g., we may assume that s and t appear in P. Then, s^R and t^R appear in P^R. We only consider the case where s is reachable from t in P, t has a predecessor u in P, and s has a successor v in P; the other cases are similar. Then, there must exist an arc $(u_1, v_1) \notin \{e_1, e_1^R, ..., e_i, e_i^R\}$ on the subpath of P from t to s (possibly, $u_1 = t$ and/or $v_1 = s$). Moreover, the arcs (u,t), (u_1,v_1), (s,v) and their mates are contained in C_{i-1}. Let C be the two-path cover of $OG'(S)$ obtained from C_{i-1} by replacing the arcs (u,t), (u_1,v_1), (s,v) and their mates with the following six arcs: e_i, (u,v_1), (u_1,v) and their mates. Since the arcs (u,t), (u_1,v_1), (s,v) and their mates each have lighter weight than e_i by the algorithm GREEDY, we have $w(C) \geq w(C_{i-1}) - 4w(e_i)$. (*Note*: Recall that $w(e_i) = w(e_i^R)$.) On the other hand, since C is a two-path cover of $OG'(S)$ in which all the arcs e_1, e_1^R, e_2, e_2^R, ..., e_i, e_i^R appear, we have $w(C) \leq w(C_i)$ by the definition of C_i. Hence, we have $w(C_{i-1}) \leq w(C) + 4w(e_i) \leq w(C_i) + 4w(e_i)$.

Case 2: s and t do not appear in the same Hamiltonian legal path in C_{i-1}. W.l.o.g., we may assume that s appears in P and t appears in P^R. Then, s^R appears in P^R and t^R appears in P. We only consider the case where t^R is reachable from s in P and both s and t^R have a successor in P; the other cases are similar. Let u be the successor of s and v be the successor of t^R in P (possibly, $u = t^R$). Then, u^R and v^R must be the predecessors of s^R and t in P^R, respectively. Moreover, the arcs (s,u), (t^R,v) and their mates are contained in C_{i-1}. Let C be the two-path cover of $OG'(S)$ obtained from C_{i-1} by replacing the arcs (s,u), (t^R,v) and their mates with the following four arcs: e_i, (u^R,v) and their mates. Since the arcs (s,u), (t^R,v) and their mates each have lighter weight than e_i by the algorithm GREEDY, we have $w(C) \geq w(C_{i-1}) - 2w(e_i)$. (*Note*: Recall that $w(e_i) = w(e_i^R)$.) On the other hand, since C is a two-path cover of $OG'(S)$ in which all the arcs e_1, e_1^R, e_2, e_2^R, ..., e_i, e_i^R appear, we have $w(C) \leq w(C_i)$ by the definition of C_i. Hence, we have $w(C_{i-1}) \leq w(C) + 2w(e_i) \leq w(C_i) + 2w(e_i)$.

By the claim, we have $w(C_0) \leq w(C_{n-1}) + \sum_{i=1}^{n-1} 4w(e_i)$. Note that $w(C_0) = w(C_{max})$ and $w(C_{n-1}) = \sum_{i=1}^{n-1} 2w(e_i)$. Therefore, $w(C_{n-1}) \geq w(C_{max})/3$. This completes the proof. ∎

Our next goal is to parallelize GREEDY. To reach this goal, we need to prove several lemmas. First, several definitions are in order. An unweighted subgraph D of $OG(S)$ is said to be *legal* if the mate of each vertex and each arc in D is also contained in D. A DPS B in D is said

to be *legal* if the mate of each arc in B is also contained in B. A *maximal* legal DPS in D is a legal DPS that is not properly contained in another legal DPS. We want to design a parallel algorithm for computing a maximal legal DPS in a legal unweighted subgraph D of $OG(S)$. To this end, we modify the RNC algorithm in subsection 6.1.3 as follows. The new input is a legal unweighted subgraph D of $OG(S)$ and step 3 is replaced with the following three lines:

3'.1. For each $s_i \in S$, randomly select one of s_i and its mate;
3'.2. Set $M'' := \{e \in M' |$ both the tail and head of e were selected in the last step $\}$;
3'.3. Set $M := M'' \cup \{e |\ e$ is in D' and e^R is in $M''\}$;

The crucial point is that for each vertex (resp., arc) in D', the vertex (resp., arc) and its mate must be removed from D' in the same call of step 5. Moreover, from Lemma 6.9, it is easy to see the following lemma:

Lemma 6.20 For each good arc (i, j) in D', $\Pr[i \in V(M'')$ or $j \in V(M'')]$ is no less than a positive constant.

By this lemma and the discussions in subsection 6.1.3, we have:

Lemma 6.21 Given a legal unweighted subgraph D of $OG(S)$ and a legal DPS F in D, a maximal legal DPS B in D with $F \subseteq B$ can be found in $O(\log n)$ expected time with $O(n+m)$ processors on a CRCW PRAM, where m is the number of arcs in D.

To decrease the number of random bits used in line 3'.1 above, we further replace lines above with the following three lines:

3".1. Use M' to construct an undirected graph $K = (M', E_K)$, where E_K consists of those edges $\{e_1, e_2\}$ such that the head or tail of e_1 is the mate of the head or tail of e_2; (Comment: The maximum degree of K is 2.)
3".2. 3-color the vertices of K to partition M' into three independent sets M'_1, M'_2, and M'_3;
3".3. Randomly set M to be one of M'_1, M'_2, and M'_3;

It is easy to construct K and 3-color it in $O(\log n)$ time with $O(n/\log n)$ processors on an EREW PRAM. Moreover, even if lines 3'.1 through 3'.3 are replaced with lines 3".1 through 3".3, Lemma 6.20 still holds. This together with the discussions in section 6.1 implies the following lemma:

Lemma 6.22 Given a legal unweighted subgraph D of $OG(S)$ and a legal DPS F in D, a maximal legal DPS B in D with $F \subseteq B$ can be found in $O(\log^2 n)$ time with $O(n^2(n+m)/\log n)$ processors on an EREW PRAM, where m is the number of arcs in D.

Now, we are ready to present a parallelized version of GREEDY. This algorithm is very similar to Algorithm 5 in [Che96].

Algorithm PARGREEDY
Input: $OG(S) = (S \cup S^R, A, ov)$.
1. Let $c = 1 + \frac{\epsilon}{3}$. In parallel, for each arc $e \in A$, set $lev(e) = \lceil \log_c ov(e) \rceil$ if $ov(e) > 1$, and set $lev(e) = 0$ otherwise.
2. Compute $MaxLev = \max\{lev(e) \mid e \in A\}$.
3. Set $B = \emptyset$ and $CurLev = MaxLev$.
4. While $CurLev \geq 0$, perform the following steps:
   4.1. Construct an unweighted digraph $D = (S \cup S^R, E)$ by setting $E = B \cup \{e \in A \mid lev(e) = CurLev\}$. (Comment: Since $ov(e) = ov(e^R)$ for each arc $e \in A$, D must be a legal unweighted subgraph of $OG(S)$.)

4.2 Compute a maximal legal DPS F in D with $B \subseteq F$ and then update B to be F.
4.3 Decrease $CurLev$ by 1.

5. Output the digraph $(S \cup S^R, B)$.

Lemma 6.23 Algorithm PARGREEDY finds a two-path cover of $OG(S)$ with weight at least $\frac{opt_{cov}(S)}{3+\epsilon}$.

Proof. The proof is very similar to that of Lemma 23 in [Che96]. Let C_{out} be the two-path cover output by PARGREEDY. Define a weighted digraph $OG'(S) = (S \cup S^R, A, w)$ by setting $w(e) = c^{lev(e)}$. Let $\succ$ be a total order on A satisfying the following condition: If either $w(e) > w(e')$, or $w(e) = w(e')$ and e is contained in C_{out} but e' is not contained in C_{out}, then $e \succ e'$. When $e \succ e'$, we say that e is *larger* than e'.

Let W be the maximum weight of a two-path cover of $OG'(S)$. Obviously, if the arcs in $OG'(S)$ are sorted in nonincreasing order using $\succ$, then given $OG'(S)$ as input to GREEDY (in place of $OG(S)$), GREEDY will output C_{out}. Thus, by Lemma 6.19, $w(C_{out}) \geq \frac{W}{3}$. On the other hand, $w(C_{out}) < c \cdot ov(C_{out})$ because $w(e) < c \cdot ov(e)$ for all $e \in A$. Therefore, $c \cdot ov(C_{out}) > \frac{W}{3} \geq \frac{opt_{cov}(S)}{3}$ since $w(e) \geq ov(e)$ for all $e \in A$. Recall that $c = 1 + \frac{\epsilon}{3}$. We now have $ov(C_{out}) > \frac{opt_{cov}(S)}{3c} = \frac{opt_{cov}(S)}{3+\epsilon}$. ∎

Theorem 6.24 There is a parallel approximation algorithm that produces a superstring of a given set S of n strings by which the amount of compression achieved is at least $\frac{1}{3+\epsilon}$ optimal for any $\epsilon > 0$. It runs in $O(\log n \cdot \log_{1+\epsilon/3} |S|)$ expected time with $O(|S|^2)$ processors on a CRCW PRAM or in $O(\log^2 n \cdot \log_{1+\epsilon/3} |S|)$ (deterministic) time with $O(|S|^2 + n^4/\log n)$ processors on an EREW PRAM.

Proof. To find a superstring of a given set S of n strings, we first construct $OG(S)$, next invoke the algorithm PARGREEDY to find a two-path cover C_{out} in $OG(S)$ with weight at least $\frac{opt_{cov}(S)}{3+\epsilon}$, and finally output the string associated with one of the two Hamiltonian legal path in C_{out}. By Fact 6.17 and Lemma 6.23, the amount of compression achieved by the output superstring is at least $\frac{1}{3+\epsilon}$ optimal.

As pointed out in [Che96], $OG(S)$ can be constructed from S in $O(\log |S|)$ time with $|S|^2$ processors on an EREW PRAM. Let us analyze the complexity of PARGREEDY. Clearly, steps 1, 2, and 3 can be done in $O(\log n)$ time with n^2 processors on an EREW PRAM. Noting that $MaxLev = O(\log_{1+\epsilon/3} |S|)$, we see that the while-loop is executed $O(\log_{1+\epsilon/3} |S|)$ times. Since steps 4.1 and 4.3 use no more resources than step 4.2, step 4 can be done in $O(\log n \cdot \log_{1+\epsilon/3} |S|)$ expected time with $O(n^2)$ processors on a CRCW PRAM by Corollary 6.21 or in $O(\log^2 n \cdot \log_{1+\epsilon/3} |S|)$ time using $O(n^4/\log n)$ processors on a CRCW PRAM by Corollary 6.22. The string associated with one of the two Hamiltonian legal path in C_{out} can be computed in $O(\log |S|)$ time with $|S|$ processors on an EREW PRAM. Thus, we can compute a superstring of S in $O(\log n \cdot \log_{1+\epsilon/3} |S|)$ expected time with $O(|S|^2)$ processors on a CRCW PRAM or in $O(\log^2 n \cdot \log_{1+\epsilon/3} |S|)$ (deterministic) time with $O(|S|^2 + n^4/\log n)$ processors on an EREW PRAM. ∎

Finally, we explain how to remove the assumption that no string in S is a palindrome. Suppose some strings in S are palindromes. We redefine $OG(S)$ as follows. For each string s in S, we introduce two vertices one of which corresponds to s and the other corresponds to s^R. That is, we treat s and s^R as different *vertices* in $OG(S)$, even if s is a palindrome. The edges of $OG(S)$ and their weights are defined as before. Redefining $OG(S)$ in this way, we give no influence on the above discussions.

