Polynomial Time Algorithms for Multi-Type Branching Processes and Stochastic Context-Free Grammars*

Kousha Etessami
School of Informatics
University of Edinburgh
Edinburgh, EH8 9AB, UK
kousha@inf.ed.ac.uk

Alistair Stewart
School of Informatics
University of Edinburgh
Edinburgh, EH8 9AB, UK
stewart.al@gmail.com

Mihalis Yannakakis
Dept. of Computer Science
Columbia University
New York, NY 10027, USA
mihalis@cs.columbia.edu

ABSTRACT

We show that one can approximate the least fixed point solution for a multivariate system of monotone probabilistic polynomial equations in time polynomial in both the encoding size of the system of equations and in $\log(1/\epsilon)$, where $\epsilon > 0$ is the desired additive error bound of the solution. (The model of computation is the standard Turing machine model.)

We use this result to resolve several open problems regarding the computational complexity of computing key quantities associated with some classic and heavily studied stochastic processes, including multi-type branching processes and stochastic context-free grammars.

Categories and Subject Descriptors

F.2.1 [**Theory of Computing**]: Analysis of Algorithms—*Numerical Algorithms and Problems*; G.3 [**Probability and Statistics**]: Stochastic Processes; G.1.5 [**Mathematics of Computing**]: Numerical Analysis—*Roots of Nonlinear Equations*

General Terms

Algorithms, Theory

1. INTRODUCTION

Some of the central computational problems associated with a number of classic stochastic processes can be rephrased as a problem of computing the non-negative *least fixed point* solution of an associated multivariate system of monotone polynomial equations.

*A full version of this paper is available at `arxiv.org/abs/1201.2374`

STOC'12, May 19–22, 2012, New York, New York, USA.

In particular, this is the case for computing the *extinction probabilities* (also called *final probabilities*) for *multi-type branching processes* (BPs), a problem which was first studied in the 1940s by Kolmogorov and Sevastyanov [20]. Branching processes are a basic stochastic model in probability theory, with applications in diverse areas ranging from population biology to the physics of nuclear chain reactions (see [18] for the classic theoretical text on BPs, and [19, 17] for some of the more recent applied textbooks on BPs). BPs describe the stochastic evolution of a population of objects of various types. In each generation, every object a of each type T gives rise to a (possibly empty) set of objects of specified types (not necessarily distinct) in the next generation, according to a given probability distribution on offsprings associated with the type T. The extinction probability, q_T, associated with type T is the probability that, starting with exactly one object of type T, the population will eventually become extinct. Computing these probabilities is fundamental to many other kinds of analyses for BPs (see, e.g., [18]). Such probabilities are in general irrational, even when the finite data describing the BP (namely, the probability distributions associated with each of the finitely many types T) are given by rational values (as is assumed usually for computational purposes). Thus, we would like to compute the probabilities approximately to some desired precision.

Another essentially equivalent problem is that of computing the probability of the language generated by a *stochastic context-free grammar* (SCFG), and more generally its *termination probabilities* (also called its *partition function*). SCFGs are a fundamental model in statistical natural language processing and in biological sequence analysis (see, e.g., [21, 6, 22, 23]). A SCFG provides a probabilistic model for the generation of strings in a language, by associating probabilities to the rules of a CFG. The termination probability of a nonterminal A is the probability that a random derivation of the SCFG starting from A eventually terminates and generates a finite-string; the total probability of the language of a SCFG is simply the termination probability for the start symbol of the SCFG. Computing these termination probabilities is again a key computational problem for the analysis of SCFGs, and is required for computing other quantities, for example the probability of generating a given string.

Despite decades of applied work on BPs and SCFGs, as

well as theoretical work on their computational problems, no polynomial time algorithm was known for computing extinction probabilities for BPs, nor for termination probabilities for SCFGs, nor even for approximating them within any nontrivial constant: prior to this work it was not even known whether one can distinguish in P-time the case where the probability is close to 0 from the case where it is close to 1.

We now describe the kinds of nonlinear equations that have to be solved in order to compute the above mentioned probabilities. Consider systems of multi-variate polynomial fixed point equations in n variables, with n equations, of the form $x_i = P_i(x),\ i = 1, \ldots, n$ where $x = (x_1, \ldots, x_n)$ denotes the vector of variables, and $P_i(x)$ is a multivariate polynomial in the variables x. We denote the entire system of equations by $x = P(x)$. The system is *monotone* if all the coefficients of the polynomials are nonnegative. It is a *probabilistic polynomial system* (PPS) if in addition the coefficients of each polynomial sum to at most 1.

It is easy to see that a system of probabilistic polynomials $P(x)$ always maps any vector in $[0,1]^n$ to another vector in $[0,1]^n$. It thus follows, by Brouwer's fixed point theorem, that a PPS $x = P(x)$ always has a solution in $[0,1]^n$. In fact, it always has a unique *least* solution, $q^* \in [0,1]^n$, which is coordinate-wise smaller than any other non-negative solution, and which is the *least fixed point* (LFP) of the monotone operator $P : [0,1]^n \to [0,1]^n$ on $[0,1]^n$. The existence of the LFP, q^*, is guaranteed by Tarski's fixed point theorem. From a BP or a SCFG we can construct easily a probabilistic polynomial system $x = P(x)$ whose LFP q^* yields precisely the vector of extinction probabilities for the BP, or termination probabilities for the SCFG. Indeed, the converse also holds: computing the extinction probabilities for a BP (termination probabilities of an SCFG) and computing the LFP of a system of probabilistic polynomial equations are equivalent problems. As we discuss below, some other stochastic models also lead to equivalent problems or to special cases.

Previous Work. As already stated, the polynomial-time computability of these basic probabilities for multi-type branching processes and SCFGs have been longstanding open problems. In [13], we studied these problems as special sub-cases of a more general class of stochastic processes called *recursive Markov chains* (RMCs), which form a natural model for analysis of probabilistic procedural programs with recursion, and we showed that these problems are equivalent to computing termination probabilities for the special subclass of *1-exit* RMCs (1-RMC). General RMCs are expressively equivalent to the model of *probabilistic pushdown systems* studied in [9, 4]. We showed that for BPs, SCFGs, and 1-RMCs, the *qualitative* problem of determining which probabilities are exactly 1 (or 0) can be solved in P-time, by exploiting basic results from the theory of branching processes. We proved however that the *decision* problem of determining whether the extinction probability of a BP (or termination probability of a SCFG or a 1-RMC) is $\geq 1/2$ is at least as hard as some longstanding open problems in the complexity of numerical computation, namely, the *square-root sum problem*, and a much more general decision problem (called PosSLP) which captures the power of unit-cost exact rational arithmetic [2], and hence it is very unlikely that the decision problem can be solved in P-time. For general RMCs we showed that in fact this hardness holds for computing *any* nontrivial *approximation* of the termination probabilities; in particular, it is PosSLP-hard to distinguish the case where a termination probability is close to 0 from the case where it is equal to 1 ([13], Theorem 5.2). No such lower bound was shown for the approximation problem for the subclass of 1-RMCs (and BPs and SCFGs). In terms of upper bounds, the best we knew so far, even for any nontrivial approximation, was that the problem is in FIXP (which is in PSPACE), i.e., it can be reduced to approximating a Nash equilibrium of a 3-player game [14]. We improve drastically on this in this paper, resolving the problem completely, by showing we can compute these probabilities in P-time to any desired accuracy.

An equivalent way to formulate the problem of computing the LFP, q^*, of a PPS, $x = P(x)$, is as a mathematical optimization problem: *minimize:* $\sum_{i=1}^{n} x_i$; *subject to:* $\{P(x) - x \leq 0;\ x \geq 0\}$. This program has a unique optimal solution, which is the LFP q^*. If the constraints were convex, then one could try to apply convex optimization methods. In general, the PPS constraints are not convex (e.g., $x_2 x_3 - x_1 \leq 0$ is not a convex constraint), however for certain restricted subclasses of PPSs they are. This is so for *backbutton processes* which were introduced and studied by Fagin et. al. in [16] and used there to analyze random walks on the web. Backbutton processes constitute a restricted subclass of SCFGs (see [13]). Fagin et. al. applied semidefinite programming to approximate the corresponding termination probabilities for backbutton processes, and used this as a basis for approximating other important quantities associated with them.

