# Space-efficient scheduling of stochastically generated tasks

Tomáš Brázdil[1]⋆, Javier Esparza[2], Stefan Kiefer[3]⋆⋆, and Michael Luttenberger[2]

[1] Faculty of Informatics, Masaryk University, Brno, Czech Republic
[2] Institut für Informatik, Technische Universität München, Germany
[3] Oxford University Computing Laboratory, UK

**Abstract.** We study the problem of scheduling tasks for execution by a processor when the tasks can stochastically generate new tasks. Tasks can be of different types, and each type has a fixed, known probability of generating other tasks. We present results on the random variable $S^\sigma$ modeling the maximal space needed by the processor to store the currently active tasks when acting under the scheduler $\sigma$. We obtain tail bounds for the distribution of $S^\sigma$ for both offline and online schedulers, and investigate the expected value $\mathbb{E}[S^\sigma]$.

## 1 Introduction

We study the problem of scheduling tasks that can stochastically generate new tasks. We assume that the execution of a task $\tau$ can generate a set of subtasks. Tasks can be of different types, and each type has a fixed, known probability of generating new subtasks. Systems of tasks can be described using a notation similar to that of stochastic grammars. For instance

$$X \xhookrightarrow{0.2} \langle X, X\rangle \qquad X \xhookrightarrow{0.3} \langle X, Y\rangle \qquad X \xhookrightarrow{0.5} \emptyset \qquad Y \xhookrightarrow{0.7} \langle X\rangle \qquad Y \xhookrightarrow{0.3} \langle Y\rangle$$

describes a system with two types of tasks. Tasks of type $X$ can generate 2 tasks of type $X$, one task of each type, or zero tasks with probabilities $0.2$, $0.3$, and $0.5$, respectively (angular brackets denote multisets). Tasks of type $Y$ can generate one task, of type $X$ or $Y$, with probability $0.7$ and $0.3$. Tasks are executed by one processor. The processor repeatedly selects a task from a pool of unprocessed tasks, processes it, and puts the generated subtasks (if any) back into the pool. The pool initially contains one task of type $X_0$, and the next task to be processed is selected by a *scheduler*.

We study random variables modeling the time and space needed to *completely* execute a task $\tau$, i.e., to empty the pool of unprocessed tasks assuming that initially the pool only contains task $\tau$. We assume that processing a task takes one time unit, and storing it in the pool takes a unit of memory. So the *completion time* is given by the total number of tasks processed, and the *completion space* by the maximum size reached by the pool during the computation. The completion time has been studied in [13], and so the bulk of the paper is devoted to studying the distribution of the completion space for different classes of schedulers.

⋆ Supported by Czech Science Foundation, grant No. P202/10/1469.
⋆⋆ Supported by the EPSRC project *Automated Verification of Probabilistic Programs*.

Our computational model is abstract, but relevant for different scenarios. In the context of search problems, a task is a problem instance, and the scheduler is part of a branch-and-bound algorithm (see e.g. [18]). In the more general context of multithreaded computations, a task models a thread, which may generate new threads. The problem of scheduling multithreaded computations space-efficiently on *multi*processor machines has been extensively studied (see e.g. [21, 6, 2, 1]). These papers assume that schedulers know nothing about the program, while we consider the case in which stochastic information on the program behaviour is available (obtained from sampling).

We study the performance of *online* schedulers that know only the past of the computation, and compare them with the *optimal offline* scheduler, which has complete information about the future. Intuitively, this scheduler has access to an oracle that knows how the stochastic choices will be resolved. The oracle can be replaced by a machine that inspects the code of a task and determines which subtasks it will generate (if any).

We consider task systems with completion probability 1, which can be further divided into those with finite and infinite expected completion time, often called *subcritical* and *critical*. Many of our results are related to the probability generating functions (pgfs) associated to a task system. The functions for the example above are $f_X(x, y) = 0.2x^2 + 0.3xy + 0.5$ and $f_Y(x, y) = 0.7x + 0.3y$, and the reader can easily guess the formal definition. The completion probability is the least fixed point of the system of pgfs [17].

Our first results (Section 3) concern the distribution of the completion space $S^{op}$ of the optimal offline scheduler $op$ on a fixed but arbitrary task system with $\boldsymbol{f}(\boldsymbol{x})$ as pgfs (in vector form). We exhibit a very surprising connection between the probabilities $\Pr[S^{op} = k]$ and the *Newton approximants* to the least fixed point of $\boldsymbol{f}(\boldsymbol{x})$ (the approximations to the least fixed point obtained by applying Newton's method for approximating a zero of a differentiable function to $\boldsymbol{f}(\boldsymbol{x}) - \boldsymbol{x} = \boldsymbol{0}$ with seed $\boldsymbol{0}$). This connection allows us to apply recent results on the convergence speed of Newton's method [19, 12], leading to tail bounds of $S^{op}$, i.e., bounds on $\Pr[S^{op} \geq k]$. We then study (Section 4) the distribution of $S^\sigma$ for an online scheduler $\sigma$, and obtain upper and lower bounds for the performance of *any* online scheduler in subcritical systems. These bounds suggest a way of assigning weights to task types reflecting how likely they are to require large space. We study *light-first* schedulers, in which "light" tasks are chosen before "heavy" tasks with larger components, and obtain an improved tail bound.

So far we have assumed that there are no dependencies between tasks, requiring a task to be executed before another. We study in Section 4.3 the case in which a task can only terminate after all the tasks it has (recursively) spawned have terminated. These are the *strict* computations studied in [6]. The optimal scheduler in this case is the *depth-first* scheduler, i.e., the one that completely executes the child task before its parent, resulting in the familiar stack-based execution. Under this scheduler our tasks are equivalent to special classes of recursive state machines [15] and probabilistic pushdown automata [14]. We determine the exact asymptotic performance of depth-first schedulers, hereby making use of recent results [8].

We restrict ourselves to the case in which a task has at most two children, i.e., all rules $X \overset{p}{\hookrightarrow} \langle X_1, \ldots, X_n \rangle$ satisfy $n \leq 2$. This case already allows to model the forking-mechanism underlying many multithreaded operating systems, e.g. Unix-like systems.

*Related work.* Space-efficient scheduling for search problems or multithreaded computations has been studied in [18, 21, 6, 2, 1]. These papers assume that nothing is known about the program generating the computations. We study the case in which statistical information is available on the probability that computations split or die. The theory of *branching processes* studies stochastic processes modeling populations whose members can reproduce or die [17, 4]. In computer science terminology, all existing work on branching processes assumes that the number of processors is *unbounded* [3, 7, 20, 22, 24, 26]. To our knowledge, we are the first to study the 1-processor case.

*Structure of the paper.* The rest of the paper is structured as follows. The preliminaries in Section 2 formalize the notions from the introduction and summarize known results on which we build. In Section 3 we study optimal offline schedulers. Section 4 is dedicated to online schedulers. First we prove performance bounds that hold uniformly for all online schedulers, then we prove improved bounds for light-first schedulers, and finally we determine the exact asymptotic behaviour of depth-first schedulers. In Section 5 we obtain several results on the expected space consumption under different schedulers. Section 6 contains some conclusions. Full proofs can be found in [9].

## 2 Preliminaries

Let $A$ be a finite set. We regard elements of $\mathbb{N}^A$ and $\mathbb{R}^A$ as *vectors* and use boldface (like $\boldsymbol{u}, \boldsymbol{v}$) to denote vectors. The vector whose components are all $0$ (resp. $1$) is denoted by $\mathbf{0}$ (resp. $\mathbf{1}$). We use angular brackets to denote multisets and often identify multisets over $A$ and vectors indexed by $A$. For instance, if $A = \{X, Y\}$ and $\boldsymbol{v} \in \mathbb{N}^A$ with $\boldsymbol{v}_X = 1$ and $\boldsymbol{v}_Y = 2$, then $\boldsymbol{v} = \langle X, Y, Y \rangle$. We often shorten $\langle a \rangle$ to $a$. $M_A^{\leq 2}$ denotes the multisets over $A$ containing at most 2 elements.

**Definition 2.1.** *A* task system *is a tuple* $\Delta = (\Gamma, \hookrightarrow, \mathit{Prob}, X_0)$ *where* $\Gamma$ *is a finite set of* task types, $\hookrightarrow \subseteq \Gamma \times M_\Gamma^{\leq 2}$ *is a set of* transition rules, $\mathit{Prob}$ *is a function assigning positive probabilities to transition rules so that for every* $X \in \Gamma$ *we have* $\sum_{X \hookrightarrow \alpha} \mathit{Prob}((X, \alpha)) = 1$, *and* $X_0 \in \Gamma$ *is the* initial type.

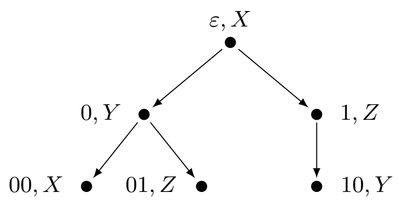
We write $X \overset{p}{\hookrightarrow} \alpha$ whenever $X \hookrightarrow \alpha$ and $\mathit{Prob}((X, \alpha)) = p$. Executions of a task system are modeled as family trees, defined as follows. Fix an arbitrary total order $\preceq$ on $\Gamma$. A *family tree* $t$ is a pair $(N, L)$ where $N \subseteq \{0, 1\}^*$ is a finite binary tree (i.e. a prefix-closed finite set of words over $\{0, 1\}$) and $L : N \hookrightarrow \Gamma$ is a labelling such that every node $w \in N$ satisfies one of the following conditions: $w$ is a leaf and $L(w) \hookrightarrow \varepsilon$, or $w$ has a unique child $w0$, and $L(w)$ satisfies $L(w) \hookrightarrow L(w0)$, or $w$ has two children $w0$ and $w1$, and $L(w0), L(w1)$ satisfy $L(w) \hookrightarrow \langle L(w0), L(w1) \rangle$ and $L(w0) \preceq L(w1)$. Given a node $w \in N$, the subtree of $t$ rooted at $w$, denoted by $t_w$, is the family tree $(N', L')$ such that $w' \in N'$ iff $ww' \in N$ and $L'(w') = L(ww')$ for every $w' \in N'$. If a tree $t$ has a subtree $t_0$ or $t_1$, we call this subtree a *child* of $t$. (So, the term *child* can refer to a node or a tree, but there will be no confusion.)

We define a function $\Pr$ which, loosely speaking, assigns to a family tree $t = (N, L)$ its probability (see the assumption below). Assume that the root of $t$ is labeled by $X$. If $t$ consists only of the root, and $X \overset{p}{\hookrightarrow} \varepsilon$, then $\Pr[t] = p$; if the root has only one child (the node 0) labeled by $Y$, and $X \overset{p}{\hookrightarrow} Y$, then $\Pr[t] = p \cdot \Pr[t_0]$; if the root has two children (the nodes 0 and 1) labeled by $Y$ and $Z$, and $X \overset{p}{\hookrightarrow} \langle Y, Z\rangle$, then $\Pr[t] = p{\cdot}\Pr[t_0]{\cdot}\Pr[t_1]$. We denote by $\mathcal{T}_X$ the set of all family trees whose root is labeled by $X$, and by $\Pr_X$ the restriction of $\Pr$ to $\mathcal{T}_X$. We drop the subscript of $\Pr_X$ if $X$ is understood.

*Example 2.2.* Figure 1 shows (a) a task system with $\Gamma = \{X, Y, Z\}$; and (b) a family tree $t$ of the system with probability $\Pr[t] = 0.25 \cdot 0.1 \cdot 0.75 \cdot 0.6 \cdot 0.4 \cdot 0.9$. The name and label of a node are written close to it.

$$X \xhookrightarrow{0.25} \langle Y, Z\rangle \qquad Y \xhookrightarrow{0.1} \langle X, Z\rangle \qquad Z \xhookrightarrow{0.4} \langle Y\rangle$$
$$X \xhookrightarrow{0.75} \emptyset \qquad Y \xhookrightarrow{0.9} \emptyset \qquad Z \xhookrightarrow{0.6} \emptyset$$

(a)



(b)

**Fig. 1.** (a) A task system. (b) A family tree.

*Assumptions.* Throughout the paper we assume that a task system $\Delta = (\Gamma, \hookrightarrow, \mathit{Prob}, X_0)$ satisfies the following two conditions for every type $X \in \Gamma$: (1) $X$ is *reachable* from $X_0$, meaning that some tree in $\mathcal{T}_{X_0}$ contains a node labeled by $X$, and (2) $\Pr[\mathcal{T}_X] = \sum_{t \in \mathcal{T}_X} \Pr[t] = 1$. So we assume that $(\mathcal{T}_X, \Pr_X)$ is a discrete probability space with $\mathcal{T}_X$ as set of elementary events and $\Pr_X$ as probability function. This is the formal counterpart to assuming that every task is completed with probability 1.

**Proposition 2.3.** *It can be decided in polynomial time whether assumptions (1) and (2) are satisfied.*

*Proof.* (1) is trivial. For (2) let the *probability generating function* (pgf) of the task system be defined as the function $\boldsymbol{f} : \mathbb{R}^\Gamma \to \mathbb{R}^\Gamma$ of $\Delta$ where for every $X \in \Gamma$

$$\boldsymbol{f}_X(\boldsymbol{v}) = \sum_{X \overset{p}{\hookrightarrow} \langle Y, Z\rangle} p \cdot \boldsymbol{v}_Y \cdot \boldsymbol{v}_Z + \sum_{X \overset{p}{\hookrightarrow} \langle Y\rangle} p \cdot \boldsymbol{v}_Y + \sum_{X \overset{p}{\hookrightarrow} \emptyset} p\,.$$

It is well known (see e.g. [17]) that (2) holds iff the least nonnegative fixed point of $\boldsymbol{f}$ equals $\mathbf{1}$, which is decidable in polynomial time [15]. $\square$

*Derivations and schedulers.* Let $t = (N, L)$ be a family tree. A *state* of $t$ is a maximal subset of $N$ in which no node is a proper prefix of another node (graphically, no node is a proper descendant of another node). The elements of a state are called *tasks*. If $s$ is a state and $w \in s$, then the *$w$-successor of $s$* is the uniquely determined state $s'$ defined as follows: if $w$ is a leaf of $N$, then $s' = s \setminus \{w\}$; if $w$ has one child $w0$, then $s' = (s \setminus \{w\}) \cup \{w0\}$; if $w$ has two children $w0$ and $w1$, then $s' = (s \setminus \{w\}) \cup \{w0, w1\}$. We write $s \Rightarrow s'$ if $s'$ is the $w$-successor of $s$ for some $w$. A *derivation of $t$* is a sequence

$s_1 \Rightarrow \ldots \Rightarrow s_k$ of states such that $s_1 = \{\epsilon\}$ and $s_k = \emptyset$. A *scheduler* is a mapping $\sigma$ that assigns to a family tree $t$ a derivation $\sigma(t)$ of $t$. If $\sigma(t) = (s_1 \Rightarrow \ldots \Rightarrow s_k)$, then for every $1 \leq i < k$ we denote by $\sigma(t)[i]$ a task of $s_i$ such that $s_{i+1}$ is the $\sigma(t)[i]$-successor of $s_i$. Intuitively, $\sigma(t)[i]$ is the task of $s_i$ scheduled by $\sigma$. This definition allows for schedulers that know the tree, and so how future tasks will behave. In Section 4 we define and study online schedulers which only know the past of the computation. Notice that schedulers are deterministic (non-randomized).

*Example 2.4.* A scheduler $\sigma_1$ may schedule the tree $t$ in Figure 1 as follows: $\{\varepsilon\} \Rightarrow \{0,1\} \Rightarrow \{0,10\} \Rightarrow \{0\} \Rightarrow \{00,01\} \Rightarrow \{01\} \Rightarrow \{\}$. Let $\sigma_2$ be the scheduler which always picks the least unprocessed task w.r.t. the lexicographical order on $\{0,1\}^*$. (This is an example of an online scheduler.) It schedules $t$ as follows: $\{\varepsilon\} \Rightarrow \{0,1\} \Rightarrow \{00,01,1\} \Rightarrow \{01,1\} \Rightarrow \{1\} \Rightarrow \{10\} \Rightarrow \{\}$.

*Time and space.* Given $X \in \Gamma$, we define a random variable $T_X$, the *completion time of* $X$, that assigns to a tree $t \in \mathcal{T}_X$ its number of nodes. Assuming that tasks are executed for one time unit before its generated subtasks are returned to the pool, $T_X$ corresponds to the time required to completely execute $X$. Our assumption (2) guarantees that $T_X$ is finite with probability 1, but its expectation $\mathbb{E}[T_X]$ may or may not be finite. A task system $\Delta$ is called *subcritical* if $\mathbb{E}[T_X]$ is finite for every $X \in \Gamma$. Otherwise it is called *critical*. If $\Delta$ is subcritical, then $\mathbb{E}[T_X]$ can be easily computed by solving a system of linear equations [13]. The notion of criticality comes from the theory of branching processes, see e.g. [17, 4]. Here we only recall the following results:

**Proposition 2.5 ([17, 15]).** *Let $\Delta$ be a task system with pgf $\boldsymbol{f}$. Denote by $\boldsymbol{f}'(\mathbf{1})$ the Jacobian matrix of partial derivatives of $\boldsymbol{f}$ evaluated at $\mathbf{1}$. If $\Delta$ is critical, then the spectral radius of $\boldsymbol{f}'(\mathbf{1})$ is equal to* $1$*; otherwise it is strictly less than* $1$. *It can be decided in polynomial time whether $\Delta$ is critical.*

A state models a pool of tasks awaiting to be scheduled. We are interested in the maximal size of the pool during the execution of a derivation. So we define the random *completion space* $S_X^\sigma$ as follows. If $\sigma(t) = (s_1 \Rightarrow \ldots \Rightarrow s_k)$, then $S_X^\sigma(t) := \max\{|s_1|, \ldots, |s_k|\}$, where $|s_i|$ is the cardinality of $s_i$. Sometimes we write $S^\sigma(t)$, meaning $S_X^\sigma(t)$ for the type $X$ labelling the root of $t$. If we write $S^\sigma$ without specifying the application to any tree, then we mean $S_{X_0}^\sigma$.

*Example 2.6.* For the schedulers of Example 2.4 we have $S^{\sigma_1}(t) = 2$ and $S^{\sigma_2}(t) = 3$.

## 3 Optimal (Offline) Schedulers

Let $S^{op}$ be the random variable that assigns to a family tree the minimal completion space of its derivations. We call $S^{op}(t)$ the *optimal completion space* of $t$. The optimal scheduler assigns to each tree a derivation with optimal completion space. In the multithreading scenario, it corresponds to a scheduler that can inspect the code of a thread and decide whether it will spawn a new thread or not. Note that, although the optimal scheduler "knows" how the stochastic choices are resolved, the optimal completion space $S^{op}(t)$ is still a random variable, because it depends on a random tree. The following proposition characterizes the optimal completion space of a tree in terms of the optimal completion space of its children.

**Proposition 3.1.** *Let $t$ be a family tree. Then*

$$S^{op}(t) = \begin{cases} \min\left\{ \begin{array}{l} \max\{S^{op}(t_0)+1, S^{op}(t_1)\}, \\ \max\{S^{op}(t_0), S^{op}(t_1)+1\} \end{array} \right\} & \text{if } t \text{ has two children } t_0, t_1 \\ S^{op}(t_0) & \text{if } t \text{ has exactly one child } t_0 \\ 1 & \text{if } t \text{ has no children.} \end{cases}$$

*Proof sketch.* The only nontrivial case is when $t$ has two children $t_0$ and $t_1$. Consider the following schedulings for $t$, where $i \in \{0,1\}$: Execute first all tasks of $t_i$ and then all tasks of $t_{1-i}$; within both $t_i$ and $t_{1-i}$, execute tasks in optimal order. While executing $t_i$, the root task of $t_{1-i}$ remains in the pool, and so the completion space is $s(i) = \max\{S^{op}(t_i)+1, S^{op}(t_{1-i})\}$. The optimal scheduler chooses the value of $i$ that minimizes $s(i)$. □

Given a type $X$, we are interested in the probabilities $\Pr[S_X^{op} \leq k]$ for $k \geq 1$. Proposition 3.1 yields a recurrence relation which at first sight seems difficult to handle. However, using results of [11, 10] we can exhibit a surprising connection between these probabilities and the pgf $\boldsymbol{f}$.

Let $\boldsymbol{\mu}$ denote the least fixed point of $\boldsymbol{f}$ and recall from the proof of Proposition 2.3 that $\boldsymbol{\mu} = \mathbf{1}$. Clearly, $\mathbf{1}$ is a zero of $\boldsymbol{f}(\boldsymbol{x}) - \boldsymbol{x}$. It has recently been shown that $\boldsymbol{\mu}$ can be computed by applying to $\boldsymbol{f}(\boldsymbol{x}) - \boldsymbol{x}$ Newton's method for approximating a zero of a differentiable function [15, 19]. More precisely, $\boldsymbol{\mu} = \lim_{k\to\infty} \boldsymbol{\nu}^{(k)}$ where

$$\boldsymbol{\nu}^{(0)} = \mathbf{0} \quad \text{and} \quad \boldsymbol{\nu}^{(k+1)} = \boldsymbol{\nu}^{(k)} + (I - \boldsymbol{f}'(\boldsymbol{\nu}^{(k)}))^{-1} \left( \boldsymbol{f}(\boldsymbol{\nu}^{(k)}) - \boldsymbol{\nu}^{(k)} \right)$$

and $\boldsymbol{f}'(\boldsymbol{\nu}^{(k)})$ denotes the Jacobian matrix of partial derivatives of $\boldsymbol{f}$ evaluated at $\boldsymbol{\nu}^{(k)}$ and $I$ the identity matrix. Computing $\boldsymbol{\mu}$, however, is in our case uninteresting, because we already know that $\boldsymbol{\mu} = \mathbf{1}$. So, why do we need Newton's method? Because the sequence of Newton approximants provides exactly the information we are looking for:

**Theorem 3.2.** $\Pr[S_X^{op} \leq k] = \boldsymbol{\nu}_X^{(k)}$ *for every type $X$ and every $k \geq 0$.*

*Proof sketch.* We illustrate the proof idea on the one-type task system with pgf $f(x) = px^2 + q$, where $q = 1-p$. Let $\mathcal{T}_{\leq k}$ and $\mathcal{T}_{=k}$ denote the sets of trees $t$ with $S^{op}(t) \leq k$ and $S^{op}(t) = k$, respectively. We show $\Pr[\mathcal{T}_{\leq k}] = \nu^{(k)}$ for all $k$ by induction on $k$. The case $k = 0$ is trivial. Assume that $\nu^{(k)} = \Pr[\mathcal{T}_{\leq k}]$ holds for some $k \geq 0$. We prove $\Pr[\mathcal{T}_{\leq k+1}] = \nu^{(k+1)}$. Notice that

$$\nu^{(k+1)} := \nu^{(k)} + \tfrac{f(\nu^{(k)}) - \nu^{(k)}}{1 - f'(\nu^{(k)})} = \nu^{(k)} + (f(\nu^{(k)}) - \nu^{(k)}) \cdot \textstyle\sum_{i=0}^{\infty} f'(\nu^{(k)})^i.$$

Let $\mathcal{B}_{k+1}^{(0)}$ be the set of trees that have two children both of which belong to $\mathcal{T}_{=k}$, and, for every $i \geq 0$, let $\mathcal{B}_{k+1}^{(i+1)}$ be the set of trees with two children, one belonging to $\mathcal{T}_{\leq k}$, the other one to $\mathcal{B}_{k+1}^{(i)}$. By Proposition 3.1 we have $\mathcal{T}_{\leq k+1} = \bigcup_{i\geq 0} \mathcal{B}_{k+1}^{(i)}$. We prove $\Pr\left[\mathcal{B}_{k+1}^{(i)}\right] = f'(\nu^{(k)})^i \; (f(\nu^{(k)} - \nu^{(k)})$ by an (inner) induction on $i$, which completes the proof. For the base $i = 0$, let $\mathcal{A}_{\leq k}$ be the set of trees with two children in $\mathcal{T}_{\leq k}$; by induction hypothesis we have $\Pr[\mathcal{A}_{\leq k}] = p\nu^{(k)}\nu^{(k)}$. In a tree of $\mathcal{A}_{\leq k}$ either (a) both children belong to $\mathcal{T}_{=k}$, and so $t \in \mathcal{B}^{(0)}_{k+1}$, or (b) at most one child belongs to $\mathcal{T}_{=k}$. By Proposition 3.1, the trees satisfying (b) belong to $\mathcal{T}_{\leq k}$. In fact, a stronger property holds: a tree of $\mathcal{T}_{\leq k}$ either satisfies (b) or it has one single node. Since the probability of the tree with one node is $q$, we get $\Pr[\mathcal{A}_{\leq k}] = \Pr\left[\mathcal{B}^{(0)}_{k+1}\right] + \Pr[\mathcal{T}_{\leq k}] - q$. Applying the induction hypothesis again we obtain $\Pr\left[\mathcal{B}^{(0)}_{k+1}\right] = p\nu^{(k)}\nu^{(k)} + q - \nu^{(k)} = f(\nu^{(k)}) - \nu^{(k)}$. For the induction step, let $i > 0$. Divide $\mathcal{B}^{(i)}_{k+1}$ into two sets, one containing the trees whose left (right) child belongs to $\mathcal{B}^{(i)}_{k+1}$ (to $\mathcal{T}_{\leq k}$), and the other the trees whose left (right) child belongs to $\mathcal{T}_{\leq k}$ (to $\mathcal{B}^{(i)}_{k+1}$). Using both induction hypotheses, we get that the probability of each set is $p\nu^{(k)} f'(\nu^{(k)})^i (f(\nu^{(k)}) - \nu^{(k)})$. So $\Pr\left[\mathcal{B}^{(i+1)}_{k+1}\right] = (2p\nu^{(k)}) \cdot f'(\nu^{(k)})^i (f(\nu^{(k)}) - \nu^{(k)})$. Since $f(x) = px^2 + q$ we have $f'(\nu^{(k)}) = 2p\nu^{(k)}$, and so $\Pr\left[\mathcal{B}^{(i+1)}_{k+1}\right] = f'(\nu^{(k)})^{i+1}(f(\nu^{(k)} - \nu^{(k)})$ as desired. $\square$

*Example 3.3.* Consider the task system $X \overset{p}{\hookrightarrow} \langle X, X\rangle,\ X \overset{q}{\hookrightarrow} \emptyset$ with pgf $f(x) = px^2 + q$, where $p$ is a parameter and $q = 1 - p$. The least fixed point of $f$ is 1 if $p \leq 1/2$ and $q/p$ otherwise. So we consider only the case $p \leq 1/2$. The system is critical for $p = 1/2$ and subcritical for $p < 1/2$. Using Newton approximants we obtain the following recurrence relation for the distribution of the optimal scheduler, where $p_k := \Pr[S^{op} \geq k] = 1 - \nu^{(k-1)}$: $p_{k+1} = (pp_k^2)/(1 - 2p + 2pp_k)$. In particular, for the critical value $p = 1/2$ we get $p_k = 2^{1-k}$ and $\mathbb{E}[S^{op}] = \sum_{k\geq 1} \Pr[S^{op} \geq k] = 2$.

Theorem 3.2 allows to compute the probability mass function of $S^{op}$. As a Newton iteration requires $\mathcal{O}(|\Gamma|^3)$ arithmetical operations, we obtain the following corollary, where by the unit cost model we refer to the cost in the Blum-Shub-Smale model, in which arithmetic operations have cost 1 independently of the size of the operands [5].

**Corollary 3.4.** $\Pr[S^{op}_X = k]$ *can be computed in time* $\mathcal{O}(k \cdot |\Gamma|^3)$ *in the unit cost model.*

It is easy to see that Newton's method converges quadratically for subcritical systems (see e.g. [23]). For critical systems, it has recently been proved that Newton's method still converges linearly [19, 12]. These results lead to tail bounds for $S^{op}_X$:

**Corollary 3.5.** *For any task system* $\Delta$ *there are real numbers* $c > 0$ *and* $0 < d < 1$ *such that* $\Pr[S^{op}_X \geq k] \leq c \cdot d^k$ *for all* $k \in \mathbb{N}$. *If* $\Delta$ *is subcritical, then there are real numbers* $c > 0$ *and* $0 < d < 1$ *such that* $\Pr[S^{op}_X \geq k] \leq c \cdot d^{2^k}$ *for all* $k \in \mathbb{N}$.

## 4 Online Schedulers

From this section on we concentrate on online schedulers that only know the past of the computation. Formally, a scheduler $\sigma$ is *online* if for every tree $t$ with $\sigma(t) = (s_1 \Rightarrow \ldots \Rightarrow s_k)$ and for every $1 \leq i < k$, the task $\sigma(t)[i]$ depends only on $s_1 \Rightarrow \ldots \Rightarrow s_i$ and on the restriction of the labelling function $L$ to $\bigcup_{j=1}^{i} s_j$.

*Compact Task Systems.* Any task system can be transformed into a so-called *compact* task system such that for every scheduler of the compact task system we can construct a scheduler of the original system with nearly the same properties. A type $W$ is *compact* if there is a rule $X \hookrightarrow \langle Y, Z \rangle$ such that $X$ is reachable from $W$. A task system is *compact* if all its types are compact. *From now on we assume that task systems are compact.* This assumption is essentially without loss of generality, as we argue in [9].

## 4.1 Tail Bounds for Online Schedulers

The following main theorem gives computable lower and upper bounds which hold uniformly for all online schedulers $\sigma$.

**Theorem 4.1.** *Let $\Delta$ be subcritical.*

– *Let $\boldsymbol{v}, \boldsymbol{w} \in (1,\infty)^\Gamma$ be vectors with $\boldsymbol{f}(\boldsymbol{v}) \le \boldsymbol{v}$ and $\boldsymbol{f}(\boldsymbol{w}) \ge \boldsymbol{w}$. Denote by $\boldsymbol{v}_{min}$ and $\boldsymbol{w}_{max}$ the least component of $\boldsymbol{v}$ and the greatest component of $\boldsymbol{w}$, respectively. Then*

$$\frac{\boldsymbol{w}_{X_0} - 1}{\boldsymbol{w}_{max}^{k+2} - 1} \le \Pr[S^\sigma \ge k] \le \frac{\boldsymbol{v}_{X_0} - 1}{\boldsymbol{v}_{min}^{k} - 1} \text{ for all online schedulers } \sigma.$$

– *Vectors $\boldsymbol{v}, \boldsymbol{w} \in (1,\infty)^\Gamma$ with $\boldsymbol{f}(\boldsymbol{v}) \le \boldsymbol{v}$ and $\boldsymbol{f}(\boldsymbol{w}) \ge \boldsymbol{w}$ exist and can be computed in polynomial time.*

*Proof sketch.* Choose $h > 1$ and $\boldsymbol{u} \in (0,\infty)^\Gamma$ such that $h^{\boldsymbol{u}_X} = \boldsymbol{v}_X$ for all $X \in \Gamma$. Define for all $i \ge 1$ the variable $m^{(i)} = \boldsymbol{z}^{(i)} \bullet \boldsymbol{u}$ where "$\bullet$" denotes the scalar product, i.e., $m^{(i)}$ measures the number of tasks at time $i$ weighted by types according to $\boldsymbol{u}$. One can show that $h^{m^{(1)}}, h^{m^{(2)}}, \ldots$ is a supermartingale for any online scheduler $\sigma$, and, using the Optional Stopping Theorem [27], that $\Pr\left[\sup_i m^{(i)} \ge x\right] \le (\boldsymbol{v}_{X_0} - 1)/(h^x - 1)$ for all $x$ (see [9] for the details and [16, 25] for a similar argument on random walks). As each type has at least weight $\boldsymbol{u}_{min}$, we have that $S^\sigma \ge k$ implies $\sup_i m^{(i)} \ge k\boldsymbol{u}_{min}$. Hence $\Pr[S^\sigma \ge k] \le \Pr\left[\sup_i m^{(i)} \ge k\boldsymbol{u}_{min}\right] \le (\boldsymbol{v}_{X_0} - 1)/(\boldsymbol{v}_{min}^k - 1)$. The lower bound is shown similarly. □

All online schedulers perform within the bounds of Theorem 4.1. For an application of the upper bound, assume one wants to provide as much space as is necessary to guarantee that, say, 99.9% of the executions of a task system can run without needing additional memory. This can be accomplished, regardless of the scheduler, by providing $k$ space units, where $k$ is chosen such that the upper bound of Theorem 4.1 is at most $0.001$.

A comparison of the lower bound with Corollary 3.5 proves for subcritical task systems that the asymptotic performance of any online scheduler $\sigma$ is far away from that of the optimal offline scheduler: the ratio $\Pr[S^\sigma \ge k] / \Pr[S^{op} \ge k]$ is unbounded.

*Example 4.2.* Consider again the task system with pgf $f(x) = px^2 + q$. For $p < 1/2$ the pgf has two fixed points, 1 and $q/p$. In particular, $q/p > 1$, so $q/p$ can be used to obtain both an upper and a lower bound for online schedulers. Since there is only one type of tasks, vectors have only one component, and the maximal and minimal components coincide; moreover, in this case the exponent $k+2$ of the lower bound can be improved to $k$. So the upper and lower bounds coincide, and we get $\Pr[S^\sigma \ge k] = \frac{q/p - 1}{(q/p)^k - 1}$ for every online scheduler $\sigma$. In particular, as one intuitively expects, all online schedulers are equivalent.[4]

[4] For this example $\Pr[S^\sigma \ge k]$ can also be computed by elementary means.

### 4.2 Tail Bounds for Light-First Schedulers

We present a class of online schedulers for which a sharper upper bound than the one given by Theorem 4.1 can be proved. It may be intuitive that a good heuristic is to pick the task with the smallest expected completion time. If we compute a vector $\boldsymbol{v}$ with $\boldsymbol{f}(\boldsymbol{v}) \leq \boldsymbol{v}$ in polynomial time according to the proof of Theorem 4.1, then the type $X_{min}$ for which $\boldsymbol{v}_{X_{min}} = \boldsymbol{v}_{min}$ holds turns out to be the type with smallest expected completion time. This suggests choosing the active type $X$ with smallest component in $\boldsymbol{v}$. So we look at $\boldsymbol{v}$ as a vector of weights, and always choose the lightest active type. In fact, for this (intuitively good) scheduler we obtain two different upper bounds.

Given a vector $\boldsymbol{v}$ with $\boldsymbol{f}(\boldsymbol{v}) \leq \boