Inverse NP Problems

Hubie Chen

Department of Computer Science, Cornell University,
Ithaca, NY 14853, USA
hubes@cs.cornell.edu

Abstract. One characterization of the class NP is as the class of all languages for which there exists a polynomial-time verifier with the following properties: for every member of the language, there exists a polynomially-sized proof causing the verifier to accept; and, for every non-member, there is no proof causing the verifier to accept. Relative to a particular verifier, every member x of the language induces a set of proofs, namely, the set of proofs causing the verifier to accept x.
This paper studies the complexity of deciding, given a set Π of proofs, whether or not there exists some x inducing Π (relative to a particular verifier). We call this decision problem the inverse problem for the verifier. We introduce a new notion of reduction suited for inverse problems, and use it to classify as coNP-complete the inverse problems for the "natural" verifiers of many NP-complete problems.

1 Introduction

By now, the complexity class NP has become one of the most pervasive and successful notions in computer science. Indeed, the intimately connected notion of NP-completeness is described as "computer science's favorite paradigm, fad, punching bag, buzzword, alibi, and intellectual export" in a lively discussion on the influence and nature of NP-completeness [11]. There are many equivalent characterizations of the class NP. According to one such characterization, a language is in NP if there is an efficient (polynomial-time) *verifier* with the following properties: for every member of the language, there exists a polynomially-sized proof causing the verifier to accept (upon being given the member along with the proof); and, for every non-member, there is no proof causing the verifier to accept. Thus, with a verifier for a language in hand, deciding membership in the language amounts to deciding whether or not a given potential member has a proof or not.

Inverse problems. Relative to a particular verifier, every member x of the language induces a *set of proofs*, namely, the set of proofs causing the verifier to accept x as a member. Call a set of proofs arising in this way *induced*. This paper studies a simply stated and natural question: what is the complexity of deciding, given a set Π of proofs, whether or not Π is induced? We call this decision problem the *inverse problem* for the verifier.

As a first example, consider the natural verifier for 3-SAT, which accepts a 3-SAT formula and an assignment to the variables of the formula when the assignment satisfies the formula; call the inverse problem for this verifier INVERSE 3-SAT. INVERSE 3-SAT is then the question of deciding, given a set Π of assignments, if there is a 3-SAT formula with *exactly* Π as its set of satisfying assignments. It is fairly straightforward to show that INVERSE 3-SAT is in coNP: for any set of assignments Π, we can efficiently compute the "candidate formula" F containing all 3-clauses satisfied by all assignments in Π. It can be verified that if *any* 3-SAT formula has Π as set of satisfying assignments, then the candidate formula is such a formula. Intuitively, this is because we placed as many clauses as possible in the candidate formula F, so any 3-SAT formula including Π in its induced set can only be less constrained – that is, admit more satisfying assignments – than F. Thus, INVERSE 3-SAT is in coNP: to show that a set Π is not in INVERSE 3-SAT, it suffices to give an assignment outside of Π satisfied by the candidate formula (of Π).

In previous work, Kavvadias and Sideri [7] considered the *inverse satisfiability problem*, which includes as a specific case the INVERSE 3-SAT problem. (It is from them that we adopt the term *inverse problem.*) They examined the class of generalized boolean satisfiability problems on which Schaefer's dichotomy theorem [12] was proved – a class which includes Horn SAT and 2-SAT, two well-known tractable subclasses of SAT; as well as not-all-equal SAT and one-in-three SAT, two intractable variants of SAT. For any satisfiability problem from this generalized class, the inverse problem is always in coNP by an argument similar to that given above. The intriguing result they obtained is that the inverse satisfiability problem is intractable (coNP-complete) if and only if the original satisfiability problem is intractable (NP-complete)! Thus, it follows from their result that INVERSE 3-SAT is coNP-complete because 3-SAT is NP-complete, as well as that INVERSE 2-SAT is in P because 2-SAT is in P.