The mathematical program for a PPS that was just described can be formulated as a geometric programming problem in posynomial form [5], after removing the variables with value 0. (Thanks to Pablo Parillo for pointing this out to us.) This in fact holds true more generally for the same mathematical program that one can associate with any monotone system of polynomial equations, $x = P(x)$, including ones that arise from the more general recursive Markov chains (RMCs). Geometric programs in posynomial form can be transformed to convex optimization problems by using a change of variables, $y_i = \log x_i$. Although well-established methods exist for tackling geometric programs based on using this transformation and applying convex optimization methods, it is important to note that in general computing a non-trivial approximation of an optimal feasible solution for a geometric program, or even just a non-trivial approximation of the optimal value (even when the program is guaranteed to be feasible), cannot be done in polynomial time (in the standard Turing model) unless PosSLP can be solved in polynomial time, i.e., unless the unit-cost exact arithmetic model can be simulated by the Turing machine model with only polynomial overhead. This is a consequence of our earlier result in [13] (Theorem 5.2), which established that obtaining *any* nontrivial approximation of the termination probabilities of a RMC is PosSLP-hard, and the fact that we can easily (in P-time) construct from any RMC a feasible geometric program whose unique optimal solution yields the vector of termination probabilities of the RMC.

There are a number of natural iterative methods that

one can try to use (and which indeed are used in practice) in order to solve the equations arising from BPs and SCFGs. The simplest such method is *value iteration*: starting with the vector $x^0 = 0$, iteratively compute the sequence $x^{i+1} := P(x^i)$, $i = 1, \ldots$. The sequence always converges monotonically to the LFP q^*. Unfortunately, it can be very slow to converge: even for the simple univariate polynomial system $x = (1/2)x^2 + 1/2$, for which $q^* = 1$, one requires 2^{i-3} iterations to exceed $1 - 1/2^{i-1}$, i.e. to get i bits of precision [13].

In [13] we provided a much better method that always converges monotonically to q^*. Namely, we showed that a decomposed variant of Newton's method can be applied to such systems of equations (and in fact, much more generally, to any monotone system of polynomial equations) $x = P(x)$, and always converges monotonically to the LFP solution q^* (if a solution exists). Optimized variants of this decomposed Newton's method have by now been implemented in several tools [24, 22], and they perform quite well in practice on many instances.

The theoretical speed of convergence of Newton's method on such monotone (probabilistic) polynomial systems was subsequently studied in much greater detail by Esparza, Kiefer, and Luttenberger in [8]. They showed that, even for Newton's method on PPSs, there are instances where exponentially many iterations of Newton's method (even with *exact arithmetic* in each iteration) are required, as a function of the encoding size of the system, in order to converge to within just one bit of precision of the solution q^*. On the upper bound side, they showed that after *some* number of iterations in an initial phase, thereafter Newton obtains an additional bit of precision per iteration (this is called *linear convergence* in the numerical analysis literature). In the special case where the input system of equations is *strongly connected*, meaning roughly that all variables depend (directly or indirectly) on each other in the system of equations $x = P(x)$, they proved an exponential upper bound on the number of iterations required in the initial phase as a function of input size. For the general case where the input system of equations is not strongly connected, they did not provide any upper bound as a function of the input size. In more recent work, Esparza et al [7] further studied probabilistic polynomial systems. They did not provide any new worst-case upper bounds on the behavior of Newton's method in this case, but they studied a modified method which is in practice more robust numerically, and they also showed that the qualitative problem of determining whether the LFP $q^* = \mathbf{1}$ is decidable in *strongly* polynomial time.

Our Results. In this paper we provide the first polynomial time algorithm for computing, to any desired accuracy, the least fixed point solution, q^*, of probabilistic polynomial systems, and thus also provide the first P-time approximation algorithm for extinction probabilities of BPs, and termination probabilities of SCFGs and 1-exit RMCs. The algorithm proceeds roughly as follows:

1. We begin with a preprocessing step, in which we determine all variables x_i which have value 0 or 1 in the LFP q^* and remove them from the system.

2. On the remaining system of equations, $x = P(x)$, with an LFP q^* such that $\mathbf{0} < q^* < \mathbf{1}$, we apply Newton's method, starting at initial vector $x^{(0)} := \mathbf{0}$. Our key result is to show that, once variables x_i with $q_i^* \in \{0, 1\}$ have been removed, Newton's method only requires polynomially many iterations (in fact, only *linearly* many iterations) as a function of both the encoding size of the equation system and of $\log(1/\epsilon)$ to converge to within additive error $\epsilon > 0$ of the vector q^*. To do this, we build on the previous works [13, 8, 14], and extend them with new techniques.

3. The result in the previous step applies to the unit-cost arithmetic RAM model of computation, where we assume that each iteration of Newton's method is carried out in *exact* arithmetic. The problem with this, of course, is that in general after only a linear number of iterations, the number of bits required to represent the rational numbers in Newton's method can be exponential in the input's encoding size. We resolve this by showing, via a careful round-off analysis, that if after each iteration of Newton's method the positive rational numbers in question are all *rounded down* to a suitably long but polynomial encoding length (as a function of both the input size and of the desired error $\epsilon > 0$), then the resulting "approximate" Newton iterations will still be well-defined and will still converge to q^*, within the desired error $\epsilon > 0$, in polynomially (in fact *linearly*) many iterations. The correctness of the rounding relies crucially on the properties of PPSs shown in step 2, and it does not work in general for other types of equation systems.[1]

Extinction probabilities of BPs and termination probabilities of SCFGs are basic quantities that are important in many other analyses of these processes. We illustrate an application of these results by solving in polynomial time some other important problems for SCFGs that are at least as hard as the termination problem. We show two results in this regard:

(1) Given a SCFG and a string, we can compute the probability of the string to any desired accuracy in polynomial time. This algorithm uses the following construction:

(2) Given an SCFG, we can compute in P-time another SCFG in Chomsky normal form (CNF) that is approximately equivalent in a well-defined sense.

These are the first P-time algorithms for these problems that work for *arbitrary* SCFGs, including grammars that contain ϵ-rules. Many tasks for SCFGs, including computation of string probabilities, become much easier for SCFGs in CNF, and in fact, many papers start by assuming that the given SCFG is in CNF. In the non-stochastic case, there are standard efficient algorithms for transforming any CFG to an equivalent one in CNF. However, for stochastic grammars this is not the case and things are much more complicated. It is known that every SCFG has an equivalent one in CNF [1], however, as remarked in [1], their proof is nonconstructive and does not yield any algorithm (not even an exponential-time algorithm). Furthermore, it is possible

[1]In particular, there are examples of PPSs which *do* have $q_i^* = 1$ for some i, such that this rounding method fails completely because of very severe *ill-conditioning* (see [8]). Also, for *quasi-birth-death* (QBDs) processes, a stochastic model studied heavily in queueing systems, different monotone polynomial equations can be associated with key probabilities, and Newton's method converges in polynomially many iterations [12], but it is an open problem whether the key probabilities for QBDs can be computed in P-time in the *Turing* model.

that even though the given SCFG has rational rule probabilities, the probabilities for any equivalent SCFG in CNF must be irrational, hence they have to be computed approximately. We provide here an efficient P-time algorithm for computing such a CNF SCFG. To do this requires our P-time algorithm for SCFG termination probabilities, and requires the development of considerable additional machinery to handle the elimination of probabilistic ϵ-rules and unary rules, while keeping numerical values polynomially bounded in size, yet ensuring that the final SCFG meets the desired accuracy bounds.

The paper is organized as follows: Section 2 gives basic definitions and background; Section 3 addresses the solution of PPSs, showing a linear bound on the number of Newton iterations; Section 4 shows a polynomial time bound in the Turing model; Section 5 describes briefly the applications to SCFGs; we make concluding remarks in Section 6. Various proofs and technical developments have been omitted, and can be found in the full version [11].

2. DEFINITIONS AND BACKGROUND

A (finite, discrete-time) ***multi-type Branching Process*** (**BP**), $G = (V, R)$, consists of a (finite) set $V = \{S_1, \ldots, S_n\}$ of *types*, and a (finite) set $R = \cup_{i=1}^{n} R_i$ of *rules*, which are partitioned into distinct rule sets, R_i, associated with each type S_i. Each rule $r \in R_i$ has the form $S_i \stackrel{p_r}{\rightarrow} \alpha_r$, where $p_r \in (0, 1]$, and α_r is a finite multiset (possibly the empty multiset) whose elements are in V. Furthermore, for every type S_i, we have $\sum_{r \in R_i} p_r = 1$. The rule $S_i \stackrel{p_r}{\rightarrow} \alpha_r$ specifies the probability with which an entity (or object) of type S_i generates the multiset α_r of offspring types in the next generation. As usual, rule probabilities p_r are assumed to be rational for computational purposes. Multisets α_r over V can be encoded by giving a vector $v(\alpha_r) \in \mathbb{N}^n$, with the i'th coordinate $v(\alpha_r)_i$ representing the number of elements of type S_i in the multiset α_r. We assume instead that the multisets α_r are represented even more succinctly in *sparse representation*, by specifying only the non-zero coordinates of the vector $v(\alpha_r)$, encoded in binary.

