ON RANDOMIZING DECISION PROBLEMS: A SURVEY OF THE THEORY OF RANDOMIZED NP

P.M.W. Knijnenburg

RUU-CS-88-15
March 1988

Rijksuniversiteit Utrecht

Vakgroep informatica

Budapestlaan 6 3584 CD Utrecht
Corr. adres: Postbus 80.012 3508 TA Utrecht
Telefoon 030-53 1454
The Netherlands

ON RANDOMIZING DECISION PROBLEMS: A SURVEY OF THE THEORY OF RANDOMIZED NP

P.M.W. Knijnenburg

Technical Report RUU-CS-88-15
March 1988

Department of Computer Science
University of Utrecht
P.O.Box 80.012, 3508 TA Utrecht
The Netherlands

ON RANDOMIZING DECISION PROBLEMS: A SURVEY OF THE THEORY OF RANDOMIZED NP

P.M.W. Knijnenburg

RUU-CS-88-15
March 1988

Rijksuniversiteit Utrecht

Vakgroep informatica

Budapestlaan 6 3584 CD Utrecht
Corr. adres: Postbus 80.012 3508 TA Utrecht
Telefoon 030-53 1454
The Netherlands

ON RANDOMIZING DECISION PROBLEMS: A SURVEY OF THE THEORY OF RANDOMIZED NP

P.M.W. Knijnenburg

Technical Report RUU-CS-88-15
March 1988

Department of Computer Science
University of Utrecht
P.O.Box 80.012, 3508 TA Utrecht
The Netherlands

Contents

Chapter 1

Introduction

Cook's theory of $\mathcal{NP}$-completeness (Cook [1], Garey and Johnson [3]) has shown that many relevant combinatorial problems are computationally equivalent, in the sense that one such problem can be polynomially reduced to another one. The theory has given rise to the claim that $\mathcal{NP}$-complete problems are not solvable in polynomial time and hence, are not tractable, unless it is proved that $\mathcal{P} = \mathcal{NP}$. Yet much effort has been spent on proving the tractability of $\mathcal{NP}$-complete problems "in practice" or "on average", apart from circumventing their intractability by the use of approximation algorithms. Several $\mathcal{NP}$-complete problems have been proved to be solvable in polynomial time on average for given probability functions on their instances (see [6]).

It is natural to ask, whether there are $\mathcal{NP}$-complete problems which are *not* polynomially solvable on average, that is: is there an $\mathcal{NP}$-complete problem D and a probability distribution μ on its instances such that no algorithm that solves the problem, can run in expected polynomial time?

To settle this question, it is likely that one would have to consider all possible algorithms for $\mathcal{NP}$-problems and solve a question very similar to "$\mathcal{P} \stackrel{?}{=} \mathcal{NP}$". Therefore, the best one can hope for is to show that certain problems are *complete* (with respect to an appropriate concept of reduction) for this class of randomized problems.

In this paper we will present a theoretical framework proposed by Levin [8] (and extended by Gurevich [4]) to deal with "Random $\mathcal{NP}$-problems", and analyse a number of arguments in detail.

In his original paper Levin defined a "randomized decision problem" as being a pair (D, μ) where D is a decision problem and μ is a probability function on its instances. He formulated the concept of "computability in *polynomial on average* time" and gave a definition of reducibility between randomized decision problems. He also gave a terse proof of the completeness of a randomized version of the *Bounded Tiling* problem (the "cryptoanalytically inclined reader" is refered to [8] for details). Johnson [6] provided some intuition for Levin's ideas and corrected a minor flaw in the definition of "polynomial on average". Finally, Gurevich [4] formalized the theory in more detail and proved some basic and more sophisticated results. In particular, he proved a theorem, which we call the Coding Theorem, which lies at the core of the theory.

In this paper we will provide some insight into the theory of randomized decision problems. Because it is needed for the theory, we will give a new proof of the $\mathcal{NP}$-completeness of Bounded Tiling with respect to polynomial time reductions, which differs slightly from earlier proofs as e.g. in [10].

This paper is essentially self-contained. In chapter 2 we give the necessary preliminaries from complexity theory. In chapter 3 we give the definitions of randomized problems and randomized $\mathcal{NP}$ ($\mathcal{RNP}$) and define a generalized notion of reduction between these problems. We also prove some useful lemmas, due to Gurevich [4]. In chapter 4 we prove the completeness of Randomized Bounded Halting and a version of Randomized Bounded Tiling with respect to Ptime reductions. Some basic extensions to the theory are given in chapter 5; we briefly discuss a notion of *logspace* reducibility for randomized problems and prove some results for a randomized version of $\mathcal{PSPACE}$.

Chapter 2

Preliminaries

We assume familiarity with the basic concepts of formal language and automata theory; see, for example, [10]. Our basic model of computation is the *Turing machine*, defined by A.M. Turing [11]. A Turing machine consists of a finite state control unit and a one-way infinite tape which is used for input and as a "memory". The tape is divided into *tape squares* which each containing one symbol from the alphabet of the Turing machine. The tape squares are accessed by a *tape head* which is able to read, write or erase one symbol and move one square to the left or to the right at a time. Formally we can define:

Definition 2.1 1. *A* Turing machine *is a quintuple* $M = (K, \Sigma, \Delta, s, h)$ *where:*

- K *is a finite set of* states, *not containing* h;
- Σ *is an* alphabet, *containing the blank symbol* $\#$, *but not containing the special symbols* L *and* R *which denote the possible movements (left and right respectively) of the tape head;*
- $s \in K$ *is the* initial state;
- h *is the* halting state;
- $\Delta \subseteq K \times \Sigma \times (K \cup \{h\}) \times (\Sigma \cup \{L, R\})$ *is the* transition relation.

2. *If for every* $p \in K$ *and* $a \in \Sigma$ *there are at most one* $q \in K$ *and* $b \in \Sigma \cup \{L, R\}$ *such that* $(p, a, q, b) \in \Delta$, *then we call the Turing machine* deterministic. *It is called* non-deterministic *otherwise.*

Let M be a Turing machine. As usual we define a *configuration* of the Turing machine as a pair $(p, \sigma \underline{a} \tau)$ where $p \in K$ and $\sigma, \tau \in \Sigma^*$ and $a \in \Sigma$. The underscore denotes the position of the tape head, which is thus implicitly encoded in the configuration. The transition relation Δ induces a relation $\vdash$ on configurations:

$$\begin{array}{lll} (p, \sigma \underline{a} \tau) \vdash (q, \sigma' \underline{b} \tau') & \text{iff} & \sigma = \sigma', \tau = \tau' \text{ and } (p, a, q, b) \in \Delta, \\ & \text{or} & \sigma a = \sigma', \tau = b\tau' \text{ and } (p, a, q, R) \in \Delta, \\ & \text{or} & \sigma = \sigma' b, \tau = a\tau' \text{ and } (p, a, q, L) \in \Delta. \end{array}$$

The relation $\vdash^n$ is recursively defined by:

$$
\begin{aligned}
C_1 \vdash^0 C_2 \quad &\text{iff} \quad C_1 = C_2, \\
C_1 \vdash^1 C_2 \quad &\text{iff} \quad C_1 \vdash C_2, \\
C_1 \vdash^{n+1} C_2 \quad &\text{iff} \quad \exists C_3 . C_1 \vdash C_3 \wedge C_3 \vdash^n C_2.
\end{aligned}
$$

for configurations C_1, C_2 and C_3. We let $\vdash^*$ denote the reflexive, transitive closure of $\vdash$. We use the following conventions:

Convention 2.2 1. *On input w the initial tape contents of a Turing machine are $\underline{\#}w\#$. Thus the initial configuration is $(s, \underline{\#}w\#)$. We assume that inputs w do not contain any occurrence of the blank symbol $\#$.*

2. *$L(M) = \{w \in \Sigma^* \mid (s, \underline{\#}w\#) \vdash^* (h, \underline{\#})\}$ is called the* language recognized by *M. We say that a Turing machine M* recognizes *(or* accepts*) w if and only if $w \in L(M)$.*

3. *With every decision problem D we associate a Turing machine M_D, called the D-*machine*, such that $L(D) = L(M_D)$, where $L(D)$ denotes the set of "yes–instances" of D.*

Lemma 2.3 *There exists a computable coding function ρ from Turing machines to binary strings.*

For a proof of lemma 2.3, see e.g. [10]. It is possible to choose ρ such that ρ^{-1} is computable and total as well.

In complexity theory one associates with every Turing machine M a function $T_M(n)$ such that M accepts in time $T_M(n)$ if and only if for every n and $w \in L(M)$ with $|w| = n$ there is an $m \leqslant T_M(n)$ such that $(s, \underline{\#}w\#) \vdash^m (h, \underline{\#})$. We can interpret this function as (a bound on) the number of steps M takes to accept w as a function of the length of w. For every decision problem D, we say that M decides D in time T if and only if M_D accepts in time $T(n)$. Likewise, $S_M(n)$ is defined as (a bound on) the number of tape squares M uses to accept inputs of length n.

From the notion of decidability in time T, we can define the class of all languages, or decision problems, decidable in time T:

$$
\begin{aligned}
DTIME(T) = \{D \mid &\text{ there is a deterministic Turing machine } M \text{ such that} \\
&M \text{ decides } D \text{ in time } T\} \\
NTIME(T) = \{D \mid &\text{ there is a nondeterministic Turing machine } M \text{ such that} \\
&M \text{ decides } D \text{ in time } T\}
\end{aligned}
$$

Note that if we replace T in the above definitions by $\mathcal{O}(T)$, the classes remain the same (see e.g. [5]). We can now define the following classes:

Definition 2.4 1. $\mathcal{P} = \bigcup_{k \geqslant 0} DTIME(n^k)$

2. $\mathcal{NP} = \bigcup_{k \geqslant 0} NTIME(n^k)$

One of the central problems in complexity theory is whether $\mathcal{P}$ is equal to $\mathcal{NP}$ or not. The reader is referred to Garey and Johnson [3] for an exposition of the theory related to this problem.

A key tool in complexity theory is the notion of a *reduction*. Given two languages A and B, we say that A *reduces to* B *via* f (notation: $A \leqslant_f B$) if there exists a function $f : \Sigma_A^* \to \Sigma_B^*$, such that $w \in A$ if and only if $f(w) \in B$ for all $w \in \Sigma_A^*$. Here Σ_A and Σ_B denote the alphabets of A and B respectively. Usually the class of functions that are allowed for reductions is further restraint to e.g. the polynomial time or logspace computable functions. It can be shown that the classes of Ptime and logspace reductions (denoted by $\leqslant_p$ and $\leqslant_{log}$, respectively) are closed under composition, hence $\leqslant_p$ and $\leqslant_{log}$ are transitive relations on languages. Let $\leqslant$ be any reducibility relation.

Definition 2.5 *Let B be a language and C a class of languages.*

1. *B is* hard *for C with respect to $\leqslant$ iff $A \leqslant B$ for all $A \in C$.*
2. *B is* complete *for C with respect to $\leqslant$ iff*
   (a) *B is hard for C with respect to $\leqslant$;*
   (b) *$B \in C$.*

Cook showed in [1] that the SATISFIABILITY problem is complete for $\mathcal{NP}$ with respect to logspace reductions and hence that all problems in this class are solvable in polynomial time if and only if SATISFIABILITY is. Or equivalently, that the complexity of SATISFIABILITY is representative for the class $\mathcal{NP}$. We end this section with a useful lemma.

Lemma 2.6 *Let A be a decision problem over the alphabet Σ. Then there is a 1-1 logspace reduction of A to a decision problem A' over $\{0,1\}$.*

Proof.

Without loss of generality, we may assume that $|\Sigma| \geq 2$. Order the strings over Σ, first by length and then lexicographically. The required reduction assigns the n^{th} string over Σ to the n^{th} binary number. □

Chapter 3

Randomization

In [6] several $\mathcal{NP}$-complete problems are given that are computable in polynomial time on average for a given probability function on its instances. In this chapter we give a formalization of the notion of randomization and define a reduction between randomized problems as proposed by Levin [8].

3.1 Randomized problems

Definition 3.1 *A* randomized decision problem *is a pair (D, μ), where D is a decision problem and μ is a probability function on the instances of D.*

For any alphabet Σ, with an ordering on its elements, we can define a ordering on strings over Σ: first by length, then lexicographically. The successor of a string x is denoted by x^+. A probability function μ on a set A is any function $\mu : A \to [0,1]$ such that $\sum_{x \in A} \mu(x) = 1$. If μ is a probability function on Σ^*, then $\mu^*(x) = \sum_{y<x} \mu(y)$ is the corresponding probability distribution. We say that μ is positive if every value of μ is positive. In the sequel we use the symbol "$\propto$" to stand for " is proportional to".

Definition 3.2 1. *A probability function μ on a set A is called* standard *if*

   (a) *A is finite and $\mu(a) \propto |A|^{-1}$ for $a \in A$;*
   (b) *$A = \mathbf{N}^+$ $(= \mathbf{N})$ and $\mu(n) \propto n^{-2}$ $(\propto (n+1)^{-2})$ for $n \in \mathbf{N}^+$ $(\in \mathbf{N})$;*
   (c) *$A = \Sigma^*$ and $\mu(w) \propto (n+1)^{-2} \times |\Sigma|^{-n}$ where $w \in \Sigma^*$ and $n = |w|$;*
   (d) *$\mu = \mu_1 \times \mu_2$ for two standard probability functions μ_1 and μ_2.*

2. *A randomized decision problem (D, μ) is called* standard *if μ is.*

Let (D, μ) be a randomized decision problem. What does computability in polynomial time on average exactly mean? The first intuition would say that there must exist an algorithm of some time complexity $T(n)$ such that:

$$\sum_{w \in J} \mu(w) T(w) = \mathcal{O}(n^k)$$

for some $k \in \mathbf{N}$ and all n. Here J denotes the set of all instances of size n and $T(w)$ is the running time of the algorithm on input w. Note however that this notion is not machine independent: suppose the Turing machine M_A computes an algorithm is time T and machine M_B simulates M_A in time T^2. Suppose further that the algorithm on M_A runs in time n with probability $1 - 2^{-n/2}$ and in time $2^{n/2}$ otherwise. Then the algorithm is polynomial on average for machine M_A but *not* for machine M_B.

We can revise our first definition somewhat by requiring that, for some $k > 0$ and all n:

$$\sum_{w \in J} \mu(w) T(w)^{1/k} = \mathcal{O}(n),$$

with J as before. Next we would like μ to cover *all* instances of the problem, not just those of size n for some n. In order to achieve this we must divide by $|w|$:

Definition 3.3 *A function T from Σ^* to nonnegative reals is* polynomial on average *with respect to μ if there exists a positive integer k such that the expectation*

$$\sum_{w \in I} \mu(w) T(w)^{1/k} / |w| < \infty$$

where I is the domain of T.

Notice that this definition is independent of the encoding of a decision problem in some alphabet. The next definition extends definition 3.3 to functions and decision problems.

Definition 3.4 *Let μ be a probability function on some Σ^*.*

1. *A decision problem D over Σ^* is* expected Ptime decidable (EPtime decidable) *with respect to μ if a Turing machine M decides D in time T_M and T_M is polynomial on average with respect to μ.*

2. *A function $f : \Sigma^* \to \Delta^*$ for some alphabet Δ is* EPtime computable *with respect to μ if a Turing machine M computes f in time T_M and T_M is polynomial on average with respect to μ.*

For the purpose of this paper we define a slightly different notion of Ptime computability (cf. Gurevich [4]).

Definition 3.5 *A function f from Σ^* to reals is* Ptime computable *if there exists a Turing machine M which, given a string $w \in \Sigma^*$ and the unary notation for a natural number k, computes the binary notation for an integer i with $|f(x) - \frac{i}{2^k}| < \frac{1}{2^k}$.*

This definition states that $f(x)$ can be approximated by a binary fraction with arbitrary precision (which must be the case for computing machinery with finite precision arithmetic). In fact, this definition is only a slight variation on the definition of a "computable number", given by Turing in [11].

Definition 3.6 *$\mathcal{RNP}$ is the class of randomized decision problems (D, μ) with $D \in \mathcal{NP}$ and μ^* is Ptime computable.*

Note that Ptime computability of μ^* implies Ptime computability of μ but the converse is not necessarily true.

3.2 Reductions

Now we have defined "randomized problems" and "expected polynomial time computability", we can define a suitable notion of reduction between two randomized problems. It is obvious that a reduction $f : A \to B$ should not only map an instance w of A to an instance $f(w)$ of B, but also should map a "likely" instance to a "likely" instance. A reduction must ensure that whenever a problem A is EPtime decidable and another problem B reduces to it, that then B is also EPtime decidable. For this purpose we introduce the notion of *domination* (cf. Levin [8], Gurevich [4]).

Definition 3.7 *Let μ_1 and μ_2 be probability functions on some Σ_1^* and Σ_2^* respectively.*

μ_2 dominates μ_1 *if $\Sigma_1 = \Sigma_2$ and there is a polynomial p such that*

$$\mu_2(w) \times p(|w|) \geq \mu_1(w)$$

for all $w \in \Sigma_1^*$.

If μ_2 dominates μ_1 then the values of μ_2 may be arbitrarily larger but only polynomially smaller than the values of μ_1.

Lemma 3.8 *Let μ_1 and μ_2 be probability functions on some Σ^*. Let μ_2 dominate μ_1. If a problem D over Σ is Ptime decidable with respect to μ_2, then D is also Ptime decidable with respect to μ_1.*

Proof.

There exist integers k and l such that the following inequalities hold:

$$\begin{aligned} \infty &> \sum_{w \in I} \mu_2(w) T^{1/k}(|w|)/|w| \\ &\geq \sum_{w \in I} \mu_2(w) \times p(|w|) \times T^{1/l}(|w|)/|w| \\ &\geq \sum_{w \in I} \mu_1(w) \times T^{1/l}(|w|)/|w| \end{aligned}$$

□

Definition 3.9 *Let μ_1 and μ_2 be probability functions on Σ_1^* and Σ_2^* respectively, and $f : \Sigma_1^* \to \Sigma_2^*$.*

1. *f* transforms *μ_1 into μ_2 if*
$$\mu_2(y) = \sum_{f(x)=y} \mu_1(x).$$
2. *f* reduces *μ_1 to μ_2 if some probability function μ dominates μ_1 and f transforms μ into μ_2.*
3. *f* Ptime reduces *a randomized decision problem (D_1, μ_1) to a randomized decision problem (D_2, μ_2) (notation: $(D_1, \mu_1) \leqslant_p (D_2, \mu_2)$) if it reduces D_1 to D_2, reduces μ_1 to μ_2 and is Ptime computable.*

We can define a notion of completeness with respect to the above defined reduction $\leqslant_p$: a randomized decision problem (D, μ) is *complete* with respect to $\leqslant_p$ for a class of languages $\mathcal{C}$ iff $(D, \mu) \in \mathcal{C}$ and $(D', \mu') \leqslant_p (D, \mu)$ for each $(D', \mu') \in \mathcal{C}$. Note that in this definition we do not restrain the probability function μ' in any way. Next it is obvious that this notion of completeness is a sound one: if (D, μ) is complete for $\mathcal{RNP}$ with respect to Ptime reductions then for any decision problem (D', μ') in $\mathcal{RNP}$, (D', μ') is Ptime decidable with respect to μ' if and only if (D, μ) is Ptime decidable with respect to μ. For the following lemma we assume that f is a function that is not a constant.

Lemma 3.10 *μ_2 dominates the result of the f-transformation of μ_1 if and only if f reduces μ_1 to μ_2.*

Proof.

($\Longrightarrow$): Let μ_2 dominate the f-transformation of μ_1. By definition we have, for some polynomial p,

$$\mu_2(y) \times p(|y|) \geq \sum_{f(x)=y} \mu_1(x)$$

hence

$$\mu_2(y) \geq \sum_{f(x)=y} \frac{\mu_1(x)}{p(|f(x)|)}$$

Now, $|f(x)| \leqslant r(|x|)$ for some polynomial r, since f is Ptime computable; let $p(r(|x|)) = q(|x|)$ for an appropriate polynomial q.

