1

The Complexity of Obtaining Solutions for Problems in NP and NL

Birgit Jenner[1]
Jacobo Torán[2]

ABSTRACT We review some of the known results about the complexity of computing solutions or proofs of membership for problems in NP. Trying to capture the complexity of this problem, we consider the classes of functions $\mathrm{FP}^{\mathrm{NP}}$, $\mathrm{FP}^{\mathrm{NP}}[f]$ (for certain bounded functions f), NPSV, and $\mathrm{FP}_{tt}^{\mathrm{NP}}$ and provide some examples of NP problems with search functions in these classes. We also consider whether NP-complete problems can have such proofs of membership. We use the problem of obtaining solutions to compare the relative powers of the function classes above . Finally, we consider the situation in the nondeterministic logarithmic space setting, showing how the complexity of obtaining solutions for NL sets compares with the NP case.

1 Introduction

Problems in the class NP have traditionally been studied from a decisional point of view. This has been so mainly because in all natural cases an algorithm providing a yes/no answer to an NP problem can be used to obtain a solution for the problem, and therefore, if there exists a polynomial-time algorithm for the decision problem, a solution can also be found in polynomial time. It is more natural, however, to consider the *search version*, studying directly the complexity of obtaining solutions for the problem. Formally, let A be a problem in NP. By the standard characterization of

[1] Wilhelm-Schickard-Institut für Informatik, Universität Tübingen, Sand 13, D–72076 Tübingen, Germany (jenner@informatik.uni-tuebingen.de). Partially supported by a Habilitationsstipendium of the Deutsche Forschungsgemeinschaft (Je 154 2/2).

[2] Abt. Theoretische Informatik, Universität Ulm, Oberer Eselsberg, D–89069 Ulm, Germany (toran@informatik.uni-ulm.de). Partially supported by DAAD Accion Integrada 322–AI–e–dr.

NP with a polynomial-bounded existential quantifier, there is a polynomial p and a polynomial-time computable relation R such that for all strings x in Σ^*

$$x \in A \Longleftrightarrow \exists y \ |y| \leq p(|x|) \text{ and } R(x,y).$$

For every instance x, the set of strings y satisfying the relation $R(x,y)$ is usually called the set of solutions or the set of proofs of membership for x in A. The *search problem* for A consists of finding a solution for an input instance x. In this chapter we survey some results about the complexity of functions computing such a solution for problems in NP. For an NP problem A characterized as above by a relation R, $\mathcal{F}_{A_R}$ denotes the set of search functions for A, defined as

$$f \in \mathcal{F}_{A_R} \Longleftrightarrow f(x) = \begin{cases} \text{some } y, & |y| \leq p(|x|) \text{ and } R(x,y), \\ & \text{if such } y \text{ exists}, \\ \text{undefined}, & \text{otherwise.} \end{cases}$$

Observe that in general a search function can be partial. This should not create a problem since we are interested in finding solutions when they exist.[3]

As we will see, the complexity of computing a search function might vary depending on what kind of solution we want and also on the complexity of the decision problem. For example, it might be harder to find a specific solution or an optimal solution in a certain sense than to find any solution at all. Also, finding solutions for NP-complete problems might be harder than finding solutions for problems in NP that do not seem to be complete. Although the question of finding solutions for NP problems is a fundamental one, up to now the complexity of the search problem has not been classified in a completely satisfactory way. As we will see in Section 2, the complexity of finding optimal solutions is well understood. However, there are many open questions related to the complexity of finding any solution at all. In the different sections of this survey, we describe some of the complexity classes of functions that arise when trying to capture the complexity of the search problem, comparing them and showing some of the results and properties known for these classes. For any of the classes we consider, we give examples of natural problems in NP that have search functions in the class.

We start by considering in Section 2 the class $\mathrm{FP}^{\mathrm{NP}}$ of functions computable in polynomial time with access to an oracle in NP. This class provides an upper bound for the complexity of the search problem. In fact, as we will see, this class captures exactly the complexity of obtaining optimal solutions for many important NP-complete problems. In other words, many NP optimization problems are complete for this class of functions. An NP

[3] Also, for simplicity, where there is no confusion, we will not include the relation R in the notation and just write $\mathcal{F}_A$.

optimization problem is defined by an NP set A together with a polynomial-time computable cost function c that associates with every instance x and every solution y of x a cost $c(x, y)$. For an input x, the optimization problem consists of finding the solution y maximizing (minimizing) $c(x, y)$ over all possible solutions. If we consider the concept of metric reducibility as a tool to compare the complexity of functions, the optimization functions for many NP-complete problems are the hardest ones in the class $\mathrm{FP}^{\mathrm{NP}}$. A natural question to ask is whether there are other search functions (not necessarily giving the optimal solution) that are easier to compute. We present a result showing that under the hypothesis NP $\neq$ co-NP, there are search functions for NP-complete problems that are not complete for $\mathrm{FP}^{\mathrm{NP}}$. On the other hand, we also show that obtaining certain concrete solutions related to counting problems is probably harder than obtaining the optimal one.

In Section 3, we consider function classes that arise when bounding the number of queries that can be made to an NP oracle. We give examples of some problems in NP with search functions in such bounded query classes, showing that the existence of search functions in these classes is related to the amount of nondeterminism needed to solve a problem in NP. In the same way as $\mathrm{FP}^{\mathrm{NP}}$ can be associated in a natural way with the complexity of obtaining optimal solutions for NP-complete problems, there is a weak variant of NP optimization that provides many examples of functions in the bounded query classes. Any NP optimization problem induces the *weak optimization* problem of finding the cost of the optimal solution for an instance (without necessarily obtaining the solution). Many NP-complete optimization problems with a polynomially bounded cost function (a cost function whose value is bounded by a polynomial in the input size) have weak optimization versions in the class $\mathrm{FP}^{\mathrm{NP}}[O(\log n)]$. We also review some recent results relating bounded queries to the approximation of weak optimization problems.

The class NPSV of NP single-valued functions is considered in Section 4. The functions in this class are defined in terms of nondeterministic transducers, a model that uses nondeterminism to compute functions in a different way than the oracle query approach. Although there are some examples of search functions in NPSV, the main result explained in this section shows that NPSV functions probably cannot compute proofs of membership for NP-complete problems, since this would imply the collapse of the polynomial-time hierarchy to its second level.

Section 5 is devoted to the class $\mathrm{FP}_{tt}^{\mathrm{NP}}$, the nonadaptive version of $\mathrm{FP}^{\mathrm{NP}}$. This class is a good candidate to improve the best known upper bound for the complexity of the search problem. We present a result showing that, relative to a random oracle, solutions for NP-complete problems can be computed in this class. On the other hand, we mention a relativized result in the other direction. We also review some results about the relative power of $\mathrm{FP}_{tt}^{\mathrm{NP}}$ and the other function classes considered in this survey.

The search problem is not particular to the class NP, and it also makes sense to define the problem for other complexity classes defined in terms of nondeterminism. In Section 6, we take a look at the search problem for nondeterministic logarithmic space (NL), comparing the situation in this setting to the results known for NP. We consider the two function classes OptL and $\mathrm{FL}^{\mathrm{NL}}$, pointing out that not all of the results for the NP case translate directly to the logarithmic space setting.

For the basic notions of complexity theory used in this survey, as well as for the definitions of some of the complexity classes we mention, we refer the reader to the introductory books in the area such as [BDG88]. [Sel94] constitutes a good introduction for definitions and relationships between function classes in the polynomial-time setting.

The functions that we consider throughout this survey are defined from Σ^* to Σ^*, that is, they operate on strings rather than on natural numbers. When necessary, we assume a standard polynomial-time computable bijection from Σ^* to the set of integer numbers.

In order to compare the complexity of different functions, we use throughout the survey the notion of *metric reducibility* [Kre88]. Given two functions f and g, we say that f is metric reducible to g if there are two functions h_1 and h_2 in FP such that for every x, $f(x) = h_1(x, g(h_2(x)))$. Intuitively, this means that f can be computed by applying function g to a polynomial-time transformation of x and then performing a polynomial-time computation. We represent a particular metric reduction by its two functions (h_1, h_2). Since the functions we are dealing with might be undefined for some inputs, we require that every reduction (h_1, h_2) should be monotonic, that is, $f(x)$ is undefined if $g(h_2(x))$ is undefined. This applies in general to all the functions considered; if a function's argument is undefined, then the function is undefined. For questions related to the definitions and relationships between function classes in the polynomial-time setting that are not treated in this chapter, we refer the reader to the survey [Sel94].

For simplicity, many of the results about the complexity of obtaining solutions for NP-complete problems are presented only for Satisfiability (SAT), the set of satisfiable Boolean formulas.