A BP, $G = (V, R)$, defines a discrete-time stochastic (Markov) process, whose states are multisets over V, or equivalently elements of $\mathbb{N}^n$. If the state at time t is α^t, then the next state α^{t+1} at time $t+1$ is determined by *independently* choosing, for each object of each type S_i in the multiset α^t, a random rule $r \in R_i$ of the form $S_i \stackrel{p_r}{\rightarrow} \alpha_r$, according to the probability p_r of that rule, yielding the multiset α_r as the "offsprings" of that object in one generation. The multiset α^{t+1} is then given by the *multiset union* of all such offspring multisets, randomly and independently chosen for each object in the multiset α^t. A trajectory (*sample path*) of this stochastic process, starting at time 0 in initial multiset α^0, is a sequence $\alpha^0, \alpha^1, \alpha^2, \ldots$ of multisets over V. Note that if ever the process reaches *extinction*, i.e., if ever $\alpha^t = \{\}$ at some time $t \geq 0$, then $\alpha^{t'} = \{\}$ for all times $t' \geq t$.

Very fundamental quantities associated with a BP, which are a key to many analyses of BPs, are its vector of ***extinction probabilities***, $q^* \in [0, 1]^n$, where q_i^* is defined as the probability that, starting with initial multiset $\alpha^0 := \{S_i\}$ at time 0, i.e., starting with a single object of type S_i, the stochastic process eventually reaches extinction, i.e., that $\alpha^t = \{\}$ at some time $t > 0$.

Given a BP, $G = (V, R)$, there is a system of polynomial equations in $n = |V|$ variables, $x = P(x)$, that we can associate with G, such that the *least* non-negative solution vector for $x = P(x)$ is the vector of extinction probabilities q^* (see, e.g., [18, 13]). Let us define these equations. For an n-vector of variables $x = (x_1, \ldots, x_n)$, and a vector $v \in \mathbb{N}^n$, we use the shorthand x^v to denote the monomial $x_1^{v_1} \ldots x_n^{v_n}$. Given BP $G = (V, R)$, we define equation $x_i = P_i(x)$ by: $x_i = \sum_{r \in R_i} p_r x^{v(\alpha_r)}$. This yields n polynomial equations in n variables, which we denote by $x = P(x)$. It is not hard to establish that $q^* = P(q^*)$. In fact, q^* is the *least* non-negative solution of $x = P(x)$. In other words, if $q' = P(q')$ for $q' \in \mathbb{R}^n_{\geq 0}$, then $q' \geq q^* \geq \mathbf{0}$.[2]

Note that this system of polynomial equations $x = P(x)$ has very special properties. Namely, (I): the coefficients and constant of each polynomial $P_i(x) = \sum_{r \in R_i} p_r x^{v(\alpha_r)}$ are nonnegative, i.e., $p_r \geq 0$ for all r. Furthermore, (II): the coefficients sum to 1, i.e., $\sum_{r \in R_i} p_r = 1$. We call $x = P(x)$ a ***probabilistic polynomial system*** of equations (**PPS**) if it has properties (I) and (II) except that for convenience we weaken (II) and also allow (II′): $\sum_{r \in R_i} p_r \leq 1$. If a system of equations $x = P(x)$ only satisfies (I), then we call it a ***monotone polynomial system*** of equations (**MPS**). For any PPS, $x = P(x)$, $P(x)$ defines a *monotone* operator $P : [0, 1]^n \rightarrow [0, 1]^n$, i.e., if $y \geq x \geq \mathbf{0}$ then $P(y) \geq P(x)$. For any BP with corresponding PPS $x = P(x)$, q^* is precisely the *least fixed point* (**LFP**) of the monotone operator $P : [0, 1]^n \rightarrow [0, 1]^n$ (see [13]). A MPS, $x = P(x)$, also defines a monotone operator $P : \mathbb{R}^n_{\geq 0} \rightarrow \mathbb{R}^n_{\geq 0}$ on the non-negative orthant $\mathbb{R}^n_{\geq 0}$. An MPS need not in general have any solution in $\mathbb{R}^n_{\geq 0}$, but when it does so, it has a *least fixed point* solution $q^* = P(q^*)$ such that $0 \leq q' = P(q')$ implies $q^* \leq q'$.

Note that *any* PPS (with rational coefficients) can be obtained as the system of equations $x = P(x)$ for a corresponding BP G (with rational rule probabilities), and vice versa.[3] Thus, the computational problem of computing the extinction probabilities of a given BP is equivalent to the problem of computing the least fixed point (LFP) solution q^* of a given PPS, $x = P(x)$. For a PPS $x = P(x)$, we shall use $|P|$ to denote the sum of the number n of variables and the numbers of bits of all the nonzero coefficients and nonzero exponents of all the polynomials in the PPS. Note that the encoding length of a PPS in sparse representation is at least $|P|$ (and at most $O(|P| \log n)$).

The probabilities q^* can in general be irrational, and even

[2] In this entire paper, when a and b are vectors or matrices of the same dimensions, we use inequality $a \geq b$ to mean coordinate-wise or entry-wise inequality, i.e., $a_i \geq b_i$ for all i, if a and b are both vectors, and $a_{i,j} \geq b_{i,j}$ for all i and j, if they are both matrices. We also use $\mathbf{0}$ (and $\mathbf{1}$) to denote an all 0 (all 1) vector or matrix of suitable dimensions.

[3] "*Leaky*" PPSs where $\sum_{r \in R_i} p_r < 1$ can be translated easily to BPs by adding an extra dummy type S_{n+1}, with rule $S_{n+1} \stackrel{1}{\rightarrow} \{S_{n+1}, S_{n+1}\}$, so $q^*_{n+1} = 0$, and adding to R_i, for each "leaky" i, the rule $S_i \stackrel{p'_i}{\rightarrow} \{S_{n+1}, S_{n+1}\}$ with probability $p'_i := (1 - \sum_{r \in R_i} p_r)$. The probabilities q^* for the BP (ignoring $q^*_{n+1} = 0$) give the LFP of the PPS .

deciding whether $q_i^* \geq 1/2$ is as hard as long standing open problems, including the *square-root sum* problem, which are not even known to be in NP (see [13]). We instead want to approximate q^* within a desired additive error $\epsilon > 0$. In other words, we want to compute a rational vector $v' \in \mathbb{Q}^n \cap [0,1]^n$ such that $\|q^* - v'\|_\infty < \epsilon$.

A PPS, $x = P(x)$, is said to be in *Simple Normal Form* (SNF) if for every $i = 1, \ldots, n$, the polynomial $P_i(x)$ has one of two forms: (1) $\underline{\text{Form}_*}$: $P_i(x) \equiv x_j x_k$ is simply a quadratic monomial; or (2) $\underline{\text{Form}_+}$: $P_i(x)$ is a *linear* expression $\sum_{j \in \mathcal{C}_i} p_{i,j} x_j + p_{i,0}$, for some rational non-negative coefficients $p_{i,j}$ and $p_{i,0}$, and some index set $\mathcal{C}_i \subseteq \{1, \ldots, n\}$, where $\sum_{j \in \mathcal{C}_i \cup \{0\}} p_{i,j} \leq 1$. We call such a linear equation *leaky* if $\sum_{j \in \mathcal{C}_i \cup \{0\}} p_{i,j} < 1$. An MPS is said to be in SNF if the same conditions hold except we do not require $\sum_{j \in \mathcal{C}_i \cup \{0\}} p_{i,j} \leq 1$. It is easy to convert any MPS or PPS to SNF form, by adding auxiliary variables and equations. The following is proved in the full version of this paper ([11]):

PROPOSITION 2.1 (CF. PROPOS. 7.3 [13]). *Every PPS (MPS), $x = P(x)$, can be transformed in P-time to an "*equivalent*" PPS (MPS, respectively), $y = Q(y)$ in SNF form, such that $|Q| \in O(|P|)$. More precisely, the variables x are a subset of the variables y, and $y = Q(y)$ has LFP $p^* \in \mathbb{R}^m_{\geq 0}$ iff $x = P(x)$ has LFP $q^* \in \mathbb{R}^n_{\geq 0}$, and projecting p^* onto the x variables yields q^*.*

The following was established in [13], showing that for PPSs we can detect whether $q_i^* = 0$ or $q_i^* = 1$ in P-time (the harder problem is checking whether $q_i^* = 1$):

PROPOSITION 2.2 ([13]). *There is a P-time algorithm that, given a PPS, $x = P(x)$, over n variables, with LFP $q^* \in \mathbb{R}^n_{\geq 0}$, determines for every $i = 1, \ldots, n$ whether $q_i^* = 0$ or $q_i^* = 1$ or $0 < q_i^* < 1$.*

Thus, for every PPS, we can detect in P-time all the variables x_j such that $q_j^* = 0$ or $q_j^* = 1$. We can then remove these variables and their corresponding equation $x_j = P_j(x)$, and substitute their values on the right hand sides (RHS) of the remaining equations. This yields a new PPS, $x' = P'(x')$, where its LFP solution, q'^*, is $\mathbf{0} < q'^* < \mathbf{1}$, which corresponds to the remaining coordinates of q^*.

We can thus henceforth assume, w.l.o.g., that any given PPS, $x = P(x)$, is in SNF form and has an LFP solution q^* such that $\mathbf{0} < q^* < \mathbf{1}$.

For a MPS or PPS, $x = P(x)$, its variable *dependency graph* is defined to be the digraph $H = (V, E)$, with vertices $V = \{x_1, \ldots, x_n\}$, such that $(x_i, x_j) \in E$ iff in $P_i(x) = \sum_{r \in R_i} p_r x^{v(\alpha_r)}$ there is a coefficient $p_r > 0$ such that $v(\alpha_r)_j > 0$. Intuitively, $(x_i, x_j) \in E$ means that x_i "depends directly" on x_j. A MPS or PPS, $x = P(x)$, is called ***strongly connected*** if its dependency graph H is strongly connected. As in [13], for analysing PPSs we will find it very useful to decompose the PPS based on the *strongly connected components* (SCCs) of its variable dependency graph.

3. POLYNOMIAL UPPER BOUNDS FOR NEWTON'S METHOD ON PPSs

To find a solution for a differentiable system of equations $F(x) = \mathbf{0}$, in n variables, *Newton's method* uses the following iteration scheme: start with some initial vector $x^{(0)} \in \mathbb{R}^n$, and for $k \geq 0$ let: $x^{(k+1)} := x^{(k)} - F'(x^{(k)})^{-1} F(x^{(k)})$, where $F'(x)$ is the Jacobian matrix of $F(x)$, with $F'(x)_{i,j} = \frac{\partial F_i(x)}{\partial x_j}$.

Let $x = P(x)$ be a given PPS (or MPS) in n variables. Let $B(x) := P'(x)$ denote the Jacobian matrix of $P(x)$. In other words, $B(x)$ is an $n \times n$ matrix such that $B(x)_{i,j} = \frac{\partial P_i(x)}{\partial x_j}$. Using Newton iteration, starting at n-vector $x^{(0)} := \mathbf{0}$, yields the following iteration:

$$x^{(k+1)} := x^{(k)} + (I - B(x^{(k)}))^{-1}(P(x^{(k)}) - x^{(k)}) \qquad (1)$$

For a vector $z \in \mathbb{R}^n$, assuming that matrix $(I - B(z))$ is non-singular, we define a single iteration of Newton's method for $x = P(x)$ on z via the following operator:

$$\mathcal{N}_P(z) := z + (I - B(z))^{-1}(P(z) - z) \qquad (2)$$

It was shown in [13] that for any MPS, $x = P(x)$, with LFP $q^* \in \mathbb{R}^N_{\geq 0}$, if we first find and remove the variables that have value 0 in the LFP, q^*, and apply a decomposed variant of Newton's method that decomposes the system according to the strongly connected components (SCCs) of the dependency graph and processes them bottom-up, then the values converge *monotonically* to q^*. PPSs are a special case of MPSs, so the same applies to PPSs. In [8], it was pointed out that if $q^* > 0$, i.e., after we remove the variables x_i where $q_i^* = 0$, decomposition into SCCs isn't strictly necessary. (Decomposition is nevertheless very useful in practice, as well as in the theoretical analysis, including in this paper.). Thus:

PROPOSITION 3.1 (CF. THM 6.1 [13] & THM 4.1 [8]). *Let $x = P(x)$ be a MPS, with LFP $q^* > \mathbf{0}$. Then starting at $x^{(0)} := \mathbf{0}$, the Newton iterations $x^{(k+1)} := \mathcal{N}_P(x^{(k)})$ are well defined and monotonically converge to q^*, i.e. $\lim_{k \to \infty} x^{(k)} = q^*$, and $x^{(k+1)} \geq x^{(k)} \geq \mathbf{0}$ for all $k \geq 0$.*

We will actually establish an extension of this result in this paper, because in Section 4 we will need to show that even when each iterate is suitably *rounded off*, the rounded Newton iterations are all well-defined and converge to q^*. The main goal of this section is to show that for PPSs, $x = P(x)$, with LFP $0 < q^* < 1$, polynomially many iterations of Newton's method, *using exact rational arithmetic*, suffice, as a function of $|P|$ and j, to compute q^* to within additive error $1/2^j$. In fact, we show a much stronger *linear* upper bound with small explicit constants:

THEOREM 3.2 (**Main Theorem of Section 3**). *Let $x = P(x)$ be any PPS in SNF form, with LFP q^*, such that $\mathbf{0} < q^* < \mathbf{1}$. If we start Newton iteration at $x^{(0)} := \mathbf{0}$, with $x^{(k+1)} := \mathcal{N}_P(x^{(k)})$, then for any integer $j \geq 0$ the following inequality holds: $\|q^* - x^{(j+4|P|)}\|_\infty \leq 2^{-j}$.*

We need a sequence of Lemmas.

LEMMA 3.3. *Let $x = P(x)$ be a MPS, with n variables, in SNF form, and let $a, b \in \mathbb{R}^n$. Then:*

$$P(a) - P(b) = B(\frac{a+b}{2})(a - b) = \frac{B(a) + B(b)}{2}(a - b)$$

PROOF. Let the function $f : \mathbb{R} \to \mathbb{R}^n$ be given by $f(t) := ta + (1-t)b = b + t(a-b)$. Define $G(t) := P(f(t))$.

From the fundamental theorem of calculus, and using the matrix form of the chain rule from multi-variable calculus (see, e.g., [3] Section 12.10), we have:

$$P(a) - P(b) = G(1) - G(0) = \int_0^1 B(f(t))(a-b)\,dt$$

By linearity, we can just take out $(a-b)$ from the integral as a constant, and we get:

$$P(a) - P(b) = (\int_0^1 B(ta + (1-t)b)\,dt)(a-b)$$

We need to show that

$$\int_0^1 B(ta + (1-t)b)\,dt = B(\frac{a+b}{2}) = \frac{B(a)+B(b)}{2}$$

Since all monomials in $P(x)$ have degree at most 2, each entry of the Jacobian matrix $B(x)$ is a polynomial of degree 1 over variables in x. For any integers i, j, with $0 \le i \le n$, $0 \le j \le n$, there are thus real values α_{ij} and β_{ij} with

$$(B(ta + (1-t)b))_{ij} = \alpha_{ij} + \beta_{ij}t$$

Then

$$(\int_0^1 B(ta + (1-t)b)\,dt)_{ij} = \int_0^1 (\alpha_{ij} + \beta_{ij}t)\,dt = \alpha_{ij} + \frac{\beta_{ij}}{2}$$

$$(B(\frac{a+b}{2}))_{ij} = \alpha_{ij} + \frac{\beta_{ij}}{2}$$

$$(\frac{B(a)+B(b)}{2})_{ij} = \frac{1}{2}((\alpha_{ij}+\beta_{ij}) + \alpha_{ij}) = \alpha_{ij} + \frac{\beta_{ij}}{2}$$

□

LEMMA 3.4. *Let $x = P(x)$ be a MPS in SNF form. Let $z \in \mathbb{R}^n$ be any vector such that $(I - B(z))$ is non-singular, and thus $\mathcal{N}_P(z)$ is defined. Then:*

$$q^* - \mathcal{N}_P(z) = (I - B(z))^{-1}\frac{B(q^*) - B(z)}{2}(q^* - z)$$

PROOF. Lemma 3.3, applied to q^* and z, gives: $q^* - P(z) = \frac{B(q^*)+B(z)}{2}(q^* - z)$. Rearranging, we get:

$$P(z) - z = (I - \frac{B(q^*) + B(z)}{2})(q^* - z) \qquad (3)$$

Replacing $(P(z) - z)$ in equation (2) by the right hand side of equation (3) and subtracting both sides of (2) from q^*, gives:

$$\begin{aligned}
& q^* - \mathcal{N}_P(z) \\
= \;& (q^* - z) - (I - B(z))^{-1}(I - \frac{B(q^*) + B(z)}{2})(q^* - z) \\
= \;& (I - B(z))^{-1}(I - B(z))(q^* - z) \\
& -(I - B(z))^{-1}(I - \frac{B(q^*) + B(z)}{2})(q^* - z) \\
= \;& (I - B(z))^{-1}((I - B(z)) - (I - \frac{B(q^*) + B(z)}{2}))(q^* - z) \\
= \;& (I - B(z))^{-1}(\frac{B(q^*) - B(z)}{2})(q^* - z)
\end{aligned}$$