6.3 Parallel algorithms for the MLF problem

6.3.1 Reducing the MLF problem to a simpler one

We first show a reduction of the MLF problem to a simpler one. To reduce the problem, we need additional definitions. A graph $G = (V, E)$ is said to be *proper* if V can be partitioned into two subsets X and Y such that for every $y \in Y$, (a) $X \cup \{y\}$ is an LFS in G and (b) $|X \cap N_G(y)| = 2$. To explicitly show this partition of V, we write $G = (X \cup Y, E)$. Hereafter, by a proper graph $G = (X \cup Y, E)$, we always mean that for every $y \in Y$, $X \cup \{y\}$ is an LFS in G and $|X \cap N_G(y)| = 2$. A *valid* MLFS in a proper graph $G = (X \cup Y, E)$ is an MLFS U in G with $X \subseteq U$. The *restricted* MLF problem is to find, given a proper graph $G = (X \cup Y, E)$, a valid MLFS in G. In the rest of this section, we describe an NC reduction from the MLF problem to the restricted MLF problem.

Algorithm *REDUCE*
Input: A graph $G = (V, E)$ with n vertices.
Output: A proper graph $H = (X \cup Y, E_H)$ such that $X \cup Y \subseteq V$, $H = G[X \cup Y]$, and every valid MLFS in H is an MLFS in G.
1. Initialize X to be an MIS in G.
2. Perform the following steps twice:
   2.1. Compute W, the set of all $v \in V - X$ such that $X \cup \{v\}$ is an LFS in G and $|N_G(v) \cap X| = 1$.
   2.2. Construct a graph $G' = (W, E_{G'})$, where $E_{G'}$ consists of all $\{v_1, v_2\}$ such that $\{v_1, v_2\} \in E$ or $N_G(v_1) \cap N_G(v_2) \cap X \neq \emptyset$.
   2.3. Compute an MIS I in G'.
   2.4. Add the vertices in I to X.
3. Set Y to be the set of those $v \in V - X$ such that $X \cup \{v\}$ is an LFS in G and $|N_G(v) \cap X| = 2$.

Let X_0 be the MIS in G found at step 1. For $i = 1, 2$, let W_i, G'_i, I_i, and X_i be the contents of the variables W, G', I, and X at the end of the ith execution of the steps 2.1 through 2.4, respectively.

Lemma 6.25 X_2 is an LFS in G.

Proof. X_0 is an MIS in G and is hence an LFS in G. We next prove that X_1 is an LFS in G. First note that $X_1 = X_0 \cup I_1$. By steps 2.1 through 2.3, each $v \in I_1$ has degree 1 in the graph $G[X_1]$. This together with the acyclicness of $G[X_0]$ implies that $G[X_1]$ is acyclic. We claim that every $u \in X_0$ has degree at most 2 in the graph $G[X_1]$. This can be seen by noting that each $u \in X_0$ is adjacent to at most one vertex $v \in I_1$ in the graph $G[X_1]$ and that $X_0 \cup \{v\}$ is an LFS in G for all $v \in I_1$. The claim implies that X_1 is an LFS in G. A similar discussion shows that X_2 is also an LFS in G. ∎

Lemma 6.26 For all $v \in V - X_2$, $X_2 \cup \{v\}$ is not an LFS in G or $|N_G(v) \cap X_2| \geq 2$.

Proof. Since X_0 is an MIS in G, it holds that $W_2 \subseteq W_1$. From this, we can observe that $V - X_2$ is the union of $V - (W_1 \cup X_0)$, $W_1 - (W_2 \cup I_1)$, and $W_2 - I_2$ which are pairwise disjoint. Let v be an arbitrary vertex in $V - X_2$. One of the following three cases must occur.

Case 1: $v \in V - (W_1 \cup X_0)$. Then, by the definition of W_1, $X_0 \cup \{v\}$ is not an LFS in G or $|N_G(v) \cap X_0| \geq 2$. Since $X_0 \subseteq X_2$, it follows that $X_2 \cup \{v\}$ is not an LFS in G or $|N_G(v) \cap X_2| \geq 2$.

Case 2: $v \in W_1 - (W_2 \cup I_1)$. Then, by the definition of W_2, $X_1 \cup \{v\}$ is not an LFS in G or $|N_G(v) \cap X_1| \geq 2$. Since $X_1 \subseteq X_2$, it follows that $X_2 \cup \{v\}$ is not an LFS in G or $|N_G(v) \cap X_2| \geq 2$.

Case 3: $v \in W_2 - I_2$. Since $v \in W_1$, $|N_G(v) \cap X_0| = 1$. Let u be the unique vertex in $N_G(v) \cap X_0$. Because $v \in W_1 - I_1$ and I_1 is an MIS in G'_1, there is a $w \in I_1$ with $\{v, w\} \in E_{G'_1}$. By the construction of G'_1, $\{v, w\}$ belongs to E or there is some $x \in X_0$ with $x \in N_G(v) \cap N_G(w)$. In the former case, $w \in N_G(v) \cap I_1$ and $\{u, w\} \subseteq N_G(v) \cap X_1$, implying that $|N_G(v) \cap X_2| \geq 2$. Thus, we may assume the latter case. Then, we have $x = u$ since $x \in X_0 \cap N_G(v) = \{u\}$. Therefore, $w \in N_{G[X_1]}(u)$. On the other hand, since $v \in W_2 - I_2$ and I_2 is an MIS in G'_2, there is a $w' \in I_2$ with $\{v, w'\} \in E_{G'_2}$. By the construction of G'_2, $\{v, w'\}$ belongs to E or there is some $y \in X_1$ with $y \in N_G(v) \cap N_G(w')$. In the former case, $w' \in N_G(v) \cap I_2$ and $\{u, w'\} \subseteq N_G(v) \cap X_2$, implying that $|N_G(v) \cap X_2| \geq 2$. Thus, we may assume the latter case. Then, we have $y = u$ since $|N_G(v) \cap X_1| = 1$ and $\{u, y\} \subseteq X_1 \cap N_G(v)$. Therefore, $w' \in N_{G[X_2]}(u)$. Recalling that $w \in N_{G[X_1]}(u)$, we now have $\{w, w'\} \subseteq N_{G[X_2]}(u)$. Hence, $\{v, w, w'\} \subseteq N_{G[X_2 \cup \{v\}]}(u)$. Obviously, $v \neq w$ and $v \neq w'$. Since $w \in I_1$, $w' \in I_2$, and $I_1 \cap I_2 = \emptyset$, it follows that $w \neq w'$. Therefore, $|N_{G[X_2 \cup \{v\}]}(u)| \geq 3$. This implies that $X_2 \cup \{v\}$ is not an LFS in G. ∎

Theorem 6.27 The MLF problem can be reduced to the restricted MLF problem in $O(\log n)$ expected time with n^2 processors on a CRCW PRAM, in $O(\log^2 n)$ time with n^4 processors on an EREW PRAM, or in $O(\log^3 n)$ time with n^2 processors on an EREW PRAM.

Proof. By Lemma 6.25 and Lemma 6.26, Algorithm *REDUCE* is clearly correct. This gives a reduction from the MLF problem to the restricted MLF problem. It remains to analyze the complexity of Algorithm *REDUCE*. Steps 1 and 2.3 can be done in $O(\log n)$ expected time with n^2 processor on a CRCW PRAM [Lub86], in $O(\log^2 n)$ time with n^4 processor on an EREW PRAM [Lub86], or in $O(\log^3 n)$ time with n^2 processor on an EREW PRAM [GS89]. It is not so difficult to see that the other steps can be done in $O(\log n)$ time using n^2 processors on an EREW PRAM. This establishes the theorem. ∎

In the next subsections, we present parallel algorithms for solving the restricted MLF problem.

6.3.2 Description of the RNC algorithm

The top-level structure of the RNC algorithm is as follows.

Algorithm *FOREST*
Input: A proper graph $G = (X \cup Y, E)$.
Output: A valid MLFS U in G.
1. Set $U := X$ and construct a new graph $G' = (Y, E')$, where E' is the set of all edges $\{y_1, y_2\}$ such that $\{y_1, y_2\} \in E$ or $X \cap N_G(y_1) \cap N_G(y_2) \neq \emptyset$.
2. While G' is not empty, perform the following steps:
   2.1. Select an independent set I in G' such that each connected component of $G[U \cup I]$ contains at most one vertex of I.
   2.2. Add the vertices in I to U.
   2.3. Delete from G' all vertices y such that $y \in I$ or $U \cup \{y\}$ is not an LFS in G.
3. Output U.

An easy induction shows that Algorithm *FOREST* correctly computes a valid MLFS of G. We need to specify how to implement the steps 2.1 and 2.3. In order to perform the two steps fast, the algorithm maintains an array R for which the following is an *invariant*: For each vertex $u \in U$, $R[u] = u$ if $\mathrm{d}_{G[U]}(u) = 0$, $R[u] = v$ if $\mathrm{d}_{G[U]}(u) = 1$, where v is the other vertex of degree 1 in the connected component of $G[U]$ containing u. (*Comment:* Vertices u in U with $\mathrm{d}_{G[U]}(u) = 2$ are irrelevant to R.) It is necessary to insert a new step between steps 1 and 2 in Algorithm

FOREST to initialize R so that the invariant is true before the first execution of the while-loop. Since the details are trivial, we omit them.

To implement step 2.1, we need a definition and a lemma due to Luby [Lub86]. Following [Koz92], we say that a vertex v in a simple graph H is *good* if $\sum_{u \in N_H(v)} \frac{1}{\mathrm{d}_H(u)} \geq \frac{1}{3}$. The following lemma is known:

Lemma 6.28 [Lub86, Koz92] Given a graph H, we can compute an independent set J in $O(1)$ expected time with $n+m$ processors on a CRCW PRAM such that for each good vertex v in H, the probability that $v \in N_H(J)$ is no less than a positive constant.

