On defining PPT-search problems

Oded Goldreich
Department of Computer Science
Weizmann Institute of Science, Rehovot, Israel.

October 23, 2024

Abstract

We propose a new definition of the class of search problems that correspond to $\mathcal{BPP}$. Specifically, a problem in this class is specified by a polynomial-time approximable function $q : \{0,1\}^* \times \{0,1\}^* \to [0,1]$ that associates, with each possible solution y to an instance x, a quality $q(x,y)$. Intuitively, quality 1 corresponds to perfectly valid solutions, quality 0 corresponds to perfectly invalid solutions, but the quality of other solutions can be anywhere in-between. The class of PPT-search problems contains q if there exists a PPT algorithm that, on input x, finds a solution with value close to $\max_y\{q(x,y)\}$.

We relate this definition to previously studied definitions of "BPP-search problems" and articulate our preference for it. More importantly, we show that any PPT-search problem can be reduced in deterministic polynomial-time to a promise problem in pr$\mathcal{BPP}$ (i.e., promise-BPP).

On the companion paper. This paper represents my perspective on a joint project conducted together with Roei Tell, whereas Roei's perspective is presented in [6]. The technical contents of both papers is similar and both papers present the foregoing definition of PPT-search problems (i.e., Definition 2.4), but the perspectives presented in the two papers differ.

> ... it is nevertheless a sacred duty to prefer the truth to one's friends.
> [Aristotle, *Nicomachean Ethics*, Book I, Chapter 6, 1st paragraph]

Contents

1 Introduction **1**

2 Definitions **1**
2.1 Towards defining PPT-search: Background 1
2.2 Towards defining PPT-search: Alternatives 3
2.3 Relating the definitions 4

3 Discussion **6**
3.1 Specifying sets and decisional problems 6
3.2 Specifying search problems 7
3.3 Defining natural complexity classes of search problems 8
3.4 Relating complexity classes of different types 10

4 Proofs **10**
4.1 Proof of Theorem 2.5 10
4.2 On the failure of other versions of Theorem 2.5 12
4.3 Emulations 13

5 Conclusion **15**

1 Introduction

While search problems are of great intuitive importance, most research in complexity theory is focused on decision problems. This phenomenon is justified by the fact that the study of search problems seems more cumbersome and that in many cases this study can be reduced to the study of decision problems. Unfortunately, the latter phenomenon does not always hold, and in such cases one is forced to deal directly with search problems.

One such case, which is the actual focus of this text, is the case of search problems that are solvable by probabilistic polynomial-time (PPT) algorithms. Such problems were the focus of [4], which provided a definition of a class called "BPP-search" (see [4, Def. 3.2]). We view this definition as a tentative proposal, and will propose alternatives that we consider better. We also propose to change the terminology, replacing the term BPP-search by PPT-search, which we consider more appropriate.

2 Definitions

We shall first review the definitions proposed in [4], and then make our own proposals and explore their consequences. A more extensive discussion of these definitions appears in Section 3.

2.1 Towards defining PPT-search: Background

Recall that search problems are typically associated with binary relations. Specifically, the search problem associated with the binary relation $R \subseteq \{0,1\}^* \times \{0,1\}^*$ consists of finding a solution y to a given instance x, where y is considered a valid solution to x if $(x,y) \in R$. More precisely, letting $R(x) \stackrel{\text{def}}{=} \{y : (x,y) \in R\}$, given an instance x such that $R(x) \neq \emptyset$, the task is find an element of $R(x)$.

Turning to PPT-search, the naive proposal is to say that a binary relation $R \subseteq \{0,1\}^* \times \{0,1\}^*$ constitutes a PPT-search problem if there exists a probabilistic polynomial-time (PPT) algorithm A such that $\Pr[A(x) \in R(x)] \geq 2/3$ for every x such that $R(x) \neq \emptyset$. One drawback of this naive definition is that it does not support *error reduction*. This is a consequence of the fact that this definition does *not* require that valid solutions can be easily recognized as valid. We consider the latter fact to be a fundamental drawback of the naive definition.

There is an easy fix for the foregoing drawbacks: Augmenting the naive definition so that the missing feature is made an explicit condition. This leads to [4, Def. 3.1], which is reproduced next.

Definition 2.1 (PPT-search problem, Type 1, reproducing [4, Def. 3.1]):[1] *The binary relation $R \subseteq \{0,1\}^* \times \{0,1\}^*$ constitutes a* PPT-search problem (of Type 1) *if the following two conditions hold.*

1. *R satisfies the naive definition* (i.e., efficiently solving R)*: There exists a PPT algorithm A such that for every $x \in \{0,1\}^*$ if $R(x) \neq \emptyset$, then $\Pr[A(x) \in R(x)] \geq 2/3$.*
2. *Efficient recognition of valid solutions* (i.e., R as a set is in $\mathcal{BPP}$)*: There exists a PPT algorithm D such that for every $x, y \in \{0,1\}^*$ if $y \in R(x)$ then $\Pr[D(x,y) = 1] \geq 2/3$ and otherwise $\Pr[D(x,y) = 1] \leq 1/3$.*

We say that a probabilistic mechanism, M, solves (T1-solves) *the search problem R if M satisfies the first condition; that is, $\Pr[M(x) \in R(x)] \geq 2/3$ for every $x \in \{0,1\}^*$ such that $R(x) \neq \emptyset$.*

The fact that the distinguishing algorithm D is allowed to be probabilistic is natural within the current mind-frame in which PPT is viewed as the yardstick for efficient computation. (Indeed, a more restricted class of PPT-search problems would arise if we were to require that D is deterministic; but, as will become clear shortly, we wish to enlarge the class of PPT-search problems rather than make it smaller.)

Before proceeding, let us note some advantages offered by Definition 2.1. First, it allows for error reduction (in Condition 1), since one can repeatedly invoke algorithm A on x (till obtaining a valid solution). Second, the current formulation, which is actually a "candid search problem" (see [3, Def. 2.30]), is computationally equivalent to one that also requires that if $R(x) \neq \emptyset$, then $\Pr[A(x) = \perp] \geq 2/3$, where $\perp \notin \{0,1\}*$ is a special failure symbol. Furthermore, these PPT-search problems can be solved by a deterministic reduction to pr$\mathcal{BPP}$ (see [4, Thm. 3.5]).[2] (This fact suffices for relating the existence of polynomial-time pseudodeterministic algorithms [2] for PPT-search problems to the question of whether every problem in pr$\mathcal{BPP}$ can be extended to a problem in $\mathcal{BPP}$ (see the first two pages of [5, Sec. 4]).)

Unfortunately, many natural approximation problems that can be solved by PPT algorithms, including the archetypical problem of estimating the acceptance probability of a Boolean circuit, can not be captured by Definition 2.1. Typically, in these cases, a perfect solution is hard to find and recognize, whereas one can easily find and recognize "good enough" solutions. The hardness of evaluating the exact quality of solutions leads to relaxing the requirement such that one is only required to distinguish good solutions from bad ones, whereas there is a gap between being good and being bad. This is captured by the formalism of promise problems.

[1] The notion of T1-solving is not stated explicitly in [4, Def. 3.1].

[2] Recall that, while $\mathcal{BPP}$ denotes the class of (pure) decision problems that can be decided by a PPT algorithm, pr$\mathcal{BPP}$ denotes the corresponding class of decisional promise problems.

While promise problems have appeared in the penultimate paragraph (i.e., candid search problems (let alone pr$\mathcal{BPP}$)), in the case of (candid) search problems the promise was on the instance x (in Condition 1) only. This formulation falls within the framework of [3, Def. 2.29], and its conceptual meaning is clear (i.e., we totally discard instances that violate the promise and do not care what the algorithm does with them). In contrast, allowing (as in [4, Def. 3.2]) a promise on the set of instance-solution pairs leads to a tri-partition of the set of possible solutions into valid, invalid, and in-between solutions. This relaxes the distinguishing task (i.e., Condition 2), which only refers to valid and invalid solutions, but raises the question of how to treat the in-between solutions in the solving task (i.e., Condition 1). The conservative approach towards the solving task, which was taken by [4, Def. 3.2], is to ignore the in-between solutions in the solving task (i.e., insists that solving means providing a valid solution).

Definition 2.2 (PPT-search problem, Type 2, reproducing [4, Def. 3.2]):[3] *Let $R_{\mathtt{yes}}, R_{\mathtt{no}} \subseteq \{0,1\}^* \times \{0,1\}^*$ be two disjoint binary relations. The $(R_{\mathtt{yes}}, R_{\mathtt{no}})$ constitutes a* PPT-search problem (of Type 2) *if the following two conditions hold.*

1. *Efficiently solving $R_{\mathtt{yes}}$: There exists a PPT algorithm A such that for every $x \in \{0,1\}^*$ if $R_{\mathtt{yes}}(x) \neq \emptyset$, then $\Pr[A(x) \in R_{\mathtt{yes}}(x)] \geq 2/3$.*
2. *Efficient distinguishing between $R_{\mathtt{yes}}$ and $R_{\mathtt{no}}$* (i.e., $(R_{\mathtt{yes}}, R_{\mathtt{no}})$ as a decisional promise problem is in pr$\mathcal{BPP}$)*: There exists a PPT algorithm D such that for every $x, y \in \{0,1\}^*$ if $y \in R_{\mathtt{yes}}(x)$ then $\Pr[D(x,y) = 1] \geq 2/3$, whereas if $y \in R_{\mathtt{no}}(x)$ then $\Pr[D(x,y) = 1] \leq 1/3$.*

We say that a probabilistic mechanism, M, solves (T2-solves) *the search problem $(R_{\mathtt{yes}}, R_{\mathtt{no}})$ if M satisfies the first condition; that is, $\Pr[M(x) \in R_{\mathtt{yes}}(x)] \geq 2/3$ for every $x \in \{0,1\}^*$ such that $R_{\mathtt{yes}}(x) \neq \emptyset$.*

The elements of $R_{\mathtt{yes}}(x)$ are viewed as valid solutions (for x), the elements of $R_{\mathtt{no}}(x)$ are viewed as invalid solutions (for x), whereas the rest are in *limbo* (and are hereafter referred to as limbo solutions). Note that Condition 1 treats the limbo solutions as invalid, whereas Condition 2 allows to treat them as "wild cards" (i.e., it views them as either valid or invalid, at whim). This discrepancy does not allow for error reduction in Condition 1, nor does it provide for the recognition of instances that have no valid solution. Furthermore, only an (unspecified) subset of these PPT-search problems can be solved by a deterministic reduction to pr$\mathcal{BPP}$ (see [4, Thm. 3.5], which refers to search problems having a "companion" that is a BPP-search problem).

2.2 Towards defining PPT-search: Alternatives

The first alternative we consider is streamlining the two conditions in Definition 2.2. Specifically, given that Condition 2 treats limbo solutions as "wild cards" (i.e., views them either valid or invalid, at whim), we allow Condition 1 to do the same. Hence, the alternative definition is as follows.

Definition 2.3 (PPT-search problem, Type 3): *A pair $(R_{\mathtt{yes}}, R_{\mathtt{no}})$ as in Definition 2.2 constitutes a* PPT-search problem (of Type 3) *if the following two conditions hold.*

1. *A relaxed solving condition: There exists a PPT algorithm A such that for every $x \in \{0,1\}^*$ if $R_{\mathtt{yes}}(x) \neq \emptyset$, then $\Pr[A(x) \in (\{0,1\}^* \setminus R_{\mathtt{no}}(x))] \geq 2/3$.*

[3] The notion of T2-solving is not stated explicitly in [4, Def. 3.2].

2. *Efficient distinguishing between $R_{\mathtt{yes}}$ and $R_{\mathtt{no}}$* (as in Definition 2.2). *There exists a PPT algorithm D such that for every $x, y \in \{0,1\}^*$ if $y \in R_{\mathtt{yes}}(x)$ then $\Pr[D(x,y)=1] \geq 2/3$, whereas if $y \in R_{\mathtt{no}}(x)$ then $\Pr[D(x,y)=1] \leq 1/3$.*

We say that a probabilistic mechanism, M, solves (T3-solves) *the search problem $(R_{\mathtt{yes}}, R_{\mathtt{no}})$ if M satisfies the first condition; that is, $\Pr[M(x) \in (\{0,1\}^* \setminus R_{\mathtt{no}}(x))] \geq 2/3$ for every $x \in \{0,1\}^*$ such that $R_{\mathtt{yes}}(x) \neq \emptyset$.*

Note that PPT-search problems of Type 2 are a special case of PPT-search problems of Type 3, since satisfying Definition 2.2 implies satisfying Definition 2.3; that is, if an algorithm T2-solves $(R_{\mathtt{yes}}, R_{\mathtt{no}})$, then it T3-solves $(R_{\mathtt{yes}}, R_{\mathtt{no}})$. (Even more evidently, the fact that PPT-search problems of Type 1 are a special case of PPT-search problems of Type 2 follows by viewing the binary relation R as a disjoint pair $(R, U \setminus R)$, where $U = \{0,1\}^* \times \{0,1\}^*$.)

Definition 2.3 enjoys greater internal consistency than Definition 2.2, yet it still leaves us wondering why is it OK to "accept" solutions that cannot be certified as valid (but can be certified as not being invalid). Furthermore (like Definition 2.2), Definition 2.3 does not support error reduction in Condition 1.

One attempt to justify the foregoing convention (i.e., the "acceptability" of limbo solutions) is saying that limbo solutions have a quality that is in-between valid and invalid. This qualitative perspective, which reflects the aforementioned trichotomy, leads to asking why not allow several levels of intermediate quality. Actually, why not view quality in a quantitative manner; that is, consider solutions of arbitrary quality ranging, say, in $[0,1]$. This leads us to the second alternative. It consists of replacing the two disjoint binary relations $R_{\mathtt{yes}}, R_{\mathtt{no}} \subseteq \{0,1\}^* \times \{0,1\}^*$ by a quality function $q : \{0,1\}^* \times \{0,1\}^* \to [0,1]$ and defining solving the instance x as finding y that almost maximizes $q(x,y)$, where the function q is easy to approximate. Specifically, we propose the following definition.

Definition 2.4 (PPT-search problem, Type 4): *The function $q : \{0,1\}^* \times \{0,1\}^* \to [0,1]$ constitutes a* PPT-search problem (of Type 4) *if the following two conditions hold.*

1. *A relaxed solving condition: There exists a PPT algorithm A such that, for every $x \in \{0,1\}^*$ and $t \in \mathbb{N}$, it holds that $\Pr[q(x, A(x,1^t)) \geq q'(x) - (1/t)] \geq 2/3$, where $q'(x) \stackrel{\text{def}}{=} \max_y\{q(x,y)\}$.*

2. *Efficient approximation of q: There exists a PPT algorithm Q such that, for every $x, y \in \{0,1\}^*$ and $t \in N$, it holds that $\Pr[|Q(x,y,1^t) - q(x,y)| \leq 1/t] \geq 2/3$.*

We say that a probabilistic mechanism, M, solves (T4-solves) *the search problem q if M satisfies the first condition; that is, $\Pr[q(x, M(x,1^t)) \geq q'(x) - (1/t)] \geq 2/3$ for every $x \in \{0,1\}^*$ and $t \in \mathbb{N}$.*

Note that Definition 2.4 allows for error reduction (see the first part of Section 4.1). Furthermore, Definition 2.1 can be cast as a special case of Definition 2.4 by considering $q(x,y) \stackrel{\text{def}}{=} 1$ if $(x,y) \in R$ and $q(x,y) \stackrel{\text{def}}{=} 0$ otherwise.

2.3 Relating the definitions

We prefer Definition 2.4 over Definitions 2.2 and 2.3, and articulate the preference in Section 3.3. However, we have no decisive argument in favor of this preference nor can we rule out the possibility

of a better alternative. One technical advantage of Definition 2.4 over Definitions 2.2 and 2.3 is that it allows for error reduction in Condition 1. Another technical advantage of Definition 2.4 is that its solving task is deterministically reducible to a (decisional promise) problem in pr$\mathcal{BPP}$.[4]

Theorem 2.5 (solving PPT-search of Type 4 is deterministically reducible to pr$\mathcal{BPP}$):[5] *Let $q : \{0,1\}^* \times \{0,1\}^* \to [0,1]$ be a PPT-search problem of Type 4. Then, T4-solving q is reducible in deterministic polynomial-time to a promise problem in* pr$\mathcal{BPP}$*; that is, there exists $\Pi \in$ pr$\mathcal{BPP}$ and a deterministic polynomial-time oracle machine M such that $q(x, M^{\Pi}(x, 1^t)) \geq q'(x) - (1/t)$ for every $x \in \{0,1\}^*$ and $t \in \mathbb{N}$.*

The proof of Theorem 2.5, which uses the ideas that underlie the proof of [4, Thm. 3.5], appears in Section 4.1. An analogous result regarding PPT-search problems of Type 2 is impossible and for Type 3 it is unlikely (see Section 4.2). In contrast, the following result, which is stated for PPT-search problems of Type 4, holds also for PPT-search problems of Type 2 and 3.

Theorem 2.6 (probabilistic FPTAS is a special case of PPT-search):[6] *Consider a fully PPT approximation scheme for a function $f : \{0,1\}^* \to [0,1]$; that is, a PPT algorithm that, on input $x \in \{0,1\}^*$ and $k \in \mathbb{N}$, returns a value in $[f(x) \pm (1/k)]$. Then, $q(x,v) \stackrel{\text{def}}{=} \max(1 - |f(x) - v|, 0)$ is a PPT-search problem of Type 4 and approximating f reduces to solving the search problem of q.*

The proof amounts to observing that $q(x,v) \approx 1$ if and only if $f(x) \approx v$. Analogous statements for PPT-search problems of Type 2 and 3 follow from the fact that PPT-search problems of Type 4 can be expressed as (or emulated by) PPT-search problems of Type 2 (resp., Type 3). The latter assertion is captured by Proposition 4.6.

Actually, it is in place to ask what do we mean by saying that one type of problems can be expressed as (or emulated by) another type of problems. Loosely speaking, our answer is that this means that for each problem Π of the first type (resp., class) there exists a problem Π' of the second type (resp., class) such that solving Π is reducible to solving Π'. This answer raises the question of what does solving a problem mean, and the default and natural answer is that it means the notion of solving that is part of the definition of the class (of problems). (In fact, we made this interpretation explicit is the last part of Definitions 2.1–2.4). Nevertheless, other notions of solvability may be meaningful too. Specifically, the notion of T3-solvability, which is part of Definition 2.3, can be used for the problems of Type 2 (defined in Definition 2.2). We may view T3-solvability of problems of Type 2 as a weak notion of solvability of problems of Type 2.

The usefulness of the foregoing proposal is demonstrated in Proposition 4.4. Loosely speaking, Proposition 4.4 asserts that *for every problem $\overline{R} = (R_{\mathtt{yes}}, R_{\mathtt{no}})$ of Type 2 there exists a problem q of Type 4 such that T3-solving $\overline{R}$ is reducible in deterministic polynomial-time to solving q.* This may be interpreted as saying that problems of Type 2 can be weakly emulated by problems of

[4] This fact allows for extending the aforementioned result of [5, Sec. 4] from PPT-search of Type 1 to PPT-search of Type 4. Recall that this result relates the existence of polynomial-time pseudodeterministic algorithms [2] for PPT-search problems to the question of whether every problem in pr$\mathcal{BPP}$ can be extended to a problem in $\mathcal{BPP}$.

[5] This result is analogous to [4, Thm. 3.5], which shows that solving an (indirectly specified) subclass of the problems captured by [4, Def. 3.2] is (deterministically) reducible to pr$\mathcal{BPP}$. Specifically, this subclass consists of search problems that have a "companion" that is a BPP-search problem, where the notion of companion is defined in [4, Obs. 3.3].

[6] Approximating the acceptance probability of a given Boolean circuit is a special case; that is $f(C) = \Pr_x[C(x)=1]$.

Type 4. Combining Proposition 4.4 with Theorem 2.5 implies that every problem $\overline{R}$ of Type 2 can be T3-solved by a deterministic polynomial-time reduction to pr$\mathcal{BPP}$.

An alternative way of interpreting Proposition 4.4 is to say that it refers to a subclass of the problems of Type 3 (i.e., the class of problems of Type 2). Hence, we can say that Proposition 4.4 identifies a subclass of problems of Type 3 that can be emulated by problems of Type 4 (in the very sense stated above), except that viewing $\overline{R}$ as a problem of Type 3 justifies considering T3-solving it. Of course, this view ignores the fact that the problem $\overline{R}$ is of Type 2 and so a stronger notion of solving is relevant to it.

3 Discussion

The purpose of this section is trying to identify the principles that underlie the definition of complexity classes. Hence, we shall start with notions that are well-known and agreed upon by all (e.g., decision problems), and proceed to the more problematic and less explored notions, while trying to identify the underlying principles. Indeed, our real focus is on PPT-search problems; this focus comes to the front in Sections 3.2 and 3.3, which contain the discussions that are most important towards evaluating the various definitions presented in Section 2.

Complexity classes are classes of sets. Such a class is typically defined by imposing conditions on a broader class of sets, which in turn are specified in some manner. So we start by recalling how sets are typically specified in the context of complexity theory.

3.1 Specifying sets and decisional problems

Decisional problems are commonly associated with sets of binary strings. Two methods are commonly used to specify such sets.[7]

Specification by recognition: The set is specified by an algorithm (i.e., an operational rule) that distinguishes members of the set from non-members.

We stress that this algorithm needs not be the most efficient algorithm for the corresponding decision problem. It is often the conceptually simplest (or most intuitive) algorithm. For example, the set of primes is defined as the set of natural numbers that have no non-trivial divisor.

Specification by verification: The set is specified by a verification procedure that accepts members of the set, when coupled with adequate proofs, but never accepts non-members. The verification procedure is typically efficient, but the length of possible proofs may varying arbitrarily.

Although this specification can be replaced by one of the former type (i.e., the decision procedure may try all possible proofs), it is often more appealing and intuitive to use the current type. In particular, even sets in $\mathcal{P}$ are often specified in this manner (which only places them in $\mathcal{NP}$). Examples include the set of satisfiable 2CNF formulae, the set of bipartite graphs, and the set of graphs having perfect matchings.

[7] In addition, sets are also specified by set-operations that involve previously specified sets.

We stress that whereas $\mathcal{P}$ is defined as the class of sets that can be recognized in polynomial-time, specific sets in $\mathcal{P}$ may be specified in other means (see the foregoing examples). In general, we distinguish between the way that specific sets are specified and the way that a complexity class is defined (and specific sets are shown to reside in it).

Decisional problems of the *pure type* (a.k.a "decision problems") consist of determining, for every given instance, whether or not it is a member of a predetermined set. In contrast, *decisional problems of the promise type* (a.k.a "promise problems" [1]) only require such distinguishing for instances that are in a (predetermined) promise set. Hence, decisional problems of the promise type are define by the specification of two sets, a main set S and a promise set P, and consist of determining membership in S for instances in P only; equivalently, distinguishing between $P \cap S$ and $P \setminus S$. Indeed, these two sets determine a three-way partition of $\{0,1\}^*$ (i.e., the partition $(P \cap S, P \setminus S, \{0,1\}^* \setminus P)$), where instances in $\{0,1\}^* \setminus P$ are said to violate the promise. We stress that when considering decision procedures for such promise problems, their behavior on instances that violate the promise is immaterial (and is being ignored).

Defining complexity classes. Usually, classes of decisional promise problems are defined by considering the *complexity of the task of distinguishing between $P \cap S$ and $P \setminus S$*. For example, pr$\mathcal{BPP}$ is defined as the collection of pairs of sets (S,P) such that there exists a PPT algorithm A such that $\Pr[A(x)=1] \geq 2/3$ if $x \in P \cap S$ and $\Pr[A(x)=1] \leq 1/3$ if $x \in P \setminus S$. Equivalently, one may require that for every $x \in P$, it holds that $\Pr[A(x)=\chi_S(x)] \geq 2/3$, where $\chi_S(x) = 1$ if $x \in S$ and $\chi_S(x) = 0$ otherwise.

3.2 Specifying search problems

Search problems are commonly associated with binary relations $R \subseteq \{0,1\}^* \times \{0,1\}^*$, which are typically *specified by recognition algorithm*. In this case, the search problem consists of finding, when given an instance x, a string y such that $(x,y) \in R$. Such a string y is called a valid solution for the instance x, and $R(x) \stackrel{\text{def}}{=} \{y : (x,y) \in R\}$ is the set of valid solutions for x. The bottom-line is that search problems are specified by specifying the binary relation that is associated with them.

A promise problem of the search type [3, Def. 2.29] is specified by a promise set $P \subseteq \{0,1\}^*$ and a binary relation R (as above), and consists of finding valid solutions for instances in P (only). We stress that the promise refers to the instance only. Hence, just as in the case of decisional promise problems, the meaning of the promise in such search problems is clear: Instances that violate the promise do not matter and the performance of search algorithms on them is ignored.

In contrast, considering the task of recognizing members of R leads to a relaxation of the promise type, but here the promise is a subset P of instance-solution pairs. In this case, the semantics of violating the promise P is less clear (i.e., what does $(x,y) \notin P$ mean?). The problem arises when trying to formulate the searching task. Specifically, given an instance x, *are we required to return an element of $R(x)$* or *are we required to return an element of $R(x) \cap P(x)$* (i.e., a solution y such that $(x,y) \in P \cap R$) or *are we also allowed to return any string not in $P(x)$*?

Indeed, the source of trouble is our choice to specify a search problem via a trichotomy of instance-solution pairs, which yields three types of solutions (i.e., valid, invalid, and in-between (a.k.a limbo)). The motivation for this choice is extending natural classes of search problems by capitalizing on the possibility that distinguishing between $P \cap R$ and $P \setminus R$ may be easier than distinguishing between R and $\{0,1\}^* \setminus R$ (e.g.,, in the context of probabilistic polynomial-time).

But is this the best way of achieving the aforementioned goal?

On the actual specification of search problems. Before addressing the foregoing question, we note that underlying Definitions 2.1–2.4 are general mechanisms for defining search problems. Such a mechanism consists of associating a search problem with a specific formal structure and defining what is considered as solving the problem. Here we explicitly state the four mechanisms that are used in these four definitions.

In Definition 2.1 a problem is specified by a binary relation $R \subseteq \{0,1\}^* \times \{0,1\}^*$, and solving the problem means providing an element of $R(x)$ when given an instance x such that $R(x) \neq \emptyset$.

In Definition 2.2 a problem is specified by a pair of disjoint binary relation $R_{\mathtt{yes}}, R_{\mathtt{no}} \subseteq \{0,1\}^* \times \{0,1\}^*$, and solving the problem means providing an element of $R_{\mathtt{yes}}(x)$ when given an instance x such that $R_{\mathtt{yes}}(x) \neq \emptyset$.

In Definition 2.3 a problem is specified ans in Definition 2.2 (i.e., by a pair of disjoint binary relation $R_{\mathtt{yes}}, R_{\mathtt{no}} \subseteq \{0,1\}^* \times \{0,1\}^*$), and solving the problem means providing an element of $\{0,1\}^* \setminus R_{\mathtt{no}}(x)$ when given an instance x such that $R_{\mathtt{yes}}(x) \neq \emptyset$.

In Definition 2.4 a problem is specified by a function $q : \{0,1\}^* \times \{0,1\}^* \to [0,1]$, and solving the problem means providing an string y such that $q(x,y) \geq q'(x) - (1/t)$ when given an instance x and a parameter $t \in \mathbb{N}$.

The definitions presented in Section 2 were obtained by imposing complexity bounds on the relevant structures as well as on the relevant solvers.

We stress that all decisional (promise) problems are specified by one type of structure (i.e., a pair of sets) and the same notion of solving (the corresponding problem). In contrast, Definitions 2.2–2.4 exhibit two different types of structures and three different notion of solving. While the structure-type used in Definitions 2.2–2.3 is aligned with the traditional structure-type used in Definition 2.1, we see no reason to prefer it over the structure-type used in Definition 2.4. That is, the trichotomy of potential solutions (i.e., valid, invalid, limbo) does not seem more appealing than the qualitative view of the quality of potential solutions.

3.3 Defining natural complexity classes of search problems

Let us start with a class of search problems that arises, implicitly, in the context of the P-vs-NP question (cf. [3, Sec. 2.1.1]). First, we define the class of search problems that consists of all binary relations for which the recognition problem can be solved in polynomial-time (cf. [3, Def. 2.3]). When considering only polynomially bounded relations (i.e., R such that $(x,y) \in R$ implies $|y| \leq \mathrm{poly}(|x|)$), this yields a class of search problems, denoted $\mathcal{PC}$, which is analogous to $\mathcal{NP}$. The P-vs-NP question is equivalent to asking whether solutions for every problem in $\mathcal{PC}$ can be found in polynomial-time, and the implicit class that emerges is the class of all search problems in $\mathcal{PC}$ for which solutions can be found in polynomial-time.

We seek to follow the same path towards defining a corresponding class for PPT. Doing so leads to [4, Def. 3.1], which is reproduced as Definition 2.1. Unfortunately, the resulting class feels too restrictive; in particular, it does not cover the natural problem of approximating the acceptance probability of a given Boolean circuit.

The path taken by [4, Def. 3.2] is the one outlined in Section 3.2; indeed, this is the path that was followed in Definitions 2.2 and 2.3. The starting point of this path is a focus on the complexity of distinguishing between valid solutions and invalid solutions. The distinguishing task is made easier by introducing limbo solutions (on which the distinguisher may answer arbitrarily), but *the treatment of these limbo solutions in the solving task is conceptually discomforting*. This discomfort is amplified by our opinion that the study of search problems should give priority to the solving task. Hence, it seems more natural to take an alternative path, one that avoids the question of how to treat limbo solutions. Such an alternative path was taken by Definition 2.4. Let us outline the principles underlying this alternative path.

The first principle is that, when considering search problems that can be solve within some resource bounds, it is natural to confine the attention to problems for which solutions can be evaluated for validity within the same resource bounds. Intuitively, searching for a solution makes less sense if one cannot evaluate the validity (or quality) of the solution once found. In other words, the solution is meaningful for the finder, because it can recognize the validity (or quality) of the solution. We call this principle the meaningfulness principle. Formally, the search version of the P-vs-NP question is obtained by applying the meaningfulness principle (i.e., restricting attention to the class $\mathcal{PC}$). The meaningfulness principle also led us to Definitions 2.1–2.4.

The second principle calls for trying to provide the most general definition that corresponds to a natural notion, and yet do so in an intuitive manner (i.e., the generalized definition should be appealing). Hence, we tried to extend Definition 2.1, which is natural *per se* but was deemed too restrictive. Unfortunately, the extensions proposed by Definitions 2.2 and 2.3 were problematic: They replaced the clear valid-vs-invalid dichotomy (regarding potential solutions) by a trichotomy that included in-between (or limbo) solutions. Definitions 2.2 and 2.3 differ in the way they handle these limbo solutions, but in both cases this treatment does not abide the meaningfulness principle: Definition 2.2 does not allow the searcher to return a limbo solution although this solution may be indistinguishable from a valid solution, whereas Definition 2.3 allows the searcher to return a limbo solution although this solution may be indistinguishable from an invalid solution.

In contrast, Definition 2.4 softens the valid-vs-invalid (qualitative) dichotomy and replaces it by a quantitative spectrum. Viewed from this perspective, perfectly valid solutions are viewed as having quality 1, evidently invalid solutions are viewed as having quality 0, whereas, in general, potential solutions have a value in $[0,1]$. The task of the searcher is to find a solution of maximal quality (or rather almost maximal quality), whereas the quality of solutions can be well-approximated (by a PPT algorithm). Hence, in our opinion, Definition 2.4 satisfies both principles stated above.

In light of the foregoing, we prefer Definition 2.4 over Definitions 2.2 and 2.3. Specifically, we consider the quantitative view of the quality of potential solutions far more appealing than the valid vs limbo vs invalid trichotomy, let alone the problematic treatment of limbo solutions. Furthermore, we consider the fact that Definition 2.4 supports error reduction, whereas Definitions 2.2 and 2.3 do not seem to do so, a fundamental issue. Supporting error reduction allows for having a class that is independent of the specific error bound used in its definition (as long as this bound is reasonable).

We note that, although not intended for that purpose, Definition 2.4 captures the natural notion of a PPT-solvable optimization problem: Specifically, for a polynomial-time computable function $f : \{0,1\}^* \times \{0,1\}^* \to [0,1]$, consider the task in which, on input x and $\epsilon > 0$, one is required to find y such that $f(x,y) \geq \max_z\{f(x,z)\} - \epsilon$ and do so in probabilistic poly$(|x|/\epsilon)$-time. Indeed, this is a special case of Definition 2.4, where $q = f$ is polynomial-time computable rather than only

PPT approximable.

3.4 Relating complexity classes of different types

A typical class of computational problems incorporates, in its definition, a notion of solving problems in the class. In the case of decisional problems solving refers to deciding membership (possibly in a relaxed manner as captured by promise problems). In the case of search problems solving refers to finding solutions. As illustrated by these cases, different types of problems call for different notions of solving these problems. As illustrated by Definitions 2.1–2.4, even different types of PPT-search problems refer to different notions of solving these problems. This fact makes relating different types of problems less straightforward than it may seem.

Indeed, the natural choice is to say that a problem Π of one class is emulated by problem Π' of another class if solving Π in the sense of the first class reduces to solving Π' in the sense of the second class. Still, saying that solving Π in some other sense reduces to solving Π' in yet another sense is also informative. In fact, Proposition 4.4 asserts that T3-solving (rather than T2-solving) a PPT-search problem of Type 2 is reducible to T4-solving a problem of Type 4. That is, the first problem is solved in a sense that is weaker than the sense associated with the corresponding class. We stress that, using the foregoing flexibility, Proposition 4.4 asserts a result that is weaker than the natural formulation. Yet, this is an informative result.

4 Proofs

This section contains proofs of the results stated in Section 2.3 and some additional observations.

4.1 Proof of Theorem 2.5

The proof of Theorem 2.5 is analogous to the proof of [4, Thm. 3.5], which refers to an indirectly specified subclass of PPT-search problems of Type 2. Towards proving Theorem 2.5, we first restate and generalize the two conditions of Definition 2.4 (when referring to the function $q : \{0,1\}^* \times \{0,1\}^* \to [0,1]$). Starting with Condition 2 and emplying error-reduction (while taking the median value), we note that this condition implies the existence of a PPT algorithm Q such that for every $x, y \in \{0,1\}^*$ and $t \in \mathbb{N}$ it holds that

$$\Pr\left[\left|Q(x,y,1^t) - q(x,y)\right| \leq \frac{1}{t}\right] \geq 1 - 2^{-t}. \tag{1}$$

Next, turning to Condition 1 and employing error-reduction (while using the solution of the maximum approximate quality[8]), we note that this condition implies the existence of a PPT algorithm F such that for every $x \in \{0,1\}^*$ and $t \in \mathbb{N}$ it holds that

$$\Pr\left[q(x,F(x,1^t)) \geq q'(x) - \frac{1}{t}\right] \geq 1 - 2^{-t}, \tag{2}$$

where $q'(x) \stackrel{\text{def}}{=} \max_y\{q(x,y)\}$. We shall also consider arbitrary probabilistic procedures F that satisfy Eq. (2). At this point, we restate Theorem 2.5 as follows.

[8] Specifically, we invoke the original solver A as well as the evaluator Q with deviation bound $1/3t$.

Theorem 4.1 (PPT solvers vs reduction to promise-BPP): *Let $q : \{0,1\}^* \times \{0,1\}^* \to [0,1]$ be a function such that there exists a PPT algorithm Q that satisfies* Eq. (1)*. Then, the following two statement are equivalent:*

1. *There exists a PPT algorithm F that solves the search problem of q; that is, F satisfies* Eq. (2)*.*
2. *There exists a deterministic polynomial-time reduction of solving the search problem of q to a decisional problem in promise-BPP; that is, there exists a deterministic polynomial-time oracle machine M and a decisional promise problem Π in promise-BPP such that M^Π satisfies* Eq. (2)*.* (Actually, since M is deterministic, $q(x, M^\Pi(x, 1^t)) \geq q'(x) - (1/t)$ always holds).

Recall that, by definition (of a reduction to a promise problem), the reduction in Statement 2 yields the correct answer regardless of how queries that violate the promise are answered. The proof of Theorem 4.1 uses the ideas that underlie the proof of [4, Thm. 3.5].

Proof: Clearly, if Statement 2 holds, then we can derive a PPT algorithm that solves q (i.e., establish Statement 1) by emulating the oracle in a straightforward manner (after adequate error reduction).

The proof of the opposite direction (i.e., Statement 1 implies Statement 2) relies on the fact that a promise-BPP oracle can be used to obtain adequate approximations to the function p defined next. Let F be a PPT algorithm that solves the search problem of q (i.e., satisfies Eq. (2)), and let F_r denote the residual deterministic algorithm obtained from F when fixing its randomness to $r \in \{0,1\}^m$, where $m = \mathrm{poly}(|x| + t)$. Consider the function $p : \{0,1\}^* \times \{1\}^* \times \{0,1\}^* \to [0,1]$ defined by

$$p(x, 1^t, r') \stackrel{\text{def}}{=} \mathrm{E}_{r'' \in \{0,1\}^{m-|r'|}} \left[q(x, F_{r'r''}(x, 1^t)) \right].$$

By the hypothesis that F satisfies Eq. (2), it follows that

$$p(x, 1^t, \lambda) \geq (1 - 2^{-t}) \cdot (q'(x) - (1/t)) > q'(x) - (2/t).$$

On the other hand, for every r', it holds that $p(x, 1^t, r') = (p(x, 1^t, r'0) + p(x, 1^t, r'1))/2$. Now, observe that p can be approximated up-to an additive error of $3/mt$ by making mt oracle calls to the decisional promise problem $\Pi = (\Pi_{\mathtt{yes}}, \Pi_{\mathtt{no}})$ such that

$$\begin{aligned} \Pi_{\mathtt{yes}} &\stackrel{\text{def}}{=} \{(x, 1^k, r', v) : |p(x, 1^k, r') - v| \leq 1/k\} \\ \Pi_{\mathtt{no}} &\stackrel{\text{def}}{=} \{(x, 1^k, r', v) : |p(x, 1^k, r') - v| > 3/k\}. \end{aligned}$$

Specifically, we approximate $p(x, 1^t, r')$ by making the queries $(x, 1^{mt}, r', (i - 0.5)/mt)$ for all $i \in [mt]$, and set the estimate as an arbitrary value $(i - 0.5)/mt$ such that the oracle answered the corresponding query with a `yes`.[9]

We shall show that solving the search problem of q is reducible (in deterministic polynomial-time) to approximating p upto an additive term of $3/mt$. Recalling that such an approximation can be obtained by making mt queries to the decisional promise problem $\Pi = (\Pi_{\mathtt{yes}}, \Pi_{\mathtt{no}})$, we shall complete the proof by showing that Π is in promise-BPP.

[9] We stress that this reduction also issue queries that violate the promise of the decisional promise problem; the result of the reduction is correct regardless of how these ("promise violating") queries are answered. Actually, such ("promise violating") queries are crucial (cf. [3, Exer. 2.36]).

Starting with the reduction (of solving q to approximating p), recall that on input $(x, 1^t)$, we seek y such that $q(x, y) \geq q'(x) - (1/t)$. Equivalently, we seek $r \in \{0,1\}^m$ such that $q(x, F_r(x, 1^t)) \geq q'(x) - (1/t)$. We find such an r in m iterations by gradually extending a candidate prefix of r by one bit in each iteration. Specifically, in the i^{th} iteration the current $(i-1)$-bit long prefix is extended by a single bit towards the i-bit long prefix of the higher value; that is, $r' \in \{0,1\}^{i-1}$ is extended to $r'b$ if the estimate of $p(x, 1^t, r'b)$ is at least as large as the estimate of $p(x, 1^t, r'\overline{b})$. Consequently, it holds that $p(x, 1^t, r'b) \geq p(x, 1^t, r') - 3/mt$. Thus, after m iterations, we obtain $r \in \{0,1\}^m$ such that $p(x, 1^t, r) \geq p(x, 1^t, \lambda) - (3m/mt) > q'(x) - (5/t)$, and output the value $F_r(x, 1^t)$ (such that $q(x, F_r(x, 1^t)) > q'(x) - (5/t)$). (By substituting t with $5t$, we can output a value that exceeds $q'(x) - (1/t)$.)

Lastly, we show that Π is in promise-BPP. This is the case because $p(x, 1^t, r')$ can be approximated up-to an additive error of $1/t$ by approximating $q(x, F_{r'r''}(x, 1^t))$ on t^3 random choices of $r'' \in \{0,1\}^{m-|r'|}$. Specifically, we invoke $Q(x, F_{r'r''}(x, 1^t), 1^{2t})$ on t^3 random choices of $r'' \in \{0,1\}^{m-|r'|}$, and use the average of the obtained values. (Indeed, each of these t^3 invocations satisfies $\Pr[|Q(x, F_{r'r''}(x, 1^t), 1^{2t}) - q(x, F_{r'r''}(x, 1^t))| \leq (1/2t)] \geq 1 - 2^{-2t}$, whereas (with probability at least $1 - \exp(-\Omega(t))$) the average of the t^3 sampled $q(x, F_{r'r''}(x, 1^t))$'s is within $1/2t$ of the expected value.) The theorem follows. ■

4.2 On the failure of other versions of Theorem 2.5

Here "Ti-search" stands for *PPT-search problem of Type i*. Recall that Theorem 2.5 asserts that solving (i.e., T4-solving) T4-search problems is reducible in deterministic polynomial-time to pr$\mathcal{BPP}$, and that an analogous result holds for T1-search problems. In contrast, we show that an analogous result does not hold for T2-search problems, and is unlikely also for Type 3. We start with the first claim, which follows from the fact that there exists T2-search problems that cannot be solved by deterministic algorithms.

Proposition 4.2 (T2-solving some T2-search problems cannot be reduced to pr$\mathcal{BPP}$): *There exists a T2-search problem $\overline{R} = (R_{\mathtt{yes}}, R_{\mathtt{no}})$ that cannot be T2-solved by a deterministic algorithm. In particular, T2-solving $\overline{R}$ cannot be reduced in deterministic polynomial-time to a promise problem in* pr$\mathcal{BPP}$.

Proof: Denoting the Kolmogorov complexity of a string x by $K(x)$, consider $R_{\mathtt{yes}} = \{(1^{|x|}, x) : K(x) > |x|/2\}$ and $R_{\mathtt{no}} = \emptyset$. Then, $\overline{R} = (R_{\mathtt{yes}}, R_{\mathtt{no}})$ satisfies the conditions of Definition 2.2 (e.g., Condition 1 is satisfied by an algorithm that, on input 1^n, generates a uniformly distributed n-bit string, whereas Condition 2 holds trivially). On the other hand, no deterministic algorithm can T2-solve $\overline{R}$, because $A(1^n) = x$ implies $K(x) \leq |A| + O(1) + \log_2 n$ (which is smaller than $|x|/2$ if $|x| = n/2$ and A is fixed). Lastly, observe that a deterministic polynomial-time reduction of T2-solving $\overline{R}$ to $\Pi \in$ pr$\mathcal{BPP}$ implies a deterministic algorithm for T2-solving $\overline{R}$, because Π can be decided by a deterministic (exponential-time) algorithm. ■

Proposition 4.3 (on deterministically T3-solving some T3-search problems):

1. *If $\mathcal{P} \neq \mathcal{NP}$, then there exists a T3-search problem $\overline{R} = (R_{\mathtt{yes}}, R_{\mathtt{no}})$ that cannot be T3-solved by a deterministic polynomial-time algorithm.*

2. *If $\mathcal{NP} \not\subseteq \mathcal{BPP}$, then there exists a T3-search problem $\overline{R} = (R_{\mathtt{yes}}, R_{\mathtt{no}})$ that cannot be T3-solved by a deterministic polynomial-time reduction to a promise problem in* pr$\mathcal{BPP}$.

Proof: Let R be the NP-witness relation of some set S in $\mathcal{NP} \setminus \mathcal{P}$, and assume (w.l.o.g.) that $(x,w) \in R$ implies $|w| = 3|x|$. Consider $R_{\mathtt{yes}} = R$ and

$$R_{\mathtt{no}} = \{(x,w) : |w| \neq 3|x|\} \cup \{(x,w) : w \in (\{0,1\}^{3|x|} \setminus R(x)) \ \ \& \ \ K(w) < |w|/2\}$$

where $K(w)$ denotes the Kolmogorov complexity of w. Hence, w is a limbo solution for x if and only if $w \in (\{0,1\}^{3|x|} \setminus R(x))$ and $K(w) < |w|/2 = 3|x|/2$

We first note that $\overline{R} = (R_{\mathtt{yes}}, R_{\mathtt{no}})$ satisfies the conditions of Definition 2.3: Condition 1 is satisfied by an algorithm that, on input x, generates a uniformly distributed $3|x|$-bit string, whereas Condition 2 holds by virtue of $R \in \mathcal{P}$. To show that no deterministic polynomial-time algorithm can T3-solve $\overline{R}$, we consider two cases regarding the possible behavior of a hypothetical deterministic polynomial-time algorithm A that T3-solves $\overline{R}$.

Case 1: A solves R. That is, for all but finitely many x's in S, it holds that $A(x) \in R(x)$. In this case, we obtain a polynomial-time decision procedure for S, in contradiction to the hypothesis $S \notin \mathcal{P}$. (On input x, this procedure invokes A on x, and accepts if and only if $(x, A(x)) \in R$.)

Case 2: A does not solve R. That is, for infinitely many x's in S, it holds that $A(x) \in \{0,1\}^{3|x|}$ and $K(A(x)) \geq 3|x|/2$. But this is impossible because $K(A(x)) \leq |x| + |A| + O(1)$.

This completes the proof of the first part of the proposition. To prove the second part, we follow a similar strategy, except that we use an NP-witness relation R of a set S in $\mathcal{NP} \setminus \mathcal{BPP}$. Letting M be a hypothetical deterministic polynomial-time reduction of T3-solving $\overline{R}$ to $\Pi \in$ pr$\mathcal{BPP}$, we consider the analogous two cases.

Case 1: M^{Π} solves R. That is, for all but finitely many x's in S, it holds that $M^{\Pi}(x) \in R(x)$. In this case, we obtain a PPT decision procedure for S, in contradiction to the hypothesis $S \notin \mathcal{BPP}$. On input x, this procedure invokes M on x, while emulating the oracle calls to Π (and accepts if and only if $(x, A(x)) \in R$).

The complementary case (i.e., Case 2) is handled exactly as before. This completes the proof of the second part of the proposition. ■

Digest. The foregoing proofs capitalize on a concern that was raised in Section 3.3: Definition 2.2 does not allow the searcher to return a limbo solution although this solution may be indistinguishable from a valid solution, whereas Definition 2.3 allows the searcher to return a limbo solution although this solution may be indistinguishable from an invalid solution. Indeed, this is exactly what happens in the proofs of Proposition 4.2 and Proposition 4.3, respectively.

4.3 Emulations

Again, "Ti-search" stands for *PPT-search problem of Type i*. Recall that both T2- and T3-search refer to $\overline{R} = (R_{\mathtt{yes}}, R_{\mathtt{no}})$ that satisfies the identical Condition 2 (i.e., $\overline{R}$ as a decisional promise problem is in pr$\mathcal{BPP}$). The difference between the types is captured by Condition 1: In T2-search

solving requires outputting a valid solution, whereas in T3-search solving allows outputting a limbo solution. In the sequel, we refer to the first notion of solving (i.e., T2-solving) as to strongly-solving, and to the second (relaxed) notion of solving (i.e., T3-solving) as weakly-solving. In addition, we generalize Definition 2.4 so as to allow also functions of the form $q : P \times \{0,1\}^* \to [0,1]$, where $P \subseteq \{0,1\}^*$ is a promise set. In this case, we require:

1. A relaxed solving condition: There exists a PPT algorithm A such that, for every $x \in P$ and $t \in \mathbb{N}$, it holds that $\Pr[q(x, A(x, 1^t)) \geq q'(x) - (1/t)] \geq 2/3$, where $q'(x) \stackrel{\text{def}}{=} \max_y\{q(x,y)\}$.
2. Efficient approximation of q: There exists a PPT algorithm Q such that, for every $x \in P$, $y \in \{0,1\}^*$ and $t \in N$, it holds that $\Pr[|Q(x,y,1^t) - q(x,y)| \leq 1/t] \geq 2/3$.

Lastly, for a binary relation R, we use the notation $S_R \stackrel{\text{def}}{=} \{x : R(x) \neq \emptyset\}$.

Proposition 4.4 (weak emulation of T2-search by T4-search): *For every T2-search problem $\overline{R} = (R_{\mathtt{yes}}, R_{\mathtt{no}})$ there exists an T4-search problem $q : S_{R_{\mathtt{yes}}} \times \{0,1\}^* \to [0,1]$ such that weakly-solving $\overline{R}$ is reducible in deterministic polynomial-time to solving q.*

Note that the hypothesis (that $\overline{R}$ is a T2-search problem) implies that $\overline{R}$ is strongly-solvable, but this does not mean that strongly-solving $\overline{R}$ is reducible in deterministic polynomial-time to solving q. We also note that the T4-search problem q depends on the algorithm that distinguishes between $R_{\mathtt{yes}}$ and $R_{\mathtt{no}}$ (rather than only depending on $(R_{\mathtt{yes}}, R_{\mathtt{no}})$).

Proof: Letting D be an algorithm distinguishing between $R_{\mathtt{yes}}$ and $R_{\mathtt{no}}$ as provided in Definition 2.2, we define the function $q : S_{R_{\mathtt{yes}}} \times \{0,1\}^* \to [0,1]$ such that

$$q(x,y) \stackrel{\text{def}}{=} \min(\Pr[D(x,y) = 1], 2/3),$$

and observe that q can be approximated as required by Condition 2 (of Definition 2.4). Letting A be the algorithm that finds solutions w.r.t to $R_{\mathtt{yes}}$ (per Condition 1 of Definition 2.2), observe that when A is invoked on an input $x \in S_{R_{\mathtt{yes}}}$, it outputs a valid solution, which has value $2/3 = q'(x)$. Hence, q constitutes a T4-search problem.

Next, we reduce weakly-solving $\overline{R}$ to solving q. On input $x \in S_{R_{\mathtt{yes}}}$, we invoke the T4-solver, denoted F, with parameter $t > 3$ (e.g., $t = 4$). In this case (w.p. at least $2/3$), F finds y such that $q(x,y) \geq q'(x) - (1/t) > 2/3 - 1/3 = 1/3$, which means that y is *not* invalid. Hence, $A(x) \stackrel{\text{def}}{=} F(x, 1^4)$ weakly-solves $\overline{R}$. ∎

Remark 4.5 (an alternative version of Proposition 4.4): *In the proof of Proposition 4.4, the function q was restricted to instances in $P = S_{R_{\mathtt{yes}}}$ in order to cope with the fact that Definition 2.2 makes no postulation about the output of the search algorithm in case the instance is not in $S_{R_{\mathtt{yes}}}$. An alternative that allows $P = \{0,1\}^*$ is to augment Definition 2.2 with the postulation that $S_{R_{\mathtt{yes}}}$ is in $\mathcal{BPP}$. The proof of Proposition 4.4 can then be adapted by redefining q such that $q(x,y) \stackrel{\text{def}}{=} 0$ for every $x \notin S_{R_{\mathtt{yes}}}$ and $y \in \{0,1\}^*$.*

Proposition 4.6 (emulation of T4-search by T2-search): *For every T4-search problem $q : P \times \{0,1\}^* \to [0,1]$ there exists a T2-search problem $\overline{R} = (R_{\mathtt{yes}}, R_{\mathtt{no}})$ such that solving q is reducible in deterministic polynomial-time to weakly-solving $\overline{R}$.*

Needless to say, this implies that solving q is reducible in deterministic polynomial-time to strongly-solving $\overline{R}$. Hence, Definitions 2.2 and 2.3 are each at least as expressive as Definition 2.4.

Proof: Given q as in the hypothesis, define

$$\begin{aligned} R_{\mathtt{yes}} &\stackrel{\text{def}}{=} \{(\langle x, 1^t\rangle, y) : q(x,y) \geq q'(x) - (1/t)\} \\ R_{\mathtt{no}} &\stackrel{\text{def}}{=} \{(\langle x, 1^t\rangle, y) : q(x,y) < q'(x) - (2/t)\}. \end{aligned}$$

Using both algorithms associated with q, we can distinguish between $R_{\mathtt{yes}}$ and $R_{\mathtt{no}}$: This is done by approximating $q(x,y)$ as well as $q'(x)$ (using Q and A, respectively). Hence, Condition 2 of Definition 2.2 is satisfied. Turing to Condition 1 (of Definition 2.2), we find a valid solution for $x' = \langle x, 1^t\rangle$ by invoking $A(x, 1^t)$, while observing that (w.p. at least 2/3) the output y satisfies $q(x,y) \geq q'(x) - (1/t)$, which means that $y \in R_{\mathtt{yes}}(x')$. Hence, $\overline{R}$ is a T2-search problem.

Next, we reduce solving q to weakly-solving $\overline{R}$. Let F denote the algorithm that weakly-solves $\overline{R}$, and note that $S_{R_{\mathtt{yes}}} = \{0,1\}^* \times \{1\}^*$. Then, on input $x \in \{0,1\}^*$ and 1^t, we invoke F on input $\langle x, 1^{2t}\rangle$, and obtain a (non-invalid) solution y, which means that $q(x,y) \geq q'(x) - (2/2t)$. Hence, $A(x,1^t) \stackrel{\text{def}}{=} F(x,1^{2t})$ solves q. ■

Digest. Combining Propositions 4.4 and 4.6 we infer that Definitions 2.2 and 2.4 can weakly emulate each other in the sense that for every problem Π in one class there is a problem Π' in the other class such that X-solving Π is reducible to X′-solving Π', where X (resp., X′) equals T3 when the PPT-search problem is of Type 2 and equals T4 otherwise (i.e., when the PPT-search problem is of Type 4). Indeed, the fact that T3-solving rather than T2-solving is used for PPT-search problem of Type 2 is a drawback.

5 Conclusion

As articulated in Section 3.3, we prefer Definition 2.4 over Definitions 2.2 and 2.3, and tentatively suggest using Definition 2.4 as the definition of PPT-search problems.

Our main conceptual consideration is that the quantitative view of the quality of potential solutions is more appealing than the qualitative trichotomy of valid vs limbo vs invalid solutions, let alone the controversial treatment of limbo solutions.

Another important conceptual consideration is having probabilistic complexity classes that are independent of the specific error bound used in their definition (as long as this bound is reasonable). This feature is provided by Definitions 2.1 and 2.4 (but not by Definitions 2.2 and 2.3), because the former classes support error reduction. (In retrospect, it may be nicer to set the quite arbitrary bound in Condition 1 of these definitions to 1/2 (rather than to 2/3).)

Lastly, in our opinion, the fact that Definition 2.4 associates search problems with bivariate functions rather than with binary relations (or pairs of disjoint binary relations) does not diminish its claim for capturing the intuitive notion of PPT-search problems. In general, the formal structure that is used to capture an intuitive notion is unimportant per se; what matters is how well does the entire mechanism (which includes also the notion of solving) refer to the intuitive notion.

References

[1] Shimon Even, Alan L. Selman, and Yacov Yacobi. The Complexity of Promise Problems with Applications to Public-Key Cryptography. *Information and Control*, Vol. 61 (2), pages 159–173, 1984.

[2] Eran Gat and Shafi Goldwasser. Probabilistic Search Algorithms with Unique Answers and Their Cryptographic Applications. *ECCC*, TR11-136, 2011.

[3] Oded Goldreich. *Computational Complexity: A Conceptual Perspective*. Cambridge University Press, 2008.

[4] Oded Goldreich. In a World of P=BPP. In *Studies in Complexity and Cryptography*, LNCS Vol. 6650, Springer, pages 191–232, 2011.

[5] Oded Goldreich. Multi-pseudodeterministic algorithms. *ECCC*, TR19-012, 2019.

[6] Roei Tell. On defining PPT-search problems and PPT-optimization problems. October 2024.