

available at www.sciencedirect.comjournal homepage: www.elsevier.com/locate/cosrev

Survey

The consequences of eliminating NP solutions[☆]

Piotr Faliszewski, Lane A. Hemaspaandra^{*}

Department of Computer Science, University of Rochester, Rochester, NY 14627, United States

ARTICLE INFO

Article history:

Received 11 July 2007

Received in revised form

1 February 2008

Accepted 1 February 2008

ABSTRACT

Given a function based on the computation of an NP machine, can one in general eliminate some solutions? That is, can one in general decrease the ambiguity? This simple question remains, even after extensive study by many researchers over many years, mostly unanswered. However, complexity-theoretic consequences and enabling conditions are known. In this tutorial-style article we look at some of those, focusing on the most natural framings: reducing the number of solutions of NP functions, refining the solutions of NP functions, and subtracting from or otherwise shrinking #P functions. We will see how small advice strings are important here, but we also will see how increasing advice size to achieve robustness is central to the proof of a key ambiguity-reduction result for NP functions.

© 2008 Elsevier Ltd. All rights reserved.

1. Introduction

Seal up the mouth of outrage for a while,
Till we can clear these ambiguities.
—Shakespeare, *Romeo and Juliet*, Scene 5, Act 3.

In everyday life it is natural to value clarity, and in particular to value those cases when a problem has a single, crisp answer. For example, if the question is, “Who won this presidential election?”, not many people would be happy with the answer “Maybe Al Gore did and maybe George W. Bush did” or with the answer “Both Al Gore and George W. Bush did”. And if a baseball team is deciding who will have the lead-off spot in the batting order, “Al and George will” is far from an acceptable outcome.

In computer science, issues of ambiguity and multiplicity are also quite central. For a regular language, we may wonder about the size of the smallest unambiguous finite automata that accepts it. For a context-free language, we may ask whether it has an unambiguous context-free grammar. For

an NP language, we may ask whether it is in UP, the unambiguous version of NP. For a multivalued NP function, we may ask whether it has a single-valued *refinement*—a pruning that on each instance on which the function has many solutions thins it down to exactly one solution.

In this paper we will survey the issue of whether multiplicities can be reduced – whether solutions can be eliminated – in the case of functions involving NP machines. In particular, we will try to gather together and put in context for the reader as many of the known results on this as this article’s space and flow constraints allow and, in some selected cases, will try to convey the flavor of representative proofs. The problem of reducing solutions might seem a technical issue interesting mostly to complexity theorists but it is in fact relevant to other areas of computer science as varied as database theory, planning systems, cryptography, average-case complexity analysis, structural complexity, parallel computation, and others. In particular, we point out how some of the problems that we discuss here in a theoretical setting either stem from considerations

[☆] This article grew out of an invited talk at the Eighth Workshop on Descriptive Complexity of Formal Systems.

^{*} Corresponding author.

URLs: <http://www.cs.rochester.edu/u/pfali/> (P. Faliszewski), <http://www.cs.rochester.edu/u/lane/> (L.A. Hemaspaandra).

in other subfields of computer science or are used in other subfields as tools. We thus hope that this survey may help increase cooperation between subareas of computer science.

We consider two natural domains in which solution reduction is studied for functions based on NP machines. The first one concerns the class NPMV and the problem of pruning the solution set (or, to use a more technical term, the problem of refining the function). We provide detailed definitions in the following section, but let us now give some intuitions. We say that a function f is multivalued if given an input argument x , $f(x)$ maps to a collection of values (mapping to zero, one, two, or even more values is allowed, and the number of values mapped to may differ for different inputs). (Although the term “multivalued function” is a long-used term in computer science, they are in fact relations rather than functions.) Let us consider an NP machine M such that on each input x each computation path of M on x can choose to output some string. We say that this machine computes a multivalued function f such that the values of $f(x)$ are exactly the strings output by computation paths of M on x . Thus, NPMV in a natural way models NP search problems, i.e., problems where we are interested in finding solutions rather than just deciding if a solution exists.

Theoreticians often neglect the study of search problems as many of the properties of search problems are readily derived from the properties of their underlying decision versions. However, issues such as solution reduction are most naturally studied directly in the context of function classes modeling search problems. In fact, researchers outside of complexity theory often model their problems in terms of classes of functions such as NPMV and #P (we will discuss #P in more detail soon). For example, Eiter et al. [15] present a language for describing planning problems and prove that finding certain plans is NPMV-complete (under an appropriate notion of completeness¹). They also show similar function-complexity results for other planning problems and other function classes. An important part of the work of Eiter et al. is that they seek plans that are in some sense optimal. That is, in their setting it is sometimes important to refine the search to focus on, e.g., shortest plans or plans with least cost; this can be viewed as a form of solution reduction.

The problem of focusing on a single solution from multiple ones appears even more sharply in the context of database systems. Given a relational database scheme (i.e., a description of the relations available in the system), a database query matching the scheme is essentially a partial recursive function that given a database outputs a relation represented as a set of tuples (an output relation). See, e.g., the paper by Chandra and Harel [9] for early

work on queries to relational databases. A nondeterministic query is defined analogously, except that it is allowed to output multiple relations, i.e., it is a multivalued function. Leone, Palopoli, and Saccà [48] argue that the ability to ask multivalued – nondeterministic – queries is important from the point of view of the expressive power of database systems. However, they point out that to execute a multivalued query on an actual database system one needs to refine it to a single-valued one, ideally in polynomial time. They show how to model the complexity of nondeterministic queries in terms of NPMV functions and describe classes of queries whose related search problems correspond to NPMV functions that can be refined by polynomial-time computable ones. Similar considerations appear in a paper by Greco and Saccà [28] in the context of the database oriented declarative programming language DATALOG. The findings of those papers and the body of this survey nicely complement each other, following Chandra and Harel’s [9] view that complexity-theoretic questions should be studied in the context of database systems (and the other way round, we should add).

The notion of refining multivalued functions with, say, single-valued ones appears in theoretically minded literature as well. We quickly mention a few examples where thinking in terms of function refinements proved useful. In the context of cryptography, Isobe et al. [41] use refinements in their study of the factoring problem versus the discrete logarithm problem. In the context of structural complexity of disjoint NP-pairs, Glaßer, Selman, and Sengupta [25] use results known about refining NPMV functions as evidence that certain hypotheses regarding pairs of disjoint NP sets are unlikely to hold. In the context of average-case complexity versus worst-case complexity considerations, Köbler and Schuler [45] study NPMV refinements of certain functions under the assumption that NP is easy on the average.

The problem of refining a multivalued function with a single-valued one is also important from the point of view of parallel algorithms. For example, consider the problem of finding a maximum matching in a graph. A given graph G can have exponentially many maximum matchings and it may be difficult to guarantee that parallel processors are cooperating in computing exactly the same one of them. Mulmuley, Vazirani, and Vazirani [53] showed how to, via assigning randomly chosen weights to the edges and considering a weighted version of the problem, guarantee that, with high probability, there is a unique solution which can be computed in parallel (see also the work of Chari, Rohatgi, and Srinivasan [10]). Earlier, in a similar vein, Valiant and Vazirani [74] showed that using a randomized reduction one can, in some (admittedly somewhat tricky) sense, with high probability refine the (potentially exponential-cardinality) solutions sets of satisfiable formulas down to one solution.

In the context of cryptography and key-agreement problems, Cai et al. [7] studied, among other issues, a notion they dub a “monic refinement” of a randomized key generator. While we will not attempt to present their results here in any detail, their high-level idea is that one can simplify the key-agreement problem if both parties involved have a randomized generator that is strongly biased toward a single value. If both parties use the generator biased in the

¹We mention in passing that, not surprisingly, there are a variety of completeness notions for function classes. For example, Krentel [46] suggested a reduction in effect very similar to a functional Turing reduction with a single query, Zankó [77] suggested a notion called functional many-one reduction, and Simon [68] suggested a very restrictive – but from the point of view of its structural properties, very satisfying – “parsimonious” reduction. Other notions exist as well. Papers studying functional reducibility notions include, for example, the work of Vollmer [75] and Fenner et al. [21].

same way then, with high probability, they obtain the same key via simply sampling from that biased generator. (We refer interested readers to their paper for full details [7].)

We stress that the above examples are meant as evidence that both the class NPMV and the notion of a multivalued function refinement are important in computer science. We do not claim that this list is in any way complete. Nonetheless, we hope it helps to highlight the significance of function refinement.

In the second part of our paper, Section 3, we focus on a different approach to solution reduction. We say that a function f is in #P if there exists an NP machine M such that for each input string x , M on x has exactly $f(x)$ accepting computation paths. Thus, while NPMV models obtaining solutions, #P models counting them. However, analogously to the case of NP, there certainly are #P functions and even #P-complete functions whose definitions center around notions relatively far from a direct discussion of simply counting accepting paths of NP machines. The classic example here is the permanent function, which was famously shown to be #P-complete by Valiant [72] and later on Zankó [77] showed it #P-complete even with respect to a less powerful reduction type than the one that Valiant had used.

The importance of #P stems, in part, from the practical significance of several well-known #P-complete functions. In particular, the problem of performing Bayesian inference is #P-complete [62] and, in the context of social sciences, many of the so-called power indices (see, e.g., [67,60]) are #P-complete as well. (A power index is a function that, in some sense, measures the power of individuals involved in a voting game.) Last but not least, the problem of model-counting for propositional logic, #SAT, is naturally #P-complete.

Such a variety of applications of #P makes it important to know this class well. Section 3 focuses on the problem of reducing the values of #P functions in some controlled way. This problem is interesting both as a fundamental issue and in terms of its possible applications. As an example, we will now consider the problem of comparing the values of the Shapley–Shubik power index of a given individual x in two voting games, G and G' . This problem (in light of a technical adjustment to the inputs that we discuss in footnote 3) models the situation in which individual x has a choice between two possible voting scenarios and he or she needs to decide in which one his or her vote (or the vote of the entities in each with which he or she will associate his or her fate) will change the outcome a greater portion of the time. Such situations are not uncommon in multiagent systems, particularly those based on the principles of social choice, and are even reasonable in everyday life. For example, Alice is considering joining the CS department at Cornell or the CS department at Columbia, and assuming that in the faculty senate of each school each department votes with weight equal to its size, and assuming Alice knows the sizes of all departments at both schools, Alice wishes to know at which school (under the standard and admittedly not perfectly real-world-capturing assumptions of power index computation about independence and randomness) her department would be the department that controls the outcome the higher percentage of the time. In a more business-oriented setting, an airline might in deciding which airline alliance to join be

interested in power-index analysis and comparison questions regarding its influence within the alliance and regarding the alliance’s overall influence.

Let us now somewhat formalize this issue and bring out a connection to whether #P is closed under subtraction. Given two weighted voting games, G and G' , an individual x involved in both games, and the function f (which we mention in passing is known to be #P-complete) computing the raw Shapley–Shubik power index,² we are interested in whether $f(G, x) > f(G', x)$, that is, whether x is influential in game G a greater portion of the time than in game G' .³ If #P were closed under subtraction (or, to be technically correct, if it were closed under proper subtraction; see Section 3), we would have a #P function $g(G, G', x) = \max(f(G, x) - f(G', x), 0)$ ⁴ and the problem of comparing x ’s performance in both

² Let us give a little bit more detail on weighted voting games and Shapley–Shubik power index here. An n -player weighted voting game is represented as a vector $[w_1, \dots, w_n; q]$ of nonnegative integers. Values w_1, \dots, w_n represent the weights of the players and q is a quota, i.e., a minimum total weight required by a coalition of players to be successful (e.g., $q = \lceil (1 + \sum w_i)/2 \rceil$ – requiring a strict majority of the total vote sum available – is a very typical setting). The raw Shapley–Shubik power index of a player i is exactly the number of permutations π of the n players for which there is a j such that $\pi(j) = i$, the coalition of players $\pi(1), \dots, \pi(j-1)$ is not successful, and the coalition of players $\pi(1), \dots, \pi(j-1), \pi(j) = i$ is successful. That is, the raw Shapley–Shubik power index measures how many times a given player is pivotal to a forming coalition. Note that in the Shapley–Shubik model, the underlying intuition is that all $n!$ orders of the relative strengths of the preferences of the players on the issue being voted on are equally likely, and so exactly one player will be the pivot relative to each permutation.

If one prefers the model in which we rather assume that each player randomly votes yes or no, and every player on a winning side whose vote when shifted to the other side changes the winning side to no longer win (a “swing” vote), one then would arrive at the raw Banzhaf index. One could repeat the entire above discussion, essentially identically, for that index. The only difference is that while the raw Shapley–Shubik is a direct scaling, depending just on the number of players, of the standard “normalized” Shapley–Shubik index, the raw Banzhaf is not. However, both raw indices are correct scalings, depending just on the number of players, of the probability that a given player is a pivot (Shapley–Shubik) or swing (Banzhaf). And so, assuming that in both the Shapley–Shubik and the Banzhaf cases we always follow the trick of footnote 3 to ensure that the games have the same number of players, the intuitive interpretation we are about to give – of $f(G, x) > f(G', x)$ as asking which game has x controlling the outcome with higher probability – is completely valid for both the Shapley–Shubik and the Banzhaf cases.

³ A “technical” but important issue: For the comparison to actually tell us which one more often (i.e., with higher probability, assuming the standard-for-power-index random/independence assumptions about players vote) changes the outcome, we need to make sure that both games have the same number of players. Fortunately, this is easy to do: If G and G' do not initially have the same number of players, boost the game with the smaller number of players up to the size of the game with the larger number of players by adding dummy 0-weight players, and then ask $f(G, x) > f(G', x)$ not for the original G and G' but for the thus-modified values.

⁴ The $\max(\dots, 0)$ is due to the fact that #P functions cannot have negative values.

games would be in NP. (Formally, the set $\text{PowerCompare}_f = \{(G, G', x) \mid f(G, x) > f(G', x)\}$ would be in NP, and that set can be used in light of footnote 3 to compare performances.) As disconcerting as the following may sound to very pure theoreticians, in practical situations one could then try to run a SAT solver to obtain the answer. (In recent years SAT solvers have reached such a level of development that they are routinely used to solve large instances of certain NP problems; in contrast, increased focus on #SAT solvers is remarkably recent and there is still a lot to be done; see, e.g., the following work for recent advances in the field and further references regarding theory and practice behind state-of-the-art #SAT solvers [12,64,70,13].) Recently, Faliszewski and Hemaspaandra [18] showed that PowerCompare_f , where f is either Shapley–Shubik or Banzhaf power index, is complete for probabilistic polynomial time (i.e., is PP-complete; we will later include a formal definition of PP). However, in Section 3 we return to this line of research and present a related open problem.

Interestingly, the issue of developing #SAT solvers (see the citations at the end of the previous paragraph) is also related to the problem of reducing the number of solutions of #P functions. For one, the classic DPLL backtracking algorithm modifies the formula as it runs to reflect the progress in search and eventually deletes solutions (though, of course, it works in worst-case exponential time). On the other hand, one of the powerful optimizations employed in state-of-the-art #SAT solvers is to decompose the given formula into two components that do not share any variables, solve each component separately, and multiply to obtain the final answer. That is, in some sense, #SAT solvers try to factor the values of #P functions (on some carefully prepared arguments). In Section 3 we, among other issues, consider the problem of integer division of #P functions which is a natural problem in the context of factoring. The results we present there may, in some sense, justify observed difficulties in applying the decomposition heuristics.

A very different application of the results from Section 3 appears in the paper by Faliszewski and Ogihara [19], where the authors analyze autoreductions of #P functions. They show a particularly simple autoreduction for #P-“parsimonious-complete” functions and argue – based on the fact that #P is unlikely to be closed under proper decrement – that this autoreduction cannot be easily simplified further.

This paper is structured as follows. Section 2 looks at solution reduction and solution refinement for NPMV functions. We will see that reducing solutions by polynomial-time computable, polynomially-bounded amounts is easy, but that refining solutions from many to one (or even from two to one) is impossible unless the polynomial hierarchy collapses. In the proof of this latter result we will see that small advice strings play a central role but that accepting an increase in advice size – in order to gain robustness – is the critical step in the proof. Section 3 focuses on the closure properties of #P – the class of functions that reflect the number of accepting paths of NP machines – with respect to decreasing operations, e.g., proper decrement, proper subtraction, minimum, integer division, etc. We will see that for each of these cases, #P is not closed under that type of decrease unless an unlikely complexity class collapse occurs.

Globally, we will take our alphabet Σ to be $\{0, 1\}$, \mathbb{N} will denote $\{0, 1, 2, \dots\}$, \mathbb{N}^+ will denote $\{1, 2, 3, \dots\}$, and for $n, m \in \mathbb{N}$, $n \dot{-} m$ denotes $\max(n - m, 0)$.

2. Reducing and refining NPMV functions

This section focuses on whether we can reduce or refine the solutions of multivalued NP functions. Let us start by defining the classes and notions that will be needed to study this question. (So that the similarities and differences between the various classes can be clearly seen, we also define in this section a few classes that will be used just in Section 3.)

2.1. Definitions

Let f be a (potentially) partial, (potentially) multivalued function. (In mathematics, one would call f a special kind of relation, but in theoretical computer science the term “multivalued function” is the norm in this context.) So, on each input x , either f is undefined or some strings from Σ^* are viewed as outputs of f . Following the standard notational approach to multivalued functions (see [66]), we will set $\text{set-}f(x) = \{y \mid y \text{ is an output of } f(x)\}$. Note in particular that $\text{set-}f(x) = \emptyset$ if and only if $f(x)$ is undefined. The “set”-notation avoids having to treat “undefined” as a special case, since it converts everything to clean output sets.

Our results in Section 3 regarding closure properties of #P functions involve the classes UP [71], SPP [57,20], PP [68, 24], C=P [68,76], and $\oplus P$ [58,26], and of these, PP and C=P will be important in the present section. For the sake of completeness, let us briefly recall the definitions of all these classes. We give the definitions in a form that is well suited for the way we use them in Section 3.

A language L belongs to the class UP if there exist a polynomial q and a polynomial-time computable binary predicate R such that, for each $x \in \Sigma^*$,

- (1) $x \in L \implies \|\{y \in \Sigma^* \mid |y| = q(|x|) \wedge R(x, y)\}\| = 1$, and
- (2) $x \notin L \implies \|\{y \in \Sigma^* \mid |y| = q(|x|) \wedge R(x, y)\}\| = 0$.

UP captures the notion of unambiguous nondeterminism. Indeed, quite typically UP is defined directly via unambiguous nondeterministic polynomial-time Turing machines. That is, a language L belongs to UP if there exists an NP Turing machine N that on input x has exactly one accepting path if $x \in L$ and that has no accepting paths if $x \notin L$. However, the first definition we gave better highlights the similarity between UP and SPP. SPP is a generalization of UP. A language L belongs to SPP if there exist two polynomials, q and p , and a polynomial-time computable binary predicate R such that, for all $x \in \Sigma^*$,

- (1) $x \in L \implies \|\{y \in \Sigma^* \mid |y| = q(|x|) \wedge R(x, y)\}\| = 2^{p(|x|)} + 1$, and
- (2) $x \notin L \implies \|\{y \in \Sigma^* \mid |y| = q(|x|) \wedge R(x, y)\}\| = 2^{p(|x|)}$.

A language L belongs to PP (probabilistic polynomial time) if there exist a polynomial q and a polynomial-time computable binary predicate R such that, for each $x \in \Sigma^*$,

$$x \in L \iff \|\{y \in \Sigma^* \mid |y| = q(|x|) \wedge R(x, y)\}\| \geq 2^{q(|x|)-1}.$$

Similarly, a language L belongs to $C=P$ if there exist a polynomial q and a polynomial-time computable binary predicate R such that, for each $x \in \Sigma^*$,

$$x \in L \iff \|\{y \in \Sigma^* \mid |y| = q(|x|) \wedge R(x, y)\}\| = 2^{q(|x|)-1}.$$

$C=P$ is typically described as capturing the power of exact counting, essentially due to the following alternate definition. A language L is in $C=P$ exactly if there exist a nondeterministic polynomial-time Turing machine N and a polynomial-time computable function f such that L is the set of all strings x for which the number of accepting paths of N on input x is exactly $f(x)$.

A language L belongs to the class $\oplus P$ (parity polynomial time) if there exists a nondeterministic polynomial-time Turing machine N such that, for each $x \in \Sigma^*$, $x \in L \iff N(x)$ has an odd number of accepting paths.

Naturally, there is a close connection between the above definitions and definitions directly employing nondeterministic Turing machines. However, the above definitions often are more convenient in the setting of closure properties, i.e., in Section 3.

The classes of multivalued and single-valued NP functions, NPMV and NPSV, were introduced in the seminal paper of Book, Long, and Selman [3]. These functions are structured as follows. Given a nondeterministic Turing machine N having a designated output tape whose running time (i.e., the maximum number of steps of any of its computation paths) is polynomially bounded in the length of its input, on a given input we view each path that rejects as having no output. We view each path that accepts as outputting (a single string, namely) whatever string is between the fixed left-end marker of the machine's output tape and the cell under the output tape head (but not including the content of either of those cells). We view the machine, overall, as computing a potentially partial, potentially multivalued function f , where the outputs of f on input x are precisely the outputs of N 's accepting paths.

NPMV is the class of all partial, multivalued functions computed by nondeterministic polynomial-time machines. NPSV is defined as the set of all NPMV functions f such that $(\forall x \in \Sigma^*)[\|\text{set-}f(x)\| \leq 1]$. $\text{NPMV}_{\text{total}}$ (respectively, $\text{NPSV}_{\text{total}}$) denotes all $f \in \text{NPMV}$ (respectively, NPSV) such that $(\forall x \in \Sigma^*)[\|\text{set-}f(x)\| \geq 1]$. These classes have been studied for many years, though under differing notations ([3], and see also the excellent survey [66]). What in this paper we for clarity denote as $\text{NPMV}_{\text{total}}$ and $\text{NPSV}_{\text{total}}$ are usually referred to simply as NPMV_t and NPSV_t in the literature.

Note that it is completely legal for different paths of an NP machine (modeling an NPMV function) to output the same value. However, since we view $\text{set-}f(x)$ as a set rather than a multiset, a given element is simply a regular member of $\text{set-}f(x)$ regardless of the (nonzero!) number of paths on which it is output. Also, note that it is completely legal for a multivalued function to happen to sometimes or always have only one output value—by multivalued we just mean “allowed to have multiple values”.

We will be concerned with eliminating solutions from NPMV functions.

Definition 2.1. Given partial, multivalued functions f and g , we say that g is a *fair reduction* of f exactly if $(\forall x \in \Sigma^*)[\text{set-}g(x) \subseteq \text{set-}f(x)]$.

Of course, every partial, multivalued function f is a fair reduction of itself, and the always undefined function is a fair reduction of all partial, multivalued functions. The latter fact makes clear why fair reductions are not the “right” notion to study: We want to prune down the number of outputs of multivalued functions—but certainly not to the point of eliminating all solutions. Rather, our dream case is to prune down from multiple solutions to one solution.

To capture the type of reduction we truly wish for, the right notion is not that of fair reduction, but rather is the notion of refinement.

Definition 2.2 ([66]). Given partial, multivalued functions f and g , we say that g is a *refinement* of f , denoted $g \subseteq_c f$, exactly if g is a fair reduction of f and $(\forall x \in \Sigma^*)[\text{set-}g(x) \neq \emptyset \text{ iff } \text{set-}f(x) \neq \emptyset]$.

