Self-reducibility structures and solutions of NP problems

J.L. Balcázar

Report LSI–88–19

Self-reducibility structures and solutions of NP problems †

José L. Balcázar
Department of Software (Lenguajes y Sistemas Informáticos)
Universidad Politécnica de Cataluña
08028 Barcelona, SPAIN

Abstract: Using polynomial time self-reducibility structures, we characterize certain "helping" notions, show how the characterization provides the main tool for the proof of known relationships between decisional and functional *NP*-complete problems, and extend this relationships to the case of optimization *NP*-complete problems.

Resum: Fent servir estructures d'auto-reduïbilitat, caracteritzem conceptes relatius al "helping", provem que la caracterització proporciona una eina bàsica per a la demostració de resultats coneguts referents a problemes *NP*-complets decisionals i funcionals, i estenem aquests resultats a problemes *NP*-complets d'optimització.

1. Introduction

A central aspect of Computational Complexity Theory consists of attempting to reach a better understanding of the mathematical phenomena that might cause the widely observed fact that some important problems are algorithmically difficult. Since in many cases there is no mathematical proof of this hardness, and evidence for that difficulty is based simply in the failure to design feasible algorithms to solve these problems, one of the lines of research tries to make apparent the underlying mathematical structures characterizing the relationships either between the instances of a given problem, or between the degrees of unfeasibility of the problems themselves.

Among the structural notions that are defined among instances of a particular problem, thus providing an internal structure on the corresponding set, a very important one is the concept of "self-reducible set", which has been used by several authors so far. Its naturalness is quite easy to argue, since a self-reducible set is one for which a recursive program can be designed without resorting to additional parameters. Of course, the correctness of the recursive program relies on a well-founded set guaranteeing termination and allowing an inductive verification argument. From the point of view of the amount of resources required to run the program, the depth of the recursion plays a crucial role; thus, bounds will be imposed on the length of the descending chains in the well-founded set (see the definitions below). It is natural to study the complexity-theoretic aspects of a concept so clearly related to basic constructions of Computer Science.

To present this concept in an adequate way, I use oracle Turing machines, in which oracle calls are identified to the calls to the recursive subroutine; this is achieved

† These results were announced at Symposium on Theoretical Aspects of Computer Science, Passau, 1987. This work was partially supported by CIRIT.

by analyzing the behavior of the oracle Turing machine when the oracle set is the accepted set itself, and requiring that the queries be smaller than the input in the corresponding well-founded set to ensure termination.

Important contributions to Complexity Theory are based on the existence of self-reducible complete sets for the most inmportant complexity classes; see [2] and the references there. Other contributions that are relevant to the present paper appear in [8] and [9], where Schnorr uses two kinds of self-reducibility to study the relationship between the decisional and functional versions of NP-complete problems. More precisely, he determines the complexity of the polynomial time Turing degree of the functional version of certain NP-complete problems, by showing that it coincides with the degree of the decisional problem: both are interrelated by polynomial time Turing reductions. A generalization of this result to all NP-complete sets appears in [3], which I extend to the case of optimization problems. I relate the structural aspects of the proof to another concept recently appeared in the literature, by showing how a technical notion present in the proof characterizes precisely a class of sets introduced by Ko [5]: the "self-1-helpers". Let us explain briefly this notion.

Intuitively, a set A "1-helps" another set B if B is Turing reducible to A in polynomial time, and moreover the machine does not "rely too much" on A. If another oracle is used which tries to lead to incorrectly accepting an input, the reduction procedure detects this fact and recovers from the error, obtaining always a correct answer; the price to be paid is a longer (e.g. exponential) running time. A set A "helps" a set B if such a situation holds, in which no errors are made at all: neither accepting nor rejecting an input; in fact, this "two sided" notion was studied first in [10], where helping was introduced. A "self-1-helper" is a set that 1-helps itself. A recent work of Ko [5] relates these forms of "helping" to self-reducible sets. He asks for structural characterizations of the classes studied. Here I show that the techniques used to strengthen the results of Schnorr rely on a clear property which characterizes exactly the "self-1-helpers" of [5]. In [3], this property is called "functional self-reducibility"; for reasons that will become apparent in the text, I prefer to name this property "having self-computable witnesses".