To implement step 2.1, we use the following three substeps:

2.1.1. Compute an independent set J of G' as in Lemma 6.28.

2.1.2. In parallel, for each vertex $u \in U$ with $\mathrm{d}_{G[U]}(u) \leq 1$, select u if $u = R[u]$, and randomly select one of u and $R[u]$ if $u < R[u]$.

2.1.3. Set $I := \{y \in J \mid$ the two vertices in $U \cap N_G(y)$ are both selected at step 2.1.2$\}$.

Lemma 6.29 For the set I computed in step 2.1.3, each connected component of $G[U \cup I]$ contains at most one vertex of I.

Proof. Fix an arbitrary connected component C of $G[U]$. Recall that C is a path. By Algorithm FOREST, only the endpoint(s) of C can be adjacent to a vertex of I in G. Moreover, if C has two endpoints, then exactly one of the two is selected in step 2.1.2. Thus, by the definition of I, at most one vertex u of C can be adjacent to a vertex of I in G, and this vertex u must be an endpoint of C. To show the lemma, it suffices to show that $|N_G(u) \cap I| = 1$. Towards a contradiction, assume that $|N_G(u) \cap I| \geq 2$. Let y_1 and y_2 be two vertices in $N_G(u) \cap I$. Note that either $u \in X$ or $u \in U - X$. In the former case, $\{y_1, y_2\}$ must be an edge in G' by step 1, contradicting that I is an independent set in G'. In the latter case, u must be in Y and hence $|N_G(u) \cap X| = 2$. Combining this with the fact that $X \subseteq U$, we see that u can not be an endpoint of C, a contradiction. This completes the proof. ∎

To update R and to implement step 2.3, we use the following substeps:

2.3.1. In parallel, for each $y \in I$, find the two vertices u_1 and u_2 in $N_G(y) \cap U$ and set $R[R[u_1]] := R[u_2]$ and $R[R[u_2]] := R[u_1]$.

2.3.2. Delete all vertices in I from G'.

2.3.3. In parallel, for each $u \in U$ with $\mathrm{d}_{G[U]}(u) = 2$, delete from G' all vertices in $N_G(u)$.

2.3.4. In parallel, for each vertex y in G', if the two vertices u_1 and u_2 in $N_G(y) \cap U$ satisfies that $R[u_1] = u_2$, then delete y from G'.

We claim that R is correctly updated in step 2.3.1. This simply follows from the fact that the two vertices in $N_G(y) \cap U$ both have degree at most 1 in $G[U]$.

Lemma 6.30 The vertices of G' deleted in step 2.3.3 or 2.3.4 are exactly those vertices y in G' such that $U \cup \{y\}$ is not an LFS in G.

Proof. It is clear that the vertices y of G' deleted in step 2.3.3 or 2.3.4 satisfy that $U \cup \{y\}$ is not an LFS in G. Let y be a vertex such that $U \cup \{y\}$ is not an LFS in G. Then, one of the following three cases must occur; (1) there is a vertex $u \in N_G(y) \cap U$ with $\mathrm{d}_{G[U \cup \{y\}]}(u) > 2$; (2) the two vertices in $N_G(y) \cap U$ are the two endpoints of a single path in $G[U]$; and (3) $\mathrm{d}_{G[U \cup \{y\}]}(y) > 2$. In the cases (1) and (2), y is deleted in step 2.3.3 and 2.3.4, respectively. We claim that the case (3) is impossible. Towards a contradiction, assume that the case (3) occurs. Then, since $|N_G(y) \cap X| = 2$, there must exist a vertex $u \in (U - X) \cap N_G(y)$. Since $U - X \subseteq Y$, u was

added to U in an earlier iteration of the while-loop in Algorithm *FOREST*. In that iteration, y must have been deleted in step 2.3.3 because $\mathrm{d}_{G[U]}(u) = 2$. This contradicts that y still remains in G' in the current iteration. ∎

6.3.3 Complexity Analysis

Suppose that the while-loop in Algorithm *FOREST* is executed t times. For $1 \leq i \leq t$, let U_i and G'_i be respectively the contents of the variables U and G' after the ith iteration, and let I_i and J_i be the two independent sets (in G'_i) computed in the ith iteration. For convenience, let U_0 and G'_0 be the contents of the variables U and G' before the first iteration. Our main task in this subsection is to prove an upper bound on the expected value of t. We begin with a helpful lemma.

Lemma 6.31 For every $0 \leq i \leq t-1$ and every good vertex v in G'_i, $\Pr[v \in N_{G'_i}(I_{i+1})]$ is no less than a positive constant c_1.

Proof. Fix an arbitrary good vertex v in G'_i. By Lemma 6.28, the probability that $v \in N_{G'_i}(J_{i+1})$ is no less than a positive constant. In other words, the probability that some neighbor of v in G'_i belongs to J_{i+1} is no less than a positive constant. On the other hand, if y is a neighbor of v and y is in J_{i+1}, then the probability that $y \in I_{i+1}$ is no less than $\frac{1}{4}$ by step 2.1.2. Therefore, it seems that the probability that some neighbor of v in G'_i belongs to I_{i+1} is no less than a positive constant. However, this intuitive inference is not correct and we need a rigorous proof. Fortunately, by step 2.1.2, we have the fact that $\Pr[y \in I_{i+1} \mid y \in J_{i+1}] \geq \frac{1}{4}$ for every vertex y in G'_i. Using this fact, we can modify the proof of Lemma 6.28 given in [Koz92] to prove that $\Pr[v \in N_{G'_i}(I_{i+1})]$ is no less than a positive constant. Since the details are not so difficult, we omit them. ∎

We proceed to the proof of the fact that the expected value of t is $O(\log n)$. To show the fact, we use a potential function argument. Before describing our potential function, we observe that for every $0 \leq i \leq t$, each vertex y in G'_i satisfies that $N_G(y) \cap U_i$ is equal to $N_G(y) \cap X$. We denote the two vertices in $N_G(y) \cap X$ by x_1^y and x_2^y. Now, we are ready to define our potential function Ψ as follows: For every $0 \leq i \leq t$,

$$\Psi(G'_i, U_i) = \sum_{\text{edge } e = \{y,z\} \text{ in } G'_i} \psi(e, U_i),$$

where $\psi(e, U_i) = (2 - \mathrm{d}_{G[U_i]}(x_1^y))(2 - \mathrm{d}_{G[U_i]}(x_2^y))(2 - \mathrm{d}_{G[U_i]}(x_1^z))(2 - \mathrm{d}_{G[U_i]}(x_2^z))$.

Lemma 6.32 For every $0 \leq i \leq t-1$, $\mathcal{E}(\Psi(G'_i, U_i) - \Psi(G'_{i+1}, U_{i+1})) \geq c_2 \cdot \Psi(G'_i, U_i)$ for some positive constant c_2.

Proof. Fix an arbitrary i with $0 \leq i \leq t-1$. Let m'_i be the number of edges in G'_i. For each edge $e = \{y, z\}$ in G'_i, let $Z_e = \psi(e, U_{i+1})$ if e is also in G'_{i+1} and $Z_e = 0$ otherwise, and let B_e be the event that $\{y, z\} \cap N_{G'_i}(I_{i+1}) \neq \emptyset$. Then, we have

$$\Psi(G'_i, U_i) - \Psi(G'_{i+1}, U_{i+1}) = \sum_{\text{edges } e \text{ in } G'_i} (\psi(e, U_i) - Z_e).$$

Fix an arbitrary edge $e = \{y, z\}$ in G'_i. Note that $\psi(e, U_i) \geq 1$. Assume that event B_e occurs. Then, $y \in N_{G'_i}(I_{i+1})$ or $z \in N_{G'_i}(I_{i+1})$. By symmetry, we may assume that $y \in N_{G'_i}(I_{i+1})$. Let w be a vertex in I_{i+1} with $y \in N_{G'_i}(w)$. Then, by the construction of G',

either $\{y, w\}$ is an edge in G or $X \cap N_G(y) \cap N_G(w) \neq \emptyset$. In the former case, the edge $\{y, w\}$ is not contained in G'_{i+1} since it must be deleted from G' in step 2.3.3 during the $i+1$st iteration, and hence $\psi(e, U_i) - Z_e = \psi(e, U_i) \geq 1$. In the latter case, $\{x_1^y, x_2^y\} \cap \{x_1^w, x_2^w\} \neq \emptyset$ and we can simply show that $\psi(e, U_i) - Z_e = \psi(e, U_i) - \psi(e, U_{i+1}) \geq 1$. Thus, in both cases, we have $\psi(e, U_i) - Z_e \geq 1$. This implies that $\mathcal{E}(\psi(e, U_i) - Z_e \mid B_e) \geq 1$. Therefore, $\mathcal{E}(\psi(e, U_i) - Z_e) \geq \mathcal{E}(\psi(e, U_i) - Z_e \mid B_e)\Pr[B_e] \geq \Pr[B_e]$. If in addition e is good, then by Lemma 6.31, $\mathcal{E}(\psi(e, U_i) - Z_e) \geq \Pr[B_e] \geq c_1$ for some constant $c_1 > 0$. Combining this with the fact that G_{i+1} is a subgraph of G_i, we now have

$$\begin{aligned}
\mathcal{E}(\Psi(G'_i, U_i) - \Psi(G'_{i+1}, U_{i+1})) &= \mathcal{E}\Big(\sum_{\text{edges } e \text{ in } G'_i - G'_{i+1}} (\psi(e, U_i) - Z_e)\Big) \\
&= \sum_{\text{edges } e \text{ in } G'_i - G'_{i+1}} \mathcal{E}(\psi(e, U_i) - Z_e) \\
&\geq \sum_{\text{good edges } e \text{ in } G'_i - G'_{i+1}} \mathcal{E}(\psi(e, U_i) - Z_e) \\
&\geq \sum_{\text{good edges } e \text{ in } G'_i - G'_{i+1}} c_1 \geq \frac{m'_i}{2} c_1.
\end{aligned}$$

The last inequality follows from the fact that at least half the edges in any simple graph are good [Koz92, MR95]. On the other hand, we have $\Psi(G'_i, U_i) \leq 16m'_i$. Thus, $\mathcal{E}(\Psi(G'_i, U_i) - \Psi(G'_{i+1}, U_{i+1})) \geq \frac{c_1}{32}\Psi(G'_i, U_i)$. This completes the proof. ∎

Lemma 6.33 $\mathcal{E}t = O(\log n)$.

Proof. Note that $\Psi(G'_0, U_0) \leq 16\binom{n}{2}$ and that the while-loop is iterated until $\Psi(G'_i, U_i) < 1$. Thus, by Lemma 6.32 and Theorem 1.3 in [MR95], we immediately have that $\mathcal{E}t \leq \int_1^{16\binom{n}{2}} \frac{1}{c_2 x} dx = O(\log n)$. This completes the proof. ∎

Theorem 6.34 An MLFS in an n-vertex graph G can be found in $O(\log n)$ expected time with n^2 processors on a CRCW PRAM.

Proof. By Theorem 6.27, it suffices to prove that Algorithm *FOREST* runs in $O(\log n)$ expected time with n^2 processors. Since there are only $O(n)$ edges between X and Y in G, step 1 takes $O(\log n)$ time with n^2 processors. Thus, by Lemma 6.33, it remains to show that steps 2.1 through 2.3 can be implemented in $O(1)$ time with n^2 processors.

By Lemma 6.28, step 2.1.1 can be done in $O(1)$ time with n^2 processors. Steps 2.1.2 and 2.1.3 use no more resources than step 2.1.1. Thus, the implementation of step 2.1 takes $O(1)$ time with n^2 processors. Implementing step 2.2 in $O(1)$ time using n processors is trivial. Now, consider the implementation of step 2.3. Recall that during any iteration of the while-loop, each vertex y in G' satisfies that $N_G(y) \cap U = N_G(y) \cap X$. Hence, the algorithm may compute $N_G(y) \cap U$ before entering the while-loop, and this takes $O(\log n)$ time with n^2 processors. Therefore, step 2.3.1 can be done in $O(1)$ time with n processors. The implementation of step 2.3.2 is trivial. It is not difficult to implement step 2.3.3 in $O(1)$ time with n^2 processors. Step 2.3.4 uses no more resources than step 2.3.1. In total, the implementation of step 2.3 takes $O(1)$ time with n^2 processors. This completes the proof. ∎

6.3.4 The NC algorithm

In this section, we obtain an NC algorithm for the MLF problem by derandomizing the randomized algorithm in section 6.3.2. Recall that the randomized algorithm consumes random bits in steps 2.1.1 and 2.1.2. We now consider how many random bits are sufficient. It has been shown in [Lub86, Koz92] that step 2.1.1 needs only $2\log n + O(1)$ random bits. We claim that step 2.1.2 needs only 1 random bit. To see this, we replace steps 2.1.2 and 2.1.3 with the following steps:

2.1.2'. Partition J into two subsets J_1 and J_2 such that both $U \cup J_1$ and $U \cup J_2$ are LFS's in G. (*Comment:* This step is possible because each connected component of $G[U \cup J]$ is a cycle or path.)

2.1.3'. Randomly set I to be J_1 or J_2.

It is easy to see that even if we do the above replacement, the algorithm remains correct, and more importantly, Lemma 6.31 and Lemma 6.32 still hold. Step 2.1.3' needs only 1 random bit, and hence the modified algorithm uses only $2\log n + O(1)$ random bits. Therefore, we can derandomize it using standard techniques. Since the details are rather standard, we omit them. Summarizing up, we obtain the following theorem:

Theorem 6.35 An MLFS in an n-vertex graph G can be found in $O(\log^2 n)$ time with $O(n^4)$ processors on an EREW PRAM.

6.4 Remarks

Restricting to bipartite graphs or sparse graphs (such as planar graphs), the MLF problem can be easily reduced to the MPS problem and hence has more efficient NC algorithms than Algorithm *FOREST*. Since the details are rather easy, we omit them here.

In this section, we point out an interesting open question related to this work. We say that a bipartite graph $G = (X, Y, E)$ is *special* if every $y \in Y$ has exactly 3 neighbors (in X). Consider the following problem: Given a special bipartite graph $G = (X, Y, E)$, we wish to find a maximal subset S of Y such that $G[X \cup S]$ is acyclic. We call this problem the *maximal triangular cactus* (MTC) problem. The MTC problem can be viewed as an extension of the restricted MLF problem. We describe an interesting application of the MTC problem below.

The *maximum planar subgraph* problem is, given a graph $G = (V, E)$, to find a maximum subset F of E such that the graph (V, F) is planar. This problem has applications in circuit layout and graph drawing, but is unfortunately NP-hard. Thus, it is of interest to design approximation algorithms for this problem. Note that the maximum planar subgraph of an n-vertex graph contains at most $3n - 5$ edges and that a spanning tree has exactly $n - 1$ edges. From this fact, we obtain a trivial approximation algorithm which always computes a planar subgraph whose size is at least $\frac{1}{3}$ optimal. Only recently, Călinescu *et al.* [CFFK96] gave the first approximation algorithm better than the trivial one.

To describe Călinescu *et al.*'s approximation algorithm, we need several definitions. A *triangular cactus* is a graph whose cycles (if any) are triangles and such that all edges appear in some cycle. A *triangular cactus in a graph* G is a spanning subgraph of G which is a triangular cactus. A triangular cactus H in G is *maximal* if there is no other triangular cactus in G which is a supergraph of H. It is easy to see that a maximal triangular cactus in a given graph can be computed in polynomial time. We are now ready to describe Călinescu *et al.*'s approximation algorithm. Given a (connected) graph G, their algorithm computes a maximal triangular cactus H in G, and then computes a spanning tree T in the graph obtained from G by merging each

connected component of H into a single vertex. The output of the algorithm is the union of H and T. In [CFFK96], it is shown that the size of the planar subgraph output by this approximation algorithm is at least $\frac{7}{18}$ optimal. We wish to parallelize this algorithm. To reach our goal, we only need to answer the question whether a maximal triangular cactus in a given graph can be computed in NC.

We claim that the problem of computing a maximal triangular cactus in a given graph can simply be reduced to the MTC problem. The reduction is as follows. Given a graph $G = (V, E)$, we construct a *special* bipartite graph $G' = (X, Y, E')$, where $X = V$, Y is the set of all triangles in G, and E' consists of all $\{x, y\}$ such that the vertex x appears in the triangle y. Clearly, there is a one-to-one correspondence between maximal triangular cactuses in G and maximal subsets S of Y such that $G'[X \cup S]$ is acyclic. Thus, the claim holds. Therefore, Călinescu *et al.*'s approximation algorithm can be parallelized if the MTC problem can be solved in NC. However, we do not know whether the MTC problem is in NC or not. This is an open question.

Chapter 7

NP-completeness of the problems on a simple structure

7.1 NP-completeness of Hi-Q

In this section, we consider a puzzle commonly known as Hi-Q. The board of the puzzle consists of 33 points arranged as in Figure 7.1 with all but one point placed with tokens. This special point, the center of the board, is named the *goal*. A move consists of jumping one token over an adjacent one onto an empty point. The jumping must be made either horizontally or vertically. When a token is jumped it is removed from the board. The objective of the puzzle is to make the board to a position with just one token on the goal. Thus a sequence of jumps which leads the board consisting of no tokens but on the goal will be an answer for the puzzle.

The puzzle is generalized in such a way that an initial position is given on the extended board of size $n \times n$, and a goal is also given on which only one token will finally be placed. Now call the puzzle *generalized Hi-Q*. The theorem in this section is the following:

Theorem 7.1 The question: Given generalized Hi-Q, is there an answer for the generalized puzzle? is NP-complete.

Proof. It is easy to see that the decision problem belongs to NP. It then remains to prove the completeness. We provide a polynomial time reducibility from the following known NP-complete problem: Hamiltonian Cycle Problem in planar digraph with degree bound two [Ple79].

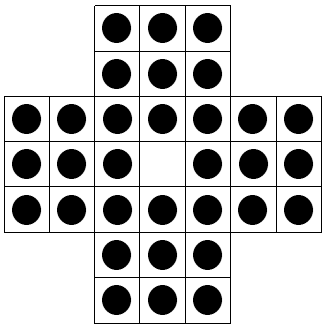


Figure 7.1: An initial board of Hi-Q

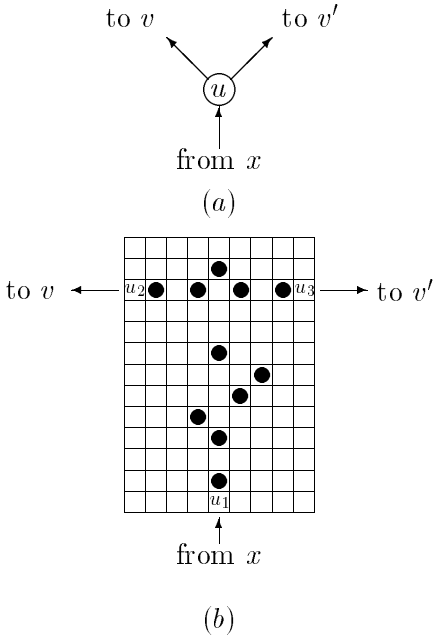


Figure 7.2: A type $\bigtriangledown$ node u of indegree one and outdegree two

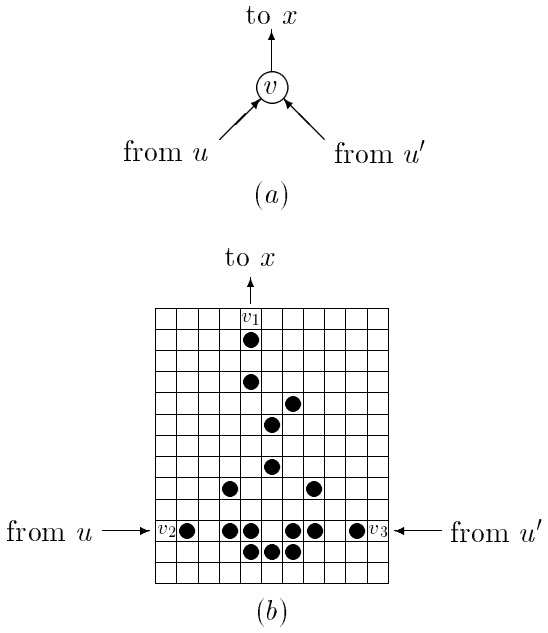


Figure 7.3: A type $\bigtriangleup$ node v of indegree two and outdegree one

Let $\mathcal{D}$ be the class of planar digraphs where each node is either indegree one and outdegree two, called type $\bigtriangledown$ (Figure 7.2(a)), or indegree two and outdegree one, called type $\bigtriangleup$ (Figure 7.3(a)).

Hamiltonian Cycle Problem of $\mathcal{D}$ is

Given: an element D of $\mathcal{D}$.

Question: Does D contain a Hamiltonian cycle?

Let D constitute a given instance of Hamiltonian Cycle Problem in planar digraph with degree bound two. We construct a corresponding instance of generalized Hi-Q. The board of the generalized puzzle will be constructed as follows:

1. For each node u of type $\bigtriangledown$, construct a pattern of token placement shown in Figure 7.2(b) and call it the *pattern* u. The points u_1 and u_2 are said to be the *exits* and u_3 be the *entry* of the pattern u. For each node v of type $\bigtriangleup$, construct similarly a pattern v as in Figure 7.3(b). The points v_2 and v_3 are the *entries* and the point v_1 is the *exit* of the pattern v.

2. For each edge (u, v), connect two patterns corresponding to u and v between the exit of u and the entry of v, using extension modules shown in Figure 7.4. Call c_1 the entry, and c_2 the exit in Figure 7.4. Two extension modules can be connected by identifying the exit of one extension module with the entry of another module. An extension module and a pattern of Figure 7.2(b) or 7.3(b) can be similarly connected.

Note that since D is planar, we can put patterns on the board so that the connection between them by the procedure 2 above may not cross.

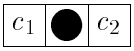
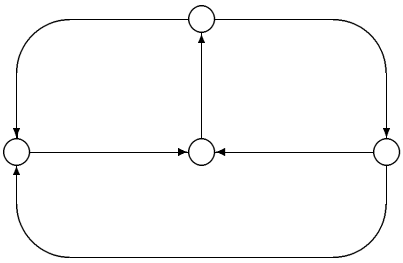
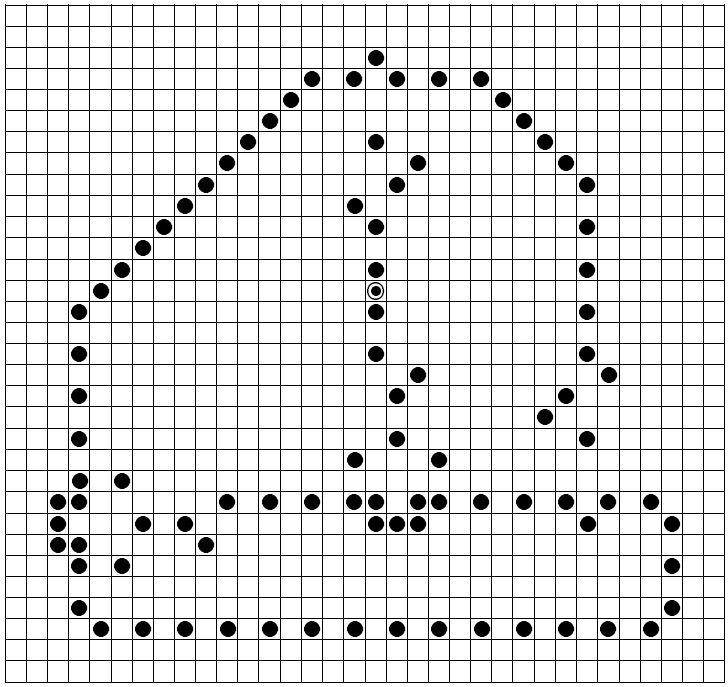


Figure 7.4: An extension module

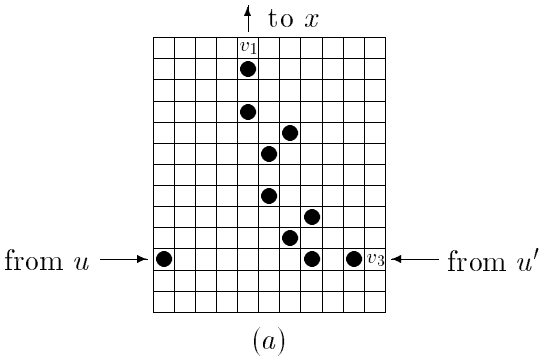


(a)



(b)

Figure 7.5: An example of the construction



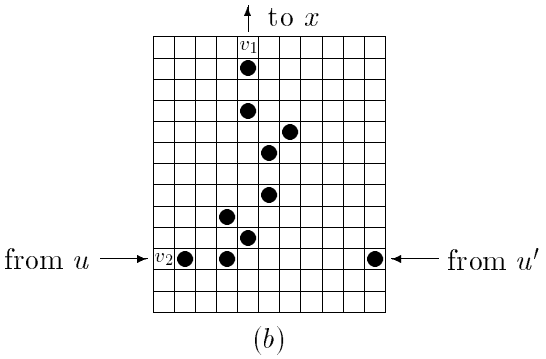
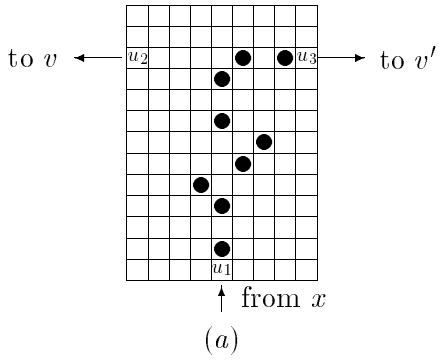


Figure 7.6: Placement after six jumps from Figure 7.3(b)



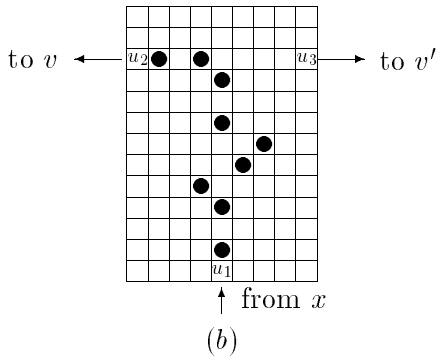


Figure 7.7: Placement after three jumps from Figure 7.2(b) with additional token on u_2 or u_3

Select one node of type $\bigtriangledown$, say z, and put a token on the entry z_1 of the pattern z. The point z_1 is the goal of the constructed generalized Hi-Q.

As an example, for a digraph shown in Figure 7.5(a), the constructed puzzle is shown in Figure 7.5(b) where a token surrounded by a circle denotes the goal.

Now we will show that D has a Hamiltonian cycle if and only if there is an answer in the constructed puzzle.

Assume that $D = (V, A)$ has a Hamiltonian cycle. For each edge $\tilde{e} = (\tilde{u}, \tilde{v})$ not on the Hamiltonian cycle, we first remove the tokens related to $\tilde{e}$ from the board: we note that $\tilde{u}$ should be of type $\bigtriangledown$ and $\tilde{v}$ be of type $\bigtriangleup$. Hence, the edge $\tilde{e} = (\tilde{u}, \tilde{v})$ should be either (u, v) or (u, v') in Figure 7.2(a), and be either (u, v) or (u', v) in Figure 7.3(a). Now suppose that $e = (u, v)$ is not on the Hamiltonian cycle, and that u, v are nodes as shown in Figure 7.2(a) and Figure 7.3(a), respectively. First, we make six jumps from Figure 7.3(b) to place a token on the entry v_2 from u as in Figure 7.6(a). (If e is of the form (u', v), a token is placed on the entry v_3 from u' as in Figure 7.6(b).) Next, the token on the entry jumps to remove the tokens on the extension modules from u to v, and it moves to the exit u_2 in the placement of Figure 7.2(b); with the token on the exit, we make three moves and the placement can be the one in Figure 7.7(a). (If e is of the form (u, v'), a token is placed on the exit u_3 to v', and then become the one in Figure 7.7(b).) After executing this procedure for every edge not on the Hamiltonian cycle, we note that all the placements of the forms Figures 7.2(b) and 7.3(b) would be the ones shown in Figure 7.7(a) or 7.7(b), and Figure 7.6(a) or 7.6(b), respectively, and that the tokens which remain on extension modules are of the edges belonging to the Hamiltonian cycle.

Then the token on the goal begins to jump to remove the tokens on the board along the direction of the edges in the Hamiltonian cycle, and finally one token can remain on the goal.

Assume that there is an answer in the constructed puzzle. We show that there is a Hamiltonian cycle in D. In order to prove this, we need some lemmas:

Lemma 7.2 Consider a placement of tokens constructed for nodes u and v shown in Figure 7.2(b) and Figure 7.3(b), respectively. We say that u_i and v_i, $1 \leq i \leq 3$ is a *gate*. Let $S \subseteq \{u_1, u_2, u_3\}$, and $T \subseteq \{v_1, v_2, v_3\}$. With an additional token on each gate of S (T), assume that every token on the figure is removed by a series of jumps, and that a token remains on each gate $\{u_1, u_2, u_3\} - S$ ($\{v_1, v_2, v_3\} - T$, respectively) thereafter. Then $S = \{u_1, u_2\}$ or $S = \{u_1, u_3\}$, and $T = \{v_2\}$ or $T = \{v_3\}$.

Proof. We first show that $T = \{v_2\}$ or $T = \{v_3\}$. It is obvious that $T \neq \phi$ and $v_1 \notin T$ from the construction of the placement of Figure 7.3(b). For $T = \{v_2, v_3\}$, we have examined every possible sequence of jumps by using a computer from the placement of Figure 7.3(b) with two additional tokens on v_2 and v_3, and found that no matter how tokens jump, at least one token remains on non-gate point in the figure. Thus, $T \neq \{v_2, v_3\}$.

We now show the proof on S. By the construction of the placement of Figure 7.2(b), we obtain that $u_1 \in S, S \neq \{u_1, u_2, u_3\}$, and $S \neq \{u_1\}$. Then the lemma follows. ∎

Remark. We note that for $T = \{v_2\}$ ($T = \{v_3\}$) in the placement of Figure 7.3(b), there are two possible independent moves which make all tokens of the placement removed: (1) a move for a token to jump to v_3 (v_2), and (2) having an additional token on v_2 (v_3, respectively), a move for a token to come up on v_1.

We say that a series of jumps is an *edge-directed move* if it removes tokens of placement of Figure 7.3(b) or Figure 7.2(b) in the direction from an entry to an exit, or if it removes tokens of extension modules in the same direction as the edge in D, where the extension modules are constructed from.

Lemma 7.3 There is exactly one edge-directed move in the constructed puzzle during the sequence of jumps to leave only one token on the goal.

Proof. Assume that an edge-directed move passes a placement of the form Figure 7.2(b) or Figure 7.3(b). Then Lemma 1 insists that after the passage there is still one edge-directed move. At the beginning of the puzzle, there is an edge-directed move starting from the goal. Therefore the lemma follows. ∎

We now proceed with the proof of the theorem. The existence of the sequence of jumps to remove all tokens but one on the goal implies that the edge-directed move visits every placement of Figures 7.2(b) and 7.3(b) one after another. Also the edge-directed move starts from the goal and ends at the goal. We have selected the node z of D of the form Figure 7.2(a) to choose z_1 as the goal. Now we can make a Hamiltonian cycle in D starting and ending at z by traversing corresponding nodes in order of placements that the edge-directed move visits. ∎

7.2 NP-complete problems on a 3-connected cubic planar graph

In this section, we show NP-completeness of the vertex cover (VC) problem on a 3-connected cubic planar graph of girth greater than 3. To this end, we first show the following lemma for a gadget shown in Figure 7.8(a).

