Electronic Colloquium on Computational Complexity, Report No. 133 (2013)

A Structured View on Weighted Counting with Relations to Quantum Computation and Applications

Cassio P. de Campos, Georgios Stamoulis, Dennis Weyland

Dalle Molle Institute for Artificial Intelligence (IDSIA)
Galleria 2, 6928 Manno-Lugano, Switzerland
{cassio,georgios,dennis}@idsia.ch

Abstract

Weighted counting problems are a natural generalization of counting problems where a weight is associated with every computational path and the goal is to compute the sum of the weights of all paths (instead of computing the number of accepting paths). We present a structured view on weighted counting by defining several classes that depend on the range of the function that assigns weights to paths and by showing the relationships between these different classes. These classes constitute strict generalizations of the usual counting problems. Weighted counting allows us to easily cast a number of famous complexity theoretic results in its terms, especially for quantum computation. Moreover, these classes are flexible enough and capture the complexity of various problems in fields such as probabilistic networks and stochastic combinatorial optimization. Using the weighted counting terminology and our results, we are able to greatly simplify and answer some long-standing open questions in those fields.

ISSN 1433-8092

1 Introduction

Counting problems play an important role in Computer Science and can be informally defined as finding the number of solutions (that is, computational paths of polynomial length that end in an accepting state) of a given combinatorial decision problem. Weighted counting problems are a natural generalization of counting problems: a weight is associated with every computational path and the goal is to compute the sum of the weights of all computational paths. Weighted counting has numerous applications in a wide variety of fields, such as quantum computation, stochastic combinatorial optimization, probabilistic graphical models, to name but a few. In many situations it is more natural and more convenient to use weighted counting problems instead of conventional counting problems. For instance, certain proofs can be simplified and stated in a more intuitive manner by using weighted counting. It also offers additional closure properties which do not hold for the conventional counting classes. Because of that and because the computational complexity of weighted counting is closely related to that of conventional counting, the former might be preferred to the latter when studying many computational complexity questions.

In this work, we give compact definitions for different classes of weighted counting problems, which differ only in the range of their weights. In particular, we study the complexity of weighted counting problems using natural, integer, rational and real numbers as weights together with some more restricted cases where weights take values on finite sets such as $\{0, 1\}$ and $\{-1, +1\}$. If one allows only positive integers as weights, it is easy to show that problems remain in #P, the class of (non-weighted) counting problems. If negative values are allowed, even if they are integers, then it is not possible anymore to guarantee that the result is a non-negative value. Therefore, these problems are not in #P for simple technical reasons. In order to study these cases, we adopt the terminology of #P[1]-equivalence to indicate that a problem is #P-hard and belongs to $\mathrm{FP}^{\#\mathrm{P}[1]}$, that is, it can be solved by using at most one call to a #P oracle and by allowing polynomial-time pre-processing of the input and post-processing of the result from that oracle. Using this terminology, we show that the problems using weights from the set $\{-1, 1\}$, and from arbitrary integers, are in fact #P[1]-equivalent, and also that the decision variants of these weighted counting problems remain in PP (the class of problems to decide whether the majority of the polynomial-time computational paths are accepting paths). While some of these results are considered to be known by the community, we are not aware of an explicit and precise treatment of all relevant cases.

The weighted counting problems just described are asking to compute results whose size is always bounded by a polynomial in the input size, but this is not necessarily the case for weighted counting problems when one allows weights to be taken from rational numbers (and obviously real numbers yield the same situation). We discuss this issue in detail and show that even the more general weighted counting problems remain #P[1]-equivalent as long as the size of the output is polynomially bounded in the input size. The corresponding decision variants remain in PP in this situation. However, if the size of the output is not bounded in this way, the problems cannot be #P[1]-equivalent anymore for obvious reasons. In this case, we show that it is still possible to compute arbitrarily good approximations for these problems by reducing them to a single call of a #P[1]-equivalent problem. As far as we know, the situation of the corresponding decision problems remains an open problem.

We also address the relations between complexity classes of weighted counting problems and those of quantum computation. We show that weighted counting can be used to represent the acceptance probability of a quantum Turing machine in a very natural way. Based on this observation, it is possible to derive complexity results regarding quantum computation using very simplified proofs. For instance, we are able to show that BQP $\in$ AWPP using a very simple and short argumentation (BQP is the class of decision problems that can be solved by a quantum Turing machine with bounded error probability and AWPP is a quite restricted gap definable counting class and the best known classical upper bound for BQP). Finally, we give a short and intuitive proof of the recently published result that quantum Turing machines with bounded error can be simulated in a time- and space-efficient way using randomized algorithms with unbounded error that have access to random access memory [38].

We conclude our paper with a detailed discussion of the implications of our results for problems in other fields such as inferences in probabilistic graphical models and stochastic combinatorial optimization problems. Using our terminology and results, it is possible to simplify complexity proofs about those problems, and most importantly to close some important open questions in these fields. Namely, upper bounds for the computational complexity of various computational tasks related to probabilistic graphical models and two-stage stochastic combinatorial optimization problems.

2 Counting Problems

Intuitively, problems in NP are related to the question of whether, for a given input, at least one solution exists (of polynomial size with respect to the input size). In 1979, Valiant [36, 37] generalized this question: instead of trying to answer if a solution exists, we want to know *how many* solutions exist. For this purpose, the complexity class #P has been defined as a class of functions that count the number of solutions (that is, accepting computational paths) of a non-deterministic polynomial-time Turing machine. The class #P can be defined as follows.

Definition 1. *#P is the class of all functions $f : \{0,1\}^\star \to \mathbb{N}$ which can be expressed in the form $f(x) = |\{u : \Sigma(x,u) = 1\}|$, where $\Sigma(x,u)$ is a predicate for strings $x \in \{0,1\}^\star$ and $u \in \{0,1\}^{p(|x|)}$ (where p is some polynomial) that can be verified in polynomial time in the lengths of x and u.*

Since its introduction, #P has been used to characterize the difficulty of counting problems. For this purpose, central is the role of #P-completeness [36, 37] capturing the computational complexity of the *permanent*. Many NP-complete problems have corresponding counting versions that are #P-complete. Surprisingly, also many "easy" problems such as Perfect Matching are #P-complete in their counting versions. The class #P is very powerful. Toda [34] showed that the entire polynomial hierarchy (PH) can be solved in polynomial time using one oracle call to a #P function.

On the negative side, it has been argued that #P might not be the correct class when we are interested in counting. Indeed #P has some (rather obvious) disadvantages: the class #P is not closed under some important operations, as for example subtractions. A direct implication of this fact is that computing the permanent of a matrix with arbitrary integer entries is not in #P for technical reasons. This has been a motivation behind the introduction of GapP [16] (intuitively, a class of functions that represent the "gap" between the number of accepting and rejecting computations of NP machines), where it is claimed that GapP constitutes a natural alternative to #P (see also [17] for a more systematic study of gap definability). Indeed, many computations (such as the permanent over arbitrary integers) outside #P fall now inside GapP. Moreover, GapP inherits the closure properties of #P, such as addition, multiplication and binomial coefficients, and is additionally also closed under subtraction [2, 16].

The closest decision problem class to #P is PP, which asks if the majority (more than half) of the computational paths of a non-deterministic polynomial-time Turing machine accept. The success of GapP is partially due to its usage to show that PP is closed under the operation of intersection (among others) [3] and also due to the fact that the definition of GapP simplifies a number of very important complexity theoretic results. For instance, Toda's famous theorem that the entire polynomial hierarchy is contained in $\mathrm{P}^{\#\mathrm{P}[1]}$ [34] can be cast in terms of GapP [35]. Also the very important result that BQP $\subseteq$ PP [1] can be greatly simplified and improved in terms of GapP. In [19], it is shown that $\mathrm{PP}^{\mathrm{BQP}} = \mathrm{PP}$ using GapP definable functions. Exploiting GapP once more, the same article shows that BQP $\subseteq$ AWPP, which is currently the best classical upper bound for BQP. Regarding the simulation of a BQP machine by a randomized, unbounded error machine, the most efficient simulation is given in a very recent article [38]. We will come back to this discussion in Section 3.2. Despite the obvious success, it seems that GapP cannot capture the complexity of *weighted counting*: every computational path has a weight and we are interested in computing the sum of weights of all such computational paths, which is precisely the focus of this work.