□

To prove their exponential upper bounds for *strongly connected* PPSs, [8] used the notion of a *cone vector* for the matrix $B(q^*)$, that is a vector $d > 0$ such that $B(q^*)d \le d$. For a strongly connected MPS, $x = P(x)$, with $q^* > 0$, the matrix $B(q^*) \ge 0$ is irreducible, and thus has a positive eigenvector. They used this eigenvector as their cone vector $d > 0$. However, such an eigenvector yields only weak (exponential) bounds. Instead, we show there is a different cone vector for $B(q^*)$, and even for $B(\frac{1}{2}(\mathbf{1}+q^*))$, that works for arbitrary (not necessarily strongly-connected) PPSs:

LEMMA 3.5. *If $x = P(x)$ is a PPS in n variables, in SNF form, with LFP $\mathbf{0} < q^* < \mathbf{1}$, and where $P(x)$ has Jacobian $B(x)$, then $\forall z \in \mathbb{R}^n$ such that $\mathbf{0} \le z \le \frac{1}{2}(\mathbf{1} + q^*)$, we have:*
$$B(z)(\mathbf{1} - q^*) \le (\mathbf{1} - q^*).$$
In particular, $B(\frac{1}{2}(\mathbf{1}+q^*))(\mathbf{1}-q^*) \le (\mathbf{1}-q^*)$, and $B(q^*)(\mathbf{1}-q^*) \le (\mathbf{1}-q^*)$.

PROOF. Lemma 3.3 applied to $\mathbf{1}$ and q^* gives: $P(\mathbf{1}) - P(q^*) = P(\mathbf{1}) - q^* = B(\frac{1}{2}(\mathbf{1}+q^*))(\mathbf{1}-q^*)$. But note that $P(\mathbf{1}) \le \mathbf{1}$, because for any PPS, since the nonnegative coefficients of each polynomial $P_i(x)$ sum to ≤ 1, $P(x)$ maps $[0,1]^n$ to $[0,1]^n$. Thus $\mathbf{1} - q^* \ge P(\mathbf{1}) - q^* = B(\frac{1}{2}(\mathbf{1} + q^*))(\mathbf{1} - q^*)$. Now observe that for $0 \le z \le \frac{1}{2}(\mathbf{1}+q^*)$, $B(\frac{1}{2}(\mathbf{1}+q^*)) \ge B(z) \ge 0$, because the entries of Jacobian $B(x)$ have nonnegative coefficients. Thus since $(\mathbf{1}-q^*) \ge 0$, we have $(\mathbf{1}-q^*) \ge B(z)(\mathbf{1}-q^*)$. □

For a square matrix A, let $\rho(A)$ denote A's spectral radius.

THEOREM 3.6. *For any PPS, $x = P(x)$, in SNF form, if we have $0 < q^* < 1$, then for all $0 \le z \le q^*$, $\rho(B(z)) < 1$ and $(I - B(z))^{-1}$ exists and is nonnegative.*

The proof (in the full version [11]) uses, among other things, Lemma 3.5, and some Perron-Frobenius theory. Note that this theorem tells us, in particular, that for *every* z (including q^*), such that $\mathbf{0} \le z \le q^*$, the Newton iteration $\mathcal{N}_P(z)$ is well-defined. This will be important in Section 4. We need the following Lemma from [8]. (To clarify our assumptions we provide a short proof here.)

LEMMA 3.7 (LEMMA 5.4 FROM [8]). *Let $x = P(x)$ be a MPS, with polynomials of degree bounded by 2, with LFP, $q^* \ge 0$. Let $B(x)$ denote the Jacobian matrix of $P(x)$. For any positive vector $\boldsymbol{d} \in \mathbb{R}^n_{>0}$ that satisfies $B(q^*)\boldsymbol{d} \le \boldsymbol{d}$, any positive real value $\lambda > 0$, and any nonnegative vector $z \in \mathbb{R}^n_{\ge 0}$, if $q^* - z \le \lambda \boldsymbol{d}$, and $(I - B(z))^{-1}$ exists and is nonnegative, then $q^* - \mathcal{N}_P(z) \le \frac{\lambda}{2}\boldsymbol{d}$.*

PROOF. By Lemma 3.4, $q^* - \mathcal{N}_P(z) = (I - B(z))^{-1}\frac{1}{2}(B(q^*) - B(z))(q^* - z)$. Note that matrix $(I - B(z))^{-1}\frac{1}{2}(B(q^*) - B(z))$ is nonnegative: we assumed $(I - B(z))^{-1} \ge 0$ and the positive coefficients in $P(x)$ and in $B(x)$ mean $(B(q^*) - B(z)) \ge 0$. This and the assumption that $q^* - z \le \lambda d$ yields: $q^* - \mathcal{N}_P(z) \le (I - B(z))^{-1}\frac{1}{2}(B(q^*) - B(z))\lambda\mathbf{d}$. We

can rearrange as follows:

$$\begin{aligned} q^* - \mathcal{N}_P(z) &\leq (I - B(z))^{-1}\frac{1}{2}(B(q^*) - B(z))\lambda\mathbf{d} \\ &= (I - B(z))^{-1}\frac{1}{2}((I - B(z)) - (I - B(q^*)))\lambda\mathbf{d} \\ &= \frac{\lambda}{2}(I - (I - B(z))^{-1}(I - B(q^*)))\mathbf{d} \\ &= \frac{\lambda}{2}\mathbf{d} - \frac{\lambda}{2}(I - B(z))^{-1}(I - B(q^*))\mathbf{d} \end{aligned}$$

If we can show that $\frac{\lambda}{2}(I - B(z))^{-1}(I - B(q^*))\mathbf{d} \geq 0$, we are done. By assumption: $(I - B(q^*))\mathbf{d} \geq 0$, and since we assumed $(I - B(z))^{-1} \geq 0$ and $\lambda > 0$, we have: $\frac{\lambda}{2}(I - B(z))^{-1}(I - B(q^*))\mathbf{d} \geq 0$. □

For a vector $b \in \mathbb{R}^n$, we shall use the following notation: $b_{\min} = \min_i b_i$, and $b_{\max} = \max_i b_i$.

COROLLARY 3.8. *Let $x = P(x)$ be MPS, with LFP $q^* > 0$, and let $B(x)$ be the Jacobian matrix for $P(x)$. Suppose there is a vector $d \in \mathbb{R}^n$, $\mathbf{0} < d \leq \mathbf{1}$, such that $B(q^*)d \leq d$. For any positive integer $j > 0$, if we perform Newton's method starting at $x^{(0)} := \mathbf{0}$, then: $\|q^* - x^{(j - \lfloor \log_2 d_{min} \rfloor)}\|_\infty \leq 2^{-j}$.*

PROOF. By induction on k, we show $q^* - x^{(k)} \leq 2^{-k}\frac{1}{d_{\min}}d$. For the base case, $k = 0$, since $d > 0$, $\frac{1}{d_{\min}}d \geq \mathbf{1} \geq q^* = q^* - x^{(0)}$. For $k > 0$, apply Lemma 3.7, setting $z := x^{(k-1)}$, $\lambda := \frac{1}{d_{\min}}2^{-(k-1)}$ and $\mathbf{d} := d$. This yields $q^* - x^{(k)} \leq \frac{\lambda}{2}\mathbf{d} = 2^{-k}\frac{1}{d_{\min}}d$. Since we assume $\|d\|_\infty \leq 1$, we have $\|2^{-(j - \lfloor \log_2 d_{\min} \rfloor)}\frac{1}{d_{\min}}d\|_\infty \leq 2^{-j}$, and thus $\|q^* - x^{(j - \lfloor \log_2 d_{\min} \rfloor)}\|_\infty \leq 2^{-j}$. □

LEMMA 3.9. *For a PPS in SNF form, with LFP q^*, where $\mathbf{0} < q^* < \mathbf{1}$, if we start Newton iteration at $x^{(0)} := \mathbf{0}$, then:*

$$\|q^* - x^{(j + \lceil (\log_2 \frac{(\mathbf{1} - q^*)_{max}}{(\mathbf{1} - q^*)_{min}}) \rceil)}\|_\infty \leq 2^{-j}$$

PROOF. For $d := \frac{\mathbf{1} - q^*}{\|\mathbf{1} - q^*\|_\infty}$, $d_{min} = \frac{(\mathbf{1} - q^*)_{\min}}{(\mathbf{1} - q^*)_{\max}}$. By Lemma 3.5, $B(q^*)d \leq d$. Apply Corollary 3.8. □

