A note on the computational complexity of a string orbit problem (DRAFT)

Tommi A. Junttila
Helsinki University of Technology,
Laboratory for Theoretical Computer Science,
P.O.Box 5400, FIN-02015 HUT, Finland

31 January 2001

Abstract

It is proven that finding the lexicographically greatest (smallest) string in the orbit of a given string under the position permuting action of a permutation group is an $\mathbf{FP}^{\mathbf{NP}}$-complete problem. This gives an affirmative answer to an open problem of Babai and Luks (1983, Proc. 15th Annual ACM STOC): determining the value of the element at a given position in the lexicographically greatest (smallest) string in the orbit of a string is a $\Delta_2\mathbf{P}$-complete problem. It is also shown that verifying whether a string is the lexicographical leader in its own orbit is a **co-NP**-complete problem. A generalized case of arbitrary polynomial time actions is also briefly discussed and an open problem is presented.

1 Introduction

Elimination of symmetric (isomorphic) objects is often used to alleviate the time and/or space requirements of combinatorial searches. The key idea is that given a permutation group acting on a set of objects, it is usually necessary to inspect only one representative object in each orbit induced by the group. To do this, one has either to (i) compare whether two objects are in the same orbit or (ii) construct a canonical representative of the object such that two objects are in the same orbit if and only if their canonical representatives are the same. Since we are usually dealing with a test of form "Given a set of already generated objects, is there an object in the set that is in the same orbit as a newly generated object?", procedures producing canonical representatives are extremely desirable as they reduce a sequence of object-wise orbit checks into one canonization and one set inclusion test. The most natural choice for a canonical representative is to choose the lexicographically greatest (smallest) object in the orbit.

This note considers the computational complexity of orbit problems in case a group is acting on strings, permuting the element positions. This kind of string orbit problems are, for instance, in the core of symmetry reduction methods designed to alleviate the state-space explosion problem appearing in the analysis of concurrent systems [5, 7, 4]. They also arise in the enumeration of combinatorial objects such as block designs, see e.g. [6].

First, for the sake of completeness, we restate a result by Clarke et al. [4]: the problem of deciding whether two strings are equivalent under the action of a group (i.e., belong to the same orbit) is polynomial time many-one equivalent to the problem of deciding whether a coset has a stabilizer for a set. This problem is polynomial time equivalent to many important problems in computational group theory. As the main result of the paper we next show that the *constructive string orbit problem*, that is, the problem of finding the lexicographically greatest (smallest) string in an orbit of a given string is an $\mathbf{FP}^{\mathbf{NP}}$-complete problem under polynomial time many-one reductions. This remains true even if the group acting on the string is an abelian 2-group. Our result is an improvement since the previous work only shows that the problem is **NP**-hard [1]. In fact, the **NP**-hardness result of Babai and Luks [1] shows that the problem of finding the lexicographically smallest (greatest) string in an orbit is at least as hard as finding the size of the largest independent set (clique) in a graph. When this is combined with the result of Krentel [10] published some years later, the constructive string orbit problem can be classified as an $\mathbf{FP}^{\mathbf{NP}[\log n]}$-hard problem. Our new result thus proves that it contains "more **NP**-completeness" and belongs to the same class as many important optimization problems such as the optimization version of the TRAVELING SALESMAN PROBLEM. We also give an affirmative answer to the open problem of Babai and Luks [1]: determining the value of an element at a given position in the lexicographic leader in an orbit of a string is a $\Delta_2\mathbf{P}$-complete problem. We also consider the problem of verifying whether a string is the lexicographical leader in its own orbit: this task is shown to be **co-NP**-complete. Finally, we briefly study the generalized case in which a group may act arbitrarily on strings, not only by permuting the element positions.

2 Preliminaries

The group of all permutations on a set A is denoted by $\mathrm{Sym}(A)$ and $G \leq H$ means that G is a sub-group of H. The following result is by Furst et al. [8].

Lemma 2.1 *Given a set of generators for a group $G \leq \mathrm{Sym}(A)$ on a finite set A, one can determine in deterministic polynomial time whether a permutation g belongs to G.*

In the rest of the paper, by group we mean a permutation group on a finite set and all the groups are assumed to be given by means of their generator sets. For permutation group algorithms the reader is referred to [3].

Let I be a finite, linearly ordered set of *indices* and Σ a finite, linearly ordered alphabet. A *Σ-string on I* is a mapping $s : I \to \Sigma$ and the set of all Σ-strings on I is denoted by Σ^I. The orders on I and Σ define the linear lexicographical order, denoted by $<$, on Σ^I. A permutation $g \in \mathrm{Sym}(I)$ *acts on a string* $s \in \Sigma^I$ resulting into the string $gs = s \circ g^{-1}$, where $\circ$ is the function composition operator. Given a group $G \leq \mathrm{Sym}(I)$, the *G-orbit* of a string $s \in \Sigma^I$ is defined by $[s]_G = \{gs \mid g \in G\}$. Two strings, s and s', are *equivalent under G*, denoted by $s \equiv_G s'$, if they belong to the same orbit, i.e., if $(\exists g \in G)(gs = s')$. As an example, let $I = \{1, 2, 3, 4\}$ with the order $1 < 2 < 3 < 4$, $\Sigma = \{a, b, c\}$ with $a < b < c$, $s = cabc = \{1 \mapsto c, 2 \mapsto a, 3 \mapsto b, 4 \mapsto c\} \in \Sigma^I$ and G a sub-group of $\mathrm{Sym}(I)$ generated by $g = \left(\begin{smallmatrix}1&2&3&4\\2&3&1&4\end{smallmatrix}\right)$. Now $gs = bcac$, $(g \circ g)s = abcc$, $(g \circ g \circ g)s = s$, $gs < s$ and $[s]_G = \{abcc, bcac, cabc\}$. The strings $abcc$ and $cabc$ are the *lexicographically smallest* and *greatest strings in the G-orbit of s*, respectively.

For computational complexity theory the reader is referred to, e.g. [9, 12]. In this paper we use *polynomial time many-one reductions.* $\mathbf{FP}^{\mathbf{NP}}$ denotes the class of function problems computable by a deterministic Turing machine in polynomial time with access to an **NP**-oracle, while $\Delta_2\mathbf{P}$ is the class of decision problems solvable by similar machinery.

3 The decision problem

We first study the computational complexity of deciding whether two strings are equivalent under the action of a group G, i.e., whether they belong to the same G-orbit.

Problem 3.1 String Orbit. *Given a group $G \leq \mathrm{Sym}(I)$ and two strings $s, s' \in \Sigma^I$, decide whether $s \equiv_G s'$.*

In the light of Lemma 2.1, the problem is obviously in **NP** since we can guess a permutation $g \in \mathrm{Sym}(I)$, check that $g \in G$ and test whether $gs = s'$, all in non-deterministic polynomial time. String Orbit is also Graph Isomorphism-hard since the adjacency matrix of a graph can be interpreted as a string while the group acting on such a string is the natural extension of the symmetric group permuting the vertices of the graph [5]. Clarke et al. [4] use the following problem to classify the computational complexity of String Orbit more accurately.

Problem 3.2 Set Stabilizer in a Coset (SSC). *Given a finite set Y, a subset $Y' \subseteq Y$, a group $G \leq \mathrm{Sym}(Y)$ and a permutation $\gamma \in \mathrm{Sym}(Y)$, decide whether there exists a $\sigma \in G\gamma$ that stabilizes the set Y', i.e., $\sigma(Y') = Y'$.*

As noted in [4], SSC is polynomial time equivalent to many important problems such as Group Intersection and Coset Intersection Emptiness in computational group theory (see also [11]).[1] These problems are believed to be harder than Graph Isomorphism but not **NP**-complete. We now restate a result by Clarke et al.:

Theorem 3.3 *[4]* String Orbit *and* SSC *are polynomial time many-one equivalent.*

4 Canonization

We now turn to the problem of finding a canonical representative of a string, which is here defined to be the lexicographically greatest string in the same orbit.

Problem 4.1 Constructive String Orbit (CSO) *[4]. Given a group $G \leq \mathrm{Sym}(I)$ and a string $s \in \Sigma^I$, find the lexicographically greatest string in the G-orbit of s.*

We also consider a decision version CSO(D) of CSO that, given a goal string s', asks whether there exists a string in the orbit of s that is lexicographically greater than or equal to s'.