3 Weighted Counting Problems

In this section we give formal definitions for general weighted counting problems and the corresponding decision variants. We then use restrictions on these general definitions to characterize different classes of (weighted) counting problems, including some well-known complexity classes. Later in the section we discuss properties of these classes and known relations among them, including results for quantum computation.

Definition 2 (Weighted Counting Problem)**.** *We are given a polynomial p and a function $w : \{0,1\}^\star \times \{0,1\}^\star \to \mathbb{R}$ that can be approximated by a polynomial time (in the size of the first two arguments and the third argument) computable function $v : \{0,1\}^\star \times \{0,1\}^\star \times \mathbb{N} \to \mathbb{Z}$, such that $|w(x,u) - v(x,u,b)/2^b| \leq 2^{-b}$ for all $x \in \{0,1\}^\star, u \in \{0,1\}^\star, b \in \mathbb{N}$. The weighted counting problem associated with p and w is to compute for $x \in \{0,1\}^\star$ the function*

$$f(x) = \sum_{u \in \{0,1\}^{p(|x|)}} w(x,u).$$

Here, x is the input to the weighted counting problem and w represents the $2^{p(|x|)}$ many weights for a given input x. With the restriction to w, imposed by the approximation property, we limit the range of w to efficiently computable numbers as defined in [28]. A similar notation in the context of quantum computation is used in [38]. For a given rational threshold value we can then define the corresponding decision problem in the following way.

Definition 3 (Weighted Counting Problem, Decision Variant)**.** *We are given a weighted counting problem defined by a polynomial p and a function $w : \{0,1\}^\star \times \{0,1\}^\star \to \mathbb{R}$ as well as a threshold value $t \in \mathbb{Q}$. The corresponding decision problem is to decide for $x \in \{0,1\}^\star$ whether $f(x) \geq t$ or not.*

As mentioned earlier, weighted counting problems may have different characteristics depending on the set from which weights are taken. For any given set $S \subseteq \mathbb{R}$, we define the class $\#\mathrm{P}_S$ to consist of all functions $f : \{0,1\}^\star \to \mathbb{R}$ corresponding to weighted counting problems where the range of w is restricted to S. The class of the corresponding decision problems is then denoted by PP_S. By using this notation, we can define the classes $\#\mathrm{P}_{\{0,1\}}$, $\#\mathrm{P}_{\{-1,1\}}$, $\#\mathrm{P}_{\mathbb{N}}$, $\#\mathrm{P}_{\mathbb{Z}}$, $\#\mathrm{P}_{\mathbb{Q}}$ and $\#\mathrm{P}_{\mathbb{R}}$, as well as the corresponding classes of decision problems $\mathrm{PP}_{\{0,1\}}$, $\mathrm{PP}_{\{-1,1\}}$, $\mathrm{PP}_{\mathbb{N}}$, $\mathrm{PP}_{\mathbb{Z}}$, $\mathrm{PP}_{\mathbb{Q}}$ and $\mathrm{PP}_{\mathbb{R}}$.

It is easy to see that $\#\mathrm{P}_{\{0,1\}}$ is equal to #P [36], the class of (non-weighted) counting problems. The same equality holds for the corresponding classes of decision problems $\mathrm{PP}_{\{0,1\}}$ and PP. Additionally, $\#\mathrm{P}_{\{-1,1\}}$ is equal to GapP [16], the closure of #P under subtraction and therefore $\mathrm{PP}_{\{-1,1\}}$ is equal to the class of decision problems corresponding to GapP.

In the following we further investigate the relations among the other classes of weighted counting problems and their corresponding decision versions. Before proceeding, we must define the terminology of #P[1]-equivalence that will be used throughout:

Definition 4. *A problem is #P[1]-equivalent if it is #P-hard and belongs to $\mathrm{FP}^{\#\mathrm{P}[1]}$ (only one call to the #P oracle is allowed).*

It is well-known that logarithmic (in the input size) many calls to a PP oracle are still in PP, that is, $\mathrm{PP} = \mathrm{P}^{\mathrm{PP}[\log]}$ [3, 18]. The corresponding question for #P is open to the best of our knowledge, in particular it is unknown whether $\mathrm{FP}^{\#\mathrm{P}[1]}$ is strictly contained in $\mathrm{FP}^{\#\mathrm{P}[2]}$ [21, 30]. Considering this fact, our results in the following sections are (to some extent) the best that can be achieved in terms of similarities among these classes.

3.1 Relations among Classes of Weighted Counting Problems

We investigate the relations between different classes of weighted counting problems and their corresponding decision variants. We first show equality of the classes $\#\mathrm{P}_{\{0,1\}}$ and $\#\mathrm{P}_{\mathbb{N}}$, as well as equality of the classes $\#\mathrm{P}_{\{-1,1\}}$ and $\#\mathrm{P}_{\mathbb{Z}}$. This also implies equality of the corresponding classes of decision problems, namely of $\mathrm{PP}_{\{0,1\}}$ and $\mathrm{PP}_{\mathbb{N}}$, and of $\mathrm{PP}_{\{-1,1\}}$ and $\mathrm{PP}_{\mathbb{Z}}$. These results are widely understood as "known" by the community, but to the best of our knowledge they have never been explicitly stated. Since the range of the functions in $\#\mathrm{P}_{\{-1,1\}}$ is not limited to non-negative integers, as it is the case for the functions in $\#\mathrm{P}_{\{0,1\}}$, these two classes cannot be technically equal. Nevertheless, it is possible to show that all these classes are #P[1]-equivalent.

We later focus on weighted counting problems in $\#\mathrm{P}_{\mathbb{Q}}$ and $\#\mathrm{P}_{\mathbb{R}}$ as well as on their corresponding decision variants. We show that the size of the output of weighted counting problems belonging to these classes is not necessarily polynomially bounded in the input size (as opposed to the more restricted classes, where this clearly holds). Therefore, it is generally not possible to give polynomial reductions from problems in $\#\mathrm{P}_{\mathbb{Q}}$ and $\#\mathrm{P}_{\mathbb{R}}$ to any of the more restricted classes. As we will see, it is still possible to reduce these problems to one call of a #P[1]-equivalent problem such that we lose only a small additive approximation error. This result then implies that $\#\mathrm{P}_{\mathbb{Q}}$ and $\#\mathrm{P}_{\mathbb{R}}$ are indeed #P[1]-equivalent as long as we focus on problems whose output is polynomially bounded in the input size. The same holds for the corresponding decision variants.

To show equality between $\#\mathrm{P}_{\{0,1\}}$ and $\#\mathrm{P}_{\mathbb{N}}$, we make use of the following property. Due to the definition of the weight function w and its approximation v, we can assure that the integer part of the weights can always be encoded using polynomially many bits. We then add this polynomial to the given polynomial p and construct a new weight function using only weights of 0 and 1, such that the overall sum does not change. We formalize these ideas in the proof of the following theorem.

Theorem 1. *$\#P_{\{0,1\}} = \#P_{\mathbb{N}}$.*

Proof. Since problems in $\#\mathrm{P}_{\{0,1\}}$ are by definition in $\#\mathrm{P}_{\mathbb{N}}$, we only have to show the other inclusion. For a weighted counting problem in $\#\mathrm{P}_{\mathbb{N}}$ defined by a polynomial p and a weight function $w : \{0,1\}^\star \times \{0,1\}^\star \to \mathbb{N}$, we construct the following weighted counting problem in $\#\mathrm{P}_{\{0,1\}}$. Take the polynomial $p' = p + q$, where q is a polynomial bounding the number of bits required to encode the integer part of the weights of our original problem. Define $w' : \{0,1\}^\star \times \{0,1\}^\star \to \{0,1\}$ by

$$w'(x,u) = \begin{cases} 1 & \text{if } \#u_2 < w(x,u_1) \\ 0 & \text{else,} \end{cases}$$

where u_1 are the first $p(|x|)$ bits of u and $\#u_2$ is the number encoded by the last $q(|x|)$ bits of u. Since the two functions, defined by the original weighted counting problem and the newly constructed weighted counting problem, are identical, we conclude the proof. ☐

A similar argument can be used to show the equality of $\#\mathrm{P}_{\{-1,1\}}$ and $\#\mathrm{P}_{\mathbb{Z}}$.