2 Computing Optimal Solutions: The Class $\mathrm{FP}^{\mathrm{NP}}$

$\mathrm{FP}^{\mathrm{NP}}$ is defined as the class of functions computable in polynomial time by a deterministic Turing machine that can query an oracle in NP. This class provides an upper bound for the complexity of the search problem since with the help of an NP oracle one can compute solutions for NP problems in the following easy way: Let A be an NP problem, and let p and R be the polynomial and the polynomial-time relation defining the problem in the existential characterization of NP. Given an instance x in A, a solution

for x can be obtained querying the set of the prefixes of the solutions

$$\text{Pref}_A = \{\langle x, u\rangle \mid x \in A \text{ and } \exists v, |uv| \leq p(|x|) \text{ and } R(x, uv)\}.$$

Clearly, Pref_A belongs to NP, and querying this set at most $p(|x|)$ times, a solution for x can be constructed by the standard prefix search method. Depending on how the prefix search is done, one can construct the lexicographically smallest or largest solution, and therefore this method is indicated to solve NP optimization problems. For simplicity, in this survey we consider only maximization problems, the minimization case is completely analogous. To obtain the best solution for an NP maximization problem Π, one can first obtain the cost of the best solution for an instance x by querying the set

$$\text{Max-Cost}_A = \{\langle x, k\rangle \mid \exists y,\ R(x, y) \text{ and } c(x, y) \geq k\}$$

at most a polynomial number of times using binary search, and then find the solution with the obtained cost by a prefix search using the set

$$\text{Pref}'_A = \{\langle x, k, u\rangle \mid x \in A \text{ and } \exists v,\ R(x, uv) \text{ and } c(x, uv) = k\}.$$

In order to capture the complexity of optimization problems, Krentel introduced in [Kre88] the complexity class OptP containing the functions computable by taking the maximum (or minimum) over a set of feasible solutions. These functions can be best understood using the *metric Turing machine* model, also introduced in [Kre88]. A metric Turing machine M is a nondeterministic polynomial-time Turing machine that at the end of a computation path might write an integer number before halting. Define $\text{opt}_M(x)$ to be the largest output written by M on any computation path on input x. By convention, if no output is written at the end of any computation path on input x, we consider that $\text{opt}_M(x)$ is undefined. OptP is the class of functions f that for some metric Turing machine M and for every instance x satisfy $f(x) = \text{opt}_M(x)$.

For example, consider the function Max-Assign, computing the maximum (that is, lexicographically largest) satisfying assignment of a Boolean formula. This function is computed by a metric Turing machine M that on input F guesses an assignment y for F, and if it satisfies the formula it outputs y (considered now as an integer). Clearly, Max-Assign $= opt_M$.

As a second example, consider the optimization version of the Clique problem, consisting of obtaining a clique of largest size in a graph (in case there is more than one such clique, the one with the largest encoding should be selected). We can construct a metric machine for this problem that on every nondeterministic computation path guesses a set of nodes, and if these nodes form a clique, then the machine outputs its size. This machine computes the cost of the best solution rather than the solution itself. However, the metric machine can be easily modified to encode in its output the

clique together with its size in a way that cliques of greater size produce a larger output.

From the considerations made before, the largest output of a metric Turing machine can be computed with adaptive queries to NP, and therefore OptP is a subclass of $\mathrm{FP}^{\mathrm{NP}}$. However, as shown by Krentel, [Kre88] the relation between both classes is stronger.

Theorem 2.1 $\mathrm{FP}^{\mathrm{NP}}$ *coincides with the metric closure of* OptP.

Proof. As we have mentioned, every function in OptP is contained in $\mathrm{FP}^{\mathrm{NP}}$. This shows one of the containments, since $\mathrm{FP}^{\mathrm{NP}}$ is closed under metric reducibility. The other containment is more interesting. Let f be a function computed in $\mathrm{FP}^{\mathrm{NP}}$ by a deterministic machine M that queries an NP set A. Let p be a polynomial and R a polynomial-time relation defining the set A. We can suppose that there is a polynomial q such that on input x the machine queries A exactly $q(|x|)$ times. We give the description of a metric machine M' that can output different values, and the largest of them coincides with the sequence of query answers of M on a given instance. On input x, M' conjectures a sequence of oracle answers $a_1, \ldots, a_{q(|x|)}$, (the a_i's take either value 0 or 1, and we identify 0 with a negative answer and 1 with a positive one). The machine simulates the (deterministic) computation of M on x following the sequence of guessed answers, and for every query q_i answered positively, it guesses a string y_i of size bounded by $p(|q_i|)$. If for every query q_i with a positive answer a_i, $R(q_i, y_i)$ holds, then M' outputs $a_1 a_2 \ldots a_{q(|x|)}$ (considered as an integer), otherwise M' halts without producing any output. Observe that although M' might output wrong sequences of oracle answers, the positive answers in all sequences are always correct. If a wrong sequence s is produced, there is always a larger sequence also given as output, namely one that provides a correct positive answer to the first wrong negative answer of s. From this, it follows that the function computing the correct sequence of oracle answers of M coincides with the largest output of this metric machine. The final observation needed to prove the theorem is that the function f is metric reducible to the correct sequence of oracle answers of M. □

Krentel [Kre88] has shown that optimization versions of important NP-complete problems such as Knapsack and 0–1 Integer Programming are metric complete for OptP, and therefore also for $\mathrm{FP}^{\mathrm{NP}}$. Gasarch, Krentel, and Rappoport [GKR95] have extended the list of NP-complete problems with optimization versions that are complete for OptP, indicating that OptP completeness is the standard behavior for the optimization version of NP-complete problems.

These results show that the functions computing the best solution for NP-complete problems are the hardest ones in $\mathrm{FP}^{\mathrm{NP}}$, and every other function in the class can be metric reduced to them. What is the situation for obtaining solutions other than the optimal? It is not hard to see that in

the way the class of search functions for an NP problem is defined, one can construct search functions that are not even recursive. A more interesting question is whether there are solutions that are easier to compute than the optimal one, or whether all of them are metric hard for $\mathrm{FP}^{\mathrm{NP}}$. Next, we show a result by Watanabe and Toda [WT93] that under the hypothesis NP $\neq$ co-NP, the optimal search functions for NP-complete problems cannot be reduced to all search function for the problem. Therefore, there might be solutions that are easier to compute. The original result is proven for *functional reducibility*, a generalization of metric reducibility that allows polynomially many (nonadaptive) functional queries. The following lemma plays an important role in the result.

Lemma 2.2 *Let f be any function and A a set in* NP. *If f is metric reducible to every search function for A, then there is a uniform metric reduction (h_1, h_2) such that f is metric reducible to every search function for A via (h_1, h_2).*

The statement that a function is reducible to all the functions in the class using always the same reduction has strong consequences.

Theorem 2.3 *If* Max-Assign *can be metric reduced to every search function for* SAT, *then* NP = coNP.

Proof. Given a Boolean formula F on variables $x_1, \ldots x_n$, let us denote by F' the formula $\neg x_0 \vee F(x_1, \ldots, x_n)$. Then, the formula F' is satisfiable; clearly, F is satisfiable if and only if the maximum satisfying assignment of F' gives to x_0 the value 1. Under the hypothesis that Max-Assign can be reduced to every search function for SAT, using the previous lemma there is a uniform metric reduction (h_1, h_2) such that for every search function for SAT g and every Boolean formula F,

$$\text{Max-Assign}(F) = h_1(F, g(h_2(F))).$$

A formula F is unsatisfiable if and only if Max-Assign(F') assigns the value 0 to x_0, and this is true if and only if for every search function for SAT g, $h_1(F', g(h_2(F')))$ assigns x_0 value 0. Observe that since (h_1, h_2) is a metric reduction (and hence is monotonic), and Max-Assign(F') is defined, $g(h_2(F'))$ is an assignment for the (satisfiable) formula $h_2(F')$. Also, since the value of $h_1(F', g(h_2(F')))$ is always the same independent of the assignment for $h_2(F')$ produced by g, we have that F is unsatisfiable if and only if there exists an assignment a for $h_2(F')$ such that $h_1(F', a)$ assigns x_0 value 0. But this in an NP predicate to decide unsatisfiability, and from this follows NP = co-NP. □

The proof of Lemma 2.2 uses the fact that the class of search functions for SAT is very broad, and therefore a reduction to every search function is a very strong hypothesis. However, it is not hard to see that for the original functional reducibility used in [WT93] the converse result also holds, that

is, if NP = co-NP, then the optimal search function for SAT could be functionally reduced to any other search function for the problem.

We finish this section by mentioning a result by Toda [Tod90] that characterizes the complexity of finding a specific solution different from the optimal one. He considers the solution that coincides with the median of the output values of a metric Turing machine and shows that the complexity of obtaining such a solution for SAT and other NP-complete problems is metric complete for the class $\mathrm{FP}^{\mathrm{PP}}$. Together with another result from Toda, the theorem stating that the polynomial-time hierarchy (PH) is Turing reducible to PP [Tod91], this implies that unless PH collapses, this specific search function cannot be computed in $\mathrm{FP}^{\mathrm{NP}}$. Related results for other specific search functions connected to counting have been obtained by Vollmer and Wagner in [VW93].

3 Bounded Queries to NP

In this section, we consider the functions that arise by bounding the access to the NP oracle in the class $\mathrm{FP}^{\mathrm{NP}}$. Let us say that a function f from $\mathbf{N}$ to $\mathbf{N}$ is *smooth* if f is nondecreasing and polynomial-time computable (with respect to the value of n). For a smooth function f, $\mathrm{FP}^{\mathrm{NP}}[f]$ denotes the class of functions that can be computed by a deterministic polynomial-time machine that on inputs of length n makes at most $f(n)$ queries to an oracle in NP. For a class $\mathcal{F}$ of functions, $\mathrm{FP}^{\mathrm{NP}}[\mathcal{F}]$ is defined in the natural way.

We have seen in the previous section that solutions to NP problems can be computed in $\mathrm{FP}^{\mathrm{NP}}$ with at most a polynomial number of queries to the oracle. Can these functions also be obtained with fewer oracle queries? Since in the procedures described, basically one oracle query is needed for each bit of the solution, the question is closely related to that of what is the shortest length for the string quantified existentially in the characterization of an NP problem or, in other words, to the question of what is the smallest amount of nondeterminism needed to solve a problem in NP. All known examples of NP-complete problems need a polynomial, or at least linear, number of nondeterministic bits for their solution. However, there are interesting examples of problems in NP that are not known to be in P but have shorter proofs of membership. For a smooth function f, define $\mathrm{NP}(f)$ to be the class of problems A in NP that for some polynomial-time computable relation R can be expressed as

$$A = \{x \mid \exists y, |y| \leq f(|x|) \text{ and } R(x, y)\}.$$

For polylogarithmic functions, these classes of bounded nondeterminism were defined by Kintala and Fisher [KF84]. It is easy to construct versions of NP problems that fall in these classes; for example, the problem of whether a graph has a clique of logarithmic size in its number of nodes is contained

in $\mathrm{NP}(\log^2 n)$. Here the solution encodes the $\log n$ nodes in the clique, and $\log n$ bits are needed to encode each node. The classes $\mathrm{NP}(f)$ restricted to polylogarithmic functions f also contain interesting natural problems, including Dominating Set for tournament graphs, VC-Dimension, and Quasigroup Isomorphism, that are not known to be solvable in polynomial time (see [MV88, DT90, PY93, Far94]).

It is straightforward to verify that a problem in NP has a proof of membership computable in $\mathrm{FP}^{\mathrm{NP}}[f]$ if and only if the problem belongs to $\mathrm{NP}(f)$. Since $\mathrm{NP}(f)$ is contained in $\mathrm{DTIME}(2^f)$ (that is, in P for $f \in O(\log n)$), probably one cannot obtain solutions for NP-complete problems in $\mathrm{FP}^{\mathrm{NP}}[f]$ for functions f much smaller than linear. This reasoning can be used to show that under the standard complexity hypothesis certain containment results among the bounded query classes of functions are not possible, as the following result from Krentel [Kre88] shows.

Theorem 3.1 *If* $\mathrm{FP}^{\mathrm{NP}}[n^{O(1)}] \subseteq \mathrm{FP}^{\mathrm{NP}}[O(\log n)]$ *then* $\mathrm{P} = \mathrm{NP}$.

The hypothesis states that membership tests for SAT can be obtained in the class $\mathrm{FP}^{\mathrm{NP}}[O(\log n)]$, but this implies $\mathrm{SAT} \in \mathrm{NP}(O(\log n))$, and this class coincides with P. In the same paper, Krentel proved a more general separation result, showing that in certain cases all the queries are necessary.

Theorem 3.2 *Let* f *be smooth and* $f(n) \leq \epsilon \log n$, *for some* $\epsilon < 1$. *If*

$$\mathrm{FP}^{\mathrm{NP}}[f(n)] \subseteq \mathrm{FP}^{\mathrm{NP}}[f(n) - 1],$$

then $\mathrm{P} = \mathrm{NP}$.[4]

Krentel asked whether similar results could be obtained for other functions greater than $\epsilon \log n$. As we will see later in Section 5, it follows from Theorem 5.5 that for functions f in $O(\log n)$ such a containment would imply the collapse of the polynomial-time hierarchy to its third level. Results implying a collapse of PH under the hypothesis of the equality of function classes in which more queries are allowed are still open.

Another motivation for studying the bounded query classes of functions is related to a weak form of optimization. Recall from Section 1 that in an optimization problem for NP there are two parameters of interest: the optimal cost and a solution achieving such a cost. We have just observed that in order to compute an optimal solution for each one of its bits, a query to NP seems to be needed. On the other hand, if we are just interested in the optimal cost achieved by any solution, how hard is this weak form of optimization? If the cost function is exponential in the input size as happens in many problems, computing a cost can be basically as hard as computing a solution. For example, computing the cost of the best tour for the Traveling Salesperson Problem is complete in $\mathrm{FP}^{\mathrm{NP}}$, and also there

[4] This result was originally proved for $\epsilon < \frac{1}{2}$. As observed by Beigel in [Bei88], the result is true for any $\epsilon < 1$.

are cases like the function Max-Assign considered in Section 2 in which the solution is encoded in the cost function. There are, however, many other optimization problems, like Max-Clique, also considered in Section 2, for which the cost function is subexponential, and therefore it is possible to compute it with less than a polynomial number of queries. In fact, Krentel [Kre88] has shown that some problems like computing the Maximum Clique Size, Chromatic Number, and Max-Sat (computing the maximum number of simultaneously satisfiable clauses in a Boolean formula in conjunctive normal form) are metric complete for the class $\mathrm{FP}^{\mathrm{NP}}[O(\log n)]$. From an approximation algorithm from Karmarkar and Karp for Bin Packing Size [KK82], it follows that this problem can be computed in $\mathrm{FP}^{\mathrm{NP}}[O(\log \log n)]$.

The number of NP queries provides a quantitative framework for comparing the complexities of these examples. Using Theorems 3.1 and 3.2, it can be concluded that the weak optimization version for the Traveling Salesperson Problem is strictly harder than Maximum Clique Size, and this problem is strictly harder than Bin Packing Size.

Recently, Chang, Gasarch and Lund [CG93, CGL94, Cha94] have pointed out a very interesting connection between bounded queries and approximation of weak optimization problems showing that there is a trade-off between the number of NP queries and the closeness of the approximation for such problems. In particular, it is shown in [Cha94] that the problem of approximating Maximum Clique Size by a constant factor is metric complete for the class $\mathrm{FP}^{\mathrm{NP}}[\log \log n + O(1)]$. Again in this setting, the quantitative nature of the bounded queries to NP can provide a framework for comparing the complexity of different approximation problems.

4 Computing Solutions Uniquely: The Class NPSV

In order to compute solutions for problems in NP, it seems that nondeterminism as a resource has to be used in one way or another. This can be done with queries to an NP oracle as explained in the previous sections, but there is a different way: using nondeterministic transducers to compute functions directly. A problem with this approach is that a nondeterministic transducer, a machine that on every path might output some value, can produce many different outputs for the same input instance and therefore does not compute a function in the usual sense, but a multivalued function ([BLS84]; see also [FHOS93, Sel94]). In order to compute functions, we should make the additional restriction that for any input given to the nondeterministic transducer, all the computation paths that produce some output value produce the same one. A computation path can also stop without producing any output. This idea defines the class NPSV, introduced by Book, Long, and Selman in [BLS84], of functions computable by polynomial-time nondeterministic transducers that for any input produce

at most one output value. If no output is produced, we consider that the function is undefined for that input.

Are there NPSV functions computing solutions for problems in NP? There are not many examples of problems that are not known to be in P but have solutions computable in NPSV. An important one is Primes, the problem of deciding whether a given number is prime. This set has proofs of membership in NPSV since it belongs to UP [FK92], the class of NP problems that can be accepted by nondeterministic machines with at most one accepting path. Clearly, every set in UP has proofs of membership in NPSV. Its complement, Composites (also known to be in UP [FK92]), has a very natural proof of membership: given an integer, its list of prime factors can be computed in NPSV.

Can NPSV functions also compute solutions for harder problems in NP? Hemaspaandra, Naik, Ogihara, and Selman [HNOS94] have shown that for NP-complete problems this is probably not the case, since it would imply the collapse of the polynomial-time hierarchy to its second level. The proof of this result applies the notion of *selectivity*, or semimembership algorithm, a recursion-theoretic concept defined in the 1960's that was translated to the polynomial-time setting by Selman in [Sel79]. We define only NPSV-selectivity, the type of selectivity that is used in the following result. We say that a set A is NPSV-selective if there is a function $f \in \text{NPSV}$, the selector, such that for every pair of strings x, y, $f(x, y) \in \{x, y\}$ or is undefined, and if at least one of the strings x or y belongs to A, then $f(x, y) \in A$.

To show that computing solutions for NP-complete problems in NPSV implies a collapse of PH, Hemaspaandra et al. [HNOS94] proved the following lemma relating the possibility of computing solutions in NPSV with NPSV-selectivity and then showed that NPSV-selective sets cannot be too hard. The result of the lemma holds in fact in both directions. For simplicity we only consider here the direction needed to prove the main result.

Lemma 4.1 *If solutions for* SAT *can be computed in* NPSV*, then* SAT *is* NPSV*-selective.*

Proof. Let f be a function in $\text{NPSV} \cap \mathcal{F}_{\text{SAT}}$, and let M be a nondeterministic transducer computing f. From M, we can construct an NPSV selector for SAT, as described by the following nondeterministic program M': On inputting a pair of Boolean formulas (F_1, F_2), let $G = F_1 \vee F_2$ and simulate M on G. If M outputs a string y (a solution for G), check whether y satisfies F_1. If this is the case, output F_1, otherwise output F_2. Clearly, M' computes an NPSV function that selects SAT. □

The next result shows that NPSV-selective sets can be computed in NP∩coNP with the help of some small amount of additional information. As defined by Karp and Lipton [KL80] for a complexity class $\mathcal{C}$, the nonuniform version of $\mathcal{C}$, $\mathcal{C}/\text{poly}$, is the class of sets A for which there is a set $B \in \mathcal{C}$, a polynomial p, and a function h such that for every string x it holds that

$|h(x)| \leq p(|x|)$, and $x \in A$ if and only if $\langle x, h(0^{|x|})\rangle \in B$. Ko proved in [Ko83] that P-selective sets are in P/poly. The proof of the next result follows similar arguments.

Theorem 4.2 NPSV-*selective sets in* NP *are in* (NP $\cap$ coNP)/poly.

Proof. Let A be a set in NP that is NPSV-selective via a selector function f. without loss of generality, we can suppose that for every pair of strings x, y, $f(x, y)$ is either undefined or $f(x, y) = f(y, x)$. Let $n \in \mathbb{N}$. We show how to construct an advice of polynomial size for all the strings in $\Sigma^{\leq n}$. Consider a directed graph G_n that has as nodes all the strings in $\Sigma^{\leq n}$ and for which there is a directed edge from node x to node y if $f(x, y) = y$. The subgraph induced by the strings of $A^{\leq n}$ is a tournament, that is, there is a directed edge between each pair of nodes in this subgraph. Tournament graphs always have a dominating set of size logarithmic in the number of nodes. Since $||A^{\leq n}|| \leq 2^{n+1}$, there is a dominating set of nodes $S \subseteq A$ of size at most $n + 1$. Observe that for each $x \in \overline{A}$ and for each $y \in S$, $f(x, y) = y$, and also, for each $z \in A \setminus S$, it holds that $f(y, z) = z$. That is, S together with the selector function can give enough information to decide A.

For two finite sets, S and T, we will say that the triple $\langle 0^n, S, T\rangle$ is an *advice* if $S \subseteq A^{\leq n}$ and T contains a proof of membership in A for each string in S. Define the set A' as

$$A' = \{\langle x, \langle 0^{|x|}, S, T\rangle\rangle \quad | \quad \langle 0^{|x|}, S, T\rangle \text{ is an advice and for some string } y \in S, f(x, y) = x\}.$$

The set A' is clearly in NP. Moreover, A' belongs also to coNP, since an instance $\langle x, \langle 0^{|x|}, S, T\rangle\rangle$ is in $\overline{A'}$ if either $\langle 0^{|x|}, S, T\rangle$ is not an advice (this can be tested in polynomial time) or for all strings $y \in S$, $f(x, y) = y$. This second fact is true, since in case $\langle 0^{|x|}, S, T\rangle$ is an advice, $S \subseteq A$, and therefore for all strings $y \in S$, $f(x, y)$ is defined.

We can define $h(0^n)$ to be the advice $\langle 0^n, S, T\rangle$, encoding in S the smallest dominating set in the subgraph induced by A in G_n, and in T the smallest proofs of membership in A for the strings in S. Clearly, for every string x, $x \in A$ if and only if $\langle x, h(0^{|x|})\rangle \in A'$, and this proves $A \in$ (NP$\cap$coNP)/poly. □

Improving a result by Karp and Lipton [KL80], Köbler and Watanabe [KW95] have shown that if SAT $\subseteq$ (NP$\cap$coNP)/poly, then PH collapses to the class $\mathrm{ZPP}^{\mathrm{NP}}$. This result, together with Lemma 4.1 and Theorem 4.2, yields the following theorem, stated in [HNOS94].

Theorem 4.3 *If solutions to* NP*-complete sets can be computed in* NPSV*, then the polynomial-time hierarchy collapses to* $\mathrm{ZPP}^{\mathrm{NP}}$.

This result has been subsequently improved in [BKT94] and recently by Ogihara [Ogi95] in a way that relates bounded queries and NPSV functions:

If solutions to NP-complete sets can be computed in $\mathrm{FP}^{\mathrm{NPSV}}[\epsilon(\log n)]$ for some $\epsilon < 1$, then the polynomial-time hierarchy collapses to Σ_2. ($\mathrm{FP}^{\mathrm{NPSV}}$ is the class of functions computed in polynomial time with the help of functional oracle queries to NPSV [FHOS93].)

5 Nonadaptive Queries to NP: The Class $\mathrm{FP}_{tt}^{\mathrm{NP}}$

The prefix search method described in Section 2 to compute the optimal solution of an NP problem queries an NP oracle in an adaptive way, and the fact that the queries made at a certain stage of the computation depend on the previous answers seems crucial. A question that has motivated important research efforts is whether NP problems have some solution that can be obtained with *nonadaptive* queries to NP, that is, whether functions in $\mathrm{FP}_{tt}^{\mathrm{NP}}$ can compute proofs of memberships for problems in NP. $\mathrm{FP}_{tt}^{\mathrm{NP}}$ is defined to be the class of functions that can be computed in polynomial time by a machine that makes only *truth-table* or *parallel* queries to NP. This means that the queries cannot depend on previous oracle answers and have to be written in a list before the machine gets any oracle answer.

It is easy to see that $\mathrm{FP}_{tt}^{\mathrm{NP}}$ contains both classes $\mathrm{FP}^{\mathrm{NP}}[O(\log n)]$ and NPSV. For the inclusion $\mathrm{FP}^{\mathrm{NP}}[O(\log n)] \subseteq \mathrm{FP}_{tt}^{\mathrm{NP}}$, observe that if an oracle machine queries some string in a computation stage, depending on the oracle answer, there are at most two new strings that can be queried in the next query stage. If only a logarithmic number of adaptive queries is allowed, a polynomial-time machine can consider the complete tree of polynomially many possible oracle queries for a given input and query them all at the same time in parallel. The inclusion $\mathrm{NPSV} \subseteq \mathrm{FP}_{tt}^{\mathrm{NP}}$ illustrates a fact that will play an important role in other results, namely, that if there is only one possible solution, then it can be computed with nonadaptive queries to NP. This is done by asking in parallel for all $i's$ up to the solution's length whether there is a solution whose ith bit is a 1.

Again, in this case there are natural problems in NP for which $\mathrm{FP}_{tt}^{\mathrm{NP}}$ is the best-known classification for the complexity of its search problem. An example is Graph Automorphism, the problem to decide whether a given graph has an automorphism different from the identity. Lozano and Torán proved in [LT92] that the smallest nontrivial permutation (in the standard order of permutations) that defines an automorphism can be found with parallel queries to Graph Automorphism. Curiously, the problem of finding the largest permutation is as hard as Graph Isomorphism, which seems to be a harder problem. These results have recently been generalized in [AA96]. Another example of a problem with solutions in $\mathrm{FP}_{tt}^{\mathrm{NP}}$ is Group Intersection, the problem to decide, given sets of generators for two permutation groups, whether there is a permutation (different from the identity) in the intersection of both groups. In [Tor95] it is shown that permutations

in the intersection group can be obtained with parallel queries to Group Intersection.

As we have mentioned, an important open question is whether solutions can be computed in $\mathrm{FP}_{tt}^{\mathrm{NP}}$ for other NP problems, in particular, for NP-complete problems.

Trying to compute optimal solutions in $\mathrm{FP}_{tt}^{\mathrm{NP}}$ may be asking for too much. This would imply that $\mathrm{FP}^{\mathrm{NP}}$ coincides with $\mathrm{FP}_{tt}^{\mathrm{NP}}$. In [Sel94] it is shown that the equality of the function classes is equivalent to the equality of the language classes P^{NP} and $\mathrm{P}_{tt}^{\mathrm{NP}}$. Although no unexpected consequences of these equalities are known, we believe that the two function classes $\mathrm{FP}^{\mathrm{NP}}$ and $\mathrm{FP}_{tt}^{\mathrm{NP}}$ are different. However, there might be other solutions different from the optimal one that could be easier to compute and that might be obtained with nonadaptive queries to NP. Watanabe and Toda [WT93] have shown that in fact this is true in almost all relativized worlds. Again, for simplicity, we state the result for the case of SAT, and in fact the original result is stronger than the one mentioned here.