Lemma 4.2 CSO(D) *is* **NP**-*complete.*

Proof. Clearly CSO(D) is in **NP** since we can guess a permutation $g \in G$ and then see if $gs \geq s'$ in polynomial time. To show **NP**-hardness we reduce from the **NP**-complete problem Clique for undirected graphs as in a proof in [1, Sec. 3.1]. Consider the adjacency matrix of an undirected graph with n nodes $\{1, 2, \ldots, n\}$, represented as a $\{0, 1\}$-string s on $I_n = \{\langle i, j\rangle \mid 1 \leq i, j \leq n\}$ (it is assumed that each node has a self-loop). Order I_n in a way that I_m, for $m < n$, (the upper left square) always precedes its complement (the elements of I_n not in I_m). Let G be the natural extension of $\mathrm{Sym}(\{1, 2, \ldots, n\})$ to I_n so that $g(\langle i, j\rangle) = \langle g(i), g(j)\rangle$ for each $g \in \mathrm{Sym}(\{1, 2, \ldots, n\})$. Since $\mathrm{Sym}(\{1, 2, \ldots, n\})$ is generated by two generators, the swapping of elements 1 and 2 and the rotation of $\{1, 2, \ldots, n\}$ by one step, G is generated by the permutations corresponding to these generators. Choose the string s' so that $s'(\langle i, j\rangle) = 1$ if $\langle i, j\rangle \in I_m$ and 0 otherwise for some $m \leq n$. Now there is a clique of size m (or greater) iff there is a string in the orbit of s that is lexicographically greater than or equal to s'. □

We now prove the main result of this paper.

[1] Here *polynomial time equivalence* between two problems means that if either has a polynomial time algorithm, then and only then both have.

Theorem 4.3 CSO *is* $\mathbf{FP^{NP}}$*-complete.*

Proof. There exists $|\Sigma|^{|I|}$ Σ-strings on I. With binary search, we can find the lexicographically greatest element by issuing $\mathcal{O}(\log(|\Sigma|^{|I|})) = \mathcal{O}(|I| \times \log(|\Sigma|))$ queries to the **NP** oracle asking the question CSO(D). Thus CSO is in $\mathbf{FP^{NP}}$. To show $\mathbf{FP^{NP}}$-hardness, we reduce from the problem Maximum Satisfying Assignment (MSA).

Problem 4.4 MSA. *Given a Boolean formula f over $x_1, \dots, x_n$, return the lexicographically greatest $x_1 \cdots x_n \in \{0,1\}^n$ that satisfies φ or 0 if φ is not satisfiable.*

MSA was shown to be $\mathbf{FP^{NP}}$-complete by Krentel [10]. It is $\mathbf{FP^{NP}}$-complete for k-CNF formulas, $k \geq 3$, too. Formally, a k-CNF formula f is a conjunction of m clauses, where each clause c_j is a disjunction of k literals (negative or positive appearances of a variable x_i). That is, $f = c_1 \wedge \cdots \wedge c_m$, where $c_j = l_{j,1} \vee \cdots \vee l_{j,k}$. For a non-empty set $X \subseteq \{x_1, \dots, x_n\}$ of Boolean variables, a valuation is a function $T : X \to \{0,1\}$. A valuation T can be inverted w.r.t. a Boolean variable $x \subseteq \{x_1, \dots, x_n\}$, resulting in a new valuation $\mathrm{inv}_x(T)$ identical to T except that the value of x is flipped if x is in X. It is clear that inversions commute and are self-inverting, i.e. $\mathrm{inv}_x(\mathrm{inv}_y(T)) = \mathrm{inv}_y(\mathrm{inv}_x(T))$ and $\mathrm{inv}_x(\mathrm{inv}_x(T)) = T$ for all $x, y \in \{x_1, \dots, x_n\}$. The valuation mapping all variables to 0 is denoted by T_0.