LEMMA 3.10. *For a strongly connected PPS, $x = P(x)$, with LFP q^*, where $0 < q^* < 1$, for any two coordinates k, l of $\mathbf{1} - q^*$:*

$$\frac{(\mathbf{1} - q^*)_k}{(\mathbf{1} - q^*)_l} \geq 2^{-(2|P|)}$$

PROOF. Lemma 3.5 says that $B(\frac{1}{2}(\mathbf{1} + q^*))(\mathbf{1} - q^*) \leq (\mathbf{1} - q^*)$. Since every entry of the vector $\frac{1}{2}(\mathbf{1} + q^*))$ is $\geq 1/2$, every non-zero entry of the matrix $B(\frac{1}{2}(\mathbf{1} + q^*))$ is at least $1/2$ times a coefficient of some monomial in some polynomial $P_i(x)$ of $P(x)$. Moreover, $B(\frac{1}{2}(\mathbf{1} + q^*))$ is irreducible. Calling the entries of $B(\frac{1}{2}(\mathbf{1} + q^*))$, $b_{i,j}$, we have a sequence of *distinct* indices, $i_1, i_2, \ldots, i_m$, with $l = i_1$, $k = i_m$, $m \leq n$, where each $b_{i_j i_{j+1}} > 0$. (Just take the "shortest positive path" from l to k.) For any j:

$$(B(\frac{1}{2}(\mathbf{1} + q^*))(\mathbf{1} - q^*))_{i_{j+1}} \geq b_{i_j i_{j+1}}(\mathbf{1} - q^*)_j$$

Using Lemma 3.5 again, $(\mathbf{1} - q^*)_{i_{j+1}} \geq b_{i_j i_{j+1}}(\mathbf{1} - q^*)_{i_j}$. By simple induction: $(\mathbf{1} - q^*)_k \geq (\prod_{j=1}^{l-1} b_{i_j i_{j+1}})(\mathbf{1} - q^*)_l$. Note that $|P|$ includes the encoding size of each positive coefficient of every polynomial $P_i(x)$. We argued before that each $b_{i_j i_{j+1}} \geq c_i/2$ for some coefficient $c_i > 0$ of some monomial in $P_i(x)$. Therefore, since each such c_i is a distinct coefficient that is accounted for in $|P|$, we must have $\prod_{j=1}^{l-1} b_{i_j i_{j+1}} \geq 2^{-(|P|+n)} \geq 2^{-(2|P|)}$, and thus we have: $(\mathbf{1} - q^*)_k \geq 2^{-(2|P|)}(\mathbf{1} - q^*)_l$. □

Combining Lemma 3.9 and 3.10 establishes the following:

THEOREM 3.11. *For a strongly connected PPS, $x = P(x)$ in n variables, in SNF form, with LFP q^*, such that $\mathbf{0} < q^* < \mathbf{1}$, if we start Newton iteration at $x^{(0)} := \mathbf{0}$, then: $\|q^* - x^{(j+2|P|)}\|_\infty \leq 2^{-j}$.*

To get a polynomial upper bound on the number of iterations of Newton's method for general PPSs, we can apply Lemma 3.9 combined with a Lemma in [14] (Lemma 7.2 of [14]), which implies that for a PPS $x = P(x)$ with n variables, in SNF form, with LFP q^*, where $q^* < \mathbf{1}$, $(\mathbf{1} - q^*)_{\min} \geq 1/2n2^{|P|^c}$ for some constant c. Instead, we prove the following much stronger result:

THEOREM 3.12. *For a PPS, $x = P(x)$ in n variables, in SNF form, with LFP q^*, such that $0 < q^* < 1$, for all $i = 1, \ldots, n$: $1 - q_i^* \geq 2^{-4|P|}$, i.e., $\|q^*\|_\infty \leq 1 - 2^{-4|P|}$.*

The proof of Theorem 3.12 is more involved and is given in the full version [11]. The proof first establishes a bound for the strongly connected case, using norm bounds on some key matrices involved, and some Perron-Frobenius theory. It then establishes the bounds for arbitrary variables using the fact that every variable must depend, via a short enough path, on some variable in a bottom SCC. We thus get the Main Theorem of this section:

PROOF OF THEOREM 3.2 **(Main Theorem of Sec. 3)**. By Lemma 3.9, $\|q^* - x^{(j + \lceil (\log \frac{(\mathbf{1} - q^*)_{\max}}{(\mathbf{1} - q^*)_{\min}}) \rceil)}\|_\infty \leq 2^{-j}$. But by Theorem 3.12, $\lceil (\log \frac{(\mathbf{1} - q^*)_{max}}{(\mathbf{1} - q^*)_{min}}) \rceil \leq \lceil \log \frac{1}{(\mathbf{1} - q^*)_{min}} \rceil \leq \lceil \log 2^{4|P|} \rceil = 4|P|$. □

4. POLYNOMIAL TIME IN THE STANDARD TURING MODEL OF COMPUTATION

The previous section showed that for a PPS, $x = P(x)$, using $(4|P| + j)$ iterations of Newton's method starting at $x^{(0)} := 0$, we obtain q^* within additive error 2^{-j}. However, performing even $|P|$ iterations of Newton's method *exactly* may not be feasible in P-time in the *Turing* model, because the encoding size of iterates $x^{(k)}$ can become very large. Specifically, by repeated squaring, the rational numbers representing the iterate $x^{(|P|)}$ may require encoding size exponential in $|P|$.

In this section, we show that we can nevertheless approximate in P-time the LFP q^* of a PPS, $x = P(x)$. We do so by showing that we can *round down* all coordinates of each Newton iterate $x^{(k)}$ to a suitable polynomial length, and still have a well-defined iteration that converges in nearly the same number of iterations to q^*. Throughout this section we assume every PPS is in SNF form.

DEFINITION 4.1. *("Rounded down Newton's method", with rounding parameter h.) Given a PPS, $x = P(x)$, with LFP q^*, where $\mathbf{0} < q^* < \mathbf{1}$, in the "rounded down Newton's method" with integer rounding parameter $h > 0$, we compute a sequence of iteration vectors $x^{[k]}$, where the initial starting vector is again $x^{[0]} := \mathbf{0}$, and such that for each $k \geq 0$, given $x^{[k]}$, we compute $x^{[k+1]}$ as follows:*

1. *First, compute $x^{\{k+1\}} := \mathcal{N}_P(x^{[k]})$, where the Newton iteration operator $\mathcal{N}_P(x)$ was defined in equation (2). (Of course we need to show that all such Newton iterations are defined.)*
2. *For each coordinate $i = 1, \ldots, n$, set $x_i^{[k+1]}$ to be equal to the maximum (non-negative) multiple of 2^{-h} which is $\leq \max(x_i^{\{k+1\}}, 0)$. (In other words, round down $x^{\{k+1\}}$ to the nearest multiple of 2^{-h}, while making sure that the result is non-negative.)*

THEOREM 4.2 (**Main Theorem of Sec. 4**). *Given a PPS, $x = P(x)$, with LFP q^*, such that $0 < q^* < 1$, if we use the rounded down Newton's method with parameter $h = j+2+4|P|$, then the iterations are all defined, for every $k \geq 0$ we have $0 \leq x^{[k]} \leq q^*$, and after $h = j + 2 + 4|P|$ iterations we have: $\|q^* - x^{[j+2+4|P|]}\|_\infty \leq 2^{-j}$.*

We prove this via some lemmas. The next lemma proves that the iterations are always well-defined, and yield vectors $x^{[k]}$ such that $\mathbf{0} \leq x^{[k]} \leq q^*$. Note however that, unlike Newton iteration using exact arithmetic, we *do not* claim (as in Proposition 3.1) that $x^{[k]}$ converges *monotonically* to q^*. It may not. It turns out we don't need this: all we need is that $0 \leq x^{[k]} \leq q^*$, for all k. In particular, it may not hold that $P(x^{[k]}) \geq x^{[k]}$. For establishing the monotone convergence of Newton's method on MPSs (Proposition 3.1), the fact that $P(x^{(k)}) \geq x^{(k)}$ is key (see [13]). Indeed, note that for PPSs, once we know that $(P(x^{(k)}) - x^{(k)}) \geq 0$, Theorem 3.6 and the defining equation of Newton iteration, (1), already proves monotone convergence: $x^{(k)}$ is well-defined and $x^{(k+1)} \geq x^{(k)} \geq 0$, for all k. However, $P(x^{[k]}) \geq x^{[k]}$ may no longer hold after rounding down. If, for instance, the polynomial $P_i(x)$ has degree 1 (i.e., has Form_+), then one can show that after any positive number of iterations $k \geq 1$, we will have that $P_i(x^{\{k\}}) = x_i^{\{k\}}$. So, if we are unlucky, rounding down each coordinate of $x^{\{k\}}$ to a multiple of 2^{-h} could indeed give $(P(x^{[k+1]}))_i < x_i^{[k+1]}$.