Theorem 5.1 *There is a polynomial-time machine M that nonadaptively queries a set in NP^X such that for almost every oracle X, M computes proofs of membership for* SAT. *That is, the Lebesgue measure of $\{X : \mathrm{FP}_{tt}^{\mathrm{NP}^X} \cap \mathcal{F}_{\mathrm{SAT}} \neq \emptyset\}$ is 1.*

Although this is a statement about almost every oracle, the result in fact has to do with random oracles, and more generally with randomized computation. In results dealing with almost every oracle, the measure of the size of a class of sets is done in terms of the probability that a random set belongs to the class. Random sets are constructed by independent series of tosses (one toss for each string in Σ^*) of an unbiased coin.

The proof of this theorem is strongly based on the well-known result by Valiant and Vazirani [VV86] that says that for any nonempty set $A \subseteq \Sigma^n$, with the help of n randomly chosen vectors from $\{0,1\}^n$ it is possible with high probability to isolate a single element of A. Roughly speaking, the idea from the proof of [WT93] is to get from the random oracle enough randomness to perform the isolation technique from Valiant and Vazirani on the set of satisfying assignments of a given formula. Once a single assignment has been selected, it can be computed with nonadaptive queries to NP as in the proof of $\mathrm{NPSV} \subseteq \mathrm{FP}_{tt}^{\mathrm{NP}}$.

An intermediate result proved by Watanabe and Toda in order to obtain Theorem 5.1 gives a clear view of the role played by randomization in the result.

Theorem 5.2 *For every polynomial p there exists a polynomial-time randomized nonadaptive machine M querying a set in* NP *such that for every Boolean formula F:*

i) if $F \in$ SAT, then $Pr\{M(F)$ outputs a satisfying assignment for $F\} \geq 1 - 2^{-p(|F|)}$;

ii) if $F \notin \mathrm{SAT}$, *then* $Pr\{M(F) \text{ is undefined}\} = 1$.

As the authors point out in [WT93], this result does not say that there is a randomized nonadaptive query machine that computes a function in $\mathcal{F}_{\mathrm{SAT}}$, since M might output different assignments for different random bits and therefore does not strictly compute a function. The existence of such a randomized machine computing a function in $\mathcal{F}_{\mathrm{SAT}}$ is still an open problem.

In spite of the fact that Theorem 5.1 holds for almost every oracle, Buhrman and Thierauf [BT96] have obtained a relativization in the opposite direction. Using results about exponential time, they have constructed an oracle A such that $\mathrm{FP}_{tt}^{\mathrm{NP}^A} \cap \mathcal{F}_{\mathrm{SAT}} = \emptyset$.

From Theorems 5.1 and 5.2, one might get the idea that if some solution for an NP-complete problem like SAT can be computed in $\mathrm{FP}_{tt}^{\mathrm{NP}}$, then it should be a "random one." On the other hand, the next result from Buhrman, Kadin, and Thierauf [BKT94] states that if a solution for SAT can be computed in $\mathrm{FP}_{tt}^{\mathrm{NP}}$, then also a very special one, namely the solution with the maximum number of 0's, can be computed in this class.

Theorem 5.3 $\mathcal{F}_{\mathrm{SAT}} \cap \mathrm{FP}_{tt}^{\mathrm{NP}} \neq \emptyset$ *if and only if satisfying assignments for* SAT *with maximum number of* 0*'s can be computed in* $\mathrm{FP}_{tt}^{\mathrm{NP}}$.

Adaptive versus nonadaptive oracle access is an important research topic. As in the adaptive case, the number of nonadaptive queries to an NP oracle is a resource that has become a subject of study. In these studies, one considers the relative power of function classes defined by bounding the number of oracle queries.

The following result from [BKS94] is the analogue of Theorem 3.2 for the nonadaptive setting.

Theorem 5.4 *Let* f *be smooth and* $f(n) \leq \epsilon \log n$ *for some* $\epsilon < 1$.
If $\mathrm{FP}_{tt}^{\mathrm{NP}}[f(n)] \subseteq \mathrm{FP}^{\mathrm{NP}}[f(n) - 1]$, *then* $\mathrm{P} = \mathrm{NP}$.

A different collapse for functions in $O(\log n)$ was obtained in [ABG94].

Theorem 5.5 *Let* f *be smooth and* $f \in O(\log n)$.
If $\mathrm{FP}_{tt}^{\mathrm{NP}}[f(n)] \subseteq \mathrm{FP}^{\mathrm{NP}}[f(n) - 1]$, *then* $\mathrm{NP} \subseteq \mathrm{coNP/poly}$ *(and* $\mathrm{PH} = \Sigma_3^p$*)*.

The question of whether $\mathrm{FP}^{\mathrm{NP}}[O(\log n)]$ and $\mathrm{FP}_{tt}^{\mathrm{NP}}$ coincide is of special interest. We have seen already that $\mathrm{FP}^{\mathrm{NP}}[O(\log n)] \subseteq \mathrm{FP}_{tt}^{\mathrm{NP}}$. Moreover, in the decisional case (that is, in the case where the functions computed are restricted to produce 0/1 outputs) both classes coincide.

If the classes $\mathrm{FP}_{tt}^{\mathrm{NP}}$ and $\mathrm{FP}^{\mathrm{NP}}[O(\log n)]$ coincide, then there is a polynomial-time algorithm that correctly decides the satisfiability of a formula with at most one satisfying assignment. (If the formula has more than one assignment, the algorithm may incorrectly decide that the formula is not satisfiable.) In order to see this, observe that we can define a function $f \in \mathrm{FP}_{tt}^{\mathrm{NP}}$ that for a Boolean formula $F(x_1, \ldots, x_n)$ outputs a string $a_1 \ldots a_n \in \{0,1\}^n$ with $a_i = 1$ if and only if there is a satisfying assignment

for F assigning value 1 to the variable x_i. In the case that F has a unique satisfying assignment, this is the value of $f(F)$. Under the hypothesis of equality of the function classes above, f is contained in $\mathrm{FP}^{\mathrm{NP}}[O(\log n)]$. Simulating all the possible values for the oracle answers of the machine computing f, one can get a list of polynomially many values, one of which satisfies the formula F. This result is stated formally using the concept of promise problems (see [ESY84]). A promise problem is a pair of sets (Q, R). A set L is called a solution to the promise problem (Q, R) if $\forall x(x \in Q \Rightarrow (x \in L \Leftrightarrow x \in R))$. 1SAT denotes the set of Boolean formulas with at most one satisfying assignment.

Theorem 5.6 *If*

$$\mathrm{FP}_{tt}^{\mathrm{NP}} \subseteq \mathrm{FP}^{\mathrm{NP}}[O(\log n)],$$

then the promise problem $(1\mathrm{SAT}, \mathrm{SAT})$ *has a solution in* P.

A polynomial-time solution for the promise problem $(1\mathrm{SAT}, \mathrm{SAT})$ would imply the unexpected consequences expressed in the following theorem.[5]

Theorem 5.7 *If the promise problem* $(1\mathrm{SAT}, \mathrm{SAT})$ *has a solution in* P, *then* FewP = P, NP = R, *and* coNP = US.

As a consequence of these two results, we state the following theorem, which summarizes the situation. This result was obtained by Selman in [Sel94].[6]

Theorem 5.8 *If*

$$\mathrm{FP}_{tt}^{\mathrm{NP}} \subseteq \mathrm{FP}^{\mathrm{NP}}[O(\log n)],$$

then FewP = P, NP = R, *and* coNP = US.

The consequence NP = R is especially strong, since it implies the collapse of the polynomial-time hierarchy. We obtained in [JT95] a different consequence of the equality of the function classes that, contrary to the previous results, does not seem to be related to the promise problem $(1\mathrm{SAT}, \mathrm{SAT})$. Namely, if $\mathrm{FP}_{tt}^{\mathrm{NP}} = \mathrm{FP}^{\mathrm{NP}}[O(\log n)]$, then a polylogarithmic amount of nondeterminism can be simulated in polynomial time, and SAT can be decided (for any k) in polynomial time with the help of only $\frac{n}{\log^k n}$ nondeterministic bits. The main tool needed for these results is the following theorem, which under the hypothesis of the equality of the function classes gives an upper bound for the complexity of selecting a Boolean formula from a list. For a smooth function f, $\mathrm{NPSV}^{\mathrm{NP}}[f]$ denotes the generalization of NPSV to functions computed by a nondeterministic single-valued machine that can

[5] Valiant and Vazirani [VV86] obtained the second and third consequences. Beigel [Bei88] observed that the weaker result UP = P follows from the equivalent hypothesis that Unique-SAT and $\overline{\mathrm{SAT}}$ are P-separable.

[6] Beigel [Bei88] and Toda [Tod91] contain other interesting applications of Theorem 5.7.

make f queries to an NP oracle before doing any nondeterministic step (see [JT95]).

Theorem 5.9 *If*

$$\mathrm{FP}_{tt}^{\mathrm{NP}} \subseteq \mathrm{FP}^{\mathrm{NP}}[O(\log n)],$$

then there is a function $f \in \mathrm{NPSV}^{\mathrm{NP}}[O(\log\log)]$ *that for a sequence of Boolean formulas* $F_1, \ldots, F_n$ *outputs one satisfiable formula from the list in case one exists.*

The proof of this result orders the set of possible answers given by the machine computing the function in $\mathrm{FP}^{\mathrm{NP}}[O(\log n)]$ forming a lattice and uses combinatorial arguments and the fact that the Set Cover Problem can be approximated by a logarithmic factor to contract this lattice. An interesting observation is that the satisfiable formula selected from a sequence $F_1, \ldots, F_n$ by the function f in the result is not necessarily the first satisfiable one in the sequence; the selection depends on how a function in $\mathrm{FP}_{tt}^{\mathrm{NP}}$ can be computed with $O(\log n)$ adaptive queries. Based on these results, we obtained some consequences that make reference to subclasses of NP with bounded nondeterminism (see Section 3), showing that under the hypothesis $\mathrm{FP}_{tt}^{\mathrm{NP}} = \mathrm{FP}^{\mathrm{NP}}[O(\log n)]$, a significant reduction in the number of nondeterministic bits in an NP computation can be obtained.

Theorem 5.10 *If*

$$\mathrm{FP}_{tt}^{\mathrm{NP}} = \mathrm{FP}^{\mathrm{NP}}[O(\log n)],$$

then for any smooth function f *and for any* $k \in \mathbf{N}$, $\mathrm{NP}(f) \subseteq \mathrm{NP}(\frac{f}{\log^k})$.

This theorem has some direct consequences. For example, from the equality of the function classes it follows that for any $k \in \mathbf{N}$, the class $\mathrm{NP}(\log^k)$ is included in P and $\mathrm{SAT} \in \mathrm{NP}(\frac{n}{\log^k n})$.

Although Theorems 5.9 and 5.10 give strong evidence that the function classes are different, it is still an open problem whether the hypothesis $\mathrm{FP}^{\mathrm{NP}}[O(\log n)] = \mathrm{FP}_{tt}^{\mathrm{NP}}$ implies $\mathrm{P} = \mathrm{NP}$. A recent result from [NS96] makes progress in this direction. We refer to the original paper for the definition of *effective inclusion*.

Theorem 5.11 *If there is a constant* k *such that* $\mathrm{FP}_{tt}^{\mathrm{NP}}[n^k]$ *is effectively included in* $\mathrm{FP}^{\mathrm{NP}}[k\lceil \log n \rceil - 1]$, *then* $\mathrm{P} = \mathrm{NP}$.

6 A Look inside Nondeterministic Logspace

As in the NP case, problems computable in nondeterministic logarithmic space (NL) are traditionally decision problems. Typical (NL-complete) examples are the Graph Accessibility Problem for directed graphs, where for a graph G it is asked whether there is a path between two given nodes s and t; or the Nonemptiness Problem for Nondeterministic Finite-state

Automata, where for a given automaton A it is asked whether $L(A)$ is nonempty.

As in the case of NP, it is more natural to consider the search versions of these problems, which compute a solution for the decision problem at hand, that is, a path from s to t in G or a witness $w \in L(A)$ for the nonemptiness problem. Again, this can best be formalized using a quantifier characterization of NL. To obtain such a characterization, it does not suffice to restrict the predicate $R(x, y)$ in the characterization of NP to be computable in logspace (this again yields a characterization of NP). Additionally, the parameter y of the input (the solution) must only be scanned one-way. To compute these predicates, we hence need a special, somewhat unnatural version of the common (deterministic) Turing machine model, with two input tapes, one of which contains x and is scanned two-way, and one of which contains y and is scanned one-way. Such machines were introduced by Lange [Lan86]) and are called *protocol machines*, because the one-way parameter y can be thought of as a protocol of an NL computation.

Let A be a problem in NL. Then, there is a polynomial p and a relation R computable by a logspace-bounded protocol machine such that $\forall x \in \Sigma^*$,

$$x \in A \Longleftrightarrow \exists \vec{y} \in \Sigma^* : \ |\vec{y}\,| \leq p(|x|) \text{ and } R(x, \vec{y}\,).$$

(Here, the right arrow above y indicates that y is only scanned one-way.) Completely analogous to the case of NP, the *search problem* for A now consists of finding a solution for an input instance x, that is, a string $\vec{y}$ such that $R(x, \vec{y}\,)$ holds. In the setting mentioned, a special case of the search problem is the *optimization problem* for A, to compute for x a solution of maximum (minimum) cost. More formally, given a cost function c computable by a logspace protocol machine, the function $f_{R,c}$ is defined by

$$f_{R,c}(x) = \begin{cases} \text{some } \vec{y}\,, & |y| \leq p(|x|), R(x, \vec{y}\,), \text{ and} \\ & c(x, \vec{y}\,) = max\{c(x, \vec{z}\,) \mid R(x, \vec{z}\,)\}, \\ 0, & \text{if no } \vec{y} \text{ with } R(x, \vec{y}\,) \text{ exists.} \end{cases}$$

(Observe that in this case the function is total. As we will see, this has to do with the fact that NL is closed under complementation).

In this section, we study the exact complexity of computing solutions for NL problems and the function classes that arise in the classification of these problems. We are especially interested to see how the situation here compares with the NP case. One might suspect that the results obtained for NP simply translate to the NL case. However, as we will see, the techniques of Theorem 2.1 cannot be applied in the logspace case.

We mainly consider two function classes, OptL and $\mathrm{FL}^{\mathrm{NL}}$. NL optimization corresponds to the class OptL and, surprisingly, there are NL optimization problems that seem to be harder than computing lexicographically optimal solutions. The complexity of obtaining such solutions is captured

by the class $\mathrm{FL}^{\mathrm{NL}}$ of functions computable by a logspace machine with access to an oracle in NL. NL optimization problems are only known to be in $\mathrm{FL}^{\mathrm{NL}}$ for polynomially bounded cost functions, and computing the cost (without the solution) of such NL optimization problems (a form of weak NL optimization) can be done with just logarithmically many queries to NL.

In order to capture the complexity of NL optimization problems, Àlvarez and Jenner [AJ93] introduced the complexity class OptL analogous to the class OptP of Krentel [Kre88]. The functions in OptL are defined using a machine model called the NL transducer. An NL *transducer* is a nondeterministic logarithmic space-bounded polynomially clocked Turing machine T with a (write-only) unbounded output tape and accepting and rejecting final states. In any move of T, T writes an output bit 0 or 1, or nothing, onto its output tape. The output of a computation of T on input x is the content of T's output tape when T halts, and if T halts in an accepting state, the output is considered to be "valid." We assume that any transducer has at least one valid output for each input. This is no restriction for NL transducers: Since NL is closed under complementation [Imm88, Sze88], the absence of a valid output can be checked by a precomputation. Observe that although the number of reachable configurations of T is bounded by a polynomial in the length of the input, due to the logarithmic space bound, T can have exponentially many valid outputs. The length of any of the outputs is always bounded by a polynomial in the length of the input.

For a transducer T, let opt_T denote the function that for an input x computes the maximum valid output value of T on x with respect to lexicographical order. The class OptL is the class of functions f such that for some NL transducer T and every input x, $f(x) = opt_T(x)$.

In the NP case, the class OptP, defined via metric Turing machines, clearly captures the idea of NP optimization problems. (More formally, the closures of these classes under metric reducibility coincide.) Moreover, the class is closely related to $\mathrm{FP}^{\mathrm{NP}}$ (see Theorem 2.1). In the NL case, some new arguments are needed to prove that both characterizations of NL optimization, via predicates or via NL transducer, basically yield the same class. We say that a function f is *logspace metric reducible* to a function g if there exist two logspace computable functions h_1, h_2 such that $f(x) = h_1(x, g(h_2(x)))$ for all x. The logspace metric closure of a class C is the class of all functions that are logspace metric reducible to a function in C.

Theorem 6.1 *The logspace metric closures of the class of* NL *optimization problems and of the class* OptL *coincide.*

Proof. First, we reduce an arbitrary function $f \in$ OptL computed by an NL transducer T to an NL optimization problem $g_{R,c}$. Define $R(x, \vec{y})$ to be the predicate stating that T on input x produces some valid output on computation $\vec{y}$, and define $c(x, \vec{y})$ to be the valid output produced by $T(x)$ following the computation $\vec{y}$ (or 0, if no valid output is produced).

Then, $g_{R,c}(x)$ defines a computation $\vec{y}$ with maximum cost, and $f(x)$ is obtained by the reduction that simply computes $c(x, \vec{y})$.

Now, we show the reduction of an arbitrary NL optimization problem $f_{R,c}$ to an opt_T function for an NL transducer T. The straightforward idea here is to use codifications of the solutions $\vec{y}$ together with their costs $c(x, \vec{y})$ as output of the transducer T. However, T can only remember costs of up to logarithmic length. The following trick achieves the result.

Without loss of generality, we can assume that there are polynomials p and q such that the solutions $\vec{y}$ satisfying $R(x, \vec{y})$ have length exactly $p(|x|)$ and the costs $c(x, \vec{y})$ have length exactly $q(|x|)$. Let M_R and T_c be the logspace protocol machines that compute R and c, respectively. Let the *position-i-configuration* of T_c or M_R on input $(x, \vec{y})$ be the configuration that the machines enter when they move the input head of the protocol tape to position i, the ith symbol of $\vec{y}$. The position-1-configuration is just the initial configuration.

On input x, T works in $p(|x|)$ stages, giving in each stage one bit of the solution. On stage i, T guesses a solution $\vec{y}$ simulating in parallel the machines M_R and T_c step by step according to the guessed bits of $\vec{y}$, and outputs $c(x, \vec{y})$ followed by the ith bit of $\vec{y}$. T discards its computation if M_R rejects; otherwise, it continues with the next stage. During this simulation, just when the ith bit of $\vec{y}$ is guessed, T checks whether both machines M_R and T_c are in their position-i-configurations, and if this is not the case, T rejects. Furthermore, T stores the position-$(i+1)$-configurations of M_R and T_c to continue the simulation in the next stage correctly.

It can be shown that after $p(|x|)$ iterations, the maximum valid output produced by T has the form $c(x, \vec{y})y_1 c(x, \vec{y})y_2 \ldots c(x, \vec{y})y_p$, where $y_1 \ldots y_p$ encodes a solution $\vec{y}$ maximizing the cost $c(x, \vec{y})$. Simple counting, and hence logspace, suffices to obtain this solution from the sequence. This shows that NL optimization problems are logspace metric reducible to functions in OptL. □

OptL provides an upper bound to all NL optimization problems. But how difficult are OptL functions to compute? One answer can be given in terms of complete problems for OptL. These include, for example, computing the Maximal Word Function for Nondeterministic Finite-state Automata [AJ93] and computing the Iterated Product of Word Matrices over $\{0,1\}^*$ using the operations maximum and concatenation [AJ95].

As best-known upper bound, it is known that OptL is included in AC^1, the class of functions computable by unbounded fan-in Boolean circuits of logarithmic depth [AJ95]. As shown by Allender, Bruschi, and Pighizzini [AW90], AC^1 contains the broad class of optimization functions defined via optimization for one-way nondeterministic auxiliary pushdown automata (and hence for the class of context-free languages).

It is an open problem whether the class OptL is included in the class DET of problems that are NC^1 reducible to computing the determinant

of an integer matrix. This class was introduced by Cook in [Coo85]. The question is equivalent to the question of whether computing the Iterated Product of Word Matrices using the operations maximum and concatenation is reducible to the normal Iterated Matrix Product. (For an overview of the complexities of various iterated matrix products see Immerman and Landau [IL95].)

The second class that we consider is $\mathrm{FL}^{\mathrm{NL}}$, the class of all functions computable by a (deterministic) logspace transducer with an oracle in NL. (FL denotes the class of all deterministic logspace functions.) Observe that an NL transducer can easily simulate an $\mathrm{FL}^{\mathrm{NL}}$ machine solving the oracle queries by NL subroutines, using again that NL is closed under complementation. (This NL transducer is in fact single-valued, that is, it computes exactly one valid value for each input). This shows that $\mathrm{FL}^{\mathrm{NL}} \subseteq \mathrm{OptL}$. Contrary to the case of OptL, $\mathrm{FL}^{\mathrm{NL}}$ is known to be included in DET [Coo85].

In the polynomial-time setting, the class OptP is included in $\mathrm{FP}^{\mathrm{NP}}$ (see Theorem 2.1). The proof of this result based on prefix search does not fully translate to the NL case, since here we can only remember prefixes up to logarithmic length. We can apply prefix search when the cost function is polynomially bounded. Similarly, a prefix technique can be used to compute solutions and lexicographically maximal solutions of NL search problems.

Theorem 6.2 *(i) The lexicographically maximal solution of any* NL *search problem can be computed in* $\mathrm{FL}^{\mathrm{NL}}$.
(ii) NL *optimization problems with polynomially-bounded cost functions can be computed in* $\mathrm{FL}^{\mathrm{NL}}$.

Proof. (i) Let f_R be an NL search problem, let M be the logspace protocol machine that accepts R, and let p be the polynomial that bounds the running time of M. Define the NL oracle $L\$P := \{z\$v \mid z \in L,\ v \in P\}$, where

$$\begin{aligned} L &:= \{\langle x,k\rangle \mid \exists \vec{y},\ |y| \le p(|x|),\ R(x,\vec{y}),\ \text{and } |\vec{y}\,| \ge k\},\ \text{and} \\ P &:= \{\langle x,k,c\rangle \mid \exists \vec{z}, |\vec{z}\,| = k \text{ and } M \text{ started in configuration } c \\ &\qquad\qquad \text{on input } (x,\vec{z}) \text{ accepts}\}. \end{aligned}$$

For input x, using L the length of the lexicographically maximal solution $\vec{y}$ that satisfies R can be computed in logspace with binary search. Then, a prefix construction using the set P yields $\vec{y}$.

(ii) This is similar to the NP case. We have only to adapt the proof to the computation of protocol machines. The maximal cost can be computed with oracle queries to a set in NL (by brute force or, more efficiently, by binary search). This value can be stored within logspace. Now, we can construct a solution with that cost analogous to the proof of Theorem 2.1 using a variant of the set P above that incorporates the cost. □

We mention that bounded query classes in the logarithmic space setting also compute some interesting functions. $\mathrm{FL}^{\mathrm{NL}}[O(\log n)]$ is the class of all $\mathrm{FL}^{\mathrm{NL}}$ functions that are computable with only logarithmically many queries. In this class, for example, the length of the shortest path for a given instance of the Graph Accessibility Problem can be computed (or the length of the shortest word accepted by a given finite-state automaton, etc.). Similarly, any weak optimization problem for polynomially-bounded cost functions belongs to $\mathrm{FL}^{\mathrm{NL}}[O(\log n)]$.

The class $\mathrm{FL}^{\mathrm{NL}}$ can compute solutions for NL decision problems and hence provides an upper bound to all NL search problems. $\mathrm{FL}^{\mathrm{NL}}$ is a very robust class that has a variety of other characterizations (see [Coo85, ABJ95, AJ95]). For example, $\mathrm{FL}^{\mathrm{NL}}$ is the closure of NL under functional AC^0- or NC^1-reducibility, via an L transducer that make one round of nonadaptive queries to an oracle in NL or via a single-valued NL transducer. These different characterizations are not very surprising considering the fact that NL is closed under complementation. (Observe that under the hypothesis NP = coNP, all the classes $\mathrm{FP}^{\mathrm{NP}}$, $\mathrm{FP}_{tt}^{\mathrm{NP}}$, and NPSV coincide.)

Examples of functional problems that are complete for the class $\mathrm{FL}^{\mathrm{NL}}$ are Iterated Boolean Matrix Product [Coo85], the Maximal Word Function for Deterministic Finite-state Automata [AJ93], and computing the Iterated Product of Word Matrices over $\{1\}^*$ using the operations maximum and concatenation [AJ95]. The latter two problems are restrictions of problems that are complete for OptL.

This again shows that there is a structural difference between the classes $\mathrm{FL}^{\mathrm{NL}}$ and OptL. Unfortunately, it is not known whether $\mathrm{OptL} \subseteq \mathrm{FL}^{\mathrm{NL}}$ implies L = NL or any similar statement that is widely believed to be false. It remains an open problem to derive such an implication.

7 Conclusions

Finding solutions for problems in NP is a fundamental problem whose complexity has not yet been classified in a completely satisfactory way. We have reviewed some of the known results related to this question by considering several important complexity classes of functions that can compute proofs of membership for different problems in NP. We have shown that although the complexity of finding optimal solutions for NP-complete problems is well understood, there are many unsolved questions related to the complexity of computing other solutions. Here, the question of whether NP-complete problems have solutions that can be computed with polynomially many nonadaptive queries to NP is of particular interest. All the known examples of problems in NP with solutions computable in $\mathrm{FP}_{tt}^{\mathrm{NP}}$ have very special counting properties (see [KST92] and the chapter by Fortnow [For] in this book.) It is unclear whether these special properties hold only for

these examples or are true for any problem with solutions computable in $\mathrm{FP}_{tt}^{\mathrm{NP}}$. Also, some relationships between function classes remain open. For example, although we believe that $\mathrm{FP}_{tt}^{\mathrm{NP}}$ is strictly contained in $\mathrm{FP}^{\mathrm{NP}}$, no strong consequences are known to follow from the equality of both classes (such as, for example, a collapse of the polynomial-time hierarchy).

The most prominent open questions about logarithmic space-bounded function classes are, on the one hand, what is the (possible) difference between the classes $\mathrm{FL}^{\mathrm{NL}}$ and OptL (that is, computing the lexicographically maximal solution versus computing an optimal one) and, on the other hand, what are the precise relationships between optimization and other important problems of the area (computing the determinant or recognition of context-free languages).

The manifold open questions that relate the complexity of computing solutions to other aspects of complexity theory demonstrate the importance of this area of research.

Acknowledgments: We would like to thank Thomas Thierauf, Klaus-Jörn Lange, and the anonymous referees for many helpful comments.