Define:

$$\mu(x) = \frac{\mu_2(f(x))}{|f^{-1}(f(x))|}.$$

(Note that f need not be a injection, so $f^{-1}(f(x))$ may consist of several values.)

Then $\mu_2(y) = \sum_{f(x)=y} \mu(x)$ and since $|f^{-1}(f(x))|$ is polynomial in $|x|$,

$$\mu(x) \times s(|x|) \geq \mu_1(x)$$

for some polynomial s. So f reduces μ_1 to μ_2.

($\Longleftarrow$): Let f reduce μ_1 to μ_2. Then there exist a probability function μ and a polynomial p' such that

$$\mu(x) \times p'(|x|) \geq \mu_1(x)$$

and

$$\mu_2(y) = \sum_{f(x)=y} \mu(x).$$

Hence

$$\mu_2(y) \geq \sum_{f(x)=y} \frac{\mu_1(x)}{p'(|x|)}$$

and there exists a polynomial p such that

$$\mu_2(y) \times p(|y|) \geq \sum_{f(x)=y} \mu_1(x).$$

So μ_2 dominates the result of the f-transformation of μ_1. □

The following lemma states that Ptime reductions are transitive relations on languages.

Lemma 3.11 *Ptime reductions are closed under composition.*

Proof.

Let f Ptime reduce (D_1, μ_1) to (D_2, μ_2) and g Ptime reduce (D_2, μ_2) to (D_3, μ_3).

Then, by lemma 3.10,

$$\mu_3(y) \times p(|y|) \geq \sum_{g(x)=y} \mu_2(x)$$

and, by definition,

$$\mu_2(y) \geq \sum_{f(x)=y} \frac{\mu_1(x)}{q(|x|)}$$

for suitable polynomials p and q. So

$$\mu_3(y) \times p(|y|) \geq \sum_{g(f(x))=y} \frac{\mu_1(x)}{q(|x|)}$$

or

$$\mu_3(y) \times s(|y|) \geq \sum_{g(f(x))=y} \mu_1(x)$$

for some polynomial s and $g \circ f$ reduces μ_1 to μ_3. By ordinary reduction, $g \circ f$ reduces D_1 to D_3 and is Ptime computable. So $g \circ f$ Ptime reduces (D_1, μ_1) to (D_3, μ_3). □

Lemma 3.12 *Every $\mathcal{RNP}$-problem (D, μ) reduces to an $\mathcal{RNP}$-problem (D', μ') over {0,1}.*

Proof.

1. By lemma 2.6, there is a problem D' over $\{0, 1\}$ and a function f that reduces D to D' and is Ptime computable. D' is obviously in $\mathcal{RNP}$.
2. Define μ' to be: $\mu'(x) = \mu(f^{-1}(x))$. Clearly, f is injective, hence μ' is well-defined.

□

Lemma 3.13 *Every $\mathcal{RNP}$-problem (D, μ) over {0,1} reduces to a $\mathcal{RNP}$-problem (D, μ') where μ' is a positive probability function and every value of μ' is a finite binary fraction.*

Proof.

μ' is constructed as follows. Let x and y be binary strings and $dx = 2^{-2|x|}$ for $x \neq e$ and $de = 1/2$, where e denotes the empty string. Without loss of generality, $\mu^*(x) < 1$ for every x. Since μ^* is Ptime computable, there is a Ptime computable function N that assigns a binary fraction $N(x) = 0.y$ with precision $2|x|$ to each binary string x. So $|y| \leqslant 2|x|$ and $|\mu^*(x) - N(x)| < dx$.

Define μ' such that:

$$4\mu'(x) = N(x^+) - N(x) + 2dx$$

Then: $4\mu'(e) = N(e^+) + 1$ since $N(e) = 0$. Now, $4\mu'(x) = N(x^+) - N(x) + 2dx > (\mu^*(x^+) - dx) - (\mu^*(x^+) - dx) + 2dx = \mu(x)$. So μ' dominates μ.

We now must check that μ' is well-defined.

$$\begin{aligned} 4\mu'^*(x) &= 4\sum_{y<x}\mu(y) \\ &= \sum_{y<x}(N(y^+) - N(y) + 2dy) \\ &= \sum_{e<y<x}(N(y^+) - N(y) + 2dy) + N(e^+) + 1 \\ &= N(x) + \sum_{e<y<x} 2dy + 1 \text{ for } x \neq e \end{aligned}$$

and

$$\lim_{n=|x|\to\infty} 4\mu'^*(x) = 1 + 2\sum_{n\geq 1} 2^{-2n} \times 2^n + 1 = 4.$$

So $\sum_{x\in J}\mu'(x) = 1$ where J is the set of all instances of the problem, and μ' is a legitimate probability function. □

By virtue of lemmas 3.12 and 3.13, we may assume that every decision problem has $\Sigma = \{0,1\}$ and that every value of its probability function is a finite binary fraction.

3.3 Standardization

The *Randomized Bounded Halting Problem, $RH(M)$,* for a nondeterministic Turing machine M with binary input alphabet is defined as follows: given a string w and the unary notation of an integer $n > |w|$, does M halt on input w within n steps? The language recognized is:

$$L(RH(M)) = \{w01^n \mid M \text{ halts on } w \text{ within } n \text{ steps}\}$$

Its associated probability function μ_{hm} is defined by:

$$\mu_{hm}(w01^n) = \frac{6}{\pi^2} \times n^{-3} \times 2^{-k}$$

where $k = |w|$. This probability function is standard: choose n with probability n^{-2}; choose $k < n$ with probability $1/n$; choose a binary string of length k with probability 2^{-k}. The following result due to Gurevich [4] is important.

Theorem 3.14 (Coding Theorem) *For every $\mathcal{RNP}$-problem (D,μ) there is a nondeterministic Turing machine M_D which depends on D such that (D,μ) is Ptime reducible to $(RH(M_D),\mu_{hm})$.*

Proof.

Let x be an instance of (D,μ). Note that μ may be an arbitrary probability function, and is not necessarily standard. We must first define an encoding e that maps x onto

an equally likely $e(x)$ with respect to a standard probability function. Let x' be the shortest binary string such that

$$\mu^*(x) < 0.x'1 \leqslant \mu^*(x^+).$$

Claim $0.x'1 - 2^{-|x'1|} \leqslant \mu^*(x)$.

Proof of claim Assume $0.x'1 - 2^{-|x'1|} > \mu^*(x)$. Surely: $0.x'1 - 2^{-|x'1|} \leqslant \mu^*(x^+)$. So $\mu^*(x) < 0.x'1 - 2^{-|x'1|} \leqslant \mu^*(x^+)$. But $0.x'1 - 2^{-|x'1|}$ is (at least one digit) shorter than x'. Contradiction.

Similarly, $0.x'1 + 2^{-|x'1|} > \mu^*(x^+)$.