Lemma 7.4 Let $U = \{v_1, v_2, v_3, v_4\}$. Then

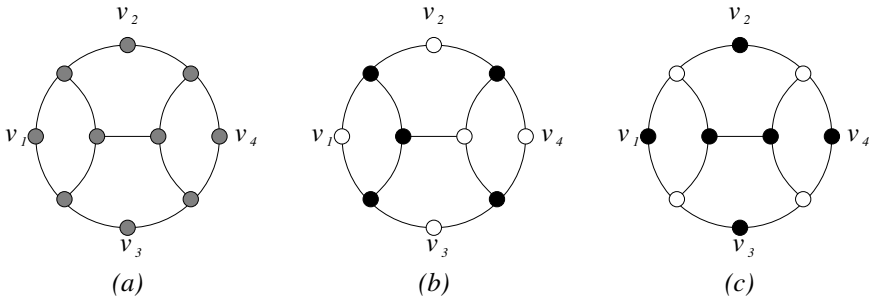


Figure 7.8: Gadget and its VCs

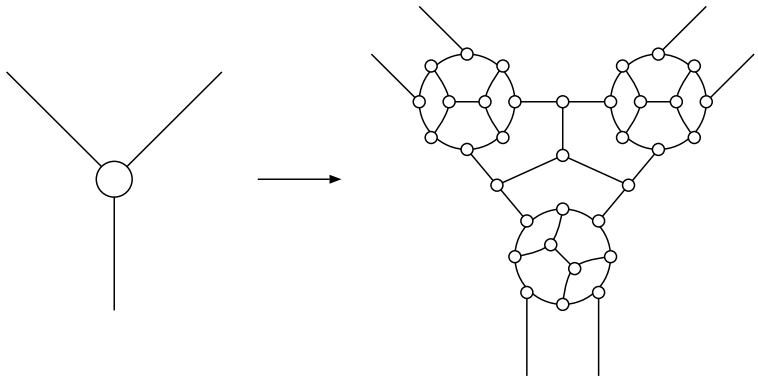


Figure 7.9: Replacement of a vertex by three gadgets

(1) the minimum VC of the gadget has size equal to 5, and it contains no vertex in U;
(2) for every VC on the gadget, if it contains at least one vertex in U, it has size greater than 5, and
(3) there exists a VC of size equal to 6, which contains every vertex in U.

Proof. We must select at least four vertices to cover eight edges around the gadget, and at least one of two center vertices to cover center edge. Thus every VC contains at least five vertices. Thus, the VC of size five is only one type shown in Figure 7.8(b) with black vertices, which contains no vertex in U. Now, it is not difficult to see that if a VC contains at least one vertex in U, it has size greater than 5. The VC shown in Figure 7.8(c) with black vertices completes the lemma. ∎

Now, we are ready to show main theorem in this section.

Theorem 7.5 The following problem is NP-complete: For a given 3-connected cubic planar graph G of girth greater than 3 and a positive integer k, is there a VC of size greater than k?

Proof. We provide a polynomial time reduction from the following known NP-complete problem [GJ79]: For a given cubic planar graph $G_0 = (V_0, E_0)$ and positive integer k_0, is there a VC of size greater than k_0? We replace all vertices of G_0 in parallel, each by three gadgets and four additional vertices, as depicted in Figure 7.9. Let $G = (V, E)$ be the resulting graph. Clearly, G is a 3-line-connected cubic planer graph with girth 4. Here, it is well-known that a cubic graph

is 3-connected if and only if it is 3-line-connected (see e.g. [BM76]). Thus, G is also 3-connected. From Lemma 7.4, it is not difficult to see that G_0 has a VC of size k_0 if and only if G has a VC of size $k_0 + 18 \mid n \mid$. ∎

For any graph $G = (V, E)$, $V' \subseteq V$ is a maximum independent set of G if and only if $V - V'$ is a VC of G [GJ79]. This implies the following corollary.

Corollary 7.6 The following problem is NP-complete: For a given 3-connected cubic planar graph G of girth greater than 3 and a positive integer k, is there a maximum independent set of size k?

Das and Goodrich proved NP-completeness of the following two problems; the VC problem on a 3-connected planar graphs, and the VC problem on a c-stellations of a 3-connected cubic planar graph for any constant $c \geq 4$. Here, a *stellation* of a face f in G as the insertion of a vertex in the interior of f that we then make adjacently to each vertex of f. A c-stellation of a graph is the result of performing c consecutive stellations on the graph (see [DG95] for detail). These VC problems contain the VC problem in Theorem 7.5. Thus, using the theorem, we can simplify the proofs in [DG95]. We remark that the theorem also simplify the proof of NP-completeness of the problem computing the asteroidal number of a graph (see [KKM96]).

7.3 NP-complete problems on a triangulation of the sphere

Whitney expressed planarity with the existence of dual graphs. Given a plane graph G, its *geometric dual graph* G^* is constructed as follows; place a vertex in each face of G (including the exterior region) and, if two regions have an edge e in common, join the corresponding vertices by an edge e^* crossing only e. The geometric dual graph of a 3-connected graph is always a simple graph (see [Har72] for detail). A *triangulation* of the sphere is a planar graph such that every face of the graph, including the exterior region, is a triangle. We first show the following lemma.

Lemma 7.7 Geometric dual graph of a 3-connected cubic planar graph is a triangulation of the sphere.

Proof. Let $G = (V, E)$ be a 3-connected cubic planar graph and $G^* = (V^*, E^*)$ be its geometric dual graph. Since G is cubic, $3 \mid V \mid = 2 \mid E \mid$ holds. By definition of the duality, $\mid E \mid = \mid E^* \mid$ holds. Since G is connected, Euler's Theorem states that $2 + \mid E \mid - \mid V \mid - \mid V^* \mid = 0$ (see [Koz92, Lecture 14] for example). Combining the equations, we get that $\mid E^* \mid = 3 \mid V^* \mid - 6$. Here, G^* is a triangulation of the sphere if and only if $\mid E^* \mid = 3 \mid V^* \mid - 6$ (see [Koz92, Theorem 14.10] for example). This completes the proof. ∎

For a given triangulation of the sphere and a positive integer k, we define two problems on them as follows.

Independent triangle set problem:

Question: Are there a triangle set of size k or more such that no two triangles in the set have an edge in common?

Shaping triangulation set problem:

Question: Are there a triangle set of size k or less such that every edge of the triangulation is had at least one triangle in the set?

We remark that the shaping triangulation problem is the problem to minimize the number of triangles to shape the triangulation of the sphere.

Theorem 7.8 Both the independent triangle set problem and the shaping triangulation problem are NP-complete.

Proof. By Lemma 7.7, the maximum independent set problem and the VC problem on a 3-connected cubic planar graph can be reduced to the independent triangle set problem, and the shaping triangulation problem, respectively. ∎

It is easy to see that the girth of a graph corresponds to the minimum degree of its geometric dual graph. Thus, the independent triangle set problem and the shaping triangulation problem remain NP-complete even if we add the restriction that the minimum degree is greater than 3 to a given triangulation.

Index

$\rightarrow$, 12, 17
$\vdash$, 8
δ-BPP, 12, 13, 22, 24
$^{\exists}\delta$-BPP, 12, 13, 23, 24, 26
δ-PP, 12, 13, 22, 24
$^{\exists}\delta$-PP, 12, 13, 23, 24, 26
δ-RP, 12, 13, 24
$^{\exists}\delta$-RP, 12, 13, 23, 24
δ-assignment, 13
δ-random source, 12, 21
δ-tree, 13, 21, 25
  best –, 14
  chimera –, 25
  clustered –, 14, 21, 25
  scattered –, 25
  worst –, 14

a-ary system, 12, 17
ABPP, 10
accepting
  – leaf, 13, 23
  – state, 9, 10
adversary argument, 30
algorithm
  approximation –, 44, 48, 54
  complete efficient –, 12
  efficient –, 11, 17
  Forest, 50
  Greedy, 45
  learning –, 29
  ParGreedy, 47
  randomized –, 32
  Reduce, 49
AM($poly$), 11
ancestor, 8
AND, 35
AND(k), 35
$\mathcal{A}_p$, 11
APP, 10
approximation algorithm, 44, 48, 54
arc, 7
  good –, 43, 47
ARP, 10, 24

best δ-tree, 14
bias, 1
  coin of –, 1, 17
  irrational –, 24, 26
  rational –, 24, 26
binary tree, 8
blank, 8
bound
  lower –, 29
  upper –, 29
BPP, 10, 13, 22, 24

chimera δ-tree, 25
choice
  0- –, 9
  1- –, 9
  nondeterministic –, 23, 24
  probabilistic –, 24
clustered δ-tree, 14, 21, 25
coin
  biased –, 1, 17
  fair –, 1
complete efficient algorithm, 12
compression, 45
computation
  efficiently –, 1
CRCW PRAM, 11, 40, 43, 48, 50, 53

de Morgan's rule, 15
degree, 7
  in–, 7
  out–, 7
  total –, 7
depth, 8
  – of a leaf, 11
  – of an efficient algorithm, 11
  – of an internal node, 11

derandomization, 44, 54
descendant, 8
deterministic Turing machine, 8
digraph, 7, 42
directed path set, 7
    maximal –, 7
DPS, 7, 47
DTM, 8

edge
    good –, 39
efficient algorithm, 11
    complete –, 12
    depth of –, 11
EREW PRAM, 11, 40, 47, 48, 54

function
    potential –, 40, 43, 52
    transition –, 8

girth, 7, 61, 64
good arc, 43, 47
good edge, 39
good vertex, 39, 43, 51
graph, 7
    3-connected –, 61, 63
    cubic –, 61, 63
    geometric dual –, 63
    overlap –, 45
    planar –, 61, 63
    proper –, 49
    representation of –, 11
    special bipartite –, 54
    underlying –, 7

Hamiltonian, 45
Hamiltonian Cycle Problem, 57
head, 7
Hi-Q, 57

ID, 8
indegree, 7
independent set, 5, 47, 50–52
    maximal –, 4
    maximum –, 6, 63
independent triangle set, 63
information theoretic lower bounds, 29
instantaneous description, 8
internal node, 8

IP, 11
isolated vertex, 7

leaf, 8
    accepting –, 13, 23
    rejecting –, 13, 23
learning algorithm, 29
legal, 45
LFS, 7
linear forest set, 7
    maximal –, 7
lower bound, 29

matching, 7, 39, 41, 42
    maximal –, 5, 7
    random –, 40, 42
mate, 45
maximal
    – directed path set, 7
    – edge-induced subgraph, 4
    – independent set, 4
    – linear forest set, 7
    – matching, 5, 7
    – path set, 7
    – triangular cactus, 54
    – vertex-induced subgraph, 4
maximum
    – assignment, 14
    – degree, 7
    – independent set, 6, 63
    – planar subgraph, 54
MDPS, 7, 42, 44
MEIS, 4
MIS, 4
MLF, 7, 49, 54
    restricted –, 49
MLFS, 7, 49
    valid –, 49
move, 8
MPS, 7, 44, 54
MTC, 54
MVIS, 4

n-sided die, 18
NC, 11, 50, 54
    – algorithm, 44, 47, 48, 50, 54
    – approximation algorithm, 44, 48, 54
neighborhood, 7
nondeterministic state, 9, 10

nondeterministic Turing machine, 9
NP, 9, 13, 23, 24, 57, 61, 63
– -complete, 57, 61, 63
NTM, 9

OR, 15, 29, 35
OR(k), 15, 29, 35
ordered tree, 8
outdegree, 7
overlap, 45
– graph, 45

P, 9, 13, 24
palindrome, 45, 48
PAR, 15, 29
PAR(k), 15, 29, 31
parallel random-access machine, 11
parent, 8
— of a subtree, 14
path, 7
path set, 7
directed –, 7
maximal –, 7
maximal directed –, 7
potential function argument, 40, 43, 52
PP, 10, 13, 24
PRAM, 11
CRCW –, 11, 40, 43, 48, 50, 53
EREW –, 11, 40, 47, 48, 50, 54
probabilistic state, 10
probabilistic Turing machine, 9
probability
– function, 9, 10
– of a δ-tree under a δ-assignment, 13
– of a leaf, 11
– of a node, 13
proper, 49
PSPACE, 9, 13, 23, 24, 26
PTM, 9

query, 15, 29

random matching, 40, 42
random source
δ- –, 12, 21
semi- –, 12, 21
randomized algorithm, 32
rejecting
– leaf, 13, 23
– state, 9, 10
reverse, 45
RNC, 11, 53
– algorithm, 39, 47, 53
root, 8
rooted directed tree, 8
RP, 10, 13, 24

$SBPP$, 12
scattered δ-tree, 25
semi-random source, 12, 21
shaping triangulation set, 63
shortest superstiring, 44
son, 8
left –, 8
right –, 8
$\mathcal{S}_p$, 12
special bipartite graph, 54
splitter, 32
SR_p, 12
state, 8
accepting –, 9, 10
final –, 8
nondeterministic –, 9, 10
probabilistic –, 10
rejecting –, 9, 10
start –, 8
STM, 10
stochastic Turing machine, 10
substring free, 45
subtree, 8
left –, 8
right –, 8
superstring, 45
superstring-with-flipping, 45
symbol, 8
input –, 8
tape –, 8

tail, 7
THR, 15, 29, 35
THR(k), 15, 29, 35
THR$_t$(k), 15, 29, 35
TM, 8
total degree, 7
transition function, 8
tree, 7, 8
binary –, 8
ordered –, 8

rooted –, 8
rooted directed –, 8
triangular cactus, 54
maximal –, 54
triangulation, 63
Turing machine, 8
deterministic –, 8
nondeterministic –, 9
probabilistic –, 9
stochastic –, 10
two-path cover, 45

underlying graph, 7
upper bound, 29

valid, 49
VC, 6, 61
vertex
– cover, 6, 61
good –, 39, 43, 51

weight, 45
winner, 15
worst δ-tree, 14, 21, 25

Bibliography

[AHK93] D. Angluin, L. Hellerstein, and M. Karpinski. Learning Read-Once Formulas with Queries. *Journal of the ACM*, 40(1):185–210, 1993.

[AKI87] H Adachi, H. Kamekawa, and S. Iwata. Shogi on $n \times n$ board is complete in exponential time (in Japanese) . *Trans. IEICE Japan*, J70-D(10):1843–1852, 10 87.

[And85] R. Anderson. A Parallel Algorithm for the Maximal Path Problem. In *Proc. 17th ACM Symp. on the Theory of Computing*, pages 33–37. ACM, 1985.

[Ang88] D. Angluin. Queries and Concept Learning. *Machine Learning*, 2:319–342, 1988.

[AW87] R. Ahlswede and I. Wegener. *Search Problems*. Wiley, 1987.

[Bab85] L. Babai. Trading Group Theory for Randomness. In *Proc. 17th ACM Symp. on the Theory of Computing*, pages 421–429. ACM, 1985.

[BDG88] J.L. Balcázar, J. Díaz, and J. Gabarró. *Structural Complexity I.* Springer-Verlag, 1988.

[BH96] N.H. Bshouty and L. Hellerstein. Attribute-efficient learning in query and mistake-bound models. In *Proc. 9th Ann. Conf. on Computing Learning Theory*, pages 235–243. ACM, 1996.

[BJL+91] A. Blum, T. Jiang, M. Li, J. Tromp, and M. Yannakakis. Linear Approximation of Shortest Superstrings. In *Proc. 23rd ACM Symp. on the Theory of Computing*, pages 328–336. ACM, 1991.

[BM76] J.A. Bondy and U.S.R. Murty. *Graph Theory with Applications.* The Macmillan Press, 1976.

[CFFK96] G. Călinescu, C.G. Fernandes, U. Finkler, and H. Karloff. A Better Approximation Algorithm for Finding Planar Subgraphs. In *Proc. 7th Ann. ACM-SIAM Symp. on Discrete Algorithms*, pages 16–25. ACM, 1996.

[CGPR94] A. Czumaj, L. Gąsieniec, M. Piotrów, and W. Rytter. Parallel and Sequential Approximation of Shortest Superstrings. In *Proc. 4th Scandinavian Workshop on Algorithm Theory*, pages 95–106. Springer-Verlag, 1994. LNCS 824.

[CH95] Z.-Z. Chen and X. He. Parallel Algorithms for Maximal Acyclic Sets. In *Proceedings of Aizu International Symposium on Parallel Algorithm/Architecture Synthesis*, pages 169–175. IEEE, 1995. to appear in *Algorithmica*.

[Che52] H. Chernoff. A Measure of Asymptotic Efficiency for Tests of a Hypothesis Based on the Sum of Observations. *Ann. of Math. Statist.*, 23:493–509, 1952.

[Che91] Z.-Z. Chen. A Randomized NC Algorithm for the Maximal Tree Cover Problem. *Information Processing Letters*, 40:241–246, 1991.

[Che95a] Z.-Z. Chen. A fast and efficient NC algorithm for maximal matching. *Information Processing Letters*, 55:303–307, 1995.

[Che95b] Z.-Z. Chen. The Maximal f-dependent Set Problem for Planar Graphs is in NC. *Theoretical Computer Science*, 143:309–318, 1995.

[Che96] Z.-Z. Chen. Parallel constructions of maximal path sets and applications to short superstrings. *Theoretical Computer Science*, 161:1–21, 1996. A preliminary version was presented at ICALP '96.

[CL95] A. Condon and R. Ladner. Interactive Proof Systems with Polynomially Bounded Strategies. *Journal of Computer and System Science*, 50:506–518, 1995.

[Cov73] T.M. Cover. On Determining the Irrationality of the Mean of a Random Variable. *The Annals of Statistics*, 1(5):862–871, 1973.

[DG95] G. Das and M.T. Goodrich. On the Complexity of Approximating and Illuminating Three-Dimensional Convex Polyhedra. In *4th International Workshop, WADS '95*, pages 74–85. Lecture Notes in Computer Science Vol. 955, Springer-Verlag, 1995.

[DGL91] K. Diks, O. Garrido, and A. Lingas. Parallel algorithms for finding maximal k-dependent sets and maximal f-matchings. In *ISA '91 Algorithms*, pages 385–395. Lecture Notes in Computer Science Vol. 557, Springer-Verlag, 1991.

[ET76] S. Even and R.E. Tarjan. A combinatorial problem which is complete in polynomial space. *Journal of the ACM*, 23(4):710–719, 1976.

[FIN+93] D. Feldman, R. Impagliazzo, M. Naor, N. Nisan, S. Rudich, and A. Shamir. On Dice and Coins: Models of Computation for Random Generation. *Information and Computation*, 104:159–174, 1993.

[FL81] A.S. Fraenkel and D. Lichtenstein. Computing a perfect strategy for $n \times n$ chess requires time exponential in n. *J. Combinatorial Theory*, 31(2):199–214, 1981.

[Fri84] J. Friedman. Constructing $O(n \log n)$ Size Monotone Formulae for the k-th Elementary Symmetric Polynomial of n Boolean Variables. In *Proc. 25th Symp. on Foundations of Computer Science*, pages 506–515. IEEE, 1984.

[Gil77] J. Gill. Computational Complexity of Probabilistic Turing Machines. *SIAM J. Comput.*, 6(4):675–695, Dec. 1977.

[GJ79] M.R. Garey and D.S. Johnson. *Computers and Intractability — A Guide to the Theory of NP-Completeness.* Freeman, 1979.

[GMR85] S. Goldwasser, S. Micali, and C. Rackoff. The Knowledge Complexity of Interactive Proof-Systems. In *Proc. 17th ACM Symp. on the Theory of Computing*, pages 291–304. ACM, 1985.

[GS86] S. Goldwasser and M. Sipser. Private Coins versus Public Coins in Interactive Proof Systems. In *Proc. 18th ACM Symp. on the Theory of Computing*, pages 59–68. ACM, 1986.

[GS89] M. Goldberg and T. Spencer. Constructing a Maximal Independent Set in Parallel. *SIAM J. Disc. Math.*, 2(3):322–328, 1989.

[Har72] F. Harary. *Graph Theory.* Addison-Wesley, 1972.

[Hof96] T. Hofmeister. Personal communication. 1996.

[HS70] W. Hoeffding and G. Simons. Unbiased Coin Tossing with a Biased Coin. *The Annals of Mathematical Statistics*, 41(2):341–352, 1970.

[HU79] J.E. Hopcroft and J.D. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley Publishing Company, 1979.

[HW91] R. Heiman and A. Wigderson. Randomized vs. Deterministic Decision Tree Complexity for Read-Once Boolean Functions. *Computational Complexity*, 1:311–329, 1991.

[HWWY94] J. Håstad, I. Wegener, N. Wurm, and S.-Z. Yi. Optimal Depth, Very Small Size Circuits for Symmetric Functions in AC^0. *Information and Computation*, 108:200–211, 1994.

[II86] A. Israeli and A. Itai. A Fast and Simple Randomized Parallel Algorithm for Maximal Matching. *Information Processing Letters*, 22:77–80, 1986.

[IK94] S. Iwata and T. Kasai. The Othello game on an $n \times n$ board is PSPACE-complete. *Theoretical Computer Science*, 123:329–340, 1994.

[IS86] A. Israeli and Y. Shiloach. An Improved Parallel Algorithm for Maximal Matching. *Information Processing Letters*, 22:57–60, 1986.

[JLD92] T. Jiang, M. Li, and D.-Z. Du. A note on shortest superstrings with flipping. *Information Processing Letters*, 44:195–199, 1992.