I present in the next section some definitions and preliminaries. Section 3 presents the definition of self-computable witnesses, the characterization, and the use of the notion in the proof of [3]. Finally, in section 4 these results are extended to the case of optimization problems, using again the same techniques.

2. Definitions and basic properties

All the sets here consist of words over the alphabet $\Gamma = \{0, 1\}$. I denote by λ the empty word. Complexity classes are defined in the usual way: among them, P, NP, $PSPACE$, and the classes of the polynomial time hierarchy. For definitions see [1] and [11]. PF is the class of all functions computable in polynomial time. $PF(A)$ is the relativization of PF to the oracle A. SAT denotes the well-known NP-complete problem of deciding the satisfiability of boolean formulas, and QBF the $PSPACE$-complete problem of deciding the truth of quantified boolean formulas. Departing from the usual convention, the NP-complete sets here are meant to be complete under polynomial time Turing reductions.

Among the several definitions of polynomial time self-reducibility proposed in the literature, the most general one is proposed in [7]. The results here do require such

a general notion, which is invariant under polynomial time isomorphism, and allows arbitrary partial orders for guaranteeing termination of the self-reduction processes if decrasing chains are short enough. Formally:

1. *Definition.* A set A is *polynomial time self-reducible* if and only if there is a polynomial time deterministic oracle Turing machine such that $A = L(M, A)$, and on each input x every word queried to the oracle is smaller than x in a (fixed) partial order satisfying:

- If x is smaller than y then $|x| \leq p(|y|)$ for some polynomial p.
- It is decidable in polynomial time whether x is smaller than y.
- Every decreasing chain is bounded in length by a polynomial of the length of its maximum element.

I will omit the words "polynomial time", which are to be assumed in every use made of any of these definitions. It can be seen that this definition captures the essential properties of the self-reducible sets (mainly, the NP-complete, co-NP-complete, and $PSPACE$-complete sets). See [2] and the references there.

Some variants of the notion of self-reducibility are obtained by imposing certain restrictions on the self-reducing machine. I define here the only one I will use, the disjunctive self-reduciblity:

2. *Definition.* The set A is *disjunctively self-reducible*, or *d-self-reducible*, if it is self-reducible and the oracle machine M witnessing this fact accepts its input whenever the oracle answers positively to any of the queries.

In [4], other forms of self-reducibility are presented (conjunctive, positive truth-table...) and some of their properties are studied. The following property is argued in this reference and also in [12], where d-self-reducibility is studied in depth.

3. *Proposition.* If A is d-self-reducible then $A \in NP$.

3. Functional NP-complete problems

Sets in NP are usually interpreted as decision problems. Functional versions of these problems are interesting too, and can be defined for every set in NP as follows. Consider any NP set A. By the quantifier characterization of NP, it follows that there is a set B in P and a polynomial p such that

$$x \in A \iff \exists y\, (|y| \leq p(|x|))\, (\langle x, y\rangle \in B) \qquad (*)$$

The word y is called a "witness" of the fact that x is in A. Intuitively, each such y is in a sense a "solution" of the problem A for input x, if the set B in P is chosen appropriately. For example, consider for B sets like

$\{\langle x, y\rangle \mid x$ encodes a boolean formula and y encodes a satisfying assignment for it$\}$

or

$\{\langle x, y\rangle \mid x$ encodes a graph and y encodes a hamiltonian circuit for it$\}$

In this way the decisional problems SAT and Hamiltonian Circuit are defined; then it is clear that a witness y encodes precisely the solution to the functional problem of finding a satisfying assignment for a formula, if one exists, or finding a hamiltonian circuit for a graph if one exists, respectively.

Consider the functional version of a generic NP set A defined as in $(*)$, which consist of, given input x, computing a solution y if such a solution exists; i.e. compute a y such that $\langle x, y\rangle \in B$. Observe, first, that this is not necessarily a single valued partial function, and second, that this problem depends on the set B in P from which A is defined by existential quantification, and on the polynomial p. I also refer to this problem as "computing witnesses". A discussion on the particular cases in which the function is defined also out of the set, or is single-valued, appears in [13].