Theorem 2. *$\#P_{\{-1,1\}} = \#P_{\mathbb{Z}}$.*

On the other hand, the classes $\#\mathrm{P}_{\{0,1\}}$ and $\#\mathrm{P}_{\{-1,1\}}$ cannot be equal for technical reasons (their ranges of output are different). Nevertheless, we can give polynomial reductions from problems of each of these classes to a single call of a problem in the other class. Here we only sketch a proof, since very similar ideas as for the previous proof are used. To show that we can reduce problems in $\#\mathrm{P}_{\{0,1\}}$ to $\#\mathrm{P}_{\{-1,1\}}$, we only have to observe that it is possible to replace a weight of 0 with two weights of values -1 and 1 (together they sum zero). In order to reduce to the opposite direction, one can add a value of 1 to every weight in the problem, obtaining a problem in $\#\mathrm{P}_{\mathbb{N}} = \#\mathrm{P}_{\{0,1\}}$ with weights in $\{0,2\}$. We can retrieve the result of the original problem in $\#\mathrm{P}_{\{-1,1\}}$ by solving this new problem and then subtracting $2^{p(|x|)}$ from the resulting value. We summarize this fact in the following theorem.

Theorem 3. *$\#P_{\{-1,1\}} = \#P_{\mathbb{Z}}$ is $\#P[1]$-equivalent.*

As already mentioned, the size of the output for problems in $\#\mathrm{P}_{\mathbb{Q}}$ and $\#\mathrm{P}_{\mathbb{R}}$ is not necessarily bounded polynomially in the input size. We treat this issue about the size of the output in more detail in the appendix, which is summarized by the following result.

Theorem 4. *Both $\#P_{\mathbb{Q}}$ and $\#P_{\mathbb{R}}$ are **not** $\#P[1]$-equivalent. Moreover, they are **not** in $FP^{\#P}$, that means they cannot even be solved with a polynomial time algorithm having access to a #P oracle.*

Nevertheless, it is possible to achieve a very good approximation result for problems in $\#\mathrm{P}_{\mathbb{R}}$ (and consequently also for those in $\#\mathrm{P}_{\mathbb{Q}}$). In the following theorem we capture the fact that we are able to approximate these problems up to a small additive approximation error using one call to a #P[1]-equivalent problem.

Theorem 5. *We are given a weighted counting problem in $\#P_{\mathbb{R}}$ defined by a polynomial p and a function $w : \{0,1\}^\star \times \{0,1\}^\star \to \mathbb{R}$ as well as a number $b \in \mathbb{N}$. The task of approximating the resulting value of the given problem up to an additive approximation error of 2^{-b} is $\#P[1]$-equivalent.*

Proof. For a given x we can solve the weighted counting problem given by the polynomial p and the integer weight function given by v where the last parameter is fixed to $p(|x|) + b$ with one call to a problem in $\#\mathrm{P}_{\mathbb{Z}}$, which is #P[1]-equivalent due to Theorem 3, followed by a division by $2^{p(|x|)+b}$. Let us call the function computed by this weighted counting problem as f'. The approximation error of the resulting value with respect to the value obtained by the original weighted counting problem satisfies

$$\begin{aligned} \left|f(x) - f'(x)\right| &= \left| \sum_{u \in \{0,1\}^{p(|x|)}} w(x,u) - \sum_{u \in \{0,1\}^{p(|x|)}} v(x,u,p(|x|)+b)/2^{p(|x|)+b} \right| \\ &\leq \sum_{u \in \{0,1\}^{p(|x|)}} \left| w(x,u) - v(x,u,p(|x|)+b)/2^{p(|x|)+b} \right| \\ &\leq \sum_{u \in \{0,1\}^{p(|x|)}} 2^{-p(|x|)-b} \leq 2^{-b}, \end{aligned}$$

which concludes the proof. □

We can now use this result to show that weighted counting problems, for which the output (even if encoded as a fraction) is bounded polynomially in the input size, are #P[1]-equivalent.

Theorem 6. *We are given a weighted counting problem in $\#P_{\mathbb{R}}$ defined by a polynomial p and a function $w : \{0,1\}^\star \times \{0,1\}^\star \to \mathbb{R}$. If the size of the output (even encoded as a fraction) is bounded by a polynomial $q(|x|)$, then the given weighted counting problem is $\#P[1]$-equivalent.*

Proof. Using Theorem 5, we compute with a single call to a problem in #P a value $f'(x)$ with an additive approximation error of at most $2^{-2q(|x|)-2}$. We know that the actual value $f(x)$ is within the interval $\left[f'(x) - 2^{-2q(|x|)-2}, f'(x) + 2^{-2q(|x|)-2}\right]$ of size $2^{-2q(|x|)-1}$. In an interval of this size there is only one rational value which can be encoded by at most $q(|x|)$ many bits. Using the continued fraction expansions of the two interval endpoints we are able to retrieve this rational value efficiently [22]. □

A similar result also holds for the corresponding decision variants. In the following theorem we show that if the size of the output is bounded polynomially in the input size, the decision variant of our weighted counting problem is in fact in PP.

Theorem 7. *We are given the decision variant of a weighted counting problem in $PP_{\mathbb{R}}$ defined by a polynomial p, a function $w : \{0,1\}^\star \times \{0,1\}^\star \to \mathbb{R}$ and a threshold value $t \in \mathbb{Q}$. If the size of the output (even encoded as a fraction) is bounded by a polynomial $q(|x|)$, then the problem is in PP.*

Proof. This proof is also based on Theorem 5. As in the previous proof, we show that we can transform our problem to a problem with integer weights, such that the result of this problem divided by a certain integer gives us a good approximation for our original problem. Here we need two additional properties. First, we want to use an approximation from above, that means an approximation which is always at least as large as the exact value. In this way the approximate result is not smaller than the given threshold value if the exact result is not smaller than the given threshold. Second, we set the accuracy of our approximation such that the approximate value remains smaller than the given threshold value if the exact value is smaller than the threshold value. We can do this efficiently, since there is a certain gap between rational values whose denominators are bounded.

Take the function $v : \{0,1\}^\star \times \{0,1\}^\star \times \mathbb{N} \to \mathbb{Z}$ from Definition 2 in order to obtain a one-side approximation for the weights. For this purpose, simply compute one additional bit using the two-side approximation given by v and shift the result accordingly. In this way, the time bounds are not changed and yet we obtain a function $v' : \{0,1\}^\star \times \{0,1\}^\star \times \mathbb{N} \to \mathbb{Z}$ which approximates the weights and satisfies $0 \leq v'(x,u,b)/2^b - w(x,u) \leq 2^{-b}$ for all $x \in \{0,1\}^\star, u \in \{0,1\}^\star, b \in \mathbb{N}$.

For a given $c \in \mathbb{N}$, we can create the following problem in $\mathrm{PP}_\mathbb{Z}$. Take the same polynomial p as for the given problem. Additionally, take $v'(\cdot,\cdot,p(|x|)+c)$ as the integer weight function and $t' = t2^{p(|x|)+c}$ as the threshold value. Since the approximation error of the weights is always to the same side, it is easy to verify the following property: if the actual result of the original problem is at least as large as the threshold value t, then the result of the new problem is at least as large as the new threshold value t'.

It remains to show that if the actual resulting value of the original problem is smaller than t, then the resulting value of the new problem is also smaller than t'. This can be achieved by using an appropriate accuracy for the approximation, namely a value of $c = q(|x|) + \lceil \log|\text{denominator}(t)| \rceil + 1$, where $\text{denominator}(t)$ is the denominator of the threshold value t. The result of the new problem divided by $2^{p(|x|)+c}$ approximates the actual value with an one-side error of at most 2^{-c}. If the actual result of the original problem is smaller than t, then it differs from t by at least 2^{-c+1}, since the encoding of the denominator of the resulting value is bounded by $q(|x|)$. The approximation of the actual result with an error of at most 2^{-c} is certainly smaller than t and therefore the result of the new problem is smaller than t'.

This concludes the proof, since we have shown that we can polynomially reduce the original problem to a singe call of a problem in $\mathrm{PP}_\mathbb{Z} = \mathrm{PP}$. $\square$

For problems in $\#\mathrm{P}_\mathbb{Q}$ and $\#\mathrm{P}_\mathbb{R}$, these results are probably the best we can hope to obtain, because problems where the size of the output (which inevitable has to be computed) is not polynomially bounded in the input size cannot be #P[1]-equivalent. The situation for the corresponding problems in $\mathrm{PP}_\mathbb{Q}$ and $\mathrm{PP}_\mathbb{R}$ might be different and remains an open problem. Furthermore, weighted counting problems generalize the closure properties of functions in #P and GapP. In addition to the closure properties of functions in GapP, weighted counting is for example also closed under certain linear combinations using rational and even efficiently computable real coefficients.

3.2 Quantum Computation

In this section we investigate the relationship between weighted counting and quantum computation. The most common computational model for quantum computations is the quantum Turing machine. Such a Turing machine is similar to a probabilistic Turing machine. Instead of (positive) transition probabilities, which preserve the L_1 norm, quantum Turing machines use transition amplitudes, which are complex numbers that preserve the L_2 norm. During the execution of a probabilistic Turing machine, the machine is at any point of time in exactly one state (according to the underlying probability distribution). The situation for quantum Turing machines is different. The machine is in a superposition of different states, whose probabilities are defined as the squares of their amplitudes. The exact nature of the state can only be determined through measurements. Such measurements are performed at the end of the computation to reveal the result, but also intermediate measurements are possible. Measuring certain bits destroys the superposition of states in the following sense. Only states which are compatible with the outcome of the measurement survive and the amplitudes change accordingly to preserve the L_2 norm. The acceptance probability of a quantum Turing machine on a given input is the probability with which the final measurements reveal an accepting configuration.

We begin this section by showing that the acceptance probability of a quantum Turing machine can be represented using weighted counting. If there are no intermediate measurements, then the acceptance probability of the given quantum Turing machine is the sum of the acceptance probabilities for accepting configurations. The acceptance probability for an accepting configuration is the absolute square of its amplitude and its amplitude is the sum of the amplitudes of computational paths leading to this state. Finally, we can efficiently compute the amplitude of a given computational path of the quantum Turing machine to any precision we need. The key observation is that we can now express the overall acceptance probability by summing over all pairs of computational paths. We call two computational paths compatible if they arrive at the same accepting configuration. The weight for a pair of computational paths that are compatible is the real part of the product of the amplitude of the first computational path and the complex conjugate of the amplitude of the second computational path. Here we are allowed to consider only the real part of that product, since the imaginary parts cancel each other out over the whole sum. The weight for a pair of non-compatible computational paths is just 0. To allow intermediate measurements, we have to slightly modify the definition of compatible computational paths, but the underlying idea does not change. In the following we state our result in a more formal way.

Theorem 8. *For a given polynomial time quantum Turing machine, the task of computing the acceptance probability on any input is a weighted counting problem in $\#P_{\mathbb{R}}$.*

Proof. We first focus on polynomial time quantum Turing machines without intermediate measurements. After that we show how the result can be extended to the more general case.

Let M be a quantum Turing machine without intermediate measurements and whose computational time is bounded from above by a polynomial t. For a given input x, let us denote the set of possible final configurations of M by C and the set of possible computational paths by P. Note that the cardinality of both sets are at most exponential in the input length and that elements from these sets can be efficiently enumerated. The amplitude for a configuration $c \in C$ is denoted by a_c and the amplitude for a computational path $p \in P$ is denoted by a_p. We can then write the acceptance probability of M for the given input x in the following way (we use the notation $p_1 \sim p_2$ to indicate that two computational paths p_1 and p_2 arrive at the same accepting configuration).

$$\begin{aligned}
\Pr(M \text{ accepts } x) &= \sum_{c\in C} \left|a_c^2\right| = \sum_{c\in C} \left| \sum_{\substack{p\,\in\,P \\ \text{arriving at } c}} a_p \right|^2 \\
&= \sum_{c\in C} \sum_{\substack{p_1,p_2\,\in\,P \\ \text{both arriving at } c}} a_{p_1}\overline{a_{p_2}} \\
&= \sum_{\substack{p_1,p_2\,\in\,P \\ p_1\sim p_2}} a_{p_1}\overline{a_{p_2}} \;=\; \sum_{\substack{p_1,p_2\,\in\,P \\ p_1\sim p_2}} \operatorname{Re}\left(a_{p_1}\overline{a_{p_2}}\right)
\end{aligned}$$

To compute the acceptance probability, we have to sum over all pairs of computational paths. If two computational paths arrive at the same accepting configuration, we call them compatible. The weight for two compatible computational paths is the real part of the product of the amplitude of the first path with the complex conjugate of the amplitude of the second part. The weight for two non-compatible paths is just 0. To approximate the product of the amplitudes of two computational paths up to a specified precision, we just have to multiply the transition amplitudes between all the computational steps of the two paths with a certain precision. That means we can express the acceptance probability of M as a weighted counting problem.

Now we still have to show that the result also holds if we allow intermediate measurements. For this purpose we have to adapt our definition of compatible computational paths. We now say that two computational paths are compatible if they arrive at the same accepting configuration and if they have the same results for intermediate measurements. We capture this extended definition of two compatible computational paths p_1 and p_2 by the same notation of $p_1 \sim p_2$. The general formula is then

$$\Pr(M \text{ accepts } x) \;=\; \sum_{\substack{p_1,p_2\,\in\,P \\ p_1\sim p_2}} \operatorname{Re}\left(a_{p_1}\overline{a_{p_2}}\right).$$

To show that this formula is indeed correct, we postpone the intermediate measurements using a standard technique. Here instead of an intermediate measurement, a CNOT operation is performed on the bits that have to be measured and special ancilla bits. Then at the very end a measurement on the ancilla bits is performed. We call this quantum Turing machine M'. M and M' have the same acceptance probabilities and in particular they accept the same language. Now our extended definition of compatible computational paths respects the following property: Two computational paths are compatible for M according to our extended definition if and only if the corresponding two computational paths are compatible for M' according the basic definition. This shows that our result also holds for the more general case in which intermediate measurements are allowed. □

Please note, that we do not impose any assumptions about the quantum Turing machines in the previous proof. In particular, we do not impose any restrictions on the transition amplitudes or on the measurements used. Based on Theorem 8, we are now able to give extremely short and intuitive proofs of the well-known facts that BQP $\in$ PP and (the stronger version) that BQP $\in$ AWPP.

Definition 5. *A language L is in BQP if it can be decided by a polynomial time quantum Turing machine with an error probability of at most* $1/3$.

Theorem 9. *BQP $\in$ PP.*

Proof. For any given language $L \in$ BQP, there exists a bounded error quantum Turing machine M which decides L. We can now apply Theorem 8 to M and obtain a weighted counting problem approximating the acceptance probabilities of M. The (approximative) decision version corresponding to this weighted counting problem is in PP and can be used to decide L due to the gap between the acceptance probabilities for words inside and outside the language. $\square$

For the second proof we need an alternative definition of the complexity class AWPP. In [15] the following definition for AWPP is given.

Definition 6. *A language L is in AWPP if and only if there exists a polynomial p and a GapP function g such that, for all $x \in \Sigma^\star$,*

$$\begin{aligned} x \in L \quad &\Rightarrow \quad 1 - c \leq g(x)/2^p \leq 1, \\ x \notin L \quad &\Rightarrow \quad 0 \leq g(x)/2^p \leq c, \end{aligned}$$

where $p = p(|x|)$ and c is a constant smaller than $1/2$.

We now give an equivalent definition based on weighted counting.