Now consider a k-CNF formula $f = c_1 \wedge \cdots \wedge c_m$ of m clauses, where each clause c_j involves Boolean variables $\mathrm{vars}(c_j) \subseteq \{x_1, \dots, x_n\}$. For each c_j, we define the set

$$C_j = \{c_j^T \mid T \text{ is a valuation for } \mathrm{vars}(c_j)\}.$$

Note that there exists at most 2^k elements in each C_j (a constant for fixed k). Let

$$I_f = \{x_1, \dots, x_n\} \cup \{\bar{x}_1, \dots, \bar{x}_n\} \cup \bigcup_{1 \leq j \leq m} C_j.$$

The group $G_f \leq \mathrm{Sym}(I_f)$ is generated by the permutations g_i, $1 \leq i \leq n$, such that (i) $g_i(x_i) = \bar{x}_i$ and $g_i(\bar{x}_i) = x_i$, (ii) $g_i(x_j) = x_j$ and $g_i(\bar{x}_j) = \bar{x}_j$ for all $j \neq i$ and (iii) $g_i(c_j^T) = c_j^{\mathrm{inv}_{x_i}(T)}$. It is easy to see that $g_i \circ g_i$ is the identity mapping on I_f for each $1 \leq i \leq n$ and $g_i \circ g_j = g_j \circ g_i$ for $1 \leq i, j \leq n$.

Order I_f such that the elements of the form $c_j^{T_0}$ are first, followed by (i) $x_1, \dots, x_n$, (ii) $\bar{x}_i$'s and (iii) finally the rest of the elements in I_f (c_j^T's having at least one variable assigned to value 1).

Take the string $s : I_f \to \{0,1\}$ such that (i) $s(x_i) = 0$, (ii) $s(\bar{x}_i) = 1$ and (iii) $s(c_j^T) = 1$ iff the clause c_j is evaluated to 1 under the valuation T. The string s corresponds to the truth valuation assigning all the variables to false ($s(\bar{x}_i) = 1$ and $s(x_i) = 0$). The elements $c_j^{T_0}$ correspond to the evaluation of the clauses in that truth valuation.

Now when a generator permutation g_i is applied to s, the valuation of the variable x_i is flipped ($s(\bar{x}_i)$ and $s(x_i)$ are swapped). At the same time, the evaluation of clauses involving variable x_i are swapped to those where the valuation of x_i is flipped (each c_j^T is swapped with $c_j^{\mathrm{inv}_{x_i}(T)}$). It is not hard to see that after applying a finite sequence $g = g_{i_1} \circ g_{i_2} \circ \cdots \circ g_{i_h}$ of generator permutations to the string s, the resulting string gs is such that the value at each position $c_j^{T_0}$ correctly describes the evaluation the clause c_j under the truth assignment described by the values in positions of form x_i and $\bar{x}_i$. Thus the group generated by the identity permutation on I_f and the generators g_i, $1 \leq i \leq n$, is such that its action on the $\{0,1\}$-string s on I_f generates all the truth assignments on the variables $\{x_1, \ldots, x_n\}$ and simultaneously keeps track of the clause evaluations in those truth assignments.

The lexicographically greatest string in the orbit of s is thus such that it either (i) does not have all the m first positions $c_j^{T_0}$ assigned to 1, in which case the original formula φ is not satisfiable, or (ii) has all the m first positions $c_j^{T_0}$ assigned to 1 and the next n positions $x_1, \ldots, x_n$ describe the lexicographically maximum $x_1 \cdots x_n \in \{0,1\}^n$ satisfying f. $\square$

Since the group G_f in the proof above is actually an abelian 2-group, CSO is $\mathbf{FP^{NP}}$-complete for such groups, too. If we were interested in the lexicographically smallest string version of the CSO instead, the proof could be made to hold by swapping the roles of 0 and 1.

Because determining whether the $x_n = 1$ in the lexicographically greatest satisfying truth assignment $x_1 \cdots x_n$ for a Boolean formula is a $\Delta_2\mathbf{P}$-complete problem [10], we are able to give an affirmative answer to the open problem of Babai and Luks [1]:

Corollary 4.5 *Given a group $G \leq \mathrm{Sym}(I)$, a string $s \in \Sigma^I$, an index $i \in I$ and an alphabet $\alpha \in \Sigma$, determining whether $s'(i) = \alpha$, where s' is lexicographically greatest (smallest) string in the G-orbit of s, is a $\Delta_2\mathbf{P}$-complete problem. This holds for abelian 2-groups, too.*

A note should be made here: choosing the lexicographically greatest (smallest) string in an orbit to be the canonical representative is probably not the most easily computable choice. This is because Blass and Gurevich [2] show that there exist equivalence relations for which lexicographical leaders are much harder to compute than arbitrary canonical representatives. And fortunately we do not sometimes need the lexicographically greatest (smallest) or even otherwise canonical representative string in the orbit. The symmetry reduction method for concurrent systems, for instance, can cope with multiple representatives per orbit. Of course, this may cause the time and/or space requirements to grow since more objects are generated.