Fix a B in P and a polynomial p. Let A be the NP set for which $(*)$ holds. I define next the functional solutions for the set A. This notion of functional solution corresponds to any partial function f such that the domain of f is A, $|f(x)| \leq p(|x|)$, and for every $x \in A$, $\langle x, f(x)\rangle \in B$. Thus, both A and the set of functional solutions are uniquely defined by B and p.

4. *Definition.* The set funct-sols(B, p) is the set

$$\{f \mid \forall x[\text{ if } \exists y\ (|y| \leq p(|x|))\ (\langle x, y\rangle \in B) \text{ then}$$
$$f(x) \text{ is defined, } |f(x)| \leq p(|x|), \text{ and } \langle x, f(x)\rangle \in B]\}$$

I want to compare sets in the following terms: given two decisional problems, I study the case in which one of them contains enough information not only to decide the other, but even to construct a solution for it. The main result in this section is a characterization of the self-1-helping property studied by Ko [5].

5. *Definition.* Given a set A in NP, say that C *provides witnesses for* A if and only if there is a set B in P and a polynomial p such that $(*)$ holds, and for which funct-sols$(B, p) \cap PF(C) \neq \emptyset$. If A provides witnesses for itself then I say that it has *self-computable witnesses.*

Thus, for a set having self-computable witnesses, the functional problem can be reduced to the decision problem, in the sense that solutions can be computed with an oracle for the set. To put it in another way, given the set A as an "untrustworthy" oracle, and given an input x which is answered positively by A, the oracle allows us to produce a polynomial time checkable proof that his answer was correct. It should be observed that the notion of a set having self-computable witnesses was already proposed, under the name of "functional self-reducibility", in an unpublished work by Borodin and Demers [3]; I consider more appropriate the naming convention used here. I prove below one of the results of their article.

As part of his work in the study of the notion of "helping", Ko has proposed the following definitions.

6. *Definition.* A set C *1-helps* a set A if and only if there is an oracle machine M such that for every oracle X, $L(M, X) = A$, and such that with oracle C, every word x

accepted by M is accepted in polynomial time. A set A is a *self-1-helper* if and only if A 1-helps itself.

The definition is from [5]; motivation and insights about this notion can be found there. Let us just mention that the definition formalizes a situation in which one has a machine that sooner or later will come up with a correct answer, but has a certain degree of interaction with an external agent which may "help" the machine so that it finishes quickly. Oracle machines M such that, for every oracle X, $L(M, X)$ is always the same are called "robust oracle machines". An equivalent formulation of 1-helping is as follows: M works always in polynomial time, $L(M, X) \subseteq A$ for every oracle X, and $L(M, C) = A$.

The announced characterization is as follows:

7. *Theorem.* A set C 1-helps a set A if and only if C provides witnesses for A. In particular, the self-1-helpers are precisely those sets having self-computable witnesses.

Proof. Assume that C 1-helps the robust machine M which always accepts A. Then A can be defined by existential quantification of the following polynomial time relation:

$$B = \{\langle x, y\rangle \mid y \text{ encodes a } p(|x|) \text{ long computation of } M \text{ accepting } x\ \}$$

where p is an appropriately selected polynomial. Each computation y corresponds to a way of answering the oracle queries. Let us show that B fulfills $(*)$: assuming that a polynomially long computation y of M accepts x, then, since M always accepts A, $x \in A$; conversely, if $x \in A$ then the computation corresponding to the oracle C finishes in time $p(|x|)$ for the appropriate polynomial p and accepts. Moreover, C allows one to construct witnesses for this way of defining A by simply simulating the computation of M with oracle C.