That is, a refinement removes zero or more values from the output set, but never removes so many as to cross from having some outputs to having no outputs. It is true that each f will trivially be a refinement of itself, but theorems about refinement generally block that case via conditions that ensure that, unless f is NPSV to begin with, solution eliminations will occur (the exact nature of which will vary from theorem statement to theorem statement).

2.2. Fairly reducing NPMV functions by small amounts can be done for free

It is rather unfortunate that the natural goal is to understand refinements rather than to understand fair reductions. The reason it is unfortunate is that eliminating small numbers of solutions turns out to be easy and consequence-free for the case of fair reductions.

Theorem 2.3. *NPMV is closed under fair reduction via proper subtraction of polynomial-time computable (or even $\text{NPSV}_{\text{total}}$ -computable⁵), polynomially value-bounded numbers of solutions. (That is, if $f \in \text{NPMV}$ and $g: \Sigma^* \rightarrow \mathbb{N}$ is a total, polynomial-time computable (or even $\text{NPSV}_{\text{total}}$ -computable) function such that for some polynomial r it holds that $(\forall x \in \Sigma^*)[g(x) \leq r(|x|)]$, then there exists some function $h \in \text{NPMV}$ such that h is a fair reduction of f and, for each $x \in \Sigma^*$, it holds that $\|\text{set-}h(x)\| = \|\text{set-}f(x)\| - g(x)$.)*

Proof. Let N be an NPTM modeling f . Compute $g(x) \in \mathbb{N}$. Since $g(x) \in \text{NPSV}_{\text{total}}$, we do this nondeterministically, and on all paths that compute an output value (and note that all those will compute the same output value) do the following. Nondeterministically guess a $1 + g(x)$ tuple of distinct computation paths of N on input x . If all $1 + g(x)$ paths are accepting paths and have pairwise distinct outputs, then on the current path output the lexicographically largest output. Otherwise, the current path has no output (rejects). It is easy to see that we have just given an NP machine whose outputs on each x are exactly all the elements of $\text{set-}f(x)$ (if any) that are not among the $g(x)$ lexicographically smallest elements of $\text{set-}f(x)$. \square

⁵ Technically, $\text{NPSV}_{\text{total}}$ functions map from Σ^* to a single-valued subset of Σ^* , but via a bit of type coercion and the standard bijection between \mathbb{N} and Σ^* we may view them in this particular setting as a way of computing a mapping from Σ^* to \mathbb{N} .

So, for example, NPMV is closed under fair reduction via proper decrement. We will see in Section 3 that #P lacks that closure unless $NP \subseteq SPP$, and for the case of refinement we will see in Section 2.3 that even refining from (at most) two solutions to (at most) one would collapse the polynomial hierarchy.

There is a class, SpanP [44], that focuses on the number of distinct outputs of NP machines. Note that Theorem 2.3, due to its focus on cardinality reduction, is close to being a theorem about SpanP being closed under proper subtraction of simple, small functions (we have not been able to yet find that theorem in the literature, but it certainly seems a natural theorem that thus might be already known), except Theorem 2.3 is in fact stronger, since Theorem 2.3 is not merely reducing the “span” of an NPMV function, but is even doing so in a way that respects solution names (that is, that employs a fair reduction).

Exactly due to this connection between fair reductions and SpanP, it is certainly true that if NPMV is closed under fair reduction via proper subtraction of polynomial-time computable functions (note that we have removed the limitation that they be small in value), then SpanP is closed under proper subtraction of polynomial-time computable functions. Ogiwara and Hemachandra [57] have shown that the latter closure is completely characterized by the complexity class collapse $NP = C \cdot NP$ (“C.” has its usual literature meaning, namely, application of the counting operator associated with PP) or, equivalently, $NP = PH = C=P = PP = CH$. So, certainly “NPMV is closed under fair reduction via proper subtraction of polynomial-time computable functions” implies that collapse. However, we observe that it is not hard to see that $NP = PH = C=P = PP = CH$ implies “NPMV is closed under fair reduction via proper subtraction of polynomial-time computable functions”. To see this, the crucial thing to note is that in PP^{NP} we can accept the set that (for a fixed machine N modeling an NPMV function, call it f) answers the question (the input to the set is $\langle x, n, y \rangle$) “On input x is it the case that $y \in \text{set-}f(x)$ and there are at least n elements in $\text{set-}f(x)$ that are lexicographically less than y ”. But $PP^{NP} \subseteq CH$, and so the assumption $NP = CH$ makes it easy to see the desired implication (similarly to the proof of Theorem 2.3, we can kill off an appropriate collection of lexicographically smallest output values). So putting together the previous comments, NPMV is closed under fair reduction via proper subtraction of polynomial-time computable functions exactly if $NP = C \cdot NP$. It is also easy to see that this remains true even if the subtracted functions are allowed to be $NPSV_{\text{total}}$ -computable: Our algorithm can simply start off by $NPSV_{\text{total}}$ -computing the number of solutions to remove, and then each path that successfully computes that value proceeds using the above approach.

Theorem 2.4. *The following conditions are equivalent.*

- (1) NPMV is closed under fair reduction via proper subtraction of polynomial-time computable functions.
- (2) NPMV is closed under fair reduction via proper subtraction of $NPSV_{\text{total}}$ -computable functions.
- (3) $NP = C \cdot NP$.

Equivalently, in light of [57] and the above discussion, NPMV is closed under fair reduction via proper subtraction of polynomial-time computable functions (or even $NPSV_{\text{total}}$ -computable functions) exactly if SpanP is closed under proper subtraction of polynomial-time computable functions. [57] provides about a dozen other statements that are equivalent to each of these statements, i.e., that are also characterized by $NP = C \cdot NP$.

Readers interested in the class defined by the closure of SpanP under subtraction (not proper subtraction, but subtraction; so this class will have functions that can take on negative values) are referred to the interesting work of Mahajan, Thierauf, and Vinodchandran [50], which provides for SpanP a “gap”-like analog that parallels the relationship of GapP to #P.

2.3. Refining NPMV functions to unique solutions collapses the polynomial hierarchy, as do many other refinement hypotheses

In this section we survey the known cases in which refining NPMV functions collapses the polynomial hierarchy.

Let $NP2V$ denote all $f \in NPMV$ satisfying $(\forall x \in \Sigma^*)[\|\text{set-}f(x)\| \leq 2]$ [35]. Among the standard classes we will employ are PH (the polynomial hierarchy: $P \cup NP \cup NP^{NP} \cup \dots$) [52,69], ZPP (zero-error probabilistic polynomial time) [24], S_2 (the symmetric version of NP^{NP}) [8,63], and $S_2^{NP \cap coNP}$ (all sets computable via relativizing S_2 with sets from $NP \cap coNP$) (see [6]). For completeness we give the definitions of S_2 and $S_2^{NP \cap coNP}$.

Definition 2.5. (1) ([8,63]) A language L is in S_2 if there exist a polynomial-time computable 3-argument boolean predicate P and a polynomial p such that, for all $x \in \Sigma^*$,

- (a) $x \in L \iff (\exists y \in \Sigma^* : |y| = p(|x|))(\forall z \in \Sigma^* : |z| = p(|x|))[P(x, y, z) = 1]$, and
- (b) $x \notin L \iff (\exists z \in \Sigma^* : |z| = p(|x|))(\forall y \in \Sigma^* : |y| = p(|x|))[P(x, y, z) = 0]$.

(2) ([6]) A language L is in $S_2^{NP \cap coNP}$ if there exist a 3-argument boolean predicate $P \in NP \cap coNP$ and a polynomial p such that, for all $x \in \Sigma^*$,

- (a) $x \in L \iff (\exists y \in \Sigma^* : |y| = p(|x|))(\forall z \in \Sigma^* : |z| = p(|x|))[P(x, y, z) = 1]$, and
- (b) $x \notin L \iff (\exists z \in \Sigma^* : |z| = p(|x|))(\forall y \in \Sigma^* : |y| = p(|x|))[P(x, y, z) = 0]$.

However, for the purposes of this paper, the only thing important to remember is that

$$NP \cup coNP \subseteq S_2 \subseteq S_2^{NP \cap coNP} \subseteq ZPP^{NP} \subseteq NP^{NP} \cap coNP^{NP}.$$

Also, we will need to draw on the notion of Karp–Lipton advice classes, in particular $(NP \cap coNP)/\text{poly}$.

Definition 2.6 (Instantiating [42] to the case of $(NP \cap coNP)/\text{poly}$). $(NP \cap coNP)/\text{poly}$ denotes each set L such that there exist a set $A \in NP \cap coNP$ and a polynomially length-bounded function $f: \Sigma^* \rightarrow \Sigma^*$ such that, $(\forall x \in \Sigma^*)[x \in L \iff \langle x, f(0^{|x|}) \rangle \in A]$.

Informally put, there is an “advice-interpreter” set in $\text{NP} \cap \text{coNP}$ (this will soon come back to haunt us!) that with the right advice (which is short and depends only on the length of x) accepts exactly A . ($\text{NP} \cap \text{coNP}$)/quadratic is analogous to [Definition 2.6](#) except with f required to be quadratically length-bounded.

We will draw on the following recent result, which we state without proof.

Theorem 2.7 ([6]). $\text{NP} \subseteq (\text{NP} \cap \text{coNP})/\text{poly} \implies \text{S}_2^{\text{NP} \cap \text{coNP}} = \text{PH}$.

Finally, we will draw on two more items. The first is the notion of NPSV-selectivity, and the second is a relatively easy lemma linking refinement to NPSV-selectivity.

Definition 2.8 ([35]). A set L is said to be NPSV-selective if there is a function $f \in \text{NPSV}$ such that

- (a) $(\forall x, y \in \Sigma^*)[\text{set-}f(x, y) \subseteq \{x, y\}]$ and
- (b) $(\forall x, y \in \Sigma^*)[\{x, y\} \cap L \neq \emptyset \implies (\text{set-}f(x, y) \neq \emptyset \wedge \text{set-}f(x, y) \subseteq A)]$.

Lemma 2.9 ([35]). *The following are equivalent.*

- (1) All NPMV functions have NPSV refinements.
- (2) All NP2V functions have NPSV refinements.
- (3) All NP sets are NPSV-selective.

Proof. (1) \implies (2) is immediate. Regarding (2) \implies (3), note that for each NP set L there clearly is an NP2V function f_L such that $(\forall x, y \in \Sigma^*)[\text{set-}f_L(x, y) = L \cap \{x, y\}]$. Note that if f_L has an NPSV refinement, that refinement proves under [Definition 2.8](#) that L is NPSV-selective. So (2) \implies (3). Regarding (3) \implies (1), let f be an NPMV function computed by NPTM N . Consider the set $A = \{\langle x, s \rangle \mid x \in \Sigma^* \text{ and } b \in \{0, 1\}^*\}$ and there is some accepting path of N on input x whose nondeterministic guess sequence has s as a prefix. Note that $A \in \text{NP}$. But if A is NPSV-selective, via NPSV function g , we can (deterministically) check whether $N(x)$ has no nondeterministic guesses and if so we are done and we output the corresponding output if any, and otherwise we run $g(\langle x, 0 \rangle, \langle x, 1 \rangle)$ and every accepting path of that checks whether the guess sequence it found is the full guess sequence for that path, and if so outputs the corresponding output if any, and otherwise continues the self-reduction process one more level (i.e., on a path that found that $\langle x, 0 \rangle$ is output by $g(x)$, move on to simulating $g(\langle x, 00 \rangle, \langle x, 01 \rangle)$). We always, in calling g , order its arguments $g(\langle x, \alpha \rangle, \langle x, \beta \rangle)$ when $\alpha \leq_{\text{lex}} \beta$. Note that the just-described process creates an NPSV refinement of f , so (3) \implies (1). \square

We finally must state the key link between refinement and hierarchy collapse.

Theorem 2.10 ([35]). $\text{NPSV-selective} \cap \text{NP} \subseteq (\text{NP} \cap \text{coNP})/\text{poly}$.

We can now show that unique solutions collapse the polynomial hierarchy. The following result is due to [35], except it is stated here with the stronger conclusion that [Theorem 2.7](#) (of [6]) gave it.

Theorem 2.11 ([35] in light of [6]). *If all NPMV functions (or even all NP2V functions) have NPSV refinements, then $\text{S}_2^{\text{NP} \cap \text{coNP}} = \text{PH}$.*

Proof (Theorem 2.11). If all NPMV functions have NPSV refinements (or even all NP2V functions do) then by [Lemma 2.9](#) all NP sets are NPSV-selective. So by [Theorem 2.10](#), $\text{NP} \subseteq (\text{NP} \cap \text{coNP})/\text{poly}$. So by [Theorem 2.7](#), $\text{S}_2^{\text{NP} \cap \text{coNP}} = \text{PH}$. \square

We are done, except for the proof of [Theorem 2.10](#). We do not formally prove that, but rather we will give a high-level exposition of the proof. (For a formal proof, see [35].) Suppose we are given an NP set L that is NPSV-selective via NPSV function f . Without the loss of generality, f is symmetric (i.e., $(\forall x, y \in \Sigma^*)[f(x, y) = f(y, x)]$)—otherwise replace f with $f'(a, b) = f(\min(a, b), \max(a, b))$, which is symmetric and which can easily be seen to be an NPSV-selector for L since f is an NPSV-selector for L . Consider for some arbitrary $n \in \mathbb{N}$ the set $L_n = L \cap \Sigma^n$. Imagine each element of L_n being a node in a tournament such that, for $a, b \in L_n$, $a \neq b$, there is an edge from a to b if and only if $b \in \text{set-}f(a, b)$. By a standard divide-and-conquer argument, originally used by Ko [43] in the related context of showing that $P\text{-selective} \subseteq P/\text{poly}$, it is easy to see (the first step is to eliminate half the graph by choosing as part of S_n some node that points to at least half the other nodes—by counting, some such node must exist) that there will be a subset, S_n , of L_n of cardinality at most $\lceil \log_2(\|L_n\| + 1) \rceil$ such that $L_n = \{y \in \Sigma^n \mid (\exists a \in S_n)[y \in \text{set-}f(a, y)]\}$. Since $\|L_n\| \leq 2^n$ and each string in S_n has n bits, clearly S_n can be coded using $O(n^2)$ bits.

It would be wonderful to declare victory now, via claiming that we have an $(\text{NP} \cap \text{coNP})/\text{quadratic}$ attack that on length n inputs uses S_n as the advice and sees whether an input x , $|x| = n$, beats at least one element of S_n . This at first would seem to work perfectly, but in fact there is a subtle yet severe problem.

That problem can be seen as follows. Imagine the allegedly “ $\text{NP} \cap \text{coNP}$ ” set of $(\text{NP} \cap \text{coNP})/\text{quadratic}$ that on input $\langle x, B \rangle$ takes the advice $B = \{a_1, a_2, \dots, a_k\}$ (allegedly $B = S_{|x|}$) and runs $f(x, a_1)$ and on each path that accepts (and thus chooses x or a_1) runs $f(x, a_2)$ and on each path that accepts... etc., etc. At the end of this sequence, assuming $B = S_{|x|}$, each path that found an accepting path for each of the k applications of f definitively knows that $x \in L$ (namely, if at least one of the k applications of f had x as an output) or knows that $x \notin L$ (namely, if none of the k applications yielded x as the output). The problem is the innocent-looking “assuming $B = S_{|x|}$!” Everything we just described works perfectly if $B = S_{|x|}$, i.e., if we are given the correct advice. But the definition of $(\text{NP} \cap \text{coNP})/\text{poly}$ requires the advice interpreter to be an $\text{NP} \cap \text{coNP}$ set, and since the alleged advice is part of that set’s input, that means we must be “ $\text{NP} \cap \text{coNP}$ ”-like *even when given lies for advice*. Informally put, “ $\text{NP} \cap \text{coNP}$ ”-like means having a machine that on each input has at least one path that accepts or rejects, each accepting or rejecting path must be correct, and paths also are allowed to neither accept nor reject. (The technical term for such “ $\text{NP} \cap \text{coNP}$ machines” is “strong” computation (see [49,65]).) However, our selector f is NPSV, not necessarily $\text{NPSV}_{\text{total}}$! If $B = S_{|x|}$ then all the a_i ’s are members of L , and so each $f(x, a_i)$ is defined since $a_i \in L \implies \text{set-}f(x, a_i) \neq \emptyset$ (by the definition of NPSV-selectivity). However, if B is untrue advice and contains some a_i that does not belong to L and $x \notin L$, then $f(x, a_i)$ may be undefined and our computation is not “ $\text{NP} \cap \text{coNP}$ ”-like and the proof is in shambles.

The fix is to trade advice size for robustness. Instead of quadratic-sized advice, $S_{|x|}$, let us instead require the advice to be $S_{|x|}$ plus, for each $a_i \in S_{|x|}$, a proof that $a_i \in L$. Since $L \in \text{NP}$, such proofs exist. Now, our “ $\text{NP} \cap \text{coNP}$ ”-like attack is home free. Given the true advice, it does the right thing, as described above. Given a giant lie – a bunch of a_i 's not all accompanied by valid membership proofs – it will detect that it is being lied to, and will reject. And, most interestingly, given a subtle lie – an advice set $B' = \{a_1, a_2, \dots, a_k\}$ whose a_i 's are all length $|x|$ strings and that all are accompanied by valid membership proofs in L but such that B' does not happen to have the property (which $S_{|x|}$ crucially does have) that $L_n = \{y \in \Sigma^n \mid (\exists a \in B)[y \in \text{set-}f(a, y)]\}$ – we will fail to detect that we are being lied to, but nonetheless will act in an “ $\text{NP} \cap \text{coNP}$ ”-like fashion, since the fact that each a_i belongs to L ensures that each application of $f(x, a_i)$ has an output. In brief, by going from quadratic to polynomial advice (the polynomial depends on the certificate size of $L \in \text{NP}$), we made our advice interpreter robust enough to weather lies.

This completes the proof sketch for [Theorem 2.10](#).

We conclude this section by briefly listing results more recent than [Theorem 2.11](#) that show additional cases where refining solutions collapses the polynomial hierarchy. To do so, we will use the notation that for each $A \subseteq \mathbb{N}^+$, $\text{NP}_A \text{V}$ denotes all NPMV functions f such that $(\forall x \in \Sigma^*)[\|\text{set-}f(x)\| \in A \cup \{0\}]$ [[37](#)]. For example, $\text{NPMV} = \text{NP}_{\mathbb{N}^+} \text{V}$, $\text{NPSV} = \text{NP}_{\{1\}} \text{V}$, and $\text{NP2V} = \text{NP}_{\{1,2\}} \text{V}$. So [Theorem 2.11](#) says $\text{NP}_{\{1,2\}} \text{V} \subseteq_c \text{NP}_{\{1\}} \text{V} \implies \text{S}_2^{\text{NP} \cap \text{coNP}} = \text{PH}$. The following results give other hypotheses sufficient to collapse the hierarchy, but unfortunately they collapse it to a level substantially higher than $\text{S}_2^{\text{NP} \cap \text{coNP}}$. The reason is that no analog of [Theorem 2.10](#) is known for these cases, and so the proofs use weaker techniques such as direct quantifier exchange.

Theorem 2.12 ([[54](#)]). Let $k \in \mathbb{N}^+$. If $\text{NP}_{\{1,2,\dots,k+1\}} \text{V} \subseteq_c \text{NP}_{\{1,2,\dots,k\}} \text{V}$, then $\text{NP}^{\text{NP}} = \text{PH}$.

Theorem 2.13 ([[55](#)]). Let $0 < \gamma < 1$. If $\text{NP}_{\mathbb{N}^+} \text{V} \subseteq_c \text{NP}_{\{1,2,\dots,\lfloor \max(1, n^\gamma) \rfloor\}} \text{V}$ (here n is the length of the input), then $\text{NP}^{\text{NP}} = \text{PH}$.

Buhrman, Kadin, and Thierauf [[5](#)] asked whether the work of Hemaspaandra et al. [[35](#)] could be extended beyond NPSV to even handle the case of limited Turing access to NPSV, and they successfully explored the case of polynomial-time Turing reductions allowed at most one query to NPSV, achieving an $\text{NP}^{\text{NP}} = \text{PH}$ consequence (and as noted in [[6](#)], their clever trick can be used in conjunction with the work of [[6](#)] to obtain even a $\text{S}_2^{\text{NP} \cap \text{coNP}} = \text{PH}$ conclusion). As shown by Ogihara [[55](#)], Ogihara's [Theorem 2.13](#) can be used to achieve an $\text{NP}^{\text{NP}} = \text{PH}$ consequence even for the case of polynomial-time Turing reductions allowed at most $(1 - \epsilon) \log(n)$ queries to NPSV (and, closely related to some of the comments above, it remains open whether that result can be extended to a $\text{S}_2^{\text{NP} \cap \text{coNP}} = \text{PH}$ conclusion).

Even more recently, the following flexible but complex cases have been established (note that [Theorem 2.15](#) implies [Theorem 2.12](#)).

Theorem 2.14 ([[37](#)]). Let $A, B \subseteq \mathbb{N}^+$ be nonempty. Suppose there exist four integers $c > 0$, $d > 0$, $e \geq 0$, and $\delta \geq 0$ satisfying the following conditions:

- (1) $d \leq c \leq 2d$ and $\delta < 2d - c$,
- (2) $c, 2d + e \in A$,
- (3) $c - \delta \leq \min\{i \mid i \in B\} \leq c$, and
- (4) $2d - (2\delta + 1) \geq \max\{i \in B \mid i \leq 2d + e\}$.

Then $\text{NP}_A \text{V} \subseteq_c \text{NP}_B \text{V}$ implies $\text{NP}^{\text{NP}} = \text{PH}$.

Theorem 2.15 ([[37](#)]). Let $k \geq 2$ and $d, 1 \leq d \leq k - 1$, be integers. Let $A, B \subseteq \mathbb{N}^+$ be such that $\binom{k-1}{k-d} \in A$, $\binom{k}{k-d} \in A$, and $\max\{i \mid i \in B \text{ and } i \leq \binom{k}{k-d}\} \leq \lceil \frac{k}{d} \rceil - 1$. Then $\text{NP}_A \text{V} \subseteq_c \text{NP}_B \text{V}$ implies $\text{NP}^{\text{NP}} = \text{PH}$.

We refer the reader to [[37](#)] for a wide variety of corollaries that follow from [Theorems 2.14](#) and [2.15](#), for proofs of these results, and for a discussion of how an even more general “lowness” result can unify and extend further these claims.

This section focused on cases where refining solutions implies hierarchy collapses. For completeness, we mention that there is a relatively general result known showing that in many cases one can nontrivially refine functions. [Theorem 2.16](#) is proven by tuple trickery reminiscent in flavor to that used in the proof of [Theorem 2.3](#).

Theorem 2.16 ([[37](#)]). Let $A \subseteq \mathbb{N}^+$ and $B \subseteq \mathbb{N}^+$ be finite sets such that $A = \{a_1, \dots, a_m\}$ with $a_1 < a_2 < \dots < a_m$. If $\|A\| = 0$ or $(\exists b_1, \dots, b_m : 0 < b_1 < \dots < b_m) \{b_1, \dots, b_m\} \subseteq B$ and $a_1 - b_1 \geq \dots \geq a_m - b_m \geq 0$, then $\text{NP}_A \text{V} \subseteq_c \text{NP}_B \text{V}$.

For example, [Theorem 2.16](#) yields $\text{NP}_{\{10,20,100\}} \text{V} \subseteq \text{NP}_{\{5,12,92\}} \text{V}$.

The most glaring open issues remaining are filling in the holes in our knowledge of which cases imply collapses of the polynomial hierarchy, and determining how dramatic a collapse is implied in each case. Regarding the first open issue, though for some cases (see [Theorem 2.11](#) through [2.15](#)) collapses of the polynomial hierarchy have been shown to follow from a certain refinement, and for some cases (see [Theorem 2.16](#)) it has been shown that a certain refinement truly does hold (and we mention in passing that [Theorem 2.16](#)'s proof applies not just in the real world but also in all relativized worlds), those theorems do not handle all cases. Can one extend the results just mentioned to the point of having a simple, transparent test, regarding A and B , such that, for all A and B satisfying the test, $\text{NP}_A \text{V} \subseteq_c \text{NP}_B \text{V}$ implies the collapse of the polynomial hierarchy, and such that, for all A and B failing to satisfy the test, $\text{NP}_A \text{V} \subseteq_c \text{NP}_B \text{V}$ holds in the real world and all relativized words? That is, can one obtain a clean, simple dichotomy theorem? Regarding the second open issue, the extent of collapses, it would be good to learn exactly what the extent is of the conditions on A and B that allow one to show that $\text{NP}_A \text{V} \subseteq_c \text{NP}_B \text{V}$ implies the collapse of the polynomial hierarchy to $\text{S}_2^{\text{NP} \cap \text{coNP}}$. For example, can the conclusions of [Theorems 2.14](#) and [2.15](#) be strengthened to yield that collapse? If so, can they be strengthened to yield a collapse to S_2 or even some class smaller than that?

3. Reducing #P functions

The goal of this section is to survey known results regarding reducing the number of solutions of #P functions. For example, given two #P functions f and g , is the function $h(x) =$

$f(x) \dot{-} g(x)$ always a #P function? Alternatively, can we show some unlikely complexity-theoretic consequence that would follow were #P to be closed under proper subtraction? We will see that the latter seems to be the case. It is particularly interesting to look at reducing the number of solutions of #P functions as opposed to simply looking at #P's closure properties because it appears that it is this controlled value reduction that makes it unlikely for #P to be closed under proper subtraction, integer division, proper decrement, and minimum. Loosely put, the results surveyed in this section suggest that given any one of a wide variety of natural value-decreasing operations, #P is unlikely to be closed under the operation.⁶ In this section we attempt to justify this loosely put intuition. (We mention that this paper's views on #P closure properties are greatly influenced by the papers of Gupta [30,29].) The class #P was defined by Valiant [73]. A function f is in #P if there exists a nondeterministic Turing machine N such that, for each string x , $f(x)$ equals the number of accepting paths of N on input x . Some typical examples of #P functions include the function that given a boolean formula ϕ returns the number of satisfying assignments of ϕ , and the function that given a graph and a positive integer k returns the number of distinct k -colorings of that graph.

We say that f is a *closure property* if there exists a positive integer i such that f is a function from \mathbb{N}^i to \mathbb{N} [57]. Within this paper we will mostly be interested in the cases $i = 2$ (e.g., proper subtraction, integer division, 2-ary minimum) and $i = 1$ (e.g., proper decrement, integer division by two). One may consider this framework to apply also to functions that take as arguments not a tuple of natural numbers but rather a tuple of strings, in such cases implicitly invoking the standard bijection between \mathbb{N} and Σ^* . In fact, we assume that coercions in either direction between \mathbb{N} and Σ^* are done implicitly via such a bijection whenever from the context it is clear that such coercing is needed, even if we do not explicitly mention it. f is said to be a *P-closure property* if f is a closure property and f is computable in polynomial time.

Definition 3.1 ([57]). Let $f: \mathbb{N}^i \rightarrow \mathbb{N}$ be a closure property. We will say that #P is *closed under f* – or, equivalently, will say that f is a *closure property of #P* – if it holds that $(\forall g_1, g_2, \dots, g_i \in \#P) [f(g_1(x), g_2(x), \dots, g_i(x)) \in \#P]$.

Note that #P has some very natural closure properties. For example, #P is closed under addition and multiplication [61]. Let f and g be two #P functions and let N_f and N_g be two nondeterministic Turing machines via which f and g are defined. #P is closed under addition because we can construct a nondeterministic Turing machine N_{f+g} that on input x nondeterministically chooses one of $N_f(x)$ and $N_g(x)$ to simulate and then nondeterministically performs that simulation. Closure under multiplication follows from the fact that on input x we can first simulate $N_f(x)$ and then

⁶ For each such operation we of course need a formal result stating that, e.g., if #P has this closure property then an unlikely complexity class collapse follows. And we mention (as an unnatural, extreme example, but one worth noting) that proper subtraction of any sufficiently large function from a given #P function in fact *does* yield a #P function, since it yields the constant function whose value is zero.

on every accepting path we can simulate $N_g(x)$. However, in this paper we are interested in those closure properties that have the potential to *decrease* values. Such closure properties include proper subtraction, proper decrement, integer division, minimum, etc. Note that we have to use *proper* subtraction/decrement and *integer* division since the codomain of #P functions is \mathbb{N} .

The nature of reducing the number of solutions of #P functions is quite different from the nature of reducing the number of outputs of NPMV functions. For example, if f were an NPSV refinement of some NPMV function g , it would be totally legal for f to have, on some (or even all) inputs, more accepting paths than g , provided that for a given input each of the accepting paths would output the same value. The same comment applies to fair reductions of solutions. In particular, by [Theorem 2.3](#) we know that NPMV is closed under fair reductions via proper decrement. However, as we will see, if #P is closed under proper decrement, then an unlikely complexity class collapse occurs.

The study of the complexity of closure properties of #P (and some other function classes that we will not discuss here—SpanP, MidP, and OptP) was initiated by Ogiwara and Hemachandra [57]. One of the most important contributions of their work was introducing the concept of a “hard” P-closure property for a given function class and showing that #P does indeed have such “hard” P-closure properties. Let $f: \mathbb{N}^i \rightarrow \mathbb{N}$ be a closure property. We will say that f is a *#P-hard P-closure property* [57] if f is a P-closure property and the following implication is true: If #P is closed under f then #P is closed under all polynomial-time computable closure properties.

Interestingly enough, #P is closed under every polynomial-time computable closure property if and only if #P is closed under every closure property f such that $f \in \#P$.⁷ While at first this might seem surprising, it in fact will be established by [Theorem 3.2](#), which itself has a relatively simple proof.

The following theorem shows that some simple operations that decrease the number of solutions of #P functions are in fact “the least likely” to be closure properties of #P among closure properties that plausibly could be closure properties of #P. One can think of this as a loose parallel to the framework of NP-completeness theory, which shows that certain NP sets – SAT, CLIQUE, etc. – are “the least likely” NP sets to belong to P.

Theorem 3.2 ([57]). *The following statements are equivalent.*

- (1) #P is closed under proper subtraction.
- (2) #P is closed under integer division.⁸
- (3) #P is closed under every nonnegative polynomial-time computable function (i.e., #P has every polynomial-time computable closure property).
- (4) #P is closed under every #P-computable function (i.e., #P has every #P-computable closure property).

⁷ Note the difference between f being a closure property of #P and $f \in \#P$ being a closure property. In the former case #P is closed under f and in the latter f is simply stated to be a #P function of type $\mathbb{N}^i \rightarrow \mathbb{N}$, for some i .

⁸ Note that for integer division, to avoid the issue of dividing by zero one would have to slightly adjust the notion of being closed under an operation. We will return to this issue later.

(5) $UP = PP$.

So proper subtraction and integer division are $\#P$ -hard P -closure properties. See [57] for additional $\#P$ -hard P -closure properties.

We will give a fairly detailed overview of the proof of Theorem 3.2 (see also [57,36]). Clearly, (4) implies (3), and (3) implies both (1) and (2). We will argue in some detail that (1) implies (5), and then we will give overviews of the proofs that (2) \implies (5) and that (5) \implies (4).

Let us show that if $\#P$ is closed under proper subtraction then $UP = PP$. Assume that $\#P$ is closed under proper subtraction. First we will show that $coNP \subseteq UP$ and then we will show that $PP \subseteq NP$. Since PP is closed under complementation, the latter implies that $PP \subseteq coNP$. Thus, we have $PP \subseteq coNP \subseteq UP$, and since $UP \subseteq PP$ holds without assumption, we have $UP = PP$.

Recall that we have assumed that $\#P$ is closed under proper subtraction. Let L be some arbitrary $coNP$ language. By the definition of $coNP$, there exists a nondeterministic polynomial-time Turing machine N_f such that if $x \in L$ then $N_f(x)$ has no accepting paths, and if $x \notin L$ then $N_f(x)$ has at least one accepting path. (In other words, \bar{L} is an NP language.) Let f be the $\#P$ function that N_f implicitly defines. Since $\#P$ is closed under proper subtraction, $g(x) = 1 - f(x)$ is a $\#P$ function, and so there exists a nondeterministic Turing machine N_g that on input x has exactly $g(x)$ accepting paths. However, it is easy to see that

$$g(x) = \begin{cases} 1 & \text{if } x \in L, \\ 0 & \text{if } x \notin L. \end{cases}$$

Thus, N_g is a nondeterministic Turing machine that establishes $L \in UP$. (We use the fact that UP can be defined via unambiguous nondeterministic polynomial-time machines.) So $coNP \subseteq UP$. (We mention in passing an alternate attack. Since below we will prove $PP \subseteq UP$ under our hypothesis, instead of proving $coNP \subseteq UP$ as we just did it would suffice to prove the weaker statement $UP = NP$, and that can be briskly seen, under our hypothesis, as follows: Given any NP machine N_f and its associated $\#P$ function f , note that if $\#P$ is closed under proper subtraction then $f - (f - 1)$ must belong to $\#P$, but this shows that $L(N_f)$ belongs to UP .)

Now, still under the assumption that $\#P$ is closed under proper subtraction, let us show that $PP \subseteq NP$. Let L be a PP language defined via a polynomial-time binary predicate R and a polynomial q such that for all n , $q(n) \geq 1$ and, for each $x \in \Sigma^*$,

$$x \in L \iff \|\{y \in \Sigma^* \mid |y| = q(|x|) \wedge R(x, y)\}\| \geq 2^{q(|x|)-1}.$$

Naturally, there exists a nondeterministic polynomial-time Turing machine N_f that on input x has exactly $\|\{y \in \Sigma^* \mid |y| = q(|x|) \wedge R(x, y)\}\|$ accepting paths. For example, on input x , this machine can guess a string y of length $q(|x|)$ and then will accept on the current path exactly if $R(x, y)$ holds. Let f be the $\#P$ function implicitly defined by N_f , and let us consider the function $g(x) = f(x) - (2^{q(|x|)-1} - 1)$. Since $2^{q(|x|)-1} - 1$ is clearly a $\#P$ function and we assumed that $\#P$ is closed under proper subtraction, g is a $\#P$ function as well. Thus, there is a nondeterministic polynomial-time Turing machine N_g that on input x has exactly $g(x)$ accepting computation

paths. However, it is easy to see that if $x \in L$ then $g(x) \geq 1$ (as then $f(x) \geq 2^{q(|x|)-1}$) and that if $x \notin L$ then $g(x) = 0$ (as then $f(x) < 2^{q(|x|)-1}$). Thus, N_g establishes $L \in NP$. Since L was chosen arbitrarily, we have $PP \subseteq NP$. This concludes the proof that if $\#P$ is closed under proper subtraction then $UP = PP$.

We will briefly argue as to why (5) implies (4). We assume that $UP = PP$ and we will show, separately for each fixed $i \in \mathbb{N}^+$, that given a $\#P$ function f , $f : \mathbb{N}^i \rightarrow \mathbb{N}$, and a sequence of i $\#P$ functions g_1, g_2, \dots, g_i it holds that $h(x) = f(g_1(x), g_2(x), \dots, g_i(x))$ is a $\#P$ function. Fix an arbitrary $i \in \mathbb{N}^+$. The idea behind the proof is to somehow get a handle on the values $g_1(x), g_2(x), \dots, g_i(x)$ and then to directly run f on those values.

As an aside, fix a nice multi-arity pairing function $\langle \dots \rangle$ (see, e.g., [31]). This function will of course take as arguments tuples of strings (but in fact, it is legal for, as is the case in the definition of L_j for example, some components to be natural numbers, in which case the standard bijection between \mathbb{N} and Σ^* will be implicitly applied) and will output a single string. Returning to our proof, how can we obtain the values $g_1(x), g_2(x), \dots, g_i(x)$? First, we observe that for each j , $1 \leq j \leq i$, the language $L_j = \{\langle x, y \rangle \mid g_j(x) \geq y\}$ is in PP . PP is closed under truth-table reductions [22]. So – though this actually requires the closure of PP just under bounded-truth-table reductions since i is fixed – the language

$$L = \{\langle x, y_1, y_2, \dots, y_i \rangle \mid (\forall j \in \{1, 2, \dots, i\})$$

$$\langle x, y_j \rangle \in L_j \wedge \langle x, y_j + 1 \rangle \notin L_j\}$$

is in PP as well. (We can decide L using $2i$ queries to languages L_1, L_2, \dots, L_i – two queries to each L_j , $1 \leq j \leq i$ – each of which can be translated into a query to some selected PP -complete language.) Clearly, we have that $\langle x, y_1, \dots, y_i \rangle \in L$ if and only if for all j , $1 \leq j \leq i$, it holds that $g_j(x) = y_j$.

Now, since $UP = PP$, we have that $L \in UP$ and so there is an unambiguous polynomial-time nondeterministic Turing machine N that accepts L . Given this machine, we are ready to show that $\#P$ is closed under f . It is enough to construct a nondeterministic Turing machine M that on input x on each of its computation paths guesses a sequence of i strings z_1, z_2, \dots, z_i (each z_j , $1 \leq j \leq i$, of appropriately polynomially bounded length) and simulates N on input $\langle x, z_1, \dots, z_i \rangle$. By the nature of L and the fact that N is unambiguous, there is only one path of M that reaches N 's acceptance, and on that path we have complete information about the values $g_1(x), g_2(x), \dots, g_i(x)$; it remains to simply run the nondeterministic polynomial-time Turing machine via which f is defined with these values as input. This concludes the proof of the implication (5) \implies (4).

Finally, we will quickly argue that if $\#P$ is closed under integer division then $UP = NP$. Note that in case of integer division we cannot really use our definition of what it means to be closed under an operation. If f and g are two $\#P$ functions then $\lfloor \frac{f(x)}{g(x)} \rfloor$ is undefined when $g(x) = 0$. Thus, we say that $\#P$ is closed under integer division if for every two $\#P$ functions f and g such that $(\forall x \in \Sigma^*) [g(x) > 0]$ the function $h(x) = \lfloor \frac{f(x)}{g(x)} \rfloor$ belongs to $\#P$.

Let L be a PP language. The key point here is that if $L \in PP$ then one can (with a bit of easy argumentation, which we will leave as an exercise, regarding item (3) below) claim that there is a positive integer k and a machine M that witnesses the membership of L in PP that has the following properties:

- (1) On input x , M has exactly $2^{|x|^k}$ computation paths, each containing exactly $|x|^k$ binary nondeterministic choices.
- (2) If on input x M has at least $2^{|x|^k-1}$ accepting paths then $x \in L$.
- (3) On each input x , M has at most $2^{|x|^k} - 1$ accepting paths.

Let f be the #P function defined by M . If #P is closed under integer division then $h(x) = \left\lfloor \frac{f(x)}{2^{|x|^k-1}} \right\rfloor$ is a #P function itself. However, we have that $h(x) = 1$ if $x \in L$ and $h(x) = 0$ otherwise. (We needed item (3) above to make sure that $h(x)$ is never greater than 1.) Clearly, the nondeterministic polynomial-time machine that defines h is a UP machine that accepts L . This concludes our proof sketch of [Theorem 3.2](#).

Note that [Theorem 3.2](#) shows that if properly subtracting a #P function from a #P function always yields a #P function, then #P is closed under every #P-computable function. Can the hypothesis here be weakened? What if we instead assume just that properly subtracting a nonnegative polynomial-time computable function from a #P function always yields a #P function? Or what if we instead assume just that properly subtracting a #P function from a nonnegative polynomial-time computable function always yields a #P function? Are these two weaker assumptions still powerful enough to cause #P to be closed under every #P-computable function? The answer is that they both are indeed strong enough to yield that. That is, it turns out that each one of them is just as demanding an assumption as the seemingly stronger assumption that properly subtracting a #P function from a #P function always yields a #P function: All three of these assumptions stand or fall based on the same complexity-class equality. Of the two extended claims one is already known, and we state it without proof ([Theorem 3.3](#)). The other claim is new to this paper, and we prove it below as [Theorem 3.4](#).

Theorem 3.3 ([57]). UP = PP if and only if for every #P function f and every polynomial-time computable function g it holds that $h(x) = f(x) \dot{-} g(x)$ is a #P function.

Theorem 3.4. UP = PP if and only if for every polynomial-time computable function f and every #P function g it holds that $h(x) = f(x) \dot{-} g(x)$ is a #P function.

To prove [Theorem 3.4](#), we will need the following two facts regarding UP, PP, and C=P.

Proposition 3.5. (1) [57, p. 304] UP = C=P if and only if UP = PP.

(2) [68] $L \in C=P$ if and only if there exist a polynomial q and a polynomial-time computable binary predicate R such that, for each $x \in \Sigma^*$,

- (a) $x \in L \implies \|\{y \in \Sigma^* \mid |y| = q(|x|) \wedge R(x, y)\}\| = 2^{q(|x|)-2}$, and
- (b) $x \notin L \implies \|\{y \in \Sigma^* \mid |y| = q(|x|) \wedge R(x, y)\}\| < 2^{q(|x|)-2}$.

Proof (Theorem 3.4). The “only if” direction follows immediately from [Theorem 3.2](#). We now will prove the “if” direction by showing that UP = C=P follows if one assumes that for every nonnegative polynomial-time computable function f and every #P function g it holds that $h(x) = f(x) \dot{-} g(x)$ is a #P function. Proving this suffices, since by [Proposition 3.5](#) we know that UP = C=P is equivalent to UP = PP.

Let L be an arbitrary C=P language. Let R be the polynomial-time predicate and q be the polynomial whose existence is guaranteed by [Proposition 3.5](#). Clearly there exists an NP machine M that, on arbitrary input x , guesses a string y from $\Sigma^{q(|x|)}$ and accepts exactly if $R(x, y)$ holds. And so there is a #P function f – namely the function defined by the number of accepting paths of M – such that

- (1) if $x \in L$ then $f(x) = 2^{q(|x|)-2}$, and
- (2) if $x \notin L$ then $f(x) < 2^{q(|x|)-2}$.

By the “if” direction’s assumption that properly subtracting #P functions from nonnegative polynomial-time computable functions only yields #P functions, we have that $g(x) = 2^{q(|x|)-2} \dot{-} f(x)$ is a #P function. Note that $g(x) = 0$ if $x \in L$ and $g(x) > 0$ if $x \notin L$. Since g is a #P function, by a second application of the “if” direction’s assumption we have that $h(x) = 1 \dot{-} g(x)$ is a #P function. However, by these constructions,

$$h(x) = \begin{cases} 1 & \text{if } x \in L, \\ 0 & \text{if } x \notin L. \end{cases}$$

So each NP machine that instantiates $h \in \#P$ is in fact a UP machine that accepts L . \square

[Theorem 3.2](#) shows that there are #P-hard P-closure properties. It is interesting to see that these hard closure properties are in fact very simple solution-reducing operations. As mentioned earlier, #P-hard P-closure properties can be viewed as analogs of complete languages, e.g., as analogs of NP-complete languages. On the other hand the closure properties that #P actually has (e.g., addition and multiplication), can be viewed as analogs of languages in P. By Ladner’s [Theorem \[47\]](#) we know that if $P \neq NP$ then there are also intermediate NP languages: Languages that are in NP–P but that are not NP-complete. Interestingly, there seem to be analogs of those in the world of closure properties as well. Some candidates for intermediate closure properties are proper decrement, minimum, and integer division by two. However, much as in the case of the graph isomorphism problem (which is suspected, but not known, to be NP-intermediate), we do not have a proof that these P-closure properties are either #P-hard or feasible, but rather have some pieces of evidence that suggest that that may be the case.

It would be nice to be able to prove that [Theorem 3.2](#) in fact says that every closure property that isn’t obviously not a closure property of #P becomes a closure property of #P if proper subtraction is a closure property of #P. The following statement, considered hand-in-hand with [Theorem 3.2](#), makes it clear that that is indeed the case.

Theorem 3.6. Let f be a closure property. If $f \notin \#P$, then #P is not closed under f .

Proof. Let f be a closure property such that $f \notin \#P$. Since f is a closure property, for some $i \geq 0$ (and in fact, $i = 0$ is impossible if $f \notin \#P$, so let us suppose $i \geq 1$) f maps from \mathbb{N}^i to \mathbb{N} . Fix any nice arity- i pairing function, $\langle \cdot \cdot \rangle$. For each $1 \leq j \leq i$, let $g_j(\langle n_1, n_2, \dots, n_i \rangle) = n_j$. Note that each g_j is a #P function. However, suppose that #P is closed under f . Then $f(g_1(\langle n_1, n_2, \dots, n_i \rangle), g_2(\langle n_1, n_2, \dots, n_i \rangle), \dots, g_i(\langle n_1, n_2, \dots, n_i \rangle))$ belongs to #P. However, $f(g_1(\langle n_1, n_2, \dots, n_i \rangle), g_2(\langle n_1, n_2, \dots, n_i \rangle), \dots, g_i(\langle n_1, n_2, \dots, n_i \rangle))$ clearly equals $f(n_1, n_2, \dots, n_i)$, and so $f(n_1, n_2, \dots, n_i) \in \#P$, a contradiction, so our supposition that #P is closed under f must be wrong. \square

As we saw in [Theorem 3.2](#), whether #P is closed under its #P-hard P-closure properties is fully characterized in terms of complexity class collapses by $UP = PP$. In the case of proper decrement, integer division by two, and minimum we do not have such complete characterizations.

Theorem 3.7. (1) (Due to Torán, as Noted in [57].) If #P is closed under proper decrement, then $NP \subseteq SPP$.

(2) ([57]) If $UP = NP$, then #P is closed under proper decrement.

(3) ([57]) If #P is closed under integer division by two (i.e., under the function $f(n) = \lfloor \frac{n}{2} \rfloor$), then $SPP = \oplus P$.

(4) ([57]) If #P is closed under minimum then $UP = NP$ and $SPP = C=P$.

The proofs of these theorems are similar in spirit to the proof of [Theorem 3.2](#). They are based on the fact that each of the closure properties that we are dealing with has some kind of a conditional behavior built in that, together with some nondeterministic Turing machine trickery, can be exploited to cause a given complexity class collapse. For example, in the case of integer division by two, we can use it (together with the fact that #P is closed under multiplication by two) to decrement a function by one, provided that the function's value is odd. (We integer-divide it by two and then multiply it by two.) In some cases, a complexity class equality can easily seem to be equivalent to a closure question. Here are two easy examples of that behavior, both of whose proofs are immediate from the definitions.

Proposition 3.8. (1) #P is closed under $\min(1, n)$ if and only if $UP = NP$.