5 Verifying canonicity

Another closely related problem is to check whether a given string is the lexicographical leader in its own orbit under the action of a given group. This kind of problems often arise in the enumeration of combinatorial objects such as block designs, see, e.g. [6].

Problem 5.1 CANONICITY VERIFICATION. *Given a group $G \leq \mathrm{Sym}(I)$ and a string $s \in \Sigma^I$, is s the lexicographically greatest (smallest) string in the G-orbit of s?*

Theorem 5.2 CANONICITY VERIFICATION *is* **co-NP**-*complete.*

Proof. Inclusion in **co-NP** is straightforward: guess a permutation g in G and check whether $gs \geq s$. To prove **co-NP**-hardness, consider a 3-CNF formula $f = c_1 \wedge \cdots \wedge c_m$ over Boolean variables $x_1, \ldots, x_n$, where $c_i = l_{i,1} \vee l_{i,2} \vee l_{i,3}$. Take the 4-CNF formula $f' = \neg x_{n+1} \vee f \equiv c'_1 \wedge \cdots \wedge c'_m$ where x_{n+1} is a new variable not occurring in f and $c'_i = \neg x_{n+1} \vee l_{i,1} \vee l_{i,2} \vee l_{i,3}$. Construct the index set $I_{f'}$ and the group $G_{f'}$ as in the proof of Theorem 4.3. Order $I_{f'}$ such that the elements of form $c_j^{T_0}$ are first, followed by x_{n+1}, then all $x_1, \ldots, x_n$ and finally the rest. Consider the string s corresponding to the truth valuation in which x_{n+1} is false and all x_is are true. This truth valuation clearly satisfies f'. Now if there is a truth assignment that satisfies f' and whose corresponding string is lexicographically greater than s, then x_{n+1} must evaluate to true in it. That is, f is satisfiable iff s is *not* the lexicographically greatest string in its orbit. $\square$

6 A generalized case

Finally, we consider a generalized case of the string orbit problem. Now a group may act on strings in an arbitrary way, not only by permuting the positions. Formally, an *action* of a group G on a set X is a mapping $H : G \times X \to X$ such that (i) $H(\mathrm{id}, x) = x$ for all $x \in X$, where id is the identity element of G, and (ii) $H(g_1 \circ g_2, x) = H(g_1, H(g_2, x))$ for all $x \in X$ and all $g_1, g_2 \in G$. We know from the basic group theory that for each $g \in G$, the function $\sigma_g : X \to X$ such that $\sigma_g(x) = H(g, x)$ is a permutation of X.Furthermore, the mapping $\phi : G \to \mathrm{Sym}(X)$ defined by $\phi(g) = \sigma_g$ is a group homomorphism and thus $\phi(G)$ is a sub-group of $\mathrm{Sym}(X)$. The G, H-orbit of $x \in X$ is $[x]_{G,H} = \{H(g, x) \mid g \in G\}$ which is the same as the $\phi(G)$-orbit of x. We say that the action $H : G \times X \to X$ is polynomial-time computable if $H \in$ **FP**.

Example 6.1 Consider the position permuting action $PP : G \times \Sigma^* \to \Sigma^*$, where $G \leq \mathrm{Sym}(\{1, 2, \ldots, n\})$, defined by $H(g, s) = s \circ g^{-1}$ if $|s| = n$

and s otherwise. (Here a string s of length $|s|$ is seen as a function $s : \{1, 2, \ldots, |s|\} \to \Sigma$.) Clearly PP is polynomial-time computable. ♣

In the following, we consider polynomial-time computable actions $H : G \times \Sigma^* \to \Sigma^*$, where $G \leq \mathrm{Sym}(A)$ for a finite set A. Note that for polynomial-time computable actions, the lengths of strings in an orbit are a polynomially related, $\forall s \in \Sigma^* : \forall s' \in [s]_{G,H} : |s'| = \mathcal{O}(|s|^k)$ for a fixed $k > 0$ depending on H.

