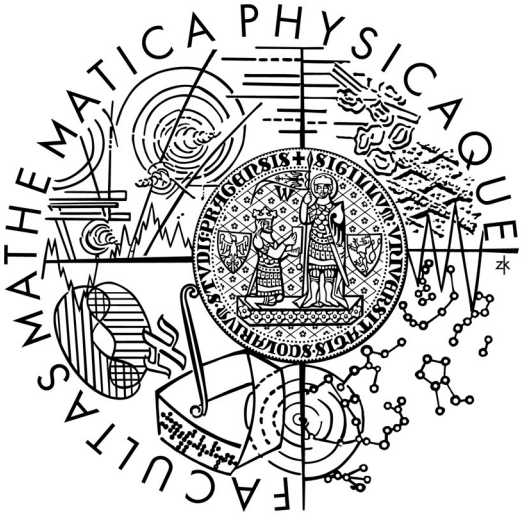
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

DIPLOMOVÁ PRÁCE



Samuel Čarnoký

Vyhledávací problémy a hledání kolizí pro hašovací funkce

Katedra algebry

Vedoucí diplomové práce: prof. RNDr. Jan Krajíček, DrSc.

Studijní program: Matematika, Matematické metody informační bezpečnosti

Ďakujem vedúcemu za trpezlivosť, cenné pripomienky a rodičom za podporu.

Prohlašuji, že jsem svou diplomovou práci napsal(a) samostatně a výhradně s použitím citovaných pramenů. Souhlasím se zapůjčováním práce.

V Praze dne 8.12.2010 Samuel Čarnoký

Contents

1 Introduction **6**

2 NP search problems **8**
2.1 Definitions 8
2.2 Relativisation 11
2.3 Examples of NP-search problems and reductions 13

3 PHP- related NP search **16**
3.1 Hash functions 16
3.2 An example of reduction 18

4 Paths in graphs **20**

5 Reduction to WPHP **27**

6 Conclusion **29**

Bibliography **30**

Název práce: Vyhledávací problémy a hledání kolizí pro hašovací funkce
Autor: Samuel Čarnoký
Katedra : Katedra algebry
Vedoucí diplomové práce: prof. RNDr. Jan Krajíček, DrSc.
e-mail vedoucího: krajicek@karlin.mff.cuni.cz

Abstrakt: Centrálnymi bodmi tejto práce sú NP vyhľadávacie problémy a existencia redukcie medzi nimi v relativizovanom zmysle. Absolútna separácia by separovala P od NP. Venujeme sa špeciálne problému hľadania kolízii v hešovacích funkciách, ktorých existencia je garantovaná známym holubníkovým princípom (PHP). Podávame stručný úvod do problematiky, definujeme rôzne NP vyhľadávacie problémy a pripomíname redukcie a separácie. Referujeme o redukcii slabej verzie PHP na hľadanie homogénneho podgrafu a prinášame vlastnú redukciu varianty PHP na problematiku súvisiacu s hľadaním ciest v grafe. Pojednávame o redukovaní hladania kolízií vo viacerých funkciach na hľadanie kolízie v jednej.

Klíčová slova: NP vyhľadávanie, redukcie, pigeonhole principle, orákula

Title: Search problems and search for collisions in hash functions
Author: Samuel Čarnoký
Department: The Department of Algebra
Supervisor: prof. RNDr. Jan Krajíček, DrSc.
Supervisor's e-mail address: krajicek@karlin.mff.cuni.cz

Abstract: Central points of this work are NP search problems and the existence of reductions amog them in the relativised world. Absolute separation would separate N from NP. In particular, we talk about the problem of finding collisions in hash functions that must exist due to the famous pigeonhole principle. We present a brief introduction into the topic, we define various NP search problems and recall reductions and separations. Reduction of weak version of PHP to a problem of finding a homogeneous subgraph is described and our own results are presented in the form of reduction of another variant of PHP to a problem related to finding paths in a graph. We talk about reducing the task of finding collisions in multiple functions into finding a collision in one function.

Keywords: NP search, reductions, pigeonhole principle, oracles

Chapter 1

Introduction

Last decades witnessed an increasing interest in complexity theory, study of computational complexity and generally what sort of problems can mankind solve in a reasonable time. Also, no less attention is dedicated to trying to know what can not be solved in time, before the results are irrelevant. Identifying hard problems is not an easy task, there is a lot of unproven conjectures. These problems could be used to our advantage, by forcing someone else to solve them, if they wish to achieve something we don't want them to. But with so many problems, proving for every one that it is hard is unrealistic, we use more sophisticated method, reducing one problem to another. Reductions among problems are a good way to place some of them, that are vague or difficult to analyze, in to the hierarchy with other problems. Then we would know that solving a problem is at least as hard as solving the one we know that is hard, meaning that it is proven to be very difficult as efficient solution hasn't been found for centuries. We can sleep peacefully knowing that if our adversary succeeds at stealing our money from the bank, he also succeeds at solving something that is considered unsolvable in a reasonable time and we can consider our lost money to be the motivation that served to advance the humanity forward. This is because mechanisms that protect our assets or our privacy are based on these hard problems. Absolute security is unrealistic in practical life, because even though Vernam's cipher, the only cipher proven to be absolutely secure, that is unbreakable given any amount of ciphertext or time, still requires the key to be as long as input, not to mention it has to be truly random. Instead, we use running time of solving algorithm to define what is infeasible. We could succeed by exhausting every possibility for a given problem but that could take more than the span of our

life or the duration of the universe, so calling a problem that we can solve, but not fast enough, to be infeasible, describes the situation well. Reductions will be the primary concern in this work, with regard to the problem which is expected to be hard, if we want to guarantee security in many real life applications. They also have a significant theoretical value because they can reveal a proof potential of some mathematical theorems, lemmas or simple facts and this will be also discussed later. We will present a framework to categorize problems, show many examples and reductions.

Chapter 2

NP search problems

We will be interested mainly in the family of problems called NP-search problems. We will soon define them and explain why they represent our needs in both practical and theoretical applications. We begin with formalizing the notion of time needed for solving a problem as it is central in complexity theory and information security. As the computational model, we will use the deterministic Turing machine, whose one step during a computation will be considered to have the unit cost. Following definitions are widely used and can be found in [7].

2.1 Definitions

Definition. *For a Turing machine M with an input $x \in \{0,1\}^*$, the time of the computation, labeled $time_M(x)$, is the total number of steps done by M on input x before it halts or ∞ if it never stops.*

Now we need to scale the time of computation to the length of the input. This leads to following definition.

Definition. *For a Turing machine M and number n, we define the time complexity of M to be the function $t_M(n) = max\{\ time_M(x) \mid x \in \{0,1\}^n\}$.*

This function represents the worst case scenario for the duration of computation on inputs with n bit length. Now we define classes of problems according to the speed of growth of $t_M(n)$.

Definition. *Let $f : \mathbb{N} \to \mathbb{N}$ to be a function. Class TIME(f(n)) consists of all problems (languages) such that for each of them exists a Turing machine providing a solution for each n bit input in $t_M(n) = O(f(n))$ time.*

Big O notation is used because we want to compare the speed of growth of functions and not actual functions. We recall from [2] that for f,g functions from natural numbers to natural numbers, f(n) = O(g(n)) as n $\to \infty$, if there exist M and n_0 such that $f(n) \leq Mg(n)$ for all $n > n_0$. Therefore, if one problem is solvable in 4n+156 steps and second in 27n +14 steps they will both belong in the same class TIME(n), called the linear time. Now we can proceed to the important class of problems, solvable in polynomial time, corresponding to what is considered feasible in real life.

Definition. *Class $P = \bigcup_{k\in\mathbb{N}} TIME(n^k)$ is the class of problems solvable in polynomial time.*

On the other hand, problems that seem to be solvable only by exhaustive methods justify next definition as trying all possibilities corresponds to exponential time.

Definition. *Class $EXP = \bigcup_{k\in\mathbb{N}} TIME(2^{n^k})$ is the class of problems solvable in exponential time.*

Now we define another important class, called NP problems. If there is a solution of a NP problem on input x, it can be verified quickly that it is indeed a solution. Having a solution is expressed by belonging to a language, more formally in the following definition.

Definition. *A problem X belongs to the class NP if there exists a polynomial time relation R(x,y) and constant c such that for all x :*

$$x \in X \Leftrightarrow \exists y(|y| \leq |x|^c) \wedge R(x, y)$$

Any such y is called a witness for $x \in X$. If someone gives us a solution i.e. a witness, we can check that it is indeed a solution in polynomial time. Hence Nondeterministic Polynomial - NP. We could construct a decision NP problem by the following way.

Definition. *Let R be a polynomial time relation and c a constant such that $R(x,y) \Rightarrow |y| \leq |x|^c$. A decision NP problem is the problem of determining whether for a given x, there is a y, such that R(x,y) holds.*

The answer is "yes" or "no". Finally, we could formulate an associated NP-search problem.

Definition. *Let R be a polynomial time relation and c a constant such that $R(x,y) \Rightarrow |y| \leq |x|^c$. A NP-search problem is the problem of finding, for a given x, a y that satisfies $R(x,y)$, if such y exists.*

We follow the nomenclature from [3] and label the class of all NP-search problems as FNP. A NP-search problem is said to be total, if for each input x, such y exists. These problems form class denoted TFNP. We can see here, that for total NP-search problems we always get answer "yes" from the decision version, but actually finding y can be difficult. For example, finding prime factors of a positive integer. Existence of a prime factorisation is guaranteed by the fact that these integers always have an unique prime factorisation. Of course, in other structures there could be more then one factorisation, yielding multiple y for an input x.

Informally, we say that problem Q_1 is polynomial-time Turing reducible to Q_2 if there is some polynomial machine M that takes input for Q_1, transforms it to an input to Q_2 and produces correct answers to Q_1 using results of Q_2. This reduction will be defined precisely in a more general form in the next section. Generally, the existence of a solution to every input of a given problem is usually guaranteed by a lemma or a theorem. This allows us to study their relative logical power using reducibility and non reducibility between associated NP-search problems.