Now, $\mu^*(x) - \mu^*(x^+) = \mu(x) < 2 \times 2^{-|x'1|}$.

Define the encoding:

$$e(x) := \textbf{if } 2^{-|x|} > \mu(x) \textbf{ then } 0x \textbf{ else } 1x'.$$

$e(x)$ is tagged by a binary digit to signal whether $e(x)$ is a "disguise" for x or not.

The desired nondeterministic Turing machine M_D is defined as follows:

on input bw, where b is a binary digit:

1. **if** $b = 0$ **then if** $2^{-|w|} \leqslant \mu(w)$ **then** loop {wrong input} **else** $x := w$; goto (4).
2. Find the unique x with $\mu^*(x) < 0.w1 \leqslant \mu^*(x^+)$. Recompute the unique shortest x' such that $\mu^* < 0.x'1 \leqslant \mu^*(x^+)$.
3. **if** $2^{-|x|} > \mu(x)$ **or** $x' \neq w$ **then** loop {wrong input}.
4. Simulate the D-machine on x.

There exists a polynomial q such that M_D has a halting computation on $e(x)$ within at most $q(|x|)$ steps if and only if M_D has a halting computation if and only if $x \in L(D)$.

The desired Ptime reduction f is defined as:

$$f(x) = e(x)01^{q(|x|)}$$

which is indeed Ptime computable. The probability function μ_{hm} of $RH(M_D)$ dominates the result of the f-transformation of μ, because:

$$\frac{\pi^2}{3} \times |f(x)|^3 \times \mu_{hm}(f(x)) > 2 \times 2^{-|e(x)|} > \mu(x)$$

hence f reduces μ to μ_{hm} by lemma 3.10. □

The main purpose of the Coding Theorem is to establish the fact that *every* $\mathcal{RNP}$-problem (D, μ), however special μ might be, Ptime reduces to an equivalent problem with a standard probability function. This fact is crucial in the next chapter.

3.4 Weak reducibility

As Levin [8] remarked in his original paper, we can modify definitions 3.7 and 3.9 somewhat to get a notion of *weak* or *expected polynomial time* reducibility. We give here a definition due to Gurevich [4].

Definition 3.15 *Let μ_1 and μ_2 be probability functions on Σ^*, D_1 and D_2 decision problems over Σ^* and $f : \Sigma^* \to \Sigma^*$.*

1. μ_1 weakly dominates μ_2 *if there is a function p such that*

$$\mu_2(w) \times p(|w|) \geq \mu_1(w)$$

   for all $w \in \Sigma^*$ and p is EPtime computable with respect to μ_2.

2. f EP-reduces μ_1 *to μ_2 if some μ weakly dominates μ_1 and f transforms μ into μ_2.*

3. f EPtime reduces *a randomized decision problem (D_1, μ_1) to a randomized decision problem (D_2, μ_2) if it reduces D_1 to D_2, EP-reduces μ_1 to μ_2 and is EPtime computable with respect to μ_1.*

It is easy to see that whenever a problem (D, μ) is EPtime decidable and a problem (E, ν) EPtime reduces to it, that (E, ν) is also EPtime decidable.

In [4] several interesting theorems are proved concerning EPtime reducibility.

Chapter 4

Completeness

In this chapter we prove the existence of two complete problems with respect to Ptime reductions in the class $\mathcal{RNP}$. The first problem is rather basic: it is the randomized version of bounded halting, a problem which, in some formulation or another, is complete for every class of decision problems. The second problem is the randomized version of bounded tiling. This problem was first claimed to be $\mathcal{NP}$-complete by Levin in [7] and was treated more thoroughly by Lewis in [9]. As Van Emde Boas has pointed out in [2], this problem could well serve as a "master reduction" in the theory of $\mathcal{NP}$-completeness, instead of e.g. SATISFIABILITY, which was first proved to be $\mathcal{NP}$-complete by Cook in [1].

4.1 Randomized Bounded Halting

Bounded Halting, BH, is defined as: given the encoding of a nondeterministic Turing machine M, a binary string w and the unary notation for an integer $n > |w|$, does M halt on input w within n steps? The language formulation is:

$$L(BH) = \{\rho(M)000w001^n \mid M \text{ halts on } w \text{ within } n \text{ steps}\}$$

where ρ is the encoding function for Turing machines defined in lemma 2.3.

Randomized Bounded Halting, RBH, is the problem BH together with a probability function μ_h on its instances defined by:

$$\mu_h(\rho(M)000w001^n) \propto \mu_{TM}(M) \times n^{-3} \times 2^{-k}$$

where $k = |w|$. μ_{TM} denotes a (standard) probability function with Ptime computable distribution for choosing a Turing machine. One possibility for μ_{TM} could be:

$$\mu_{TM}(M) = m^{-2} \times \left(2^{8(m^2+m)}\right)^{-1}$$

where $m = |K|$: choose a number of states m; choose for the transition relation Δ any subset of $K \times \{0,1\} \times (K \cup \{h\}) \times \{0,1,L,R\}$ with equal probability.

Theorem 4.1 *(RBH, μ_h) is Ptime complete for $\mathcal{RNP}$.*

Proof.

Let (D, μ) be any $\mathcal{RNP}$ decision problem. (D, μ) Ptime reduces to $(RH(M_D), \mu_{hm})$ for the machine M_D defined in the Coding Theorem. We now reduce $(RH(M_D), \mu_{hm})$ to (RBH, μ_h) via the following function f:

$$f(w001^n) = \rho(M_D)000w001^n.$$

As $\rho(M_D)$ is a constant for each problem D, this reduction is trivially Ptime computable. $w \in D$ if and only if $\rho(M_D)000w001^{p(|w|)} \in RBH$ for some polynomial p and the probability function of RBH, μ_h, dominates the result of the transformation via f of μ_{hm}. It is evident that $RBH \in \mathcal{RNP}$. □

4.2 Bounded Tiling

As we have defined a computation by a Turing machine by the transitive, reflexive closure of the relation $\vdash$ on configurations, we can picture a computation by enumerating all steps. As is wellknown, a Turing machine which runs in time T can at most reach, or look at, T tape squares. So a $T \times T$ square, each row containing the contents of the tape at a given point in time, gives a clear visual description of a computation of a particular machine. We can generalize this observation and define a *tiling problem* which can be solved if and only if the Turing machine has a halting computation, by coding each configuration of that machine as an unambiguous row in the $T \times T$ square to be tiled. This gives a direct correspondence between the Bounded Halting and the Bounded Tiling problem.

Definition 4.2 1. *A* legal tile set *is a triple* (D, H, V) *where:*

- D *is a finite set of legal tiles;*
- $H, V \subseteq D \times D$.

2. *A* initial tiling *for a legal tile set is a pair* (Δ, d) *where:*

- $\Delta \subseteq \mathbb{N} \times \mathbb{N}$;
- $d : \Delta \to D$.

3. *A* tiling system $\mathcal{D} = (D, H, V, \Delta, d)$ *is a legal tile set together with an initial tiling.*

4. *An* $s \times s$ tiling by $\mathcal{D}$ *subject to an initial tiling, is a function*

$$f : \{0, 1, \cdots, s-1\} \times \{0, 1, \cdots, s-1\} \to D$$

such that

$$\begin{array}{rl} f(\delta) = d(\delta) & \text{for all } \delta \in \Delta \cap \{0, \ldots, s-1\} \times \{0, \ldots, s-1\}; \\ (f(m,n), f(m+1,n)) \in H & \text{for all } 0 \leqslant m, n < s; \\ (f(m,n), f(m,n+1)) \in V & \text{for all } 0 \leqslant m, n < s. \end{array}$$

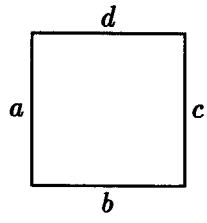

Figure 4.1: a tile

That is, an $s \times s$ tiling by $\mathcal{D}$ is just a tiling of the square of size s in the lower left corner of the first quadrant, obeying all the required constraints on the layout of the tiles. As usual, we picture tiles as squares with markings on their sides (see figure 4.1). H and V are just an abstraction of these markings denoting which tiles can border each other in the horizontal and vertical direction, respectively. For convenience we omit these relations and think of them as embodied in the specification of $\mathcal{D}$.

Lemma 4.3 *There exists an injective logspace computable coding σ of tiling systems to binary strings. $|\sigma(\mathcal{D})|$ is polynomial in the size of the tiling system $\mathcal{D}$.*

Bounded Tiling, BT, is defined as: given (the encoding of) a tiling system $\mathcal{D}$ (i.e. a set of legal tiles and an initial $(s \times s)$ tiling) and the unary notation of an integer s, does there exist a $s \times s$ tiling by $\mathcal{D}$ subject to the initial tiling? The language recognized is:

$$L(BT) = \{\sigma(\mathcal{D})001^s \mid \text{there exists a } s \times s \text{ tiling by } \mathcal{D}\}.$$

Theorem 4.4 *BT is $\leqslant_p$-complete for $\mathcal{NP}$.*

Proof.

The proof presented here is slightly different from the proof given in e.g. [10]. The main difference is the way in which we "code" the step counting function into the tiling system. The idea behind the proof is that we let the tiled square represent the space/time history of a computation by a Turing machine M. Each row of the square represents a configuration of M. If we device a neat mapping from configurations to the tiled rows, we can tile an $s \times s$ square if and only if M has a computation, consisting of s steps and using s tape squares. Next we must ensure that M has a *halting* computation within s steps and using at most s tape squares, by forcing that the last row to be tiled corresponds with the halted configuration.

The first thing to do is to construct a tiling system for a given Turing machine M. Let $M = (K, \Sigma, \Delta, s, h)$. According to lemma 3.12 we may assume $\Sigma = \{0, 1, \#\}$ where $\#$ denotes the blank symbol. Then $\mathcal{D}_M = (D_M, H, V, \Delta_M, d)$ is the tiling system corresponding to M containing the following tiles:

1. For each $b \in \Sigma$, the tile of figure 4.2 is in D_M. The tile simply communicates any unchanged symbol upwards from configuration to configuration.

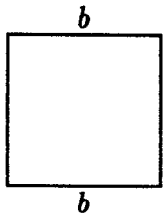

Figure 4.2: unchanged tape contents

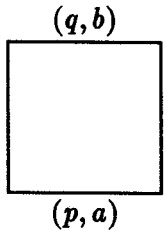

Figure 4.3: tape head and states

2. For each $a \in \Sigma$ and $p \in K$ such that $(p, a, q, b) \in \Delta$, for some $a \in \Sigma, p \in k$, the tile of figure 4.3 is in D_M. This tile correspond to the position of the tape head and the state associated with a configuration.

3. For each $p \in K$ and $a \in \Sigma$ such that $(p, a, q, R) \in \Delta$ for some $q \in K$, the tiles of figure 4.4 are in D_M. These tiles corresponds to the movement of the tape head one position from left to right.

4. Tiles similar to those of (3) for the case in which $(p, a, q, L) \in \Delta$, as illustrated in figure 4.5.

5. The above tiles do the bulk of the simulation of M by $\mathcal{D}_M$. Here we give the tiles that correspond to the initial configuration of M. For the starting state s and for each $b \in \{0, 1\}$, D contains the tiles shown in figure 4.6.

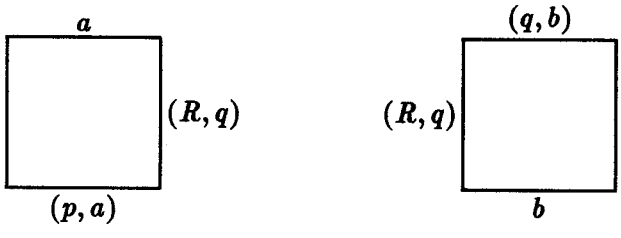

Figure 4.4: right movement

Figure 4.5: left movement

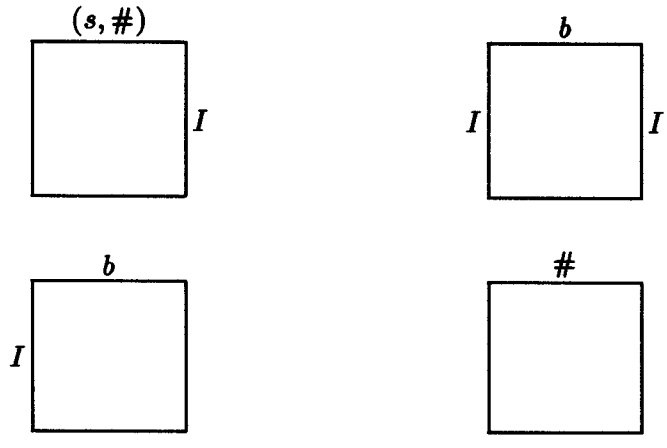

Figure 4.6: initial configuration

Figure 4.7: halting configuration

6. Next we need the two tiles for the halting configuration as shown in figure 4.7.

Any input w for M can be coded by the tiles of figure 4.6 in an obvious way, that is, we define the initial tiling to contain (the encoding of) the configuration $(s, \underline{\#}w\#)$: let $w = b_1 \dots b_n$. Then:

$$\begin{array}{lcl} d(0,0) & = & \text{the first tile of figure 4.6} \\ d(i,0) & = & \text{the second tile of figure 4.6 corresponding to } b_i \text{ for } 1 \leqslant i < n \\ d(n,0) & = & \text{the third tile of figure 4.6 corresponding to } b_n. \end{array}$$

Next we have to keep track of the number of steps M has taken. In [10] this is done by equiping each tile with natural numbers ("colors") k and $k+1$ which signal that these tiles are used in the k^{th} step of the computation. This results in an abundance of tiles. We give here a more restricted means to enforce that the computation halts within s steps.

In order to enforce that the last row will be tiled with a tiling corresponding to the halted configuration, we simply define the initial tile in the left upper square to be the second tile given in figure 4.7, that is:

$$d(0,s) = \text{the second tile of figure 4.7}$$

So the last row can only be tiled if the Turing machine has reached a halting configuration at this point.

The following facts can easily be proved by induction:

1. Each row of the tiled square contains exactly one tile with an encoding of the tape head and a state;
2. Each row of the tiled plane corresponds to a configuration of the Turing machine M;
3. If α and β are the k^{th} and $(k+1)^{\text{th}}$ row of the square, then for the corresponding configurations C_α and C_β: $C_\alpha \vdash_M C_\beta$.
4. There exists a $s \times s$ tiling of the square if and only if M accepts w in time s.

To complete the proof of theorem 4.4, let $D \in \mathcal{NP}$ be any decision problem with its associated D-machine M_D. For any binary string w, $w \in L(D)$ if and only if M_D has a halting computation on w within $p(|w|)$ steps for some polynomial p if and only if there exists a tiling of the $(p(|w|) \times p(|w|)$ square by the tiling system $\mathcal{D}_{M_D}$ defined above. The required reduction f is given by:

$$f(w) = \sigma(\mathcal{D}_{M_D})001^{p(|w|)}$$

which is Ptime computable. □

We now give an interesting corollary to theorem 4.4 which proves a claim from Levin [7].

Corollary 4.5 *There exists a set of legal tiles (D', H, V) such that the problem: "given an initial tiling and the unary notation of an integer s, does there exists a $s \times s$ tiling by the corresponding tiling system $\mathcal{D}'$" is $\mathcal{NP}$-complete.*

Proof.

BH is Ptime complete for $\mathcal{NP}$. Consider the machine for BH, M_{BH}. We can construct a tiling system $\mathcal{D}'$ that mimics the steps of M_{BH}. So, for any decision problem D in $\mathcal{NP}$, $w \in D$ if and only if M_D halts on w within $p(|w|)$ steps if and only if M_{BH} halts on $\rho(M_D)000w001^{p(|w|)}$ within $q(|w|)$ steps for some appropriate polynomial q, if and only if there exists a tiling by $\mathcal{D}'$ of the $q(|w|) \times q(|w|)$ square with an initial tiling that corresponds to

$$\rho(M_D)000w001^{p(|w|)}.$$

It is easy to see that this implies a Ptime reduction. □

4.3 A non-standard reduction

In the preceding sections we have defined a probability function on instances of a decision problem which we may assume to be over {0,1}. We called this probability function standard if the probability of a binary string w was proportional to $(|w|+1)^{-2} \times 2^{-|w|}$. We further acted as if the Turing machine M associated with the problem had the same probability function on its inputs as the problem on its instances. Yet the input alphabet of M is $\{0, 1, \#\}$. We don't randomly generate strings of a certain length over this alphabet, but over {0,1}. In a sense, we assume that M has the magical capability to reject garbage as input. For a given Turing machine M we can easily construct a new Turing machine M' that inspects its input explicitly in order to discriminate against wrongly formatted input and then simulates M. The inspection only requires $\mathcal{O}(n)$ time. It is clear that M' runs in polynomial on average time if and only if M does. Formally we can say:

Theorem 4.6 *For any decision problem (D, μ) over {0,1} in $\mathcal{RNP}$, there exists a Turing machine M with alphabet $\{0, 1, \#\}$ and a function f which is Ptime computable such that*

1. $x \in L(D) \iff f(x) \in L(M)$;

2. *the probability function μ_M for the inputs for M is standard with respect to the tape alphabet of M, that is, the probability of an input w is proportional to $(|w|+1)^{-2} \times 2^{-|w|}$;*

3. *D is EPtime decidable if and only if T_M is polynomial on average.*

Notice however that f is not a Ptime reduction in the sense of definition 3.9: the probability function μ_M is exponentially smaller than the probability function μ.

4.4 Randomized Bounded Tiling

Randomized Bounded Tiling (RBT) is the problem BT together with a probability distribution μ_t on its instances. We define μ_t to be the following standard probability function:

$$\mu_t(\sigma(\mathcal{D})001^s) = \mu_T(\mathcal{D}) \times s^{-2}$$

where μ_T is a standard probability function for choosing a tiling system. We can define μ_T to be:

$$\mu_T(\mathcal{D}) = k^{-2} \times \left(2^{k^4}\right)^{-1} \times \mu_d(d).$$

μ_T is indeed standard: choose a number of "colors" k and a subset of all possible tiles using k colors for the set of legal tiles D; μ_d denotes the probability of choosing an initial tiling: we want to choose a subset $\Delta \subseteq \mathbf{N} \times \mathbf{N}$ and a function $d : \Delta \to D$. But alas, this construction goes awry as the probability of choosing a subset Δ is (at least) proportional to $\left(2^{s^2}\right)^{-1}$ rendering the joint probability of choosing a tiling system exponentially smaller than the probability of choosing an input for the problem we want reduce to it. We therefore define Δ to be exactly the squares we need in the reduction: $s+1$ squares on the lefthand part of the first row and one square in the upperleft corner. Note that μ_d becomes zero if, at some point, we can't choose a next tile. The following result was originally proved by Levin [8].

Theorem 4.7 *RBT is Ptime complete for $\mathcal{RNP}$.*

Proof.

1. BT is $\leqslant_p$-complete for $\mathcal{NP}$;

2. Let (D, μ) be any decision problem in $\mathcal{RNP}$. Then, by theorem 4.6, (D, μ) is EPtime decidable if and only if the Turing machine M_D runs in polynomial on average time. We reduce the halting problem of this machine to RBT. Let f be the reduction defined in theorem 4.4. Then μ_T dominates the result of the f-transformation of μ: the probability of choosing a set of legal tiles is a constant; the probability of choosing an initial tiling is proportional to the probability of choosing the "input" of length k over (the representations for) $\{0, 1, \#\}$. □

From corollary 4.5 we immediately get the following corollary.

Corollary 4.8 *There exists a legal tile set (D, H, V) and a probability function μ on initial tilings such that the problem "given an initial tiling I with probability $\mu(I)$ and the unary notation of an integer s, does there exist a $s \times s$ tiling by the corresponding tiling system $\mathcal{D}$" is $\mathcal{RNP}$-complete.*

Chapter 5

Odds and ends

In this chapter we discuss some further observations and results. In particular, we investigate logspace reducibility and a randomized version of $\mathcal{PSPACE}$. For this we extend definition 2.4 to classes which are computable within a certain amount of space.