Problem 6.2 String Orbit under Polynomial-Time Action. *Given two strings $s, s' \in \Sigma^*$, a group $G \leq \mathrm{Sym}(A)$ for a finite set A and a polynomial time computable action $H : G \times \Sigma^* \to \Sigma^*$, decide whether s and s' are in the same G, H-orbit.*

Obviously this problem is in **NP**: just guess a permutation $g \in G$, compute $H(g, s)$ and compare with s'. Furthermore, it is at least as hard as String Orbit since String Orbit is a special case of it. On the other hand, it is open whether the problem is **NP**-complete, as hard as String Orbit or somewhere in between. We now consider the canonization version.

Problem 6.3 Constructive String Orbit under Polynomial-Time Action. *Given a string $s \in \Sigma^*$, a group $G \leq \mathrm{Sym}(A)$ for a finite set A and a polynomial time computable action $H : G \times \Sigma^* \to \Sigma^*$, find the lexicographically greatest string in the G, H-orbit of s.*

The corresponding decision version asks, given s, G and H as above and a goal string $s' \in \Sigma^*$, whether there exists a string s'' in the G, H-orbit of s which is lexicographically greater than or equal to s'. Clearly it is **NP**-complete. It is **NP**-hard because CSO(D) is its special case and it is in **NP** because we can (i) guess a permutation $g \in G$, (ii) compute the string $s'' = H(g, s)$ and (iii) compare whether $s'' \geq s'$, all in non-deterministic polynomial time. We can therefore conclude that Constructive String Orbit under Polynomial-Time Action is $\mathbf{FP}^{\mathbf{NP}}$-complete: the $\mathbf{FP}^{\mathbf{NP}}$-hardness follows from the fact that CSO is its special case — on the other hand, we can find the lexicographically greatest string in the G, H-orbit of s with $\mathcal{O}(\log |\Sigma|^{|s|^k}) = \mathcal{O}(|s|^k \log |\Sigma|)$ queries to its decision version oracle. Also the problem canonicity verification extended to polynomial time actions clearly stays **co-NP**-complete.

Acknowledgements The author wishes to thank Prof. Patric Östergård and Prof. Ilkka Niemelä for comments. The financial support of Helsinki Graduate School in Computer Science and Engineering (HeCSE), the Academy of Finland (project no. 47754) and Tekniikan Edistämissäätiö ("Foundation of Technology") is gratefully acknowledged.

References

[1] L. Babai and E. M. Luks. Canonical labeling of graphs. In *Fifteenth Annual ACM Symposium on Theory of Computing*, pages 171–183. ACM, 1983.

[2] A. Blass and Y. Gurevich. Equivalence relations, invariants, and normal forms. *SIAM Journal on Computing*, 13(4):682–689, 1984.

[3] G. Butler. *Fundamental Algorithms for Permutation Groups*, volume 559 of *LNCS*. Springer-Verlag, 1991.

[4] E. M. Clarke, E. A. Emerson, S. Jha, and A. P. Sistla. Symmetry reductions in model checking. In A. J. Hu and M. Y. Vardi, editors, *Computer Aided Verification (CAV'98)*, volume 1427 of *LNCS*, pages 147–158. Springer, 1998.

[5] E. M. Clarke, R. Enders, T. Filkorn, and S. Jha. Exploiting symmetry in temporal logic model checking. *Formal Methods in System Design*, 9(1/2):77–104, 1996.

[6] P. C. Denny and P. B. Gibbons. Case studies and new results in combinatorial enumeration. *Journal of Combinatorial Design*, 8(4):239–260, 2000.

[7] E. A. Emerson and A. P. Sistla. Symmetry and model checking. *Formal Methods in System Design*, 9(1/2):105–131, 1996.

[8] M. Furst, J. Hopcroft, and E. Luks. Polynomial-time algorithms for permutation groups. In *21st Annual Symposium on Foundations of Computer Science*, pages 36–41. IEEE, 1980.

[9] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness.* W. H. Freeman and Company, San Francisco, 1979.

[10] M. W. Krentel. The complexity of optimization problems. *Journal of Computer and System Sciences*, 36(3):490–509, 1988.

[11] E. M. Luks. Permutation groups and polynomial-time computation. Technical Report CIS 93–01, Department of Computer and Information Science, University of Oregon, 1993. Also published in *Groups and Computation*, L. Finkelstein and W. M. Kantor, Eds. DIMACS Series in Discrete Mathematics and Theoretical Computer Science, DIMACS, 1993.

[12] C. H. Papadimitriou. *Computational Complexity.* Addison-Wesley, Reading, Massachusetts, 1995.