To show the converse, let $B \in P$ defining A such that C provides witnesses for it. Consider a machine M that, on input x, first assumes that its oracle is C and uses it to construct a polynomially long witness, by simulating the machine which computes a solution; then it checks in polynomial time that the witness obtained is indeed a solution, i.e. to check that the oracle was correct. If no witness is found, or if the witness produced fails to be really a solution, then M decides whether x is in A by performing an exhaustive search for solutions, thus simulating an exponential time machine for A. Such a machine always accepts A, and is polynomial time bounded on inputs in A if the right oracle, C, is available. ∎

The remaining of this section is devoted to formulating a known result in terms of self-computable witnesses, as a previous step for the results to be presented in the next section. In [9] it is shown that for "self-transformable" problems (a weaker form of disjunctive self-reducibility) there is a functional solution that is "equally hard" to compute as the decisional form of the problem: i.e., "self-transformable" problems have self-computable witnesses. This encompasses most of the known NP sets. In [3], this result is generalized so that it includes all NP-complete sets, without the need of checking whether they are self-transformable. Note that it is not known whether all the NP-complete sets (even with respect to the stronger m-reducibility) are self-reducible. The proof is based on a prefix searching technique as in [13], using as a technical concept the following notion.

8. *Definition.* The set prefix-sols(B,p) is the set

$$\{\langle x,z\rangle \mid \exists y\ (|y| \leq p(|x|))\ \langle x,y\rangle \in B \text{ such that } z \text{ is a prefix of } y\ \}$$

The interest of this set lies in the following observation, which is stated in [12]:

9. *Lemma.* For every $B \in P$ and polynomial p, prefix-sols(B,p) is d-self-reducible.

The proof is easy: y is a prefix of a solution if and only if either it is a solution, or ya is a prefix of a solution for some symbol a. In [12] it is also shown that every d-self-reducible set is (disjunctively) Turing equivalent to a set of the form prefix-sols(B,p), and a deep study of the structural properties of sets of the form prefix-sols(B,p) is conducted.

I present next the following theorem, taken from [3]:

10. *Theorem.* Let A be any NP-complete set, and let B in P and p such that $(*)$ holds. Then funct-sols$(B,p) \cap PF(A) \neq \emptyset$, and therefore A has self-computable witnesses.

The interpretation of this result is that there is a functional solution of the NP-complete set A which is "no much harder to compute" than A itself, since this functional solution is Turing reducible to A in polynomial time.

Proof. By proposition 3 and lemma 9, since A is NP-complete, prefix-sols(B,p) is Turing reducible to A, say via machine M. Then a functional solution f for A can be computed by keeping a prefix of $f(x)$ in a local variable, and extending it a bit at a time using the machine M and the oracle A to ensure that the extension is always a prefix of a solution. ∎

This proof method will be applied again in the next section to optimization versions of the NP-complete problems.

4. Optimization NP-complete problems

For optimization problems, instead of computing one among a set of solutions, an optimal solution has to be selected. I consider only minimization problems; it is straightforward to adapt all the results to maximization problems. The decisional statement of these NP-complete problems is: given an input x, and an integer k, is there a solution with cost smaller than k?

Of course, the practical interest is not to compute a solution below a given cost, but finding the less expensive one. It is known that for some particular cases, a polynomial time algorithm for the decisional problem provides a polynomial time algorithm for the minimization problem (see, e.g., [6], pp. 185–188). I prove that for all such NP-complete sets, finding this solution is again "no much harder than" (i.e. polynomial time Turing reducible to) deciding whether a solution exists.

I define minimization NP problems in the most intuitive way, as a set of pairs

$$\{\langle x,k\rangle \mid \exists y\ (|y| \leq p(|x|))\ \langle x,y\rangle \in B \text{ with } \text{cost}(y) \leq k\}$$

where "cost" is a polynomial time computable function from Γ^* to Γ^* measuring the cost of the solution y. The range of this function is usually interpreted as an integer or a real number. To this set one can associate the set of functions computing optimal solutions, which depends of B, the polynomial p, and the cost function.

11. *Definition.* The set opt-sols(B, p, cost) is the set

$$\{f \mid f \in \text{funct-sols}(B, p), \text{ and } \forall g \in \text{funct-sols}(B, p), \forall x, \text{cost}(f(x)) \leq \text{cost}(g(x))\}$$

In order to show that some optimal solution is computable in polynomial time with the oracle set A, I define two "prefix" sets:

12. *Definition.* The sets prefix-sols(B, p, cost) and prefix-opt(B, p, cost) are defined as follows:

$$\text{prefix-sols}(B, p, \text{cost}) = \{\langle x, k, z\rangle \mid \exists y\ (|y| \leq p(|x|))\ \langle x, y\rangle \in B \text{ such that } \text{cost}(y) \leq k \text{ and } z \text{ is a prefix of } y\ \}$$

$$\text{prefix-opt}(B, p, \text{cost}) = \{\langle x, z\rangle \mid \exists y\ (|y| \leq p(|x|))\ \langle x, y\rangle \in B \text{ such that } z \text{ is a prefix of } y, \text{ and } \forall w\ (|w| \leq p(|x|))\ \langle x, w\rangle \in B, \text{cost}(y) \leq \text{cost}(w)\}$$

Again, the following result holds. Its proof is analogous to that of lemma 9:

13. *Lemma.* The set prefix-sols(B, p, cost) is d-self-reducible.

The next theorem formalizes the standard way to prove such a result for particular problems. See [6].

14. *Theorem.* The set prefix-opt(B, p, cost) is Turing reducible in polynomial time to the set prefix-sols(B, p, cost).

Proof. The reduction procedure goes as follows. First, one identifies the optimal cost k in polynomial time by searching for the maximum k such that $\langle x, k, \lambda\rangle$ is in prefix-sols(B, p, cost). This can be done in polynomial time using binary search, which requires a time logarithmic in the range of k, which in turn is exponential in $|x|$ since "cost" is computable in polynomial time. Once the optimal value of k is known, use the fact that $\langle x, z\rangle$ is in prefix-opt(B, p, cost) if and only if $\langle x, k, z\rangle$ is in prefix-sols(B, p, cost). ∎

Now I state the main result about optimization problems.

15. *Theorem.* If A is a *NP*-complete minimization problem, defined by the set B in P, polynomial p, and cost function "cost", then opt-sols$(B, p, \text{cost}) \cap PF(A) \neq \emptyset$.

Proof. A function in opt-sols giving optimal solutions can be obtained from prefix-opt by extending bitwise a prefix. By theorem 14, oracle queries to prefix-opt(B, p, cost) can be answered in polynomial time with an oracle for prefix-sols(B, p, cost), which by lemma 13 and proposition 3 is in *NP*. Since A is *NP*-complete, the reduction to A can be used for solving queries to prefix-sols(B, p, cost). By the transitivity of the polynomial time Turing reducibility, this allows one to construct an optimal solution in polynomial time with oracle A. ∎

References

[1] J.L. Balcázar: "Self-Reducibility". Submitted for publication. Report de recerca LSI-88-18, UPC.

[2] J.L. Balcázar, J. Díaz, J. Gabarró: *Structural Complexity I.* Springer-Verlag, EATCS Monographs 11 (1988).

[3] A. Borodin, A. Demers: "Some comments on functional self-reducibility and the NP hierarchy". Technical Report 76-284, Dept. Computer Science, Cornell University (1976).

[4] Ker-I Ko: "On self-reducibility and weak p-selectivity". *J. Comp. Sys. Sci.* 26, 2 (1983), 209-221.

[5] Ker-I Ko: "On helping by robust oracle machines". *Theor. Comp. Sci.* 52 (1987), 15–36.

[6] K. Mehlhorn: *Data structures and algorithms*, vol. 2 "Graph algorithms and NP-completeness". EATCS Monographs, Springer-Verlag 1984.

[7] A. Meyer, M. Paterson: "With what frequency are apparently intractable problems difficult?". M.I.T. Technical report TM-126 (1979).

[8] C.P. Schnorr: "Optimal algorithms for self-reducible problems". ICALP 1976, ed. S. Michaelson and R. Milner, Edinburgh Univ. Press, 322-337.

[9] C.P. Schnorr: "On self-transformable combinatorial problems". Symposium on Mathematische Optimierung, Oberwolfach 1979.

[10] U. Schöning: "Robust algorithms: a different approach to oracles". *Theor. Comp. Sci.* 40 (1985), 57–66.

[11] U. Schöning: *Complexity and structure.* Springer-Verlag, Lecture Notes in Computer Science 211 (1986).

[12] A. Selman: "Natural self-reducible sets". Technical Report NU-CCS-87-21, Northeastern University (1987).

[13] L. Valiant: "The relative complexity of checking and evaluating". *Inf. Proc. Letters* 5 (1976), 20-23.