References

[AA96] M. Agrawal and V. Arvind. A note on decision vs. search for graph automorphism. *Proceedings of the 11th Annual IEEE Conference on Computational Complexity*, pp. 272–277, 1996.

[AW90] E. Allender, D. Bruschi, and G. Pighizzini. The complexity of computing maximal word functions. *Computational Complexity*, 3: 368–391, 1993.

[AJ93] C. Àlvarez and B. Jenner. A very hard log-space counting class. *Theoretical Computer Science*, 107: 3–30, 1993.

[AJ95] C. Àlvarez and B. Jenner. A note on logspace optimization. *Computational Complexity*, 5: 155–167, 1995.

[ABJ95] C. Àlvarez, J. Balcázar, and B. Jenner. Adaptive logspace reducibility and parallel time. *Mathematical Systems Theory*, 28: 117–140, 1995.

[ABG94] A. Amir, R. Beigel, and B. Gasarch. Some connections between bounded query classes and non-uniform complexity. Manuscript, 1994. See also *Proceedings of the 5th Annual IEEE Conference on Structure in Complexity Theory*, pp. 232–244, 1990.

[BDG88] J. Balcázar, J. Diaz, and J. Gabarró. *Structural Complexity Theory I*, Springer-Verlag, Berlin, 1988.

[Bei88] R. Beigel. NP-hard sets are p-superterse unless R=NP. Technical Report TR4, Department of Computer Science, John Hopkins University, Baltimore, 1988.

[BKS94] R. Beigel, M. Kummer, and F. Stephan. Approximable sets. *Proceedings of the 9th Annual IEEE Conference on Structure in Complexity Theory*, pp. 12–23, 1994.

[BLS84] R. Book, T. Long, and A. Selman. Quantitative relativizations of complexity classes. *SIAM Journal on Computing*, 13: 461–487, 1984.

[BKT94] H. Buhrman, J. Kadin, and T. Thierauf. On functions computable with nonadaptive queries to NP. *Proceedings of the 9th Annual IEEE Conference on Structure in Complexity Theory*, pp. 43–53, 1994.

[BT96] H. Buhrman and T. Thierauf. The complexity of generating and checking proofs of membership. *Proceedings of the 13th Annual Symposium on Theoretical Aspects of Computer Science*, Lecture Notes in Computer Science 1046, pp. 75–86, Springer-Verlag, Berlin, 1996.

[Cha94] R. Chang. On the query complexity of clique size and maximum satisfiability. *Proceedings of the 9th Annual IEEE Conference on Structure in Complexity Theory*, pp. 31–42, 1994.

[CG93] R. Chang and W. Gasarch. On bounded queries and approximation. *Proceedings of the 36th Annual IEEE Symposium on Foundations of Computer Science*, pp. 547–556, 1993.

[CGL94] R. Chang, W. Gasarch, and C. Lund. On bounded queries and approximation. Technical Report TR CS-94-05, Computer Science Department, University of Maryland, College Park, 1994.

[Coo85] S. Cook. A taxonomy of problems with fast parallel algorithms. *Information and Control*, 64: 2–22, 1985.

[DT90] J. Díaz and J. Torán. Classes of bounded nondeterminism. *Mathematical Systems Theory*, 23: 21–32, 1990.

[ESY84] S. Even, A. Selman, and Y. Yacobi. The complexity of promise problems with applications to public-key cryptography. *Information and Control*, 61: 114–133, 1984.

[Far94] G. Farr. On problems with short certificates. *Acta Informatica*, 31: 479–502, 1994.

[FK92] M. Fellows and N. Koblitz. Self-witnessing polynomial-time complexity and prime factorization. *Proceedings of the 7th Annual IEEE Conference on Structure in Complexity Theory*, pp. 107–110, 1992.

[FHOS93] S. Fenner, S. Homer, M. Ogiwara, and A. Selman. On using oracles that compute values. *Proceedings of the 10th Annual Symposium on Theoretical Aspects of Computer Science*, Lecture Notes in Computer Science 665, pp. 398–408. Springer-Verlag, Berlin, 1993.

[For] L. Fortnow. Counting complexity. Chapter 4, this volume.

[GKR95] W. Gasarch, M. Krentel, and K. Rappoport. OptP-completeness as the normal behavior of NP-complete problems. *Mathematical Systems Theory*, 28: 487–515, 1995.

[HNOS94] L. Hemaspaandra, A. Naik, M. Ogihara, and A. Selman. Computing solutions uniquely collapses the polynomial hierarchy. *SIAM Journal on Computing*, 25(4): 697-708, 1996.

[Imm88] N. Immerman. Nondeterministic space is closed under complement. *SIAM Journal on Computing*, 17: 935–938, 1988.

[IL95] N. Immerman and S. Landau. The complexity of iterated multiplication. *Information and Computation*, 116: 103–116, 1995.

[JT95] B. Jenner and J. Torán. Computing functions with parallel queries to NP. *Theoretical Computer Science*, 141: 175–193, 1995.

[KK82] N. Karmarkar and R. Karp. An efficient approximation scheme for the one-dimensional bin-packing problem. *Proceedings of the 23rd*

Annual IEEE Symposium on Foundations of Computer Science, pp. 312–320, 1982.

[KL80] R. Karp and R. Lipton. Some connections between nonuniform and uniform complexity classes. *Proceedings of the 12th Annual ACM Symposium on the Theory of Computing*, pp. 302–309, 1980. [Also published as: Turing machines that take advice, *L'Enseignement Mathématique*, 28: 191–210, 1982.]

[KF84] C. Kintala and P. Fisher. Refining nondeterminism in relativized complexity classes, *SIAM Journal on Computing*, 13: 329–337, 1984.

[Ko83] K. Ko. On self-reducibility and weak P-selectivity. *Journal of Computer and System Sciences*, 26: 209–211, 1983.

[KST92] J. Köbler, U. Schöning, and J. Torán, The Graph Isomorphism problem is low for PP. *Computational Complexity*, 2: 301–330, 1992.

[KW95] J. Köbler and O. Watanabe. New consequences of NP having small circuits. *Proceedings of the 22nd International Colloquium on Automata, Languages, and Programming*, Lecture Notes in Computer Science 944, pp. 196–208, Springer-Verlag, 1995.

[Kre88] M.W. Krentel. The complexity of optimization problems, *Journal of Computer and System Sciences*, 36: 490–509, 1988.

[Lan86] K.-J. Lange. Two characterizations of the logarithmic alternation hierarchy. *Proceedings of the 12th International Symposium on Mathematical Foundations of Computer Science*, Lecture Notes in Computer Science 233, pp. 518–526, Springer-Verlag, Berlin, 1986.

[LT92] A. Lozano and J. Torán. On the non-uniform complexity of the graph isomorphism problem. *Proceedings of the 7th Annual IEEE Conference on Structure in Complexity Theory*, pp. 118–131, 1992.

[MV88] N. Meggido and U. Vishkin. On finding a minimum dominating set in a tournament. *Theoretical Computer Science*, 61: 307–316, 1988.

[NS96] A. Naik and A. Selman. A note on p-selective sets and on adaptive versus nonadaptive queries to NP. *Proceedings of the 11th Annual IEEE Conference on Computational Complexity*, pp. 224–232, 1996.

[Ogi95] M. Ogihara. Functions computable with limited access to NP. Technical Report 583, Department of Computer Science, University of Rochester, 1995.

[PY93] C. Papadimitriou and M. Yannakakis. On limited nondeterminism and the complexity of the VC-dimension. *Proceedings of the 8th Annual IEEE Conference on Structure in Complexity Theory*, pp. 12–19, 1993.

[Sel79] A. Selman. P-selective sets, tally languages, and the behavior of polynomial time reducibilities. *Mathematical Systems Theory*, 13: 55–65, 1979.

[Sel94] A. Selman. A Taxonomy of Complexity Classes of Functions. *Journal of Computer and System Sciences*, 48: 357–381, 1994.

[Sze88] R. Szelepcsényi. The method of forced enumeration for nondeterministic automata. *Acta Informatica*, 26: 279–284, 1988.

[Tod90] S. Toda. The complexity of finding medians. *Proceedings of the 31st Annual IEEE Symposium on Foundations of Computer Science*, pp. 778–787, 1990.

[Tod91] S. Toda. PP is as hard as the polynomial time hierarchy. *SIAM Journal on Computing*, 20: 865–877, 1991.

[Tod91b] S. Toda. On polynomial-time truth-table reducibilities of intractable sets to P-selective sets, *Mathematical Systems Theory*, 24: 69–82, 1991.

[Tor95] J. Torán. Solutions for the Group Intersection Problem can be obtained with parallel queries, Manuscript, December 1995.

[VV86] L. Valiant and V. Vazirani. NP is as easy as detecting unique solutions. *Theoretical Computer Science*, 47: 85–93, 1986.

[VW93] H. Vollmer and K. Wagner. The complexity of finding middle elements. *International Journal of Foundations of Computer Science*, 4: 293–307, 1993.

[WT93] O. Watanabe and S. Toda. Structural analysis of the complexity of inverse functions. *Mathematical Systems Theory*, 26: 203–214, 1993.