Definition 5.1 1. $DLOG = DSPACE(\log(n))$

2. $\mathcal{PSPACE} = \bigcup_{k \geq 0} DSPACE(n^k)$.

5.1 logspace reducibility

The definition of $DLOG$ immediately gives rise to a notion of *logspace reducibility* between randomized decision problems: f logspace reduces (D_1, μ_1) to (D_2, μ_2) (notation: $(D_1, \mu_1) \leqslant_{log} (D_2, \mu_2)$) if it reduces D_1 to D_2, reduces μ_1 to μ_2 and $f \in DLOG$.

As $\mathcal{NP}$ has $\leqslant_{log}$-complete problems (the original proof by Cook [1] of the $\mathcal{NP}$-completeness of SATISFIABILITY used logspace reducibility), it is a natural question to ask whether $\mathcal{RNP}$ has logspace complete problems. The answer seems to be "no", unless we severely restrict the definition of $\mathcal{RNP}$. Recall that in the proof of the Coding Theorem, which is at the basis of the theory, we must explicitly compute μ^* for a given problem (D, μ). Unless μ^* is logspace computable, this reduction can't be logspace computable. At the same time, it seems highly unlikely that we can prove a problem complete for $\mathcal{RNP}$ *without* consulting μ^* for every problem $(D, \mu) \in \mathcal{RNP}$.

Therefore, the best result we can formulate, is:

Corollary 5.2 *For any problem $(D, \mu) \in \mathcal{RNP}$, if μ^* is logspace computable, then $(D, \mu) \leqslant_{log} (RH, \mu_h)$ and $(D, \mu) \leqslant_{log} (RBT, \mu_t)$.*

Proof.

Immediate from the proofs of theorems 4.1 and 4.7. □

This result isn't quite as bad as it seems: it simply states that logspace reducibility doesn't seem to be an appropriate notion of reducibility among randomized problems.

As we are studying expected Ptime computability, Ptime reducibility is good enough for us.

5.2 Randomized $\mathcal{PSPACE}$

Another question raised in complexity theory is "$\mathcal{P} \stackrel{?}{=} \mathcal{PSPACE}$". Albeit not as "practical" as $\mathcal{RNP}$, a randomized version of $\mathcal{PSPACE}$ therefore is a natural extension of the theory of the previous chapters.

Definition 5.3 *A randomized decision problem (D, μ) is in $\mathcal{RPSPACE}$ if $D \in \mathcal{PSPACE}$ and μ^* is Ptime computable.*

We let $RH2$ be the randomized version of the bounded halting problem for $\mathcal{PSPACE}$, defined as: given the encoding of a Turing machine M, a binary string w and the unary notation for an integer $n > |w|$, does M halt on w within n space?

Theorem 5.4 *1. Every $\mathcal{RPSPACE}$ problem is Ptime reducible to a standard $\mathcal{RPSPACE}$ problem.*

2. $RH2$ is Ptime complete for $\mathcal{RPSPACE}$.

Proof.

Immediate from the Coding Theorem and theorem 4.1: just replace "in $q(|w|)$ steps" by "within $q(|w|)$ space". □

Define $RT2$ as the $\mathcal{PSPACE}$ equivalent of RT: given a finite set of legal tiles, and two tiled rows U and V of length N, does there exists an integer M such that it is possible to tile an $M \times N$ rectangle with U (V) serving as the top (bottom) row and with "white" left and right borders?

Theorem 5.5 *$RT2$ is Ptime complete for $\mathcal{RPSPACE}$.*

Proof.

1. $RT2$ is Ptime complete for $\mathcal{PSPACE}$ (see [2]).
   For any problem $(D, \mu) \in \mathcal{PSPACE}$, define the reduction f:
   $$f(w) = \rho(D_{M_D})00I00F001^n$$
   where:
   $$I \sim (s, \underline{\#}w\#^{n-|w|-1})$$
   $$F \sim (h, \underline{\#}\#^{n-1})$$
   and D is the set of tiles defined in the proof of theorem 4.4.
2. This reduction works. □

Chapter 6

Discussion

In this paper we have given an overview of the theory of decision problems with a probability function on their instances, called "randomized problems"as developed by Levin [8] and Gurevich [4]. We have defined a notion of reducibility between these problems and proved the existence of complete problems in the class "randomized $\mathcal{NP}$" with respect to this reduction. We also gave some extensions to the basic theory.

The theory of $\mathcal{RNP}$is a "practical" approach to the complexity of decision problems; it is concerned with "average case", not "worst case" behavior. The latter results, as is often argued, in true $\mathcal{NP}$.

A question could be where to locate this new class in the complexity hierarchy. It is obvious that a randomized problem (D, μ) is EPtime decidable for every probability function μ if $D \in \mathcal{P}$. It is known that for certain problems with an associated probability function μ, (D, μ) is EPtime decidable if $D \in \mathcal{NP}$. Even if $\mathcal{P} \neq \mathcal{NP}$, the problems in $\mathcal{RNP}$ could be EPtime decidable. This would give quite a relief to system programmers, but adds little insight to the $\mathcal{P}$ versus $\mathcal{NP}$ question.

On the other hand, as Gurevich has shown in [4] the notion of EPtime reductions leads to some interesting results in "ordinary" complexity theory.

Acknowledgements

The author wishes to thank Jan van Leeuwen for helpful discussions.

Bibliography

[1] S.A. Cook, The complexity of theorem-proving procedures, *Proc. ACM STOC* **3** (1971), 151–158.

[2] P. van Emde Boas, Dominoes are forever, Techn. rep. 83-04, Dept. of Mathematics, University of Amsterdam (1983).

[3] M.R. Garey and D.S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-completeness* (W.H. Freeman & Co., New York, 1979).

[4] Y. Gurevich, Complete and incomplete randomized NP problems, *Proc. IEEE FOCS* **28** (1987), 111–117.

[5] J.E. Hopcroft and J.D. Ullman, *Introduction to Automata Theory, Languages and Computation* (Addison Wesley, Reading, MA, 1979).

[6] D.S. Johnson, The NP-Completeness Column: An Ongoing Guide, *J. Algorithms* **5** (1984), 284–299.

[7] L.L. Levin, Universal sequential search problems, *Problems of Information Transmission* **9** (1973), 265–266.

[8] L.L. Levin, Average case complete problems, *SIAM J. Comput.*, **15** (1986), 285–286.

[9] H.R. Lewis, Complexity of solvable cases of the decision problem for the predicate calculus, *Proc. IEEE FOCS* **19** (1978), 35–47.

[10] H.R. Lewis and C.H. Papadimitriou, *Elements of the theory of computation* (Prentice-Hall Inc., Englewood Cliffs, New Jersey, 1981).

[11] A.M. Turing, On computable numbers, with an application to the Entscheidungsproblem, *Proc. London Math. Soc.*, 2 **42** (1936), 230–265 and **43** (1936), 544–546.