2.2 Relativisation

It is time now to generalize our notion of unit computing cost. Absolute separation between two NP-search problems, that is a proof that one is not reducible to another, would imply $P \neq NP$. Before we define reduction, lets consider that our machine computing a given problem has also access to an oracle which provides an information about the input also at unit cost. For example, if input is a graph, the oracle could provide edge relation information for a given pair of vertices. The input to the problem is now not only x but also an oracle O, and the set of solutions is denoted S(x,O). As the output of the oracle could be very long, for example if our queries would return all neighbours of a given vertex, we define the class NPo-search in the following way.

Definition. *Let x stand for binary strings and O for oracles representing functions from strings to strings. FNPo is the class of all problems with input (x,O) and set of possible solutions S(x,O) where $y \in S(x, O)$ is a predicate that is computable in deterministic time polynomial in $|x|$, calls to O at unit cost and all elements of S(x,O) are polynomialy bounded in $|x|$.*

A problem from class FNPo is called total if S(x,O) is nonempty for all x and O. Finally, class TFNPo denotes all these problems. This oracle based definitions relativize the world we work in and allow us to construct separations. We shall now describe how reductions are done.

Definition. *Let $X_1, X_2 \in TFNPo$. We say that X_1 is many-one reducible to X_2 if there exist a polynomial time Turing machines M and M*, such that M on input (x, O_1) for X_1 and using an oracle providing solutions to X_2 outputs some $y \in S(x, O_1)$. Machine M* is used to simulate O_2 for X_2 using only x and O_1.*

Machine M presents an input (z, O_2) to X_2 and receives $u \in S(z, O_2)$. Using (x, O_1) and whatever correct solution u to X_2 it must produce a correct solution $y \in S(x, O_1)$. Figure 2.1 describes this process.

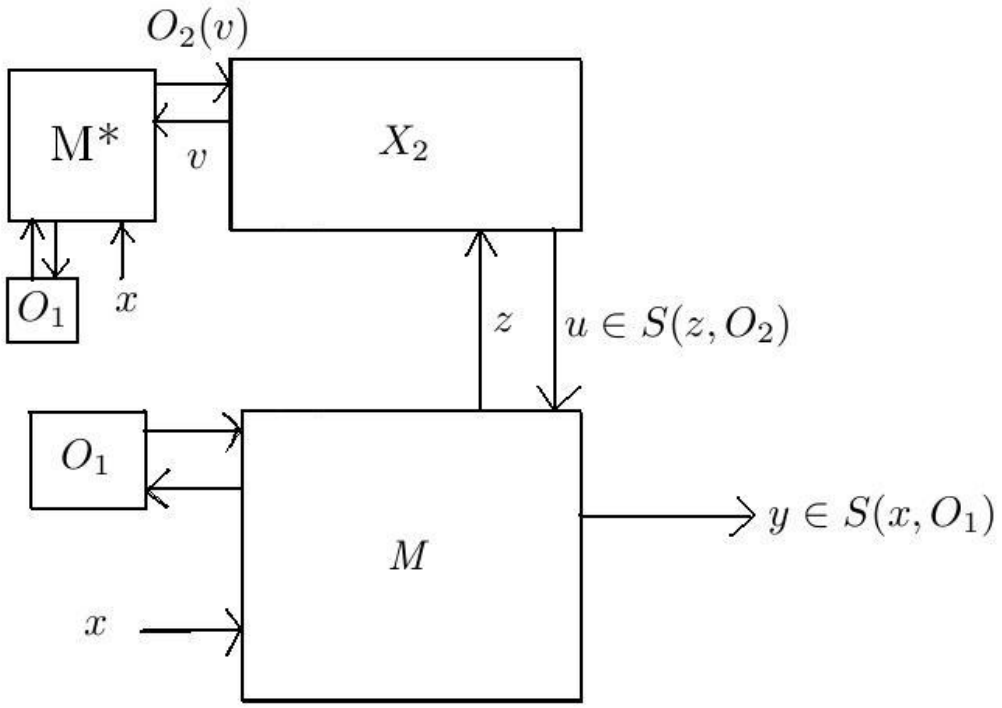


Figure 2.1: Reduction of X_1 to X_2

2.3 Examples of NP-search problems and reductions

Now we have all we need to formulate some NP-search problems and illustrate according to [1], how reductions are done. For each problem, we will provide the theoretical result that guarantees its totality. We will be trying to find a substructure on a graph that is given by an oracle. This oracle provides local information about the graph, given a vertex it returns set of neighbouring vertices, with possible direction information. There is a constant bound on degree of each vertex. Last problem will be using a function defined by an oracle. All these problems are members of TFNPo class. For input x, we will take 0^n.

Definition. *LEAF is the problem of finding a leaf on an undirected graph with vertices labeled* $\{0,1\}^n$ *and with degree* ≤ 2*. Solution is any leaf* $c \neq 0^n$ *or* 0^n*, if* 0^n *is not a leaf.*

This problem uses a known fact, that every graph has an even number of odd-degree nodes. So if LEAF is presented with a graph and a vertex 0^n that is not a leaf, it returns 0^n, but if 0^n is a leaf, then there must be another for LEAF to find.

Definition. *SOURCE.OR.SINK is the problem of finding a source or sink on a directed graph with vertices labeled* $\{0,1\}^n$ *with in-degree* ≤ 1 *and out-degree* ≤ 1*. Solution is any source or sink* $c \neq 0^n$ *or* 0^n*, if* 0^n *is not a source.*

This is a directed version of LEAF, relying on a lemma, that every directed graph with an imbalanced node, must have another imbalanced node. Again, if 0^n is a source, there has to be another source or sink.

Definition. *SINK is the problem of finding a sink on a directed graph with vertices labeled* $\{0,1\}^n$ *with in-degree* ≤ 1 *and out-degree* ≤ 1*. Solution is any sink* $c \neq 0^n$ *or* 0^n*, if* 0^n *is not a source.*

If there is a source, there has to be a sink. Now we define PIGEON, a problem central to this work.

Definition. *PIGEON is the problem of finding a collision in a function* $f : \{0,1\}^n \to \{0,1\}^n$*. Solution is any pair* (c, c')*,* $c \neq c'$ *with* $f(c) = f(c') \neq 0^n$ *or any* c'' *with* $f(c'') = 0^n$*.*

Famous pigeon hole principle guarantees the existence of a collision. It says that a function, whose domain is larger than its range cannot be injective. Here, if we omit 0^n from the range PIGEON finds the collision. The reason why we use the word collision is not accidental, it comes from terminology of cryptography and finding them in cryptographic hash functions is considered to be very difficult and assumed infeasibility to find them is crucial to information security. Now we shall present some simple reductions and separation results.

Reduction. *SOURCE.OR.SINK is many-one reducible to SINK.*

Proof. Machine M used for reduction only needs to directly feed input for SOURCE.OR.SINK to the SINK problem and received result is directly valid for SOURCE.OR.SINK. There is no transformation necessary. □

Reduction. *SOURCE.OR.SINK is many-one reducible to LEAF.*

Proof. All M has to do now, is to ignore the direction information of input graph for SOURCE.OR.SINK and present it to the LEAF problem. Then, when the leaf is found, look back at the direction it had and return source or sink vertex. Ignoring the information can be done in polynomial time. □

Reduction. *SINK is many-one reducible to PIGEON.*

Proof. M needs to present function a f for PIGEON problem derived from input graph to the SINK problem. Let us consider following definition of function f : if there is an edge from v to u, then f(v)=u, if v is isolated then f(v)=v and finally if v is a sink then f(v)= 0^n . As the input graph has indegree ≤ 1 there can be no collisions and PIGEON outputs a c'' *with* $f(c'') = 0^n$ and that is a sink. □

We could see in the previous proof how a function oracle is polynomially simulated by the oracle for the graph.

The main result for all problems defined above, is that reversed reductions for those we presented, do not exist. For each problem, we could define the class of problems reducible to it.

The separation results of classes from [1] are illustrated on figure 2.2 .

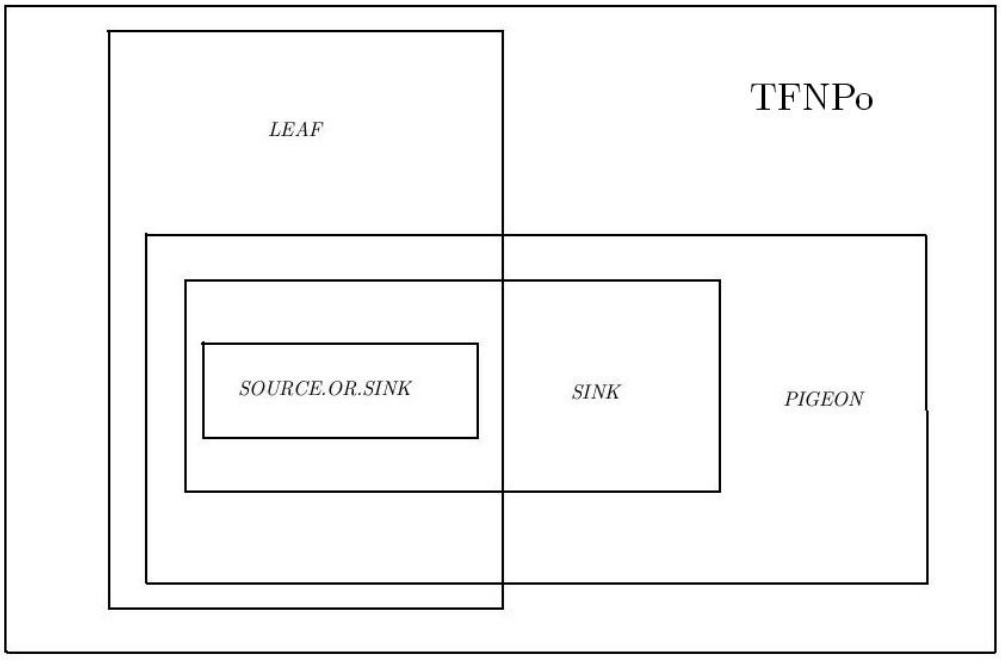


Figure 2.2: Separation results

Chapter 3

PHP- related NP search

In this chapter, we devote our attention to NP-search problems based on pigeonhole principle. First, we must understand why is it so interesting and how heavily modern cryptography relies on the assumed difficulty of finding collisions.

3.1 Hash functions

A function f is called one way, if for any $x \in X$ it is easy to compute f(x) but for a randomly selected y in the range of f we are unable to compute in feasible time a x such that f(x)=y. Also, a function f is called collision free, if we are unable to find in a feasible time distinct $x, x' \in X$, such that f(x)=f(x'). Suppose we have X, set of binary strings of length $m > n$. Function $h : X \to \{0,1\}^n$ is called a cryptographic hash function if its one way and collision free. In cryptographic applications, we would like h to behave as a random function, returning random values from its range, but for a given input x return always the same f(x). Usually, m is much bigger than n so the number of collisions is immense. But its still hoped that it is computationally infeasible to find one. From the birthday paradox, we can show that finding a collision can be done done in approximately $2^n/2$ time. If h behaves as a random oracle we do not know any faster way [5].

Cryptographic hash functions are used widely, because they provide a tool for representing larger set of data by a much smaller one , still retaining a practically unique correspondence. Hashes of passwords are stored in databases and compared to the hash of the user password provided during

login. Contracts are hashed and then the hash is signed to speed up the process, integrity of data is derived from the integrity of its hash.

3.2 An example of reduction

Knowing that domain of a function f, if used as a hash function will be much larger than its range we are allowed to formulate WPHP, a problem based on weak pigeonhole principle, coming from [8].

Definition. *Given an oracle C with strings* $\{0,1\}^m$ *as inputs and* $\{0,1\}^n$ *as outputs,* $n < m$*, the WPHP problem has to find u,v distinct elemnets of* $\{0,1\}^m$ *such that C(u)=C(v).*

Note that if $n < m$, the domain of the function given by C is at least twice as large than its range. Therefore, we denote this as weak pigeonhole principle, as opposed to domain being larger by at least one element for PHP to hold. Next definition and the following reduction can be found in [6].

Definition. *Problem RAM is the task to find a homogeneous subgraph (clique or an independent set) of size m/2 in a graph with vertices labeled by strings from* $\{0,1\}^m$*. The edge relation is given by an oracle D with 2m input bits.*

Existence of a homogeneous subgraph of size m/2, in the graph with at least 2^m vertices, is guaranteed by the Ramsey theorem. Vaguely, we are guaranteed to find a small island of order in an arbitrary chaotic environment. On the other hand, Razborov [9] has shown, that there is a graph on 2^n vertices not containing any homogeneous subgraph of size 2n, where edge relationship can be computed in non-uniform polynomial time. Previous results of Erdos [4] were used.

Reduction. *WPHP is many-one reducible to RAM.*

Proof. Suppose we have E, input to WPHP with m=n+1. Let us consider following map:

$$E(E(x_1, ..., x_{n+1}), x_{n+2}).$$

This maps $\{0,1\}^{n+2}$ to $\{0,1\}^n$ and if collision is found in this map, then it yields a collision in E. We can repeat this polynomially many times until $m \geq 4n$. Lets continue now with this amplified map and find a collision. We construct a graph G on 2^m vertices, with edge relation defined as

$$D(u,v) := R(E(u), E(v))$$

, edge relationship in G is given by before mentioned Razborov graph, after vertices are transformed by E. Now we present G to RAM and obtain a

homogeneous subgraph H of size m/2. If E was injective on H we would also get a homogeneous subgraph in Razborov graph with $m/2 \geq 2n$ vertices. We know that this is not possible, therefore there must be a collision in E on H. H is polynomially large in the size of input, so all we need to do is to check this small set and find where is the collision. □

Chapter 4

Paths in graphs

In this chapter, we will define a new NP search problem that, if solved, could also be used to find collisions. We will talk about paths in graphs. The next lemma is posed as an exercise in [3], the proving method below is ours.

Lemma. *Every graph with v vertices and $e > \frac{3}{2}(v-1)$ edges contains two vertices joined by at least three vertex independent paths.*

Proof. Let us look at cycles, closed paths with no vertex repetition in such graph. We suppose there is not any pair of vertices joined by three independent paths. This implies that every two cycles in the graph can share at most one common vertex. If there were two cycles with two common vertices, we could find two vertices and three independent paths joining them in the following manner. We would choose a vertex belonging to one cycle, but not to another. This point exists as we have two different cycles. We would follow from this point on the cycle to both directions until we arrive to the first vertex belonging to the other cycle. These two points of intersection of cycles would would be joined by three vertex independent paths. First is the path already taken containing the starting vertex. Remaining two form the second cycle.

In particular, there are no cycles that share a common edge. Each edge belongs to a maximum of one cycle. We now have an idea how this type of graph has to look. There can be isolated points, paths that don't form cycles, cycles of different lengths. Figure 4.1 illustrates the situation for n=18.

We need now to count how many edges this kind of graph can contain at maximum. To determine this bound we could regroup cycles and edges that

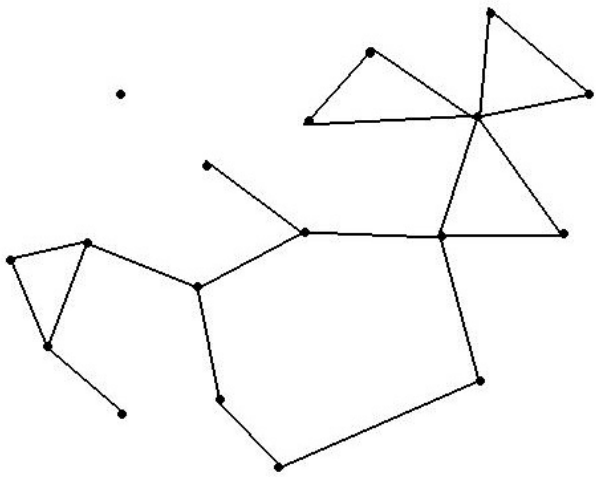


Figure 4.1: Example for n=18

are not part of any cycle so they all meet in one vertex. This will not change the number of edges nor vertices. See figure 4.2 for the regrouped graph.

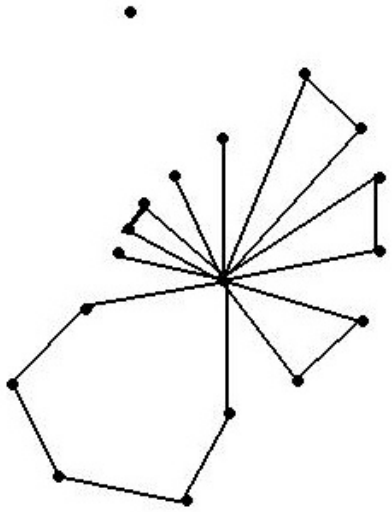


Figure 4.2: Regrouping

If the vertices would be grouped into pairs and each pair would form a triangle with the central point, we would squeeze in at most $\frac{3}{2}(v-1)$ edges. So our original graph will contain $e \leq \frac{3}{2}(v-1)$. □

Definition. *WWPHP is the weak version of WPHP, mapping function is now* $f : \{0,1\}^{n-1} \to \{0,1\}^{n-3}$. *Domain is four times larger than range.*

The lemma above offers a foundation for another NP-search problem, but sadly it cannot be used directly because paths would be given as set of consecutive vertices and that could mean exponential size. Nonetheless we will present the modified version that is NP, we will be using only small sections of this paths.

Definition. *A two-fork in a graph is set of 14 points, not necessarily distinct, labeled*

$$\{V_1, V_{1a1}, V_{1a2}, V_{1b1}, V_{1b2}, V_{1c1}, V_{1c2}, V_2, V_{2a1}, V_{2a2}, V_{2b1}, V_{2b2}, V_{2c1}, V_{2c2}\}$$

with the following properties: $V_1 \neq V_2$ *, vertices* $V_{1a1}, V_{1b1}, V_{1c1}$ *are all three*

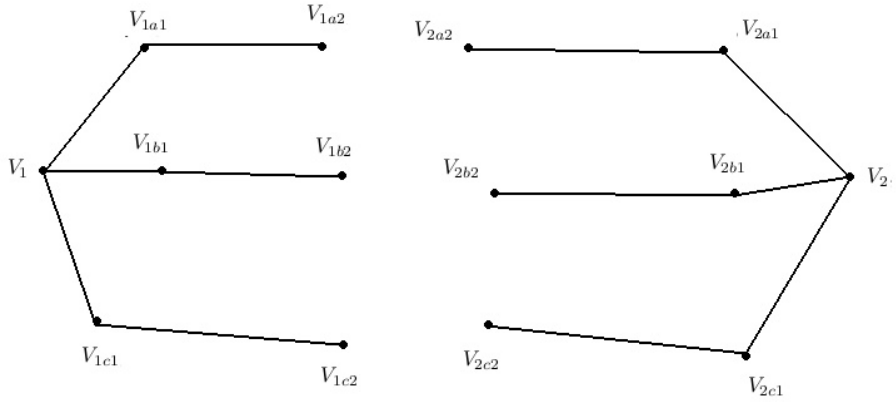


Figure 4.3: Two-fork

distinct or and also $V_{2a1}, V_{2b1}, V_{2c1}$ *are all three distinct. Edges in this structure are given by figure 4.3.*

Again, all 14 vertices need not to be distinct. In fact this structure could degenerate into five vertices. (figure 4.4) It could also degenerate into 4 vertices in the case of one arm leading directly from $V_1$to V_2. We will refer to V_1 and V_2 as base vertices.

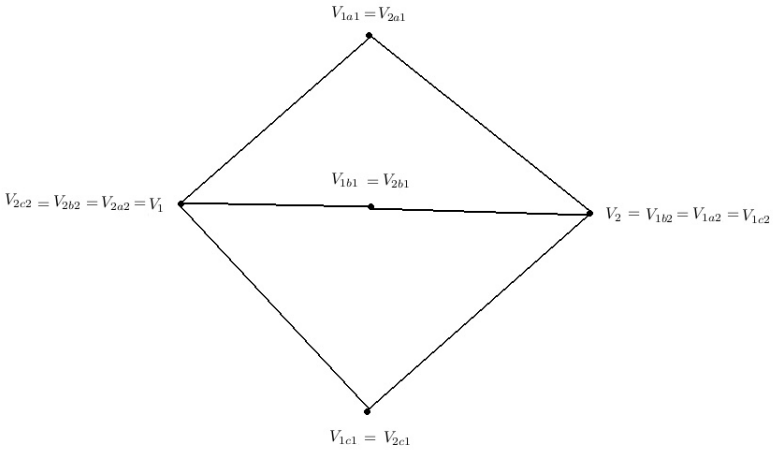


Figure 4.4: Degenerated two-fork

Definition. *2FORK is the problem of finding a two-fork in a graph with* $v = 2^n$ *vertices and* $\frac{3}{2}v$ *edges.*

This problem is in NP, because the witness is of polynomial size and can be verified in polynomial time. It is total, because there is enough edges for Lemma 1 to guarantee the existence of two vertices connected by three independent paths which implies that there is a two-fork. We are using beginnings of these paths to form it.

Reduction. *WWPHP is many-one reducible to 2FORK.*

Proof. We want to find collision in $f : \{0,1\}^{n-1} \to \{0,1\}^{n-3}$. These sets can be viewed as natural numbers, taking the binary string as a corresponding binary expression. Let us consider the following graph G with $V = 2^n$ vertices. First 2^{n-1} of them will represent domain of f, D. Next 2^{n-3} will be its range R . Edge relation between two vertices is computed as follows: There is an edge between u and v if and only if one of them, say u, is in D and one of the following conditions is met for the other:

$v = f(u) + 2^{n-1}$

$v = 2^{n-1} + 2^{n-3}$

$v = \lfloor u/2 \rfloor + 2^{n-1} + 2^{n-3} + 1$

First condition means that there is an edge between vertices that encode the function. Second condition guarantees that there is an edge between each vertex in D and special vertex S. Third condition pairs vertices in D and ensures there are two edges from each pair leading to a point in set Z defined as $G \setminus \{D \cup R \cup S\}$, illustrated in figure 4.5 by the thickest lines.

This is how the edge relation oracle is simulated by oracle for function f and this simulation needs only polynomial time in n because we are only adding and comparing at most n bit numbers.

For each vertex in D there are 3 edges, one leading to R, one leading to a special vertex $S = 2^{n-1} + 2^{n-3}$ and one to the set Z. Also note that each vertex in Z is connected to a maximum of 2 vertices in D. This graph contains exactly $\frac{3}{2}V$ edges. Therefore it can be presented to 2FORK, which outputs a two-fork T.

To find a collision, we consider two cases. If one of base vertices of T is in R its neighbouring vertices produce a triple collision.

We consider now the second case, that neither V_1 nor V_2 are in R. Than one of them, say V_1, has to be in D because there is only one vertex with degree > 2 in $G \setminus \{D \cup R\}$ and that is S. We have a vertex $V_1 \in D$ from T. Collision will be found in one of the arms leading from V_1 and back to D. It must lead back to D, because the graph G is bipartite. And by assuming that base vertices of T are not in R, collision is still found, even if they are connected with one edge. In this case, base vertices must be the vertex S and a vertex in D. This concludes the proof of reduction. □

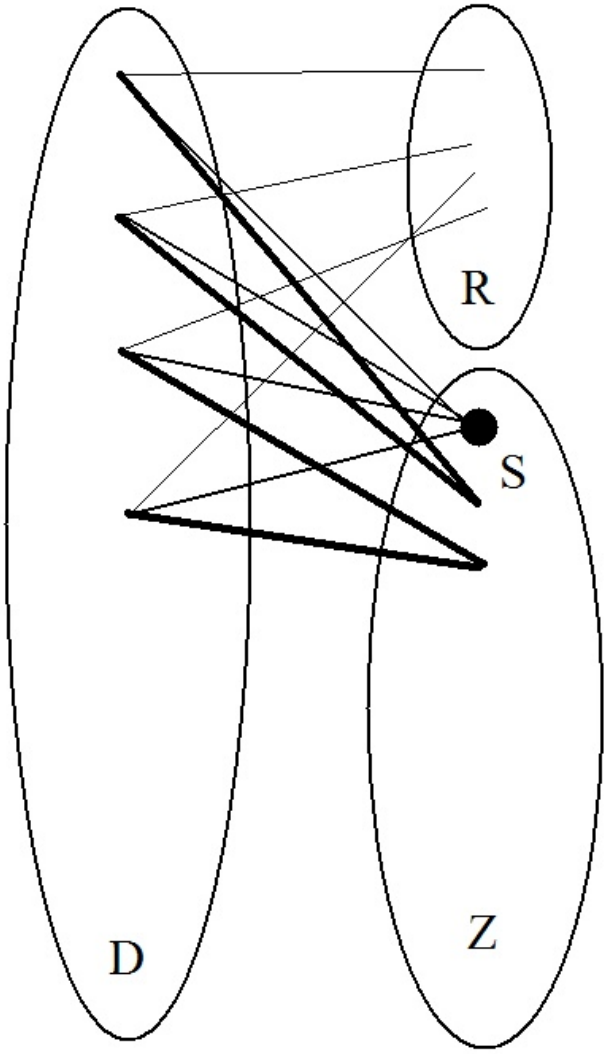


Figure 4.5: Edges in G, thickness according to type.