To give another example of an inverse problem, we begin with the language K-CLIQUE consisting of pairs $\langle G, k\rangle$ such that G is an undirected graph with a clique of size k. Consider the verifier for this language which accepts H as a proof for $\langle G, k\rangle$ if H is a size k subset of the vertex set of G that forms a clique in G. Call the inverse problem for this verifier INVERSE K-CLIQUE. Then, a well-formed input to the INVERSE K-CLIQUE problem is a set $\Pi = \{H_1, \ldots, H_m\}$, where each H_i is a set of size k; and the problem is to decide whether or not there is a graph G such that $\{H_1, \ldots, H_m\}$ is exactly the set of k-cliques in G. In analogy to the "candidate formula" for an instance of INVERSE 3-SAT, we can compute from an instance Π of INVERSE K-CLIQUE a "candidate graph" G with the property that if *any* graph has Π as its set of k-cliques, then G is such a graph: our candidate graph has vertex set $\cup_{i=1}^{m} H_i$, with an edge between two vertices if the two vertices are both contained in H_i, for some i. Thus, INVERSE K-CLIQUE is in coNP: to show that Π is not in INVERSE K-CLIQUE, it suffices to demonstrate that the candidate graph for Π has a k-clique not in Π. It will be demonstrated in this paper that INVERSE K-CLIQUE is also coNP-complete.

These examples might seem to be evidence for the conjecture that the inverse problem for verifiers accepting NP-complete languages will always be coNP-complete. However, a fairly succinct argument demonstrates that this conjecture fails. Suppose we take the verifier for K-CLIQUE and define a new verifier which accepts only "padded" proofs of the form $\langle s, H \rangle$ where H is a clique and s is a string with, say, length equal to $|H|$. Then, the inverse problem becomes "easier": for every clique H accepted by the first verifier, exponentially many proofs of the form $\langle s, H \rangle$ are accepted by the second verifier. If padding of sufficient length is applied to the proofs of the verifier for K-CLIQUE, we obtain a verifier for K-CLIQUE with inverse complexity in P. Of course, such a padded verifier may seem to be quite artificial. For many NP-complete languages L, one may feel that there is a "natural" verifier for L (such as the verifiers described in the above examples), and we will show that the inverse problem for many such verifiers is coNP-complete. Nonetheless, we emphasize that for each language L in NP, there is more than one verifier[1] for L, and that the inverse problem is defined relative to a *verifier* for an NP language, and not just an NP language by itself.

Related work and contributions. Our study of inverse problems contributes to a line of research which aims to understand the properties of the natural verifiers for NP-complete languages. This research program has its origins in the work done on the counting class $\sharp$P by Simon and Valiant [13, 14]; one important outcome of this work was the observation that many natural verifiers are related by *parsimonious reductions*, a stronger notion of many-one reduction that preserves the number of proofs. A notion of reduction even stronger than parsimonious reduction was studied by Lynch and Lipton [10]; they required that when a language member x many-one reduces to another language y, there be an efficiently computable function mapping proofs for x to proofs for y. Fischer, Hemaspaandra, and Torenvliet [5] went one step further to study what they called *witness-isomorphic reductions*, which require that there be efficiently computable isomorphisms between the proof sets of language members reduced to one another. Other work along this line includes [1], in which Agrawal and Biswas gave sufficient conditions for a verifier to accept a NP-complete language. These sufficient conditions allow one to derive many of the known NP-completeness results in a uniform manner. Our work complements these previous results by identifying an entirely new feature shared by many of the natural verifiers for NP-complete languages – namely, coNP-complete "inverse complexity."

This paper is also a contribution to the study of *structure identification* [3], where the goal is to decide, given a set of data points, whether or not the set is meaningfully structured in a way that admits a description of desirable type. More broadly, we believe that the study of inverse complexity sheds new light on the multitude of *natural properties of classical combinatorial structures* yielding NP-complete decision problems.