LEMMA 4.3. *If we start the rounded down Newton method with $x^{[0]} := \mathbf{0}$ on a PPS, $x = P(x)$, with LFP q^*, $\mathbf{0} < q^* < \mathbf{1}$, then for all $k \geq 0$, $x^{[k]}$ is well-defined and $0 \leq x^{[k]} \leq q^*$.*

PROOF. We prove this by induction on k. The base case $x^{[0]} = 0$ is immediate. Suppose the claim holds for k and thus $0 \leq x^{[k]} \leq q^*$. Lemma 3.4 tells us that

$$q^* - x^{\{k+1\}} = (I - B(x^{[k]}))^{-1} \frac{B(q^*) - B(x^{[k]})}{2}(q^* - x^{[k]})$$

Now the fact that $0 \leq x^{[k]} \leq q^*$ yields that each of the following inequalities hold: $(q^* - x^{[k]}) \geq 0$, $B(q^*) - B(x^{[k]}) \geq 0$. Furthermore, by Theorem 3.6, we have that $\rho(B(x^{[k]})) < 1$, and thus that $(I - B(x^{[k]}))$ is non-singular and $(I - B(x^{[k]}))^{-1} \geq 0$. We thus conclude that $q^* - x^{\{k+1\}} \geq 0$, i.e., that $x^{\{k\}} \leq q^*$. The rounding down ensures that $0 \leq x_i^{[k+1]} \leq x_i^{\{k+1\}}$ unless $x_i^{\{k+1\}} < 0$, in which case $x_i^{[k+1]} = 0$. in both cases, we have that $0 \leq x^{[k+1]} \leq q^*$. □

The next key lemma shows that the rounded version still makes good progress towards the LFP.

LEMMA 4.4. *For a PPS, $x = P(x)$, with LFP q^*, such that $0 < q^* < 1$, if we apply the rounded down Newton's method with parameter h, starting at $x^{[0]} := \mathbf{0}$, then for all $j' \geq 0$, we have:*

$$\|q^* - x^{[j'+1]}\|_\infty \leq 2^{-j'} + 2^{-h+1+4|P|}$$

PROOF. Since $x^{[0]} := 0$:

$$q^* - x^{[0]} = q^* \leq \mathbf{1} \leq \frac{1}{(\mathbf{1} - q^*)_{\min}}(\mathbf{1} - q^*) \tag{4}$$

For any $k \geq 0$, if $q^* - x^{[k]} \leq \lambda(\mathbf{1} - q^*)$, then by Lemma 3.7 we have:

$$q^* - x^{\{k+1\}} \leq (\frac{\lambda}{2})(\mathbf{1} - q^*) \tag{5}$$

Observe that after every iteration $k > 0$, in every coordinate i we have:

$$x_i^{[k]} \geq x_i^{\{k\}} - 2^{-h} \tag{6}$$

This holds simply because we are rounding down $x_i^{\{k\}}$ by at most 2^{-h}, unless it is negative in which case $x_i^{[k]} = 0 > x_i^{\{k\}}$. Combining the two inequalities (5) and (6) yields the following inequality:

$$q^* - x^{[k+1]} \leq (\frac{\lambda}{2})(\mathbf{1} - q^*) + 2^{-h}\mathbf{1} \leq (\frac{\lambda}{2} + \frac{2^{-h}}{(\mathbf{1} - q^*)_{\min}})(\mathbf{1} - q^*)$$

Taking inequality (4) as the base case (with $\lambda = \frac{1}{(\mathbf{1}-q^*)_{\min}}$), by induction on k, for all $k \geq 0$:

$$q^* - x^{[k+1]} \leq (2^{-k} + \sum_{i=0}^{k} 2^{-(h+i)}) \frac{1}{(\mathbf{1} - q^*)_{\min}}(\mathbf{1} - q^*)$$

But $\sum_{i=0}^{k} 2^{-(h+i)} \leq 2^{-h+1}$ and $\frac{\|\mathbf{1}-q^*\|_\infty}{(\mathbf{1}-q^*)_{\min}} \leq \frac{1}{(\mathbf{1}-q^*)_{\min}} \leq 2^{4|P|}$, by Theorem 3.12. Thus:

$$q^* - x^{[k+1]} \leq (2^{-k} + 2^{-h+1})2^{4|P|}\mathbf{1}$$

Clearly, we have $q^* - x^{[k]} \geq 0$ for all k. Thus we have shown that for all $k \geq 0$:

$$\|q^* - x^{[k+1]}\|_\infty \leq (2^{-k} + 2^{-h+1})2^{4|P|} = 2^{-k} + 2^{-h+1+4|P|}.$$

□

We can now prove the main theorem:

PROOF OF THEOREM 4.2 (**Main Theorem of Sec. 4**). In Lemma 4.4 let $j' := j + 4|P| + 1$ and $h := j + 2 + 4|P|$. We have: $\|q^* - x^{[j+2+4|P|]}\|_\infty \leq 2^{-(j+1+4|P|)} + 2^{-(j+1)} \leq 2^{-(j+1)} + 2^{-(j+1)} = 2^{-j}$. □

COROLLARY 4.5. *Given any PPS, $x = P(x)$, with LFP q^*, we can approximate q^* within additive error 2^{-j} in time polynomial in $|P|$ and j (in the standard Turing model of computation). More precisely, we can compute a vector v, $\mathbf{0} \leq v \leq q^*$, such that $\|q^* - v\|_\infty \leq 1/2^{-j}$.*

PROOF. Firstly, by Propositions 2.1 and 2.2, we can assume $x = P(x)$ is in SNF form, and that $\mathbf{0} < q^* < \mathbf{1}$. By Theorem 4.2, the rounded down Newton's method with parameter $h = j + 2 + 4|P|$, for $h = j + 2 + 4|P|$ iterations, computes a rational vector $v = x^{[h]}$ such that $v \in [0,1]^n$, and $\|q^* - v\|_\infty \leq 1/2^{-h}$.

Furthermore, for all k, with $0 \leq k \leq h$, $x^{[k]}$ has encoding size polynomial in $|P|$ and j. We then simply need to note that all the linear algebra operations, that is: matrix multiplication, addition, and matrix inversion, required in a single iteration of Newton's method, can be performed exactly on rational inputs in polynomial time and yield rational results with a polynomial size. □

5. APPLICATION TO PROBABILISTIC PARSING OF GENERAL SCFGs

We briefly describe applications to some important problems for stochastic context-free grammars (SCFGs). For definitions, background, and a detailed treatment we refer to the full version [11].

We are given a SCFG, $G = (V, \Sigma, R, S)$, with set V of nonterminals, set Σ of terminals, set R of probabilistic rules with rational probabilities, and start symbol $S \in V$. The SCFG induces probabilities on terminal strings, where the probability $p_{G,w}$ of string $w \in \Sigma^*$ is the probability that G derives w. A basic computational problem is: given a SCFG G and string w, compute the probability $p_{G,w}$ of w. This probability is generally irrational, thus we want to compute it approximately to desired accuracy $\delta > 0$. We give the first polynomial-time algorithm for this problem that works for arbitrary SCFGs.

THEOREM 5.1. *There is a polynomial-time algorithm that, given as input a SCFG G, a string w and a rational $\delta > 0$ in binary representation, approximates the probability $p_{G,w}$ within δ, i.e., computes a value v such that $|v - p_{G,w}| < \delta$.*

The heart of the algorithm involves the transformation of the given SCFG G to another "approximately equivalent" SCFG G' with rational rule probabilities that is in Chomsky Normal Form (CNF). More precisely, for every SCFG G there is a SCFG G'' in CNF that is *equivalent* to G in the sense that it gives the same probability to all the strings of Σ^*. However, it may be the case that any such grammar must have irrational probabilities, and thus cannot be computed explicitly. Our algorithm computes a CNF SCFG grammar G' that has the same structure as (i.e., the same rules as) such an equivalent CNF grammar G'', and has rational rule probabilities that approximate the probabilities of G'' to sufficient accuracy δ (we say that G' *δ-approximates* G'') such that $p_{G',w}$ provides the desired approximation to $p_{G,w}$, for all strings w up to a desired length N.

THEOREM 5.2. *There is a polynomial-time algorithm that, given a SCFG G, a natural number N in unary, and a rational $\delta > 0$ in binary, computes a new SCFG G' in CNF that δ-approximates a SCFG in CNF that is equivalent to G, and furthermore $|p_{G,w} - p_{G',w}| \leq \delta$ for all strings w of length at most N.*

The algorithm for Theorem 5.2 involves a series of transformations. There are two complex steps in the series. The first is elimination of ϵ-rules. This requires introduction of irrational rule probabilities if we were to preserve equivalence, and thus can only be done approximately. We effectively show that computation of the desired rule probabilities in this transformation can be reduced to computation of termination probabilities for certain auxiliary SCFGs; furthermore, the structure of the construction has the property that the reduction essentially preserves approximations. The construction is based on building a *conditioned* SCFG, G^c. For every rule $A \to \gamma$ of the original SCFG, G, there can be several corresponding rules associated with nonterminal A in G^c, and these rules have as their sum total probability the *conditional* probability of the rule $A \to \gamma$ in G, conditioned on the event that the string generated by a parse tree rooted at A is non-empty. The needed conditional probabilities for rules in G^c can be expressed via rational expressions involving termination probabilities of auxiliary SCFGs. (Our proof of this is related in spirit to, but quite different in its details from, a *conditioned summary chain* construction given in [15] for recursive Markov chains.)

The second complicated step is the elimination of unary rules. This requires the solution of certain linear systems whose coefficients are irrational (and hence can only be approximated) and furthermore some of the coefficients may be extremely small (doubly exponentially small) positive values, which could potentially cause the system to be very ill-conditioned. We show that fortunately, despite the small coefficients, bad ill-conditioning does not happen. We do so via a careful analysis of the structure of the constructed grammar and the associated linear systems. The details are quite involved and are given in the full version [11].

Once we have an approximately equivalent CNF SCFG G', we can compute $p_{G',w}$ using a well-known variant of the CKY parsing algorithm, which runs in polynomial time in the unit-cost RAM model, but in principle the numbers involved may become exponentially long in their encoding size. We show that by suitable rounding we can do the computation approximately with sufficient accuracy to obtain a desired approximation of the probability $p_{G,w}$ in P-time in the Turing model, thus proving Theorem 5.1.

6. CONCLUSIONS

We showed that we can compute to within any desired precision the least fixed point solution of a multivariate system of monotone probabilistic polynomial equations, in time polynomial in the encoding size of the system and in the number of bits of precision, in the standard Turing model of computation. Applications of this result include efficient P-time computation, to within any desired precision, of the extinction probabilities of multi-type branching processes, the termination probabilities of 1-exit recursive Markov chains, and the probability of the language generated by a stochastic context-free grammar. We also established some further applications for SCFGs, namely we gave a P-time algorithm for computing approximately the probability of a given string, and for computing an approximately equivalent SCFG in Chomsky Normal Form.

There are several directions for further research that deserve investigation. One is to identify other broad, interesting classes of systems of equations that are similarly tractable.

On the one hand, we know that for the broader class of monotone polynomial systems, it is PosSLP-hard to approximate the least fixed point within any nontrivial accuracy [13]. On the positive side, we have very recently extended significantly the results of this paper, by obtaining P-time algorithms for computing the least fixed point solution of max (min) probabilistic polynomial systems which are augmented with the max (respectively, min) operator. These results allow us to approximate in P-time the optimal values of, and to compute ϵ-optimal strategies for, branching Markov decision processes and 1-exit recursive Markov decision processes [10], where the objective is to maximize, or to minimize, the probability of extinction and termination. Our P-time results involve a generalization of Newton's method to the setting of max-(min-)PPSs.

Another direction is to develop polynomial-time algorithms for other important problems concerning SCFGs. We note that Chomsky Normal Form is used as the starting point in the literature for several other such problems on SCFGs. Can we solve these problems for general SCFGs? Having an approximately equivalent grammar in CNF does not automatically yield approximate solutions to these problems. However, we expect that Theorem 5.2 and the techniques we developed in this paper will help in the development of P-time algorithms for various other problems that work for arbitrary SCFGs.

Finally, another interesting direction is to explore further the implications of our results in the area of verification of probabilistic systems, for the (approximate) model checking of LTL and ω-regular properties for 1-exit recursive Markov chains (see [15]). Such questions are related to computing (approximately) the probability that an SCFG generates a string in a given regular language; several problems on SCFGs that have been studied in the literature can be cast in this framework for different types of regular languages.

Acknowledgements

Research partially supported by NSF Grant CCF-1017955.

7. REFERENCES

[1] S. P. Abney, D. A. McAllester, and F. Pereira. Relating probabilistic grammars and automata. In *Proc. of 27th Meeting of the Ass. for Computational Linguistics (ACL'99)*, pages 542–549, 1999.

[2] E. Allender, P. Bürgisser, J. Kjeldgaard-Pedersen, and P. B. Miltersen. On the complexity of numerical analysis. *SIAM J. Comput.*, 38(5):1987–2006, 2009.

[3] T. Apostol. *Mathematical Analysis*. Addison-Wesley, 2nd edition, 1974.

[4] T. Brázdil, J. Esparza, and A. Kučera. Analysis and prediction of the long-run behavior of probabilistic sequential programs with recursion. In *Proc. FOCS*, pages 521–530, 2005.

[5] R. J. Duffin, E. L. Peterson, and C. Zener. *Geometric Programming*. John Wiley and Sons, 1967.

[6] R. Durbin, S. R. Eddy, A. Krogh, and G. Mitchison. *Biological Sequence Analysis: Probabilistic models of Proteins and Nucleic Acids*. Cambridge U. Press, 1999.

[7] J. Esparza, A. Gaiser, and S. Kiefer. Computing least fixed points of probabilistic systems of polynomials. In *Proc. 27th STACS*, pages 359–370, 2010.

[8] J. Esparza, S. Kiefer, and M. Luttenberger. Computing the least fixed point of positive polynomial systems. *SIAM Journal on Computing*, 39(6):2282–2355, 2010.

[9] J. Esparza, A. Kučera, and R. Mayr. Model checking probabilistic pushdown automata. *Logical Methods in Computer Science*, 2(1):1 – 31, 2006.

[10] K. Etessami, A. Stewart, and M. Yannakakis. Polynomial-time algorithms for branching Markov decision processes and probabilistic min(max) polynomial Bellman equations. ArXiv:1202.4798, 2012.

[11] K. Etessami, A. Stewart, and M. Yannakakis. Polynomial-time algorithms for multi-type branching processes and stochastic context-free grammars. In *Proc. 44th ACM Symposium on Theory of Computing (STOC)*, 2012. ArXiv:1201.2374 (full preprint of this paper).

[12] K. Etessami, D. Wojtczak, and M. Yannakakis. Quasi-birth-death processes, tree-like QBDs, probabilistic 1-counter automata, and pushdown systems. *Perform. Eval.*, 67(9):837–857, 2010.

[13] K. Etessami and M. Yannakakis. Recursive markov chains, stochastic grammars, and monotone systems of nonlinear equations. *Journal of the ACM*, 56(1), 2009.

[14] K. Etessami and M. Yannakakis. On the complexity of Nash equilibria and other fixed points. *SIAM Journal on Computing*, 39(6):2531–2597, 2010.

[15] K. Etessami and M. Yannakakis. Model checking of recursive probabilistic systems. *ACM Transactions on Computational Logic*, 13(2), 2012 (available online).

[16] R. Fagin, A. Karlin, J. Kleinberg, P. Raghavan, S. Rajagopalan, R. Rubinfeld, M. Sudan, and A. Tomkins. Random walks with "back buttons". In *Proc. ACM Symp. on Theory of Computing (STOC)*, pages 484–493, 2000.

[17] P. Haccou, P. Jagers, and V. A. Vatutin. *Branching Processes: Variation, Growth, and Extinction of Populations*. Cambridge U. Press, 2005.

[18] T. E. Harris. *The Theory of Branching Processes*. Springer-Verlag, 1963.

[19] M. Kimmel and D. E. Axelrod. *Branching processes in biology*. Springer, 2002.

[20] A. N. Kolmogorov and B. A. Sevastyanov. The calculation of final probabilities for branching random processes. *Doklady*, 56:783–786, 1947. (Russian).

[21] C. Manning and H. Schütze. *Foundations of Statistical Natural Language Processing*. MIT Press, 1999.

[22] M.-J. Nederhof and G. Satta. Computing partition functions of PCFGs. *Research on Language and Computation*, 6(2):139–162, 2008.

[23] M.-J. Nederhof and G. Satta. Probabilistic parsing. *New Developments in Formal Languages and Applications*, 113:229–258, 2008.

[24] D. Wojtczak and K. Etessami. Premo: an analyzer for probabilistic recursive models. In *Proc. 13th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, pages 66–71, 2007.