Chapter 5

Reduction to WPHP

We could ask a question, whether finding collision in multiple functions could be reduced into finding a collision in one. More precisely, functions

$$f_i : \{0,1\}^{n+1} \to \{0,1\}^n$$

for $i = 1, ..., l \leq n^k$. Note that number of functions is polynomial in the size of the input. We would like to find one function

$$F : \{0,1\}^{m+1} \to \{0,1\}^m$$

with such m, that if given a collision in F, we would be able to find collisions in f_i for all i. We expect m to be larger than n, but only polynomially. The following construction shows how to create such F from functions f_i.

In first step we do an amplification of input for functions f_i. We define

$$f_i^* : \{0,1\}^{m+1} \to \{0,1\}^n$$

with $m := n^{k+1}$ int the following way:

$$f_i^*(x_1, x_2, ..., x_n, x_{n+1}, ..., x_{m+1}) = f_i(...f_i(f_i(f_i(x_1, x_2, ..., x_{n+1}), x_{n+2}), x_{n+3})..., x_{m+1})$$

Lemma. *Given a collision for f_i^* it is possible to compute a collision in f_i in polynomial time.*

Proof. Having a collision in f_i^* means having nonequal $u_1, u_2 \in \{0,1\}^{m+1}$ such that $f_i^*(u_1) = f_i^*(u_2)$. Because $u_1 \neq u_2$ we could find the least significant bit where they differ on position n+1+q. First n values of $v_1$and v_2, the

collision in f_i, can be computed by iterating q steps in definition of f_i^* and the bits on position n+1 of v_1 and v_2 will hold the values of the least significant bit found above. If u_1 and u_2 differ for the first time in the position n+1, or a more significant position, the first n+1 bits of u_1 and u_2 will yield the collision directly. □

We can now define function $F : \{0,1\}^{m+1} \to \{0,1\}^m$. For $u \in \{0,1\}^{m+1}$

$$F(u) = (f_1^*(u), f_2^*(u), ..., f_{n^k}^*(u)).$$

Notice that values of F $\in \{0,1\}^{n*(n^k)=m}$.

Reduction. *The problem of finding collisions in all $f_i : \{0,1\}^{n+1} \to \{0,1\}^n$ is many-one reducible to finding a collision in F.*

Proof. Given $z_1, z_2 \in \{0,1\}^{m+1}$ collision in F, we directly obtain collisions for all f_i^* and using the lemma which we have proven above, we can compute collisions for all f_i in time polynomial in n. □

It appears open whether a similar reduction is possible for ordinary PHP (i.e. problem PIGEON) and for polynomially adaptive queries (as opposed to parallel, done above for WPHP).

Chapter 6

Conclusion

In this work, we presented the notion of NP-search problems, their important role in characterizing complexity of various everyday computation tasks and we recalled several reductions to illustrate. Then we centered our attention to PHP based search problems and their relationship to a problem from graph theory. We recalled the reduction to finding a homogeneous subgraph, and found a new reduction to paths related structures in graphs. We also presented a problem that can be reduced to PHP, the LEAF. This places pigeon hole principle into the hierarchy of NP search problems and their classes. Finally, in the last chapter, we examined how can many parallel queries to WPHP be substituted by one that is only polynomially larger.

Bibliography

[1] Beame, Cook, Impagliazzo, Edmonds, Pitassi *The Relative Complexity of NP Search Problems*, Proceedings of the twenty-seventh annual ACM symposium on Theory of computing 1997 p. 1-28

[2] Paul E. Black *big-O notation* in Dictionary of Algorithms and Data Structures, U.S. National Institute of Standards and Technology 2006

[3] B. Bollobas *Modern graph theory*, GTM 184, Springer, 1998, p. 136

[4] P. Erdos *Some remarks on the theory of graphs*, Bull. of the AMS, 1947 p.292-294

[5] Kazuhiro Suzuki, Dongvu Tonien, Kaoru Kurosawa and Koji Toyota *Birthday Paradox for Multi-collisions*, Oxford University Press 2008

[6] J. Krajicek *Structured pigeonhole principle, search problems and hard tautologies*, J. of Symbolic Logic, 70(2), 2005, p.619-630

[7] C.H. Papadimitriou *Computational Complexity* Addison-Wesley, 1994

[8] J. B. Paris, A. J. Wilkie, and A. Woods *Provability of the pigeonhole principle and the existence of infinitely many primes*, Journal of Symbolic Logic, 53, (1988), p.1235-1244

[9] A. A. Razborov *Formulas of bounded depth and some combinatorial problems*, Vopr. Kibern., 1988 p.149-166