Theorem 10. *A language L is in AWPP if and only if there exists a function g corresponding to a weighted counting problem such that, for all $x \in \Sigma^\star$,*

$$\begin{aligned} x \in L \quad &\Rightarrow \quad 1 - c \leq g(x) \leq 1, \\ x \notin L \quad &\Rightarrow \quad 0 \leq g(x) \leq c, \end{aligned}$$

where c is a constant smaller than $1/2$.

Proof. Due to Definition 6, it is clear that for a language $L \in$ AWPP there exists a function g corresponding to a weighted counting problem with the above properties. Now let us assume that the above properties are fulfilled for a function g corresponding to a weighted counting problem. We can approximate the weighted counting problem using a GapP function and a power of 2 as a scaling factor. For a sufficiently good approximation we are then able to fulfill the requirements of Definition 6 with a constant smaller than $1/2$. $\square$

This alternative definition can now be used for a simplified proof of BQP $\in$ AWPP.

Theorem 11. *BQP $\in$ AWPP.*

Proof. For any given language $L \in$ BQP, there exists a bounded error quantum Turing machine M which decides L. By applying Theorem 8 to M, we obtain a weighted counting problem approximating the acceptance probability of M. Since M has a bounded error probability, we can show $L \in$ AWPP due to Theorem 10. $\square$

Using the framework of weighted counting, it might be also possible to give simplified proofs for other results in the field of quantum computation. As an example, we give a shorter proof for a recent result [38] in the remaining part of this section. In [38], it is shown among other results that quantum Turing machines with bounded error can be simulated in a time- and space-efficient way using randomized algorithms with unbounded error that have access to random access memory. The most important version of this result has been stated in the following way.

Theorem 12 (Randomized Simulation, Theorem 1.1 in [38])**.** *Every language solvable by a bounded-error quantum algorithm running in time $t \geq \log n$ and space $s \geq \log n$ with algebraic transition amplitudes is also solvable by an unbounded-error randomized algorithm running in time $\mathcal{O}(t \log t)$ and space $\mathcal{O}(s + \log t)$, provided t and s are constructible by a deterministic algorithm with the latter time and space bounds.*

Proof. Let M be a given quantum Turing machine. We use again the formulation of the acceptance probability as a weighted counting problem:

$$\Pr(M \text{ accepts } x) \;=\; \sum_{\substack{p_1, p_2 \,\in\, P \\ p_1 \sim p_2}} \operatorname{Re}\left(a_{p_1} \overline{a_{p_2}}\right).$$

Based on this formulation, we can easily build a randomized algorithm with unbounded error: the problem of computing the acceptance probability of M is in $\#\mathrm{P}_{\mathbb{R}}$. The corresponding decision version is in $\mathrm{PP}_{\mathbb{R}}$ and a proper approximation resides in PP, which is a randomized algorithm with unbounded error. To show the desired time and space bounds, we have to bound the running time and the space used for the computation of each of the summands. For this purpose, one can use approximations of the constant number of transition amplitudes of M with a precision of $\mathcal{O}(\log t)$ bits. According to [5], the resulting quantum Turing machine is a good approximation to M. Note that the result was originally stated for quantum Turing machines without intermediate measurements, but with a similar argument as in the proof of Theorem 8, this result can be extended to the more general case. We now continue our investigations with this machine. We basically follow the proof of [38], but we employ our new terminology, which can help some passages to be stated in a much simpler way.

In a preprocessing step, we compute the (constant number of) transition amplitudes of M up to a precision of $\mathcal{O}(\log t)$ bits. This can be done for algebraic transition amplitudes in a computational time of $\mathcal{O}(\text{polylog}(t))$ and space $\mathcal{O}(\log t)$. Due to the random access of the randomized machine, we can retrieve these values efficiently later whenever they are required. Now the idea is to simulate all pairs of computational paths in parallel, step by step. Again due to the random access of the randomized machine, we can jump between the two computational paths without an additional overhead. For the moment we additionally keep track of the transition amplitudes of the paths. This causes some overhead that would invalidate the overall results, but we demonstrate how to avoid such overhead later. Whenever a quantum measurement occurs, we check if the two computational paths would result in the same measurement. If this is indeed the case, then we continue with the simulation, otherwise we stop and assign to this path a weight of 0. At the end, we check if both paths have arrived at the same accepting configuration. If that is the case, then we assign the real part of the product of the amplitude of the first path with the complex conjugate of the amplitude of the second part as the weight, otherwise we use a weight of 0. The resulting randomized algorithm shows the desired behavior, but does not respect the required time and space bounds due to the overhead needed to keep track of the transition amplitudes of the two computational paths. In the following we show how to fix this issue.

In order to avoid the overhead needed to keep track of the transition amplitudes of the two computational paths, one must realize that it is not even necessary to keep track of the weights throughout the whole computation. Instead, we may split the amplitudes in positive/negative and real/imaginary parts and perform random branchings at every computational step, instead of multiplying with the transition amplitudes. For that to work, the resulting weights of all the paths have to be adapted accordingly. If we look at time and space requirements of this approach, we see that, at each computational step, we have to generate $\mathcal{O}(\log t)$ random bits for the branching, which results in a computational time of $\mathcal{O}(t \log t)$ and a space of $\mathcal{O}(s + \log t)$, as desired. $\square$

We shall emphasize that these bounds rely heavily on the fact that random access to memory is granted to the randomized algorithm. Otherwise, we would obtain slightly weaker bounds for the general case. With a more elaborate approach it would still be possible to obtain the same bounds without random access to memory if the number of quantum measurements is $\mathcal{O}((\log t)^2)$. In this case, we have to move the pre-computed transition amplitudes during the simulation process to be always able to access them efficiently. This approach does not yield any overhead. Additionally, we simulate each of the computational paths until $\mathcal{O}(\log t)$ measurements occur. We must keep track of the measurement results and switch to the other path. In total, there are at most $\mathcal{O}(\log t)$ switches which cause a time overhead of $\mathcal{O}(s)$, which can be bounded from above by $\mathcal{O}(t)$.

3.3 Related work

There are some results concerning weighted counting complexity for specific problems. However, to the best of our knowledge, there is no systematic discussion on the aforementioned classes in the literature so far. For example, in the classical "textbook" proof that computing the permanent of a matrix $M \in \mathbb{Z}^{n\times n}$ is #P[1]-equivalent (using our notation) [4], first a reduction to the computation of the permanent of another matrix $M' \in \mathbb{Z}^{n\times n}_{\geq 0}$ is performed, and from there a reduction to the computation of the permanent of a matrix M'' with 0/1 entries is shown. This is an example of an specific counting problem over arbitrary integers that has been reduced to a classic 0/1 counting problem.

In another line of research, the complexity of counting versions of Constraint Satisfiability Problems (CSP) were investigated. A CSP can be formulated as follows: Let D be an arbitrary finite set called the domain set. Let $\mathcal{R} = \{R_1, R_2, \ldots, R_k\}$ be an arbitrary finite set of relations on D. Each R_i has an arity $n_i \in \mathbb{N}$. As input we have variables $x_1, \ldots, x_n$ over D, and a collection of constraints $R \in \mathcal{R}$, each applied to a sequence of variables. The question is whether there is an assignment that satisfies all the constraints and the corresponding counting version asks to compute the number of satisfying assignments. In other words, one could identify each $R \in \mathcal{R}$ with a binary valued function $\phi(R)$. Then a counting CSP problem can be seen as the evaluation of the following so-called *partition function* on an input instance I, $P(I) = \sum_{x_i \in D} \prod_{f\in I} f(x_{i_1}, \ldots, x_{i_r})$, where f has arity r and is applied to variables $x_{i_1}, \ldots, x_{i_r}$. Of course if f is 0/1 valued, this counts the number of solutions. If ϕ can take arbitrary values, then we have a *weighted* CSP problem. See for example [7] and [12].

Feder and Vardi [14] conjectured that a CSP over constraints $\mathcal{R}$ is either in P or NP-hard. This is a major open problem in the TCS community. In a very recent breakthrough, Cai and Chen [9] proved a dichotomy theorem for the counting versions of CSPs over complex weights: Given any finite D and any finite set of constraint functions $\mathcal{F}$, where $f_i : D^{n_i} \to \mathbb{C}$, the problem of computing the partition function $P(I)$ is either in P or #P-hard. On the negative side, the criteria are not known to be computable, but still the dichotomy exists. See also [12, 10, 7] for earlier results in the same line of research. Another similar dichotomy conjecture, but yet to be proved, exists for #CSP, namely that any #CSP problem is either in FP or #P-hard, which are in some sense the analogues of P and NP for counting problems. See [8] for some partial results regarding this conjecture.

Another related result appears in the (also very recent) work of of Bläser and Curticapean [6], in which the #W[1]-hardness of the weighted counting of all k-matchings in a bipartite graph has been shown. Here, the weight of a matching is simply the product of the weights of the edges that constitute this matching.

In yet another line of research, and more relevant to our study, an important class of weighted counting problems has been defined by Goldberg and Jerrum [20].[1] This is the class of functions $f : \Sigma^* \to \mathbb{Q}$ over a domain Σ that can be written as the division of a #P function by an integer value computable in polynomial time. This class was used to classify the complexity of approximating the value of the Tutte polynomials of a graph that takes as argument arbitrary rational numbers. The Tutte polynomial of a given graph G is a two-variable polynomial $T_G(x, y)$. Usually, the arguments x, y are integers (not necessarily positive), but the authors were interested in the most general case where x, y are arbitrary rationals. Tutte polynomials can encode a large number of interesting graph theoretic properties. For example, given a graph G, $T(1, 1)$ counts the number of spanning trees in G, $T(2, 1)$ counts the number of forests in G, and so on. Tutte polynomials are also very closely connected to chromatic polynomials, flow polynomials, etc. This class is a strict subclass of the class $\#\mathrm{P}_{\mathbb{Q}}$ defined here earlier. The complexity of exactly evaluating the Tutte polynomial is #P-hard [23], except for some threshold cases. The authors of this latter work were interested in some dichotomy results and they significantly widened the cases where there exists a Fully Polynomial-time Randomized Approximation Scheme (FPRAS) for $T_G(x, y)$, as well as showed that some other particular cases do not have a FPRAS (modulo RP $\neq$ NP).

4 Applications of Weighted Counting

In this section we take our previous discussions and results into two very relevant problems. First we discuss on inferences in the so called probabilistic graphical models, which appear in abundance in artificial intelligence, data mining and machine learning. Then we talk about stochastic combinatorial optimization problems, which represent a very important class of problems in operations research, with applications in numerous fields. Our approach focuses on using the complexity results presented so far to simplify or even to prove new complexity results for this problems.

[1]This class was also denoted by $\#\mathrm{P}_{\mathbb{Q}}$ in their work. As discussed above, it is **strictly** included in the class $\#\mathrm{P}_{\mathbb{Q}}$ defined in this work. In any case, throughout the whole paper $\#\mathrm{P}_{\mathbb{Q}}$ always refers to the class of weighted counting problems with rational weights defined in this work and there should not be any danger of confusion.

4.1 Probabilistic Graphical Models

We present an application of our results to prove that *predictive inferences in probabilistic graphical models* (and especially in Bayesian networks [24, 29]) is #P[1]-equivalent (predictive inferences are also called *belief updating* in this context). Historically, the community working with probabilistic graphical models has been used to cite papers that only partially resolve this question [26, 27, 32]. The most cited work is due to Roth [32], where hardness for #P is demonstrated, but membership is only superficially discussed and no formal proof is given. This issue is acknowledged many years later by Kwisthout [25], who in an attempt to close this question proves that predictive inference is in the so-called #P *modulo normalization* class. This situation is indeed inevitable, because the output here should be a probability value, so the problem cannot be in #P for obvious technical reasons. However, using our terminology and results, we can state that such problems are #P[1]-equivalent. By applying Theorem 6, we obtain a simple alternative (and yet more powerful) proof than [25], as we discuss in the sequel of this section.

Let $\mathcal{J} = \{1, \ldots, n\}$ and $X_{\mathcal{J}} = (X_1, \ldots, X_n)$ be a vector of discrete random variables, $\mathcal{J}_1, \ldots, \mathcal{J}_m$ be a collection of index sets satisfying $\mathcal{J}_1 \cup \cdots \cup \mathcal{J}_m = \mathcal{J}$, and $\mathcal{P} = \{\phi_1, \ldots, \phi_m\}$ be a set of functions over vectors $X_{\mathcal{J}_1}, \ldots, X_{\mathcal{J}_m}$ to non-negative rational numbers, respectively. We call $\mathcal{P}$ a *probabilistic graphical model* for $X_{\mathcal{J}}$ if the functions in $\mathcal{P}$ specify a joint probability distribution over assignments $x_{\mathcal{J}} \in X_{\mathcal{J}}$ by

$$\Pr(X_{\mathcal{J}} = x_{\mathcal{J}}) = \frac{1}{Z} \prod_{i \in \{1,\ldots,m\}} \phi_i(x_{\mathcal{J}_i}),$$

where $Z = \sum_{x_{\mathcal{J}} \in X_{\mathcal{J}}} \prod_{i \in \{1,\ldots,m\}} \phi_i(x_{\mathcal{J}_i})$ is a normalizing value known as the *partition function* (this is somewhat similar to the definition used earlier for CSPs). By default, the functions in $\mathcal{P}$ are assumed to be given by explicit tables with the mapping from elements $x_{\mathcal{J}}$ to non-negative rationals. When that is not the case, then we will assume that functions in $\mathcal{P}$ can be approximately computed as described in Theorem 5. In this case, we call the models using such functions $\mathcal{P}$ as *generalized probabilistic graphical models* (this is an extension of the usual definition commonly found in the literature [24] in order to allow more elaborate functions). Note that the well-known Bayesian networks [29] are nicely encompassed as subcase. A Bayesian network is a probabilistic graphical model that satisfies the following properties: (i) it has exactly one ϕ_i for each X_i; (ii) $\mathcal{J}_i$ must be such that $i \in \mathcal{J}_i$, and such that $j \notin \mathcal{J}_i$ whenever $j > i$; (iii) $\sum_{x_i \in X_i} \phi_i(x_{\mathcal{J}_i}) = 1$. Such restrictions naturally imply $Z = 1$ and induce conditional stochastic independences among variables $X_{\mathcal{J}}$ of the domain.

The *predictive inference* task can be succinctly defined as: given $\mathcal{P}$, compute Z. In fact, this is also known as the *partition function computation*, which turns out to be a general task that can be used to compute the probability value $\Pr(X_{\mathcal{J}'} = x_{\mathcal{J}'})$ for any event $x_{\mathcal{J}'}$ of interest: One has simply to take their specification of the probabilistic graphical model and include into it the indicator functions $\prod_{j \in \mathcal{J}'} \psi_{x_j}(X_j = x_j)$. It is not hard to check that Z equals to $\Pr(X_{\mathcal{J}'} = x_{\mathcal{J}'})$ for a Bayesian network that is extended with these new indicator functions. Because of that, we call this probabilistic graphical model where Z equals to a probability value of interest as *queried Bayesian network*.

The complexity of predictive inference depends on how the input is encoded. When all functions in $\mathcal{P}$ are given by numbers directly encoded in the input, it is easy to show that the output has size that is polynomial in the input size (one could multiply all rational numbers in the input by their least common multiple in order to achieve an input defined solely by integers – this is done in polynomial time because all numbers are explicitly given). Hence the following theorem holds for any probabilistic graphical model, including queried Bayesian networks.

Theorem 13. *Given a probabilistic graphical model defined by $\mathcal{P}$, the predictive inference (a.k.a. computation of the partition function) is $\#P[1]$-equivalent.*

In the case of the generalized probabilistic graphical models, where there are no restrictions on the input functions besides those of Theorem 5, we might end up with a large output size, but we can nevertheless show membership in $\#\mathrm{P}_{\mathbb{Q}}$, which is straightforward.

Theorem 14. *Given a generalized probabilistic graphical model defined by $\mathcal{P}$, the predictive inference is in $\#P_{\mathbb{Q}}$.*

Proof. Take the computation paths to correspond to values $x_{\mathcal{J}} \in X_{\mathcal{J}}$ and define the weight w of a path $x_{\mathcal{J}}$ to be the product $\prod_i \phi_i(x_{\mathcal{J}_i})$ of rational functions in $\mathcal{P}$. The sum of rational weights gives exactly the desired value of Z. $\square$

Regarding the decision version of *predictive inference*, where one queries whether Z is greater than a given threshold, the membership in PP is often attributed to Littman et al. [26], where pertinence of a similar (yet not equal) problem, namely probabilistic acyclic planning, is shown by the construction of a non-deterministic Turing machine with probability of acceptance greater than half. Such result, if manipulated properly, implies membership for the predictive inference in probabilistic graphical models too, but it is valid only in cases where the encoding of the instances satisfy some (restrictive) properties. This issue makes that result only partially satisfactory. Recently, Kwisthout [25] settles the membership of predictive inference in PP for queried Bayesian networks, but it does not extend to the generality of probabilistic graphical models as defined here. Hence, we obtain a stronger membership result, because we require only the output size to be polynomially bounded in the input size. Moreover, this is not restricted to queried Bayesian networks but works for any probabilistic graphical model, including those with functions that are parametric and shortly encoded (as long as they can be well approximated in polynomial time). In summary, our results lead to a simple proof that generalizes previous results for this problem [11, 25, 26, 27].

Theorem 15. *Given a generalized probabilistic graphical model defined by $\mathcal{P}$ such that its output size is known to be polynomial in the input size, the decision version of predictive inference is in PP.*

Proof. By applying Theorems 14 and 7, the result follows. $\square$

4.2 Stochastic Combinatorial Optimization

In this section we present another application of our results, this time in the context of discrete two-stage stochastic combinatorial optimization problems [33]. Dyer and Stougie [13] have shown that discrete two-stage stochastic combinatorial optimization problems are #P-hard in general. In order to obtain this result, they make use of some artificial stochastic combinatorial optimization problems. The result has then been strengthened in [39], where #P-hardness has been shown for a practically relevant stochastic vehicle routing problem. These results are both imposing lower bounds on the computational complexity of stochastic combinatorial optimization problems. Here we complement them with upper bounds for the computational complexity of many tasks related to discrete two-stage stochastic combinatorial optimization problems.

Two-stage stochastic combinatorial optimization problems contain uncertainty in terms of stochastic data in their problem formulation. This uncertainty can for example be given by probability distributions over events of the domain. We call a specific realization of the random events a scenario and we assume that the number of different scenarios is bounded exponentially in the input size. We further assume that these scenarios can be enumerated efficiently and that the probability that a given scenario occurs can be computed in polynomial time. In the first stage a decision is made solely based on the given information, without knowing the actual realizations of the random events. This first-stage decision imposes certain costs. In the second stage, after the realization of the random events is revealed, another decision has to be taken based on the first-stage decision and on the revealed information. This second-stage decision can for example be used to guarantee feasibility of the final solution or to react to certain random events. The second-stage decision causes additional costs which are usually called recourse costs. The overall goal is to find a solution for the given two-stage stochastic combinatorial optimization problem which minimizes the total expected costs, which is defined by the costs of the first-stage decision plus the expected costs of the second-stage decision.

In this context the actual solution for a two-stage stochastic combinatorial optimization problems is usually the first-stage decision. This will become more clear with the following additional assumptions. Assume now that the costs for a first-stage decision can be computed in polynomial time and that for a given first-stage decision and a given scenario the corresponding recourse costs can also be computed in polynomial time. Given a solution we can compute the expected costs in the following way. By definition we are able to compute the costs imposed by the first-stage decision in polynomial time. We then enumerate the at most exponentially many scenarios and add to the total costs for each scenario the recourse costs of this scenario multiplied with the probability that this scenario occurs. Using Theorem 5 we can prove the following result for the evaluation of solutions.

Theorem 16. *We are given a discrete two-stage stochastic combinatorial optimization problem (respecting our assumptions) and a value $b \in \mathbb{N}$. The task of computing the expected costs for a solution up to an additive error of 2^{-b} is $\#P[1]$-equivalent.*

For most of the discrete two-stage stochastic combinatorial optimization problems it holds that the expected costs for any solution can be encoded using at most polynomially many bits in the input size. In fact, we are not aware of any problem of practical relevance in which this is not the case. Using this additional assumption we can prove the following results.

Theorem 17. *We are given a discrete two-stage stochastic combinatorial optimization problem (respecting our extended assumptions). Then the following results regarding the computational complexity of different computational tasks related to the given problem hold:*

(i) The task of computing the expected costs for a solution is $\#P[1]$-equivalent.

(ii) The problem of deciding if a given solution has expected costs of at most $t \in \mathbb{Q}$ is in PP.

(iii) The problem of deciding if a solution with expected costs bounded by $t \in \mathbb{Q}$ exists is in $NP^{\#P[1]}$.

(iv) The problem of computing a solution with minimum expected costs can be solved in polynomial time with access to an $NP^{\#P[1]}$ oracle.

These results open some interesting paths for further research. First of all, they describe upper bounds for the complexity of various computational tasks related to two-stage stochastic combinatorial optimization. In [39], it has already been shown that the upper bound of the first result in Theorem 17 is tight for a practically relevant problem. It remains to answer whether there are also practically relevant problems whose decision/optimization variants match the corresponding upper bounds given here.

5 Conclusions

We have presented a structured view on weighted counting. We have shown that weighted counting problems are a natural generalization of counting problems and that in many cases the computational complexity of weighted counting problems corresponds to that of conventional counting problems. The computational complexity of decision problems in $\mathrm{PP}_{\mathbb{Q}}$ and $\mathrm{PP}_{\mathbb{R}}$, where the size of the output is not necessarily polynomially bounded in the input size, remains an interesting open problem. As for conventional counting problems, it is also of great interest to improve our understanding of the (polynomial time) approximability and inapproximability of weighted counting problems.

Additionally, we have seen that weighted counting problems arise in many different fields. Using the framework of weighted counting we could give more intuitive and simpler proofs for known results in quantum computation. We could even obtain new results regarding probabilistic graphical models and two-stage stochastic combinatorial optimization based on weighted counting. Finally, we believe that our results regarding weighted counting have many more applications in other situations and fields and we hope that our structured approach to weighted counting might help in revealing such relations in the near future.

Acknowledgments

The first author has been supported by the Hasler Foundation under grant number 10030 and the work of the second author has been partially supported by the Swiss National Science Foundation under project number 200020-122110/1 ("Approximation Algorithms for Machine Scheduling Through Theory and Experiments III").

References

[1] L.M. Adleman, J. DeMarrais, and M.A. Huang. Quantum computability. *SIAM J. Comput.*, 26(5):1524–1540, 1997.

[2] R. Beigel. Closure properties of GapP and #P. In *ISTCS*, pages 144–146, 1997.

[3] R. Beigel, N. Reingold, and D.A. Spielman. PP is closed under intersection. *J. Comput. Syst. Sci.*, 50(2):191–202, 1995.

[4] A. Ben-Dor and S. Halevi. Zero-one permanent is #P-complete, a simpler proof. In *ISTCS*, pages 108–117, 1993.

[5] E. Bernstein and U. Vazirani. Quantum complexity theory. In *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, pages 11–20. ACM, 1993.

[6] M. Bläser and R. Curticapean. Weighted counting of k-matchings is #W[1]-hard. In D.M. Thilikos and G.J. Woeginger, editors, *IPEC*, volume 7535 of *Lecture Notes in Computer Science*, pages 171–181. Springer, 2012.

[7] A. Bulatov, M. Dyer, L.A. Goldberg, M. Jalsenius, M. Jerrum, and D. Richerby. The complexity of weighted and unweighted #CSP. *J. Comput. Syst. Sci.*, 78(2):681–688, 2012.

[8] A.A. Bulatov and V. Dalmau. Towards a dichotomy theorem for the counting constraint satisfaction problem. In *FOCS*, pages 562–571. IEEE Computer Society, 2003.