We note that inverse complexity has consequences for another task involving computationally difficult problems – the task of generating multiple solutions

[1] In fact, there are infinitely many.

to a search problem, which is of both theoretical and practical interest (see, for example, [6, 9, 8, 4]). Here, a relevant problem is to decide whether or not all proofs of a language member have (at some point in time) been generated – precisely, to decide, given a language member x and a non-empty set of proofs Π of x, whether or not Π is the full set of proofs induced by x. Hardness of the inverse problem implies hardness of this decision problem, under a mild assumption.

Results and paper organization. The notation and conventions used throughout the paper are fixed in Section 2. In this section, we also define the notion of a *candidate function* for a verifier, examples of which include the "candidate formula" and "candidate graph" procedures given above.

In Section 3, we prove that INVERSE K-CLIQUE is coNP-complete. We also formulate a sufficient condition for an inverse problem to be in P, and give an example application of the condition by showing that INVERSE BIPARTITE MATCHING is in P.

We then develop a notion of reduction (Section 4) which allows us to compare the complexity of inverse problems with relative ease. In particular, we define what we call a π-*reduction* between verifiers with candidate functions, and show that the existence of a π-*function* implies the existence of a π-reduction. By giving a sequence of π-functions, we leverage the hardness of INVERSE K-CLIQUE and INVERSE 3-SAT to show the coNP-completeness of many other inverse problems.

We have hinted that the "inverse complexity" of a language depends strongly on the verifier used to accept the language. In the full version of this paper, we formally demonstrate and study this dependence. Among other results, we show that there exists a verifier with a Σ_2^p-complete inverse problem, giving a new and natural example of a Σ_2^p-complete problem. We also prove that for all natural NP-complete languages L and *any* language A in NP, there is a verifier for L with inverse complexity A.

2 Preliminaries

We assume familiarity with basic notions of complexity theory, such as the complexity classes P, NP, coNP, Σ_2^p, and Π_2^p, described for instance in [2]. We write $A \leq_m^p B$ if there is a many-one polynomial time reduction from A to B, and $A \equiv_m^p B$ if $A \leq_m^p B$ and $B \leq_m^p A$. The power set of a set S is denoted by $\mathcal{P}(S)$. Throughout this paper, Σ denotes a fixed finite alphabet. When $S \subseteq \Sigma^*$ is a set of strings, $\|S\|$ denotes $\|S\| = \sum_{x \in S} |x|$, that is, the sum of the lengths of the strings in S. (We assume that such a set $S \subseteq \Sigma^*$ is represented by a string of length linear in $\|S\|$.)

Definition 1. *A relation $R \subseteq \Sigma^* \times \Sigma^*$ is a* verifier *if membership in R is decidable in polynomial time and there is a polynomial p such that (for all $x, \pi \in \Sigma^*$) $R(x, \pi)$ implies that $|\pi| \leq p(|x|)$. When R is a verifier, we let $R(x)$ denote the set of proofs for $x \in \Sigma^*$, that is, $\{\pi \in \Sigma^* : R(x, \pi)\}$; and, we let $L(R)$*

denote the set $\{x \in \Sigma^* : \exists \pi \in \Sigma^* \text{ such that } R(x, \pi)\}$, *which we call the language associated with or accepted by* R.

We now define the inverse problem corresponding to a verifier, which is the problem of focal interest in this paper.

Definition 2. *Suppose that* R *is a verifier. Let* R^{-1} *denote the language*

$$\{\Pi \subseteq \Sigma^* : \exists x \in L(R) \text{ such that } R(x) = \Pi\}.$$

We call this language the inverse problem *for* R.

When R is a verifier, we will refer to the members of the language $L(R)$ as theorems; when x is a theorem, the elements of $R(x)$ are said to be its proofs. Using this terminology, the inverse problem is the question of deciding, given a set of proofs Π, whether or not there is a theorem x with exactly Π as its set of proofs. A natural initial consideration is whether or not there is an upper bound on the complexity of R^{-1}. We can obtain such an upper bound when we place a restriction which we call fairness on R. All verifiers considered in this paper obey this restriction.