(2) #P is closed under $1 \dot{-} n$ if and only if $UP = \text{coNP}$.

Proof. The proof of the first equivalence is immediate and we will omit it. However, for the sake of completeness, let us quickly prove the second equivalence. As part of the proof of [Theorem 3.2](#), following [36], we have already observed that if #P is closed under $1 \dot{-} n$ then $\text{coNP} \subseteq UP$. Clearly, $\text{coNP} \subseteq UP$ implies $UP = \text{coNP}$. It remains to show that if $UP = \text{coNP}$ then #P is closed under $1 \dot{-} n$. Let f be an arbitrary #P function and let L_f be the language $\{x \mid f(x) > 0\}$. Naturally, L_f belongs to NP and so its complement, $\overline{L}_f = \{x \mid f(x) = 0\}$, is in coNP. Since we assumed that $UP = \text{coNP}$ we have $\overline{L}_f \in UP$, and thus there is a UP machine M_h that accepts \overline{L}_f . We can view M_h as instantiating a #P function h such that

$$h(x) = \begin{cases} 1 & \text{if } f(x) = 0, \\ 0 & \text{if } f(x) > 0. \end{cases}$$

So the theorem is proven, since clearly $h(x) = 1 \dot{-} f(x)$. \square

As a quick example of how equivalences such as those of [Proposition 3.8](#) can help us understand the relationships between closure properties, consider the following assertion: If #P is closed under $1 \dot{-} n$ then #P is closed under $\min(1, n)$. Faced with that assertion, one might not (though see the next paragraph) immediately know whether it was true. However, in light of [Proposition 3.8](#) one instantly knows that the assertion is true, since the assertion becomes just another way of expressing the obvious fact that $UP = \text{coNP}$ implies $UP = NP$.

The observations of the previous paragraph should be contrasted with the somewhat different – and very general

and attractive (at least when it happens to work) – approach of using equalities to link closure properties in a way that spans all function classes. For example, for all natural numbers n it clearly holds that $\min(1, n) = 1 \dot{-} (1 \dot{-} n)$. From that equality we may in one fell swoop conclude that for every class \mathcal{F} of functions mapping from Σ^* to \mathbb{N} it holds that if \mathcal{F} is closed under $1 \dot{-} n$ then \mathcal{F} is also closed under $\min(1, n)$. Since in light of [Theorem 3.7](#) proper subtraction is unlikely to be a closure property of #P, it is natural to seek to work around that by making the question a bit more flexible, and some papers have sought to do so. Building on the ideas of [57], interesting work of Gupta [30,29] has defined and studied analogous notions for the class GapP and for a quotient-based class he introduced, and also has introduced the notion of seeking not to exactly compute a closure but rather to approximate it with high probability on each input. It turns out that in these cases the results one gets are, loosely put, analogous to the vanilla #P cases.

Hertrampf, Vollmer, and Wagner [40] have pursued yet another approach—one that contrasts with and complements the “characterize potential closures in terms of complexity class equalities” approach of Ogiwara and Hemachandra, and of the work of Gupta discussed in the previous paragraph. In particular, Hertrampf, Vollmer, and Wagner look at relativization, and elegantly characterize and capture the class of all closure properties that hold for #P robustly, i.e., that hold for #P under all possible relativizations.

Finally, Ogihara et al. [56] have developed a different and very interesting way of looking at closure properties of #P. We have seen that #P is not closed under proper subtraction. However, they note that in a certain sense it is “close” to being closed under proper subtraction. In particular, given two #P functions f and g one can easily see that there exists a polynomial p (for example, any polynomial that is at least one more than the maximum of the nondeterminism polynomials of two fixed machines modeling f and g) such that $h(x) = 2^{p(|x|)} + (f(x) - g(x))$ is a #P function. Thus, if we were allowed to perform some amount of postcomputation, then we could retrieve the value $f(x) \dot{-} g(x)$ from $h(x)$.

Many interesting open directions exist in the study of the closure properties of #P functions. One particularly attractive one is to “close the gaps” in characterizations involving intermediate closure properties. For example, as noted in [Theorem 3.7](#), $NP \subseteq UP$ is a sufficient condition for #P being closed under proper decrement, and $NP \subseteq SPP$ is a necessary condition for #P being closed under proper decrement. Can one find a complexity class containment that completely characterizes whether #P is closed under proper decrement? Similarly, for each of the other natural but seemingly intermediate closure properties, what are the best necessary conditions and the best sufficient conditions that one can obtain? And can one improve these to the point of having, for each natural intermediate property, a complexity class containment that completely characterizes whether #P is closed under that intermediate property?

A different direction of study stems from our power-index discussion from the introduction. We have noted that if #P were closed under subtraction then the problem of deciding in which among two voting games a given agent is more often influential would be in NP and after seeing

the results of this section we know that this problem would even be in UP. Recently, Faliszewski and Hemaspaandra [18] showed that this problem is PP-complete. However, this line of research still has many interesting open problems and we describe one of them below.

In the introduction we were discussing the “raw” version of the Shapley–Shubik power index. However, often it is useful to talk of a normalized version. Let f be the function that given an n -player weighted voting game G and an individual x in this game returns x 's raw Shapley–Shubik power index. It is easy to see that

$$\sum_{x \text{ is a player in } G} f(G, x) = n!$$

The normalized version of the Shapley–Shubik index is $g(G, x) = \frac{f(G, x)}{n!}$ (where n is the number of players in G). Thanks to normalization, we can compare the values of the power index in games with different numbers of players. In multiagent systems, the normalized Shapley–Shubik power index is often used to divide the payoff of the players involved in the game. A typical method is to have agents' payoffs be proportional to the value of their power indices. We assume that the total payoff to split is constant and does not depend on the number of players involved. As observed by Bachrach and Elkind [2], this model of splitting payoffs may create incentives for so-called false-name manipulation. In particular, consider a weighted voting game G and two of G 's players, x and y . The total payoff of x and y is proportional to the sum of their normalized Shapley–Shubik power indices. However, x and y might pretend that they are a single agent z whose weight is the sum of the weights of x and y . Let G' be a weighted voting game identical to G except that the players x and y are replaced by the player z . If it happens to be the case that z 's normalized Shapley–Shubik index in G' is higher than the sum of the normalized indices of x and y in G , then agents x and y have an incentive to pretend that they are z to obtain a higher payoff. Bachrach and Elkind [2] point out that joining two agents like this can in some cases increase the value of their joint power index but in some cases can decrease it. Thus agents x and y would very much like to know whether it is beneficial for them to present themselves as a single entity. Following Bachrach and Elkind [2], we ask about the exact complexity of testing whether the sum of the normalized Shapley–Shubik power indices of two players is smaller than the normalized Shapley–Shubik power index of a player formed by joining them (all the other players remain unchanged). Via the method of Faliszewski and Hemaspaandra [18], it is easy to show that the problem belongs to the complexity class PP. However, it remains open whether the problem is PP-complete. Perhaps, the problem is complete for some other class? As a starting point for readers interested in power indices and their complexity, we mention the paper of Shapley and Shubik [67] (for the Shapley–Shubik power index), the paper of Dubey and Shapley [14] (for the Banzhaf power index), the book of Garey and Johnson [23] and the work of Prasad and Kelly [60] and Matsui and Matsui [51] (for completeness results regarding the Shapley–Shubik and Banzhaf power indices), and the work of Hemaspaandra et al. [38] (for some experimental results involving power indices, US Congressional Apportionment, and further references).

4. Final comments

In this paper, we looked at the issue of elimination of solutions in some differing contexts. Although the paper is primarily tutorial-like, some results are to the best of our knowledge new to this paper, in particular [Theorems 3.4](#) and [3.6](#), [Proposition 3.8](#), and all of [Section 2.2](#).

Readers interested in the original or alternate treatments of the work covered in parts of [Section 2](#) are referred to the various original literature papers cited in that section, and also to [\[36,39\]](#). Among the most interesting related open issues are whether the collapses to NP^{NP} can be strengthened to collapses to $\text{S}_2^{\text{NP} \cap \text{coNP}}$. Readers interested in the original or alternate treatments of the work covered in parts of [Section 3](#) are similarly referred to the various original literature papers cited in that section, and also to [\[32,36\]](#). Among the most interesting related open issues are whether complete “complexity class collapse” characterizations can be found for the intermediate closure properties of #P. Fertile ground for additional research on the complexity of eliminating solutions includes the recently defined models of cluster-computed functions [\[34,33\]](#) (and [\[33\]](#) starts in that direction by looking at decrementation) and interval functions [\[34\]](#).

Inter-area and even inter-disciplinary connections sometimes really do happen, e.g., the connection between compilers and automata theory is a compelling case, there is a dynamic interchange of ideas between algorithmics – particularly strikingly for the case of SAT solvers – and statistical physics (see, e.g., [\[4,59,27,1\]](#)), the theory of semi-feasible algorithms is strongly tied to tournament theory (see, e.g., [\[43,39\]](#)), and social choice theory in recent years has been increasingly tied to complexity and computational issues (see, e.g., [\[16,11\]](#)). It is our hope both that knowledge of the material of this survey on solution reduction may be of use to people in domains other than complexity theory where solution-reduction issues occur, and that perhaps people in other areas of computer science will develop new insights that may advance the field's fundamental understanding of solution reduction.

Acknowledgements

We thank Edith Elkind for helpful discussions that led to one of the open problems in [Section 3](#), and we are grateful to the *Computer Science Review* referee and editors for helpful comments and suggestions. The first author was supported in part by grant NSF-CCF-0426761. The second author was supported in part by grant NSF-CCF-0426761, a TransCoop grant, and a Friedrich Wilhelm Bessel Research Award.

REFERENCES

- [1] D. Achlioptas, A. Naor, Y. Peres, Rigorous location of phase transitions in hard optimization problems, *Nature* 435 (2005) 759–764.
- [2] Y. Bachrach, E. Elkind, Divide and conquer: False-name manipulations in weighted voting games, in: *Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems*, May 2008 (in press).

- [3] R. Book, T. Long, A. Selman, Quantitative relativizations of complexity classes, *SIAM Journal on Computing* 13 (3) (1984) 461–487.
- [4] A. Braunstein, M. Mézard, R. Zecchina, Survey propagation: An algorithm for satisfiability, *Random Structures and Algorithms* 27 (2) (2005) 201–226.
- [5] H. Buhrman, J. Kadin, T. Thierauf, Functions computable with nonadaptive queries to NP, *Theory of Computing Systems* 31 (1) (1998) 77–92.
- [6] J. Cai, V. Chakaravarthy, L. Hemaspaandra, M. Ogihara, Competing provers yield improved Karp–Lipton collapse results, *Information and Computation* 198 (1) (2005) 1–23.
- [7] J. Cai, R. Lipton, L. Longpré, M. Ogihara, K. Regan, D. Sivakumar, Communication complexity of key agreement on limited ranges, in: *Proceedings of the 12th Annual Symposium on Theoretical Aspects of Computer Science*, in: *Lecture Notes in Computer Science*, vol. 900, Springer-Verlag, 1995, pp. 38–49.
- [8] R. Canetti, More on BPP and the polynomial-time hierarchy, *Information Processing Letters* 57 (5) (1996) 237–241.
- [9] A. Chandra, D. Harel, Computable queries for relational data bases, *Journal of Computer and System Sciences* 21 (2) (1980) 156–178.
- [10] S. Chari, P. Rohatgi, A. Srinivasan, Randomness-optimal unique element isolation, with applications to perfect matching and related problems, *SIAM Journal on Computing* 24 (5) (1995) 1036–1050.
- [11] Y. Chevaleyre, U. Endriss, J. Lang, N. Maudet, A short introduction to computational social choice, in: *Proceedings of the 33rd International Conference on Current Trends in Theory and Practice of Computer Science*, Springer-Verlag, 2007.
- [12] A. Darwiche, A compiler for deterministic decomposable negation normal form, in: *Proceedings of the 18th National Conference on Artificial Intelligence*, AAAI Press, 2002, pp. 627–634.
- [13] J. Davies, F. Bacchus, Using more reasoning to improve #SAT solving, in: *Proceedings of the 22nd AAAI Conference on Artificial Intelligence*, AAAI Press, 2007, pp. 185–190.
- [14] P. Dubey, L. Shapley, Mathematical properties of the Banzhaf power index, *Mathematics of Operations Research* 4 (2) (1979) 99–131.
- [15] T. Eiter, W. Faber, N. Leone, G. Pfeifer, A. Polleres, Answer set planning under action costs, *Journal of Artificial Intelligence Research* 19 (2003) 25–71.
- [16] P. Faliszewski, E. Hemaspaandra, L. Hemaspaandra, J. Rothe, A richer understanding of the complexity of election systems, in: S. Ravi, S. Shukla (Eds.), *Fundamental Problems in Computing: Essays in Honor of Professor Daniel J. Rosenkrantz*, Springer (in press). Preliminary version available as [17].
- [17] P. Faliszewski, E. Hemaspaandra, L. Hemaspaandra, J. Rothe, A richer understanding of the complexity of election systems, Technical Report TR-903, Department of Computer Science, University of Rochester, Rochester, NY, September 2006.
- [18] P. Faliszewski, L. Hemaspaandra, The complexity of power-index comparison, in: *Proceedings of the 4th International Conference on Algorithmic Aspects in Information and Management*, in *Lecture Notes in Computer Science*, Springer, 2008 (in press).
- [19] P. Faliszewski, M. Ogihara, On the autoreducibility of functions, Technical Report TR-912, Department of Computer Science, University of Rochester, Rochester, NY, January 2007.
- [20] S. Fenner, L. Fortnow, S. Kurtz, Gap-definable counting classes, *Journal of Computer and System Sciences* 48 (1) (1994) 116–148.
- [21] S. Fenner, F. Green, S. Homer, A. Selman, T. Thierauf, H. Vollmer, Complements of multivalued functions, *Chicago Journal of Theoretical Computer Science* 1999 (3) (1999). URL cs.uchicago.edu/publications/cjtcs/articles/1999/3/contents.html.
- [22] L. Fortnow, N. Reingold, PP is closed under truth-table reductions, *Information and Computation* 124 (1) (1996) 1–6.
- [23] M. Garey, D. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*, W. H. Freeman and Company, 1979.
- [24] J. Gill, Computational complexity of probabilistic Turing machines, *SIAM Journal on Computing* 6 (4) (1977) 675–695.
- [25] C. Glaßer, A. Selman, S. Sengupta, Reductions between disjoint NP-pairs, *Information and Computation* 200 (2) (2005) 247–267.
- [26] L. Goldschlager, I. Parberry, On the construction of parallel computers from various bases of boolean functions, *Theoretical Computer Science* 43 (1) (1986) 43–58.
- [27] C. Gomes, B. Selman, Computational science: Can get satisfaction, *Nature* 435 (2005) 751–752.
- [28] S. Greco, D. Saccà, Search and optimization problems in Datalog, in: *Computational Logic: Logic Programming and Beyond: Essays in Honour of Robert A. Kowalski, Part II*, in: *Lecture Notes in Computer Science*, vol. 2408, Springer-Verlag, 2002, pp. 61–82.
- [29] S. Gupta, On the closure of certain function classes under integer division by polynomially bounded functions, *Information Processing Letters* 44 (2) (1992) 205–210.
- [30] S. Gupta, Closure properties and witness reduction, *Journal of Computer and System Sciences* 50 (3) (1995) 412–432.
- [31] Y. Han, L. Hemaspaandra, T. Thierauf, Threshold computation and cryptographic security, *SIAM Journal on Computing* 26 (1) (1997) 59–78.
- [32] L. Hemachandra, M. Ogiwara, Is #P closed under subtraction? in: G. Rozenberg, A. Salomaa (Eds.), *Current Trends in Theoretical Computer Science: Essays and Tutorials*, World Scientific, 1993, pp. 523–536.
- [33] L. Hemaspaandra, C. Homan, S. Kosub, Cluster computing and the power of edge recognition, *Information and Computation* 205 (8) (2007) 1274–1293.
- [34] L. Hemaspaandra, C. Homan, S. Kosub, K. Wagner, The complexity of computing the size of an interval, *SIAM Journal on Computing* 36 (5) (2006) 1264–1300.
- [35] L. Hemaspaandra, A. Naik, M. Ogihara, A. Selman, Computing solutions uniquely collapses the polynomial hierarchy, *SIAM Journal on Computing* 25 (4) (1996) 697–708.
- [36] L. Hemaspaandra, M. Ogihara, *The Complexity Theory Companion*, Springer-Verlag, 2002.
- [37] L. Hemaspaandra, M. Ogihara, G. Wechsung, Reducing the number of solutions of NP functions, *Journal of Computer and System Sciences* 64 (2) (2002) 311–328.
- [38] L. Hemaspaandra, K. Rajasethupathy, P. Sethupathy, M. Zimand, Power balance and apportionment algorithms for the United States Congress, *ACM Journal of Experimental Algorithmics* 3 (1) (1998). URL jea.acm.org/1998/HemaspaandraPower.
- [39] L. Hemaspaandra, L. Torenvliet, *Theory of Semi-Feasible Algorithms*, Springer-Verlag, 2003.
- [40] U. Hertrampf, H. Vollmer, K. Wagner, On the power of number-theoretic operations with respect to counting, in: *Proceedings of the 10th Structure in Complexity Theory Conference*, IEEE Computer Society Press, 1995, pp. 299–314.
- [41] S. Isobe, W. Kumagai, M. Mambo, H. Shizuya, Toward separating integer factoring from discrete logarithm, *IEICE Transactions on Communications, Electronics, Information, and Systems* E90-A (1) (2007) 48–53.

- [42] R. Karp, R. Lipton, Some connections between nonuniform and uniform complexity classes, in: Proceedings of the 12th ACM Symposium on Theory of Computing, ACM Press, 1980, pp. 302–309. An extended version has also appeared as: Turing machines that take advice, *L'Enseignement Mathématique*, 2nd series, 28:191–209, 1982.
- [43] K. Ko, On self-reducibility and weak P-selectivity, *Journal of Computer and System Sciences* 26 (2) (1983) 209–221.
- [44] J. Köbler, U. Schöning, J. Torán, On counting and approximation, *Acta Informatica* 26 (4) (1989) 363–379.
- [45] J. Köbler, R. Schuler, Average-case intractability vs. worst-case intractability, *Information and Computation* 190 (1) (2004) 1–17.
- [46] M. Krentel, The complexity of optimization problems, *Journal of Computer and System Sciences* 36 (3) (1988) 490–509.
- [47] R. Ladner, On the structure of polynomial time reducibility, *Journal of the ACM* 22 (1) (1975) 155–171.
- [48] N. Leone, L. Palopoli, D. Saccà, On the complexity of search queries, in: *Fundamentals of Information Systems*, Kluwer Academic Publishers, 1999, pp. 113–128.
- [49] T. Long, Strong nondeterministic polynomial-time reducibilities, *Theoretical Computer Science* 21 (1) (1982) 1–25.
- [50] M. Mahajan, T. Thierauf, N. Vinodchandran, A note on SpanP functions, *Information Processing Letters* 51 (1) (1994) 7–10.
- [51] Y. Matsui, T. Matsui, NP-completeness for calculating power indices of weighted majority games, *Theoretical Computer Science* 263 (1–2) (2001) 305–310.
- [52] A. Meyer, L. Stockmeyer, The equivalence problem for regular expressions with squaring requires exponential space, in: *Proceedings of the 13th IEEE Symposium on Switching and Automata Theory*, October 1972, pp. 125–129.
- [53] K. Mulmuley, U. Vazirani, V. Vazirani, Matching is as easy as matrix inversion, *Combinatorica* 7 (1) (1987) 105–113.
- [54] A. Naik, J. Rogers, J. Royer, A. Selman, A hierarchy based on output multiplicity, *Theoretical Computer Science* 207 (1) (1998) 131–157.
- [55] M. Ogihara, Functions computable with limited access to NP, *Information Processing Letters* 58 (1) (1996) 35–38.
- [56] M. Ogihara, T. Thierauf, S. Toda, O. Watanabe, On closure properties of #P in the context of $P\#P$, *Journal of Computer and System Sciences* 53 (2) (1996) 171–179.
- [57] M. Ogiwara, L. Hemachandra, A complexity theory for feasible closure properties, *Journal of Computer and System Sciences* 46 (3) (1993) 295–325.
- [58] C. Papadimitriou, S. Zachos, Two remarks on the power of counting, in: *Proceedings 6th GI Conference on Theoretical Computer Science*, in: *Lecture Notes in Computer Science*, vol. 145, Springer-Verlag, 1983, pp. 269–276.
- [59] A. Percus, G. Istrate, C. Moore (Eds.), *Computational Complexity and Statistical Physics*, Oxford University Press, 2006.
- [60] K. Prasad, J. Kelly, NP-completeness of some problems concerning voting games, *International Journal of Game Theory* 19 (1) (1990) 1–9.
- [61] K. Regan, Enumeration problems. Manuscript, 1982; revised, 1985.
- [62] D. Roth, On the hardness of approximate reasoning, *Artificial Intelligence* 82 (1–2) (1996) 273–302.
- [63] A. Russell, R. Sundaram, Symmetric alternation captures BPP, *Computational Complexity* 7 (2) (1998) 152–162.
- [64] T. Sang, F. Bacchus, P. Beame, H. Kautz, T. Pitassi, Combining component caching and clause learning for effective model counting, in: *Proceedings of the 7th International Conference on Theory and Applications of Satisfiability Testing*, May 2004, pp. 20–28.
- [65] A. Selman, Polynomial time enumeration reducibility, *SIAM Journal on Computing* 7 (4) (1978) 440–457.
- [66] A. Selman, A taxonomy of complexity classes of functions, *Journal of Computer and System Sciences* 48 (2) (1994) 357–381.
- [67] L. Shapley, M. Shubik, A method of evaluating the distribution of power in a committee system, *American Political Science Review* 48 (1954) 787–792.
- [68] J. Simon, On some central problems in computational complexity, Ph.D. Thesis, Cornell University, Ithaca, NY, January 1975. Available as Cornell Department of Computer Science Technical Report TR75-224.
- [69] L. Stockmeyer, The polynomial-time hierarchy, *Theoretical Computer Science* 3 (1) (1976) 1–22.
- [70] M. Thurley, Combining component caching and clause learning for effective model counting, in: *Proceedings of the 9th International Conference on Theory and Applications of Satisfiability Testing*, August 2006, pp. 424–429.
- [71] L. Valiant, The relative complexity of checking and evaluating, *Information Processing Letters* 5 (1) (1976) 20–23.
- [72] L. Valiant, The complexity of computing the permanent, *Theoretical Computer Science* 8 (2) (1979) 189–201.
- [73] L. Valiant, The complexity of enumeration and reliability problems, *SIAM Journal on Computing* 8 (3) (1979) 410–421.
- [74] L. Valiant, V. Vazirani, NP is as easy as detecting unique solutions, *Theoretical Computer Science* 47 (3) (1986) 85–93.
- [75] H. Vollmer, On different reducibility notions for function classes, in: *Proceedings of the 11th Annual Symposium on Theoretical Aspects of Computer Science*, in: *Lecture Notes in Computer Science*, vol. 775, Springer-Verlag, 1994, pp. 449–460.
- [76] K. Wagner, The complexity of combinatorial problems with succinct input representations, *Acta Informatica* 23 (3) (1986) 325–356.
- [77] V. Zankó, #P-completeness via many-one reductions, *International Journal of Foundations of Computer Science* 2 (1) (1991) 76–82.