[Joh90] D.S. Johnson. A Catalog of Complexity Classes. In J. van Leeuwen, editor, *The Handbook of Theoretical Computer Science, Vol. I: Algorithms and Complexity*, pages 69–161. Elsevier, 1990.

[KKM96] T. Kloks, D. Kratsch, and H. Müller. Asteroidal sets in graphs. Technical Report Memorandum No. 1347, Universiteit Twente, 1996.

[Koz92] D.C. Kozen. *The Design and Analysis of Algorithms.* Springer-Verlag, 1992.

[KPS94] S.R. Kosaraju, J.K. Park, and C. Stein. Long Tours and Short Superstrings. In *Proc. 35th Symp. on Foundations of Computer Science*, pages 166–177. IEEE, 1994.

[KR90] R.M. Karp and V. Ramachandran. Parallel Algorithms for Shared-Memory Machines. In J. van Leeuwen, editor, *The Handbook of Theoretical Computer Science, Vol. I: Algorithms and Complexity*, pages 870–941. Elsevier, 1990.

[KR91] P. Kelsen and V. Ramachandran. On Finding Minimal 2-Connected Subgraphs. In *Proc. 2nd Ann. ACM-SIAM Symp. on Discrete Algorithms*, pages 178–187. ACM, 1991.

[KW85] R.M. Karp and A. Wigderson. A Fast Parallel Algorithm for the Maximal Independent Set Problem. *Journal of the ACM*, 32(4):762–773, 1985.

[KY76] D.E. Knuth and A.C. Yao. The complexity of nonuniform random number generation. In J.F.Traub, editor, *Algorithms and Complexity, New Directions and Results*, pages 357–428. Academic, New York, 1976.

[Lit88] N. Littlestone. Learning Quickly When Irrelevant Attributes Abound: A New Linear-threshold Algorithm. *Machine Learning*, 2:285–318, 1988.

[LR97] M. Liśkiewicz and R. Reischuk. Computational Limitations of Stochastic Turing Machines and Arthur-Merlin Games with Small Space Bounds. manuscript, July 1997.

[Lub86] M. Luby. A Simple Parallel Algorithm for the Maximal Independent Set Problem. *SIAM Journal on Computing*, 15(4):1036–1053, 1986.

[Lut90] J.H. Lutz. Pseudorandom Sources for BPP. *Journal of Computer and System Science*, 41:307–320, 1990.

[Meh82] K. Mehlhorn. On the Program size of Perfect and Universal Hash functions. In *Proc. 23rd Symp. on Foundations of Computer Science*, pages 170–175. IEEE, 1982.

[MR95] R. Motwani and P. Raghavan. *Randomized Algorithms*. Cambridge, 1995. 1995.

[Neu51] J. von Neumann. Various techniques used in connection with random digits. Notes by G.E.Forsythe. National Bureau of Standards. *Applied Math Series*, 12:36–38, 1951. Reprinted in von Neumann's Collected Works 5 (Pergamon Press, 1963), 768–770.

[NSS95] M. Naor, L.J. Schulman, and A. Srinivasan. Splitters and near-optimal derandomization. In *Proc. 36th Symp. on Foundations of Computer Science*, pages 182–191. IEEE, 1995.

[Pap83] C.H. Papadimitriou. Games Against Nature. In *Proc. 24th Symp. on Foundations of Computer Science*, pages 446–450. IEEE, 1983.

[Pap94] C.H. Papadimitriou. *Computational Complexity*. Addison-Wesley Publishing Company, 1994.

[Pic65] C. Picard. *Théorie des Questionnaires*. Gauthier-Villars, 1965.

[Ple79] J. Plesník. The NP-completeness of the Hamiltonian Cycle Problem in Planar Digraphs with Degree Bound Two. *Information Processing Letters*, 8(4):199–201, 1979.

[PS90] F.P. Preparata and M.I. Shamos. *Computational Geometry: An Introduction*. Springer-Verlag, 3rd edition, 1990.

[PV93] D. Pearson and V.V. Vazirani. Efficient Sequential and Parallel Algorithms for Maximal Bipartite Sets. *Journal of Algorithms*, 14:171–179, 1993.

[Rei80] S. Reisch. Gobang ist PSPACE–vollständig. *Acta Informatica*, 13(1):59–66, 1980.

[Rob83] J.M. Robson. The complexity of GO. *Proc. IFIP 1983*, pages 413–417, 1983.

[Rob84] J.M. Robson. N by N checkers is EXPTIME complete. *SIAM Journal on Computing*, 13(2):252–267, 1984.

[RW91] P. Ragde and A. Wigderson. Linear-size constant-depth polylog-threshold circuits. *Information Processing Letters*, 39:143–146, 1991.

[San71] E.S. Santos. Computability by Probabilistic Turing Machines. *Transactions of the American Mathematical Society*, 159:165–184, Sep. 1971.

[SC79] L.J. Stockmeyer and A.K. Chandra. Provably difficult combinatorial games. *SIAM Journal on Computing*, 8(2):151–174, 1979.

[Sch78] T.J. Schaefer. On the Complexity of Some Two-Person Perfect-Information Games. *Journal of Computer and System Science*, 16:185–225, 1978.

[Sha90] A. Shamir. IP=PSPACE. In *Proc. 31st Symp. on Foundations of Computer Science*, pages 11–15. IEEE, 1990.

[SM95] T. Shoudai and S. Miyano. Using Maximal Independent Sets to Solve Problems in Parallel. *Theoretical Computer Science*, 148:57–65, 1995.

[SV84] M.S. Satha and U.V. Vazirani. Generating Quasi-random Sequences from Slightly random Sources. In *Proc. 25th Symp. on Foundations of Computer Science*, pages 434–440. IEEE, 1984.

[SV86] M.S. Satha and U.V. Vazirani. Generating Quasi-random Sequences from Semi-random Sources. *Journal of Computer and System Science*, 33:75–87, 1986.

[SW84] Q.F. Stout and B. Warren. Tree Algorithms for Unbiased Coin Tossing with a Biased Coin. *The Annals of Probability*, 12(1):212–222, 1984.

[SW86] M. Saks and A. Wigderson. Probabilistic Boolean Decision Trees and the Complexity of Evaluating Game Trees. In *Proc. 27th Symp. on Foundations of Computer Science*, pages 29–38. IEEE, 1986.

[Tur89] J.S. Turner. Approximation Algorithms for the Shortest Common Superstring Problem. *Information and Computation*, 83:1–20, 1989.

[TY93] S.-H. Teng and F. Yao. Approximating Shortest Superstrings. In *Proc. 34th Symp. on Foundations of Computer Science*, pages 158–165. IEEE, 1993.

[UC97] R. Uehara and Z.-Z. Chen. Parallel Algorithms for Maximal Linear Forests. *The Transactions of the IEICE*, E80-A(4):627–634, 1997. A preliminary version was presented at Second Workshop on Randomized Parallel Computing of Eleventh International Parallel Processing Symposium (IPPS '97).

[UCH96] R. Uehara, Z.-Z. Chen, and X. He. Fast *RNC* and *NC* Algorithms for Finding a Maximal Set of Paths with an Application. In *Second Annual International Computing and Combinatorics Conference (COCOON '96)*, pages 209–218. Lecture Notes in Computer Science Vol. 1090, Springer-Verlag, 1996.

[UCH97] R. Uehara, Z.-Z. Chen, and X. He. Fast *RNC* and *NC* Algorithms for Maximal Path Sets. *Theoretical Computer Science*, 1997. to appear.

[Ueh95a] R. Uehara. Complexity Classes Characterized by Semi-Random Sources. In *Fundamental Studies on Computational Complexity*, pages 1–14. Research Institute for Mathematical Sciences, Vol. 943, Kyoto University, Kyoto, Japan, 1995.

[Ueh95b] R. Uehara. Efficient Simulations by a Biased Coin. *Information Processing Letters*, 56:245–248, 1995.

[Ueh97] R. Uehara. Collapse of PP with a Semi-Random Source to BPP. *Information Processing Letters*, 61:83–87, 1997.

[UI90] R. Uehara and S. Iwata. Generalized Hi-Q is NP-Complete. *The Transactions of the IEICE*, E73(2):270–273, 1990.

[UTW97] R. Uehara, K. Tsuchida, and I. Wegener. Optimal Attribute-Efficient Learning of Disjunction, Parity, and Threshold Functions. In *Third Europian Conference on Computational Learning Theory (EuroCOLT '97)*, pages 171–184. Lecture Notes in Artificial Intelligence Vol. 1208, Springer-Verlag, 1997.

[Vaz86] U. Vazirani. *Randomness, Adversaries and Computation.* PhD thesis, University of California, Berkeley, 1986.

[VV85] U.V. Vazirani and V.V. Vazirani. Random Polynomial Time is Equal to Slightly-random Polynomial Time. In *Proc. 26th Symp. on Foundations of Computer Science*, pages 417–428. IEEE, 1985.

[Weg87] I. Wegener. *The Complexity of Boolean Functions.* Wiley-Teubner, 1987.

List of Publications Related to the Thesis

[1] R. Uehara. Efficient Simulations by a Biased Coin. *Information Processing Letters*, 56:245–248, 1995. (Relation to Chapter 3.)

[2] R. Uehara. Collapse of PP with a Semi-Random Source to BPP. *Information Processing Letters*, 61:83–87, 1997. (Relation to Chapter 4.)

[3] R. Uehara, K. Tsuchida, and I. Wegener. Optimal Attribute-Efficient Learning of Disjunction, Parity, and Threshold Functions. In *Third Europian Conference on Computational Learning Theory (EuroCOLT '97)*, pages 171–184. Lecture Notes in Artificial Intelligence Vol. 1208, Springer-Verlag, 1997. (Relation to Chapter 5.)

[4] R. Uehara, Z.-Z. Chen, and X. He. Fast RNC and NC Algorithms for Maximal Path Sets. *Theoretical Computer Science*, 1997. to appear. (A preliminary version was presented at Second Annual International Computing and Combinatorics Conference (COCOON '96), pages 209–218. Lecture Notes in Computer Science Vol. 1090, Springer-Verlag, 1996.) (Relation to Section 6.1 and Section 6.2.)

[5] R. Uehara and Z.-Z. Chen. Parallel Algorithms for Maximal Linear Forests. *The Transactions of the IEICE*, E80-A(4):627–634, 1997. (A preliminary version was presented at Second Workshop on Randomized Parallel Computing of Eleventh International Parallel Processing Symposium (IPPS '97).) (Relation to Section 6.3 and Section 6.4.)

[6] R. Uehara and S. Iwata. Generalized Hi-Q is NP-Complete. *The Transactions of the IEICE*, E73(2):270–273, 1990. (Relation to Chapter 7.)