Definition 3. *A verifier* R *is a* fair verifier *if there is a polynomial* q *such that for all* $x \in L(R)$, *there exists* $x' \in L(R)$ *where* $|x'| \leq q(\|R(x)\|)$ *and* $R(x') = R(x)$.

Put differently, R is fair if for all sets of proofs Π, when there exists a theorem x with *exactly* Π as its set of proofs (that is, $\Pi = R(x)$), then there exists such an x with length bounded above by a polynomial in $\|\Pi\|$. Suppose R is a fair verifier. In deciding whether or not a set of proofs Π is in R^{-1}, we need only consider as potential theorems the x which are bounded above in length by a polynomial in $\|\Pi\|$, the length of the representation of Π. Moreover, by the definition of a verifier, a proof of x (that is, an element of $R(x)$) has length polynomial in $|x|$. These two facts lead to the following observation.

Observation 1 *If* R *is a fair verifier, then* R^{-1} *is in* Σ_2^p.

We note that without the restriction of fairness, the complexity of R^{-1} can be non-recursive – in fact, as high as the halting problem for general Turing machines.[2]

Definition 4. *A non-empty set* $\Pi \subseteq \Sigma^*$ *is* well-formed *(relative to an NP verifier* R*) if there exists* $x \in \Sigma^*$ *such that* $\Pi \subseteq R(x)$.

[2] For any recursively enumerable language L (such as the halting problem), there exists a verifier R such that $R^{-1} = L$. To see this, let M be a Turing machine accepting L. Define the verifier $R(x, \pi)$ to be true when x is of the form $\langle 0^n, \pi\rangle$ and M accepts when simulated for time n on input π: then, $R^{-1} = L$. Since R^{-1} is always recursively enumerable, we obtain that a language is recursively enumerable if and only if it is equal to R^{-1} for some verifier R.

The inverse problem is only "interesting" on well-formed sets, since a set that is not well-formed cannot be in R^{-1}. Checking well-formedness of a set can be done in polynomial time for all verifiers considered in this paper.

We now introduce the notion of a *candidate function* for a verifier R. This notion can be thought of as a generalization of the "candidate formula" and "candidate graph" procedures described in the introduction. A candidate function efficiently maps a set of proofs Π to a "candidate theorem" having the property that if any theorem has Π as its set of proofs, then the candidate theorem is such a theorem. In addition, the candidate theorem for Π is optimistic in the sense that all proofs in Π are proofs of it.

Definition 5. *Let R be a verifier. A polynomial time computable function $C : \mathcal{P}(\Sigma^*) \to \Sigma^*$ is a* candidate function *for R if the following two conditions hold.*

1. *For all well-formed $\Pi \subseteq \Sigma^*$, all proofs in Π are proofs for $C(\Pi)$: $\Pi \subseteq R(C(\Pi))$.*
2. *For all well-formed $\Pi \subseteq \Sigma^*$, if there exists an $x \in L(R)$ such that $\Pi = R(x)$, then $\Pi = R(C(\Pi))$.*

Note that when $C(\Pi)$ has exactly Π as its set of proofs, there is no requirement that $C(\Pi)$ be the unique theorem with Π as its set of proofs. Indeed, there will generally not be such a unique theorem.

When a verifier has a candidate function, we can give an even better upper bound on the complexity of its inverse problem than that of Observation 1. (Notice that every verifier with a candidate function is fair.)

Observation 2 *If R is a verifier with a candidate function, then R^{-1} is in coNP.*

This is because we no longer have to search over all theorems x to find a match for Π, but rather can simply compute the candidate theorem in polynomial time, and then check whether or not its set of proofs is Π.