[9] J.Y. Cai and X. Chen. Complexity of counting CSP with complex weights. In H.J. Karloff and T. Pitassi, editors, *STOC*, pages 909–920. ACM, 2012.

[10] J.Y. Cai, X. Chen, and P. Lu. Non-negatively weighted #CSP: An effective complexity dichotomy. In *IEEE Conference on Computational Complexity*, pages 45–54, 2011.

[11] A. Darwiche. *Modeling and Reasoning with Bayesian Networks.* Cambridge University Press, 2009.

[12] M. Dyer, L.A. Goldberg, and M. Jerrum. The complexity of weighted boolean CSP. *SIAM J. Comput.*, 38(5):1970–1986, 2009.

[13] M. Dyer and L. Stougie. Computational complexity of stochastic programming problems. *Mathematical Programming*, 106(3):423–432, 2006.

[14] T. Feder and M.Y. Vardi. The computational structure of monotone monadic SNP and constraint satisfaction: A study through datalog and group theory. *SIAM J. Comput.*, 28(1):57–104, 1998.

[15] S.A. Fenner. PP-lowness and a simple definition of AWPP. *Theory of Computing Systems*, 36(2):199–212, 2003.

[16] S.A. Fenner, L. Fortnow, and S.A. Kurtz. Gap-definable counting classes. *J. Comput. Syst. Sci.*, 48(1):116–148, 1994.

[17] S.A. Fenner, L. Fortnow, and L. Li. Gap-definability as a closure property. *Inf. Comput.*, 130(1):1–17, 1996.

[18] L. Fortnow and N. Reingold. PP is closed under truth-table reductions. *Information and Computation*, 124(1):1 – 6, 1996.

[19] L. Fortnow and J.D. Rogers. Complexity limitations on quantum computation. *J. Comput. Syst. Sci.*, 59(2):240–252, 1999.

[20] L.A. Goldberg and M. Jerrum. Inapproximability of the tutte polynomial. *Inf. Comput.*, 206(7):908–929, 2008.

[21] F. Green, J. Kobler, K.W. Regan, T. Schwentick, and J. Toran. The power of the middle bit of a #P function. *Journal of Computer and System Sciences*, 50(3):456 – 467, 1995.

[22] G.H. Hardy and E.M. Wright. *An introduction to the theory of numbers.* Oxford University Press, 1979.

[23] F. Jaeger, D.L. Vertigan, and D.J.A. Welsh. On the computational complexity of the Jones and Tutte polynomials. *Mathematical Proceedings of the Cambridge Philosophical Society*, 108(1):35–53, 1990.

[24] D. Koller and N. Friedman. *Probabilistic Graphical Models.* MIT press, 2009.

[25] J. Kwisthout. The computational complexity of probabilistic inference. Technical report, Faculty of Science, Radboud University Nijmegen, 2011.

[26] M. Littman, J. Goldsmith, and M. Mundhenk. The computational complexity of probabilistic planning. *Journal of Artificial Intelligence Research*, 9:1–36, 1998.

[27] M. Littman, S.M. Majercik, and T. Pitassi. Stochastic boolean satisfiability. *Journal of Automated Reasoning*, 27(3):251–296, 2001.

[28] M.L. Minsky. *Computation.* Prentice-Hall, 1967.

[29] J. Pearl. *Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference.* Morgan Kaufmann, San Mateo, 1988.

[30] K.W. Regan and T. Schwentick. On the power of one bit of a #P function. In *Proceedings of the Fourth Italian Conference on Theoretical Computer Science*, pages 317–329. World Scientific, 1992.

[31] J. Rosser and L. Schoenfeld. Approximate formulas for some functions of prime numbers. *Illinois Journal of Mathematics*, 6:64–94, 1962.

[32] D. Roth. On the hardness of approximate reasoning. *Artificial Intelligence*, 82(1-2):273–302, 1996.

[33] L. Stougie and M.H. Van Der Vlerk. *Stochastic integer programming.* Institute of Actuarial Sciences & Econometrics, University of Amsterdam, 1996.

[34] S. Toda. PP is as hard as the polynomial-time hierarchy. *SIAM J. Comput.*, 20(5):865–877, 1991.

[35] S. Toda and M. Ogiwara. Counting classes are at least as hard as the polynomial-time hierarchy. *SIAM J. Comput.*, 21(2):316–328, 1992.

[36] L.G. Valiant. The complexity of computing the permanent. *Theor. Comput. Sci.*, 8:189–201, 1979.

[37] L.G. Valiant. The complexity of enumeration and reliability problems. *SIAM J. Comput.*, 8(3):410–421, 1979.

[38] D. van Melkebeek and T. Watson. Time-space efficient simulations of quantum computations. *Theory of Computing*, 8(1):1–51, 2012.

[39] D. Weyland, R. Montemanni, and L.M. Gambardella. Hardness results for the probabilistic traveling salesman problem with deadlines. In *Proceedings of ISCO 2012 - The 2nd International Symposium on Combinatorial Optimization*, 2012.

A The Size of the Output of Problems in $\#\mathrm{P}_{\mathbb{Q}}$ and $\#\mathrm{P}_{\mathbb{R}}$

In this appendix we show that for weighted counting problems in $\#\mathrm{P}_{\mathbb{Q}}$ (and therefore also for weighted counting problems in $\#\mathrm{P}_{\mathbb{R}}$) the size of the output, encoded as a fraction, is not necessarily bounded by a polynomial in the input size. We focus on the following simple problem. According to Definition 2, take the polynomial p to be the identity function and take the weight function $w : \{0,1\}^\star \times \{0,1\}^\star \to \mathbb{Q}$ such that

$$w(x,u) = \begin{cases} 1/(\#u+1), & \text{if } \#u+1 \in \mathbb{P} \text{ and } \#u < x \\ 0, & \text{otherwise.} \end{cases}$$

Here $\#u$ is the number represented by the bitstring u and $\mathbb{P}$ is the set of prime numbers. Note that this function fulfills the approximation properties of Definition 2. For a given input $x \in \mathbb{N}$ of size $\lceil \log x \rceil$, the task is to compute the value

$$f(x) = \sum_{\substack{p\in\mathbb{P}\\ p\le x}} 1/p = \left(\sum_{\substack{p\in\mathbb{P}\\ p\le x}} \prod_{\substack{q\in\mathbb{P}\\ q\le x, q\ne p}} q \right) / \prod_{\substack{p\in\mathbb{P}\\ p\le x}} p.$$

Note that this fraction cannot be simplified. The only numbers that divide the denominator are prime numbers of value at most x. Each of these prime numbers divides all parts of the sum except one and therefore it does not divide the whole sum.

To show that the result cannot be represented efficiently with respect to the input size, we show that it is not possible to represent the denominator efficiently with respect to the input size. For this purpose we present a lower bound for the product of all prime numbers between x/e and x for sufficiently large x. Using this lower bound we can then bound the whole product appearing in the denominator of the output from below.

Lemma 1. *The number of prime numbers between x/e and x is bounded from below by $x/(3\ln x)$ for sufficiently large x.*

Proof. Let $\pi(x)$ denote the prime-counting function. Due to [31], we have for $x \geq 17$ that

$$\pi(x) > \frac{x}{\ln x} \quad \text{and} \quad \pi(x/e) < 1.25506 \frac{x/e}{\ln(x/e)}.$$

For the number of prime numbers between x/e and x we then have

$$\pi(x) - \pi(x/e) > \frac{x}{\ln x} - 1.25506\frac{x/e}{\ln(x/e)} > \frac{x}{\ln x} - 1.25506\frac{x/e}{3/4 \cdot \ln x} > \frac{1}{3}\frac{x}{\ln x}.$$

Hence our claim holds for sufficiently large x. □

This means that the product of all the prime numbers between x/e and x is bounded from below by $(x/e)^{x/(3\ln x)}$. Unfortunately, this value is doubly exponential in the input size of $\lceil \log x \rceil$ and a representation of this value would require exponentially many bits in the input size. Note that the result computed by a conventional counting problem or a counting problem using integer weights can always be represented using only polynomially many bits in the input size. Hence, a polynomial reduction between these classes can not exist in general. This result is summarized by Theorem 4 in this paper.