Definition 6. *Let R be a verifier. Define the* exhaustive proof problem *for R to be the language*

$$\textsc{Exhaustive}(R) = \{\langle x, \Pi \rangle \in \Sigma^* \times \mathcal{P}(\Sigma^*) : R(x) = \Pi, \Pi \neq \emptyset\}$$

In other words, the exhaustive proof problem is to determine, given an x and a *nonempty* set of proofs Π for x, whether or not Π contains all proofs of x (relative to R). This is the case when π is a proof of x if and only if π is in Π (for all π). To verify this one needs only examine π of length polynomial in x (by definition of a verifier), leading to the following observation.

Observation 3 *If R is a verifier, then* $\textsc{Exhaustive}(R)$ *is in coNP.*

Notice that without the restriction that the given set of proofs must be nonempty, the exhaustive proof problem for the verifier R of any NP-complete language L would trivially be coNP-complete. In this case, the reduction mapping a string x to the pair $\langle x, \emptyset \rangle$ would reduce co-L to the exhaustive proof problem for R. Although this trivial reduction fails for the given definition of the exhaustive proof problem, we can nonetheless establish hardness of this problem by using the hardness of the corresponding inverse problem.

Lemma 1. *If R is a verifier with candidate function, then $R^{-1} \leq_m^p$ EXHAUSTIVE(R).*

3 Inverse Problems

In this section, we give some initial complexity classifications of inverse problems. The hardness results given here will serve as starting points for deriving further hardness results. First, we note a theorem established in previous work by Kavvadias and Sideri.

Theorem 4. *[7]* INVERSE 3-SAT *is coNP-complete.*

Next, we establish that the inverse clique problem is coNP-hard. This is done by reducing from the clique problem itself. The idea is to take a graph G (an instance of the clique problem) and "expose" its edges by creating a clique for each edge of G. The resulting candidate graph corresponding to the set of cliques Π has, for each clique in the original graph G, a clique not in Π.

Theorem 5. INVERSE K-CLIQUE *is coNP-complete.*

It generally seems to be the case that the inverse problem for a natural verifier accepting an NP-complete language is coNP-complete: in the next section, we give many examples where this is true. However, there is at least one exception to this apparent rule of thumb.

Observation 6 INVERSE CIRCUIT-SAT *is in P.*

This is because *every* well-formed set Π has a circuit with exactly Π as the satisfying assignments, namely, a DNF circuit directly encoding Π. One might contrast this observation with Theorem 4; intuitively, the difference is that the class of boolean circuits is capable of expressing all possible sets of satisfying assignments, whereas the sets of satisfying assignments expressible by 3-SAT formulas is restricted.

We can place the complexity of other inverse problems inside P by making use of a link between inverse problems and the notion of *output polynomial time*. (This notion has been discussed in previous work, for example [6].)

Definition 7. *Say that a verifier R has an* output polynomial time algorithm *if given x, the set $R(x)$ can be computed in time polynomial in $|x| + \|R(x)\|$.*

When R has an output polynomial time algorithm, both the exhaustive proof problem and inverse problem for R become easy. The following lemma is implicit in [7].

Lemma 2. *Suppose that R is a verifier with an output polynomial time algorithm. Then,* EXHAUSTIVE(R) *is in P. If in addition R has a candidate function, then R^{-1} is in P.*

This lemma allows us to establish the tractability of INVERSE BIPARTITE MATCHING, as there is an output polynomial time algorithm for BIPARTITE MATCHING.[3]

Theorem 7. INVERSE BIPARTITE MATCHING *is in P.*

4 π-reductions

We now develop a notion of reduction between pairs of verifiers with candidate functions called π-reduction, which allows us to compare the complexity of the respective inverse problems. When there is a π-reduction from one verifier to another, the inverse problem of the first reduces in polynomial time to the inverse problem of the second. In addition, we show that the existence of a particular type of function, which we call a π-function, is sufficient for a π-reduction to exist; giving such π-functions will be our main tool for showing hardness of inverse problems. Throughout this section, we assume that R and S are verifiers with candidate functions C_R and C_S, respectively.

Definition 8. *A polynomial time computable function $g : \mathcal{P}(\Sigma^*) \to \mathcal{P}(\Sigma^*)$ is a* π-reduction *from R to S if the following three conditions hold.*

1. *For all well-formed $\Pi \subseteq \Sigma^*$, g preserves the number of proofs: $|\Pi| = |g(\Pi)|$.*
2. *For all well-formed $\Pi \subseteq \Sigma^*$, the candidate theorems for Π and $g(\Pi)$ have the same number of proofs: $|R(C_R(\Pi))| = |S(C_S(g(\Pi)))|$.*
3. *If $\Pi \subseteq \Sigma^*$ is not well-formed, then $g(\Pi)$ is not well-formed.*

Lemma 3. *If g is a π-reduction from R to S, then $R^{-1} \leq_m^p S^{-1}$ via g.*

Definition 9. *Suppose $f : \Sigma^* \times \Sigma^* \to \Sigma^*$ is a partial function computable in polynomial time, and let $f_x : \Sigma^* \to \Sigma^*$ denote the function defined by $f_x(\pi) = f(x, \pi)$. Let us say that f is a* π-function *from R to S (relative to C_R and C_S) if for all well-formed $\Pi \subseteq \Sigma^*$, when x and y are set as $x = C_R(\Pi)$ and $y = C_S(f_x(\Pi))$, f_x is a bijection between $R(x)$ and $S(y)$.*

[3] Although it would be tempting to conjecture that there is a correspondence between the complexity of the counting problem corresponding to a verifier R (given x, compute $|R(x)|$) and the inverse problem R^{-1}, a general correspondence seems unlikely in light of Theorem 4, Theorem 5, Observation 6 and Theorem 7. The counting versions of 3-SAT, K-CLIQUE, CIRCUIT-SAT, and BIPARTITE MATCHING are all ♯P-complete [14]. Moreover, the decision problems for 3-SAT and CIRCUIT-SAT are both NP-complete, so even attempting to predict the inverse complexity based on *both* the counting complexity and decision complexity seems difficult.

Lemma 4. *Suppose that there exists a π-function from R to S (relative to C_R and C_S). Then there exists a π-reduction from R to S, and $R^{-1} \leq_m^p S^{-1}$.*

Having developed a notion of reduction between inverse problems, we can classify the complexity of many natural inverse problems with relative ease.

Theorem 8. INVERSE EXACT COVER*,* INVERSE VERTEX COVER*,* INVERSE KNAPSACK*,* INVERSE STEINER TREE*, and* INVERSE PARTITION *are all coNP-complete.*

In the full version of this paper, we give precise descriptions of the verifiers that are addressed by Theorem 8. All of these verifiers have candidate functions, and thus the corresponding inverse problems are all in coNP. We establish that the inverse problems are all coNP-hard by starting with the fact that INVERSE 3-SAT and INVERSE K-CLIQUE are coNP-hard (Theorems 4 and 5), and then giving a sequence of π-functions. We note that parsimonious reductions h between R and S such that there is an efficient mapping from proofs of $x \in L(R)$ to proofs of $h(x)$ are often (but not always) of great help in developing π-functions.

Corollary 1. EXHAUSTIVE 3-SAT*,* EXHAUSTIVE CIRCUIT-SAT*,* EXHAUSTIVE K-CLIQUE*,* EXHAUSTIVE EXACT COVER*,* EXHAUSTIVE VERTEX COVER*,* EXHAUSTIVE KNAPSACK*,* EXHAUSTIVE STEINER TREE*, and* EXHAUSTIVE PARTITION *are all coNP-complete.*[4]

5 Conclusions

In this paper, we formalized the inverse problem for an arbitrary NP verifier, which includes as particular cases the inverse satisfiability problems studied by Kavvadias and Sideri [7]. We developed a notion of reduction for inverse problems and used it to classify the inverse complexity of many natural verifiers. We also formally demonstrated that the "inverse complexity" of a verifier for language L cannot be predicted from L, but rather, exhibits a strong dependence on the choice of verifier used to accept L.

There are some interesting directions for future work. Can the intuitive claim that natural verifiers for NP-complete problems tend to have coNP-complete inverse problems, be formalized and proved? (Obviously, a formalized notion of "natural verifier" having the property that all natural verifiers have coNP-complete inverse problems would have to exclude the verifier for CIRCUIT-SAT, by Observation 6.) Along these lines, it would be of interest to classify the inverse complexity of the natural verifiers for GRAPH 3-COLORABILITY and HAMILTONIAN CIRCUIT. In addition, one could study inverse problems for other complexity classes definable by nondeterministic machines, such as NL and Σ_2^p.

[4] For some (but not all) of the listed problems, we are aware of elementary proofs of coNP-completeness where the reduction is from the complement of the original language. For instance, CO-K-CLIQUE reduces to EXHAUSTIVE K-CLIQUE by mapping $\langle G, k\rangle$ to $\langle\langle G \cup C, k\rangle, V(C)\rangle$, where C is a clique of size k with vertex set disjoint from that of G.

Acknowledgements. The author wishes to thank Andrew Blumberg, Carla Gomes, Dexter Kozen, Riccardo Pucella, Tim Roughgarden, and Bart Selman for useful discussions and comments. The author was supported by a NSF Graduate Research Fellowship.

References

1. M. Agrawal and S. Biswas. Universal relations. In *Proc. 7th Structure in Complexity Theory Conference*, pages 207–220, 1992.
2. J. L. Balcázar, J. Díaz, and J. Gabarró. *Structural Complexity I.* Texts in Theoretical Computer Science – An EATCS series. Springer-Verlag, Berlin, 2nd edition, 1995.
3. R. Dechter and J. Pearl. Structure identification in relational data. *Artificial Intelligence*, 58:237–270, 1992.
4. Thomas Eiter and Kazuhisa Makino. On computing all abductive explanations. In *AAAI/IAAI*, pages 62–67, 2002.
5. S. Fischer, L. Hemaspaandra, and L. Torenvliet. Witness-isomorphic reductions and local search. *Complexity, Logic and Recursion Theory, Lecture Notes in Pure and Applied Mathematics, A. Sorbi (ed.)*, pages 207–223, 1997.
6. D. S. Johnson, C. H. Papadimitriou, and M. Yannakakis. On generating all maximal independent sets. *Information Processing Letters*, 27:119–123, 1988.
7. D. Kavvadias and M. Sideri. The inverse satisfiability problem. *SIAM Journal on Computing*, 28(1):152–163, 1998.
8. Dimitris J. Kavvadias, Martha Sideri, and Elias C. Stavropoulos. Generating all maximal models of a Boolean expression. *Information Processing Letters*, 74(3–4):157–162, 2000.
9. A. Kwan, E. P. K. Tsang, and J. E. Borrett. Phase transition in finding multiple solutions in constraint satisfaction problems. In *Workshop on Studying and Solving Really Hard Problems, First International Conference on Principles and Practice of Constraint Programming*, pages 119–126, 1995.
10. N. Lynch and R. Lipton. On structure preserving reductions. *SIAM J. Comp.*, 7(2):119–125, 1978.
11. C. Papadimitriou. NP-completeness: A retrospective. In *Proceedings of the 24th International Colloquium on Automata, Languages and Programming*, volume 1256 of *Lecture Notes in Computer Science*, pages 2–6. Springer, 1997.
12. T. J. Schaefer. The complexity of satisfiability problems. In *Proc. 10th Annual ACM Symposium on Theory of Computing*, pages 216–226, 1978.
13. J. Simon. On some central problems in computational complexity. Technical report, Cornell University, 1975.
14. L. G. Valiant. The complexity of computing the permanent. *Theoretical Computer Science*, 8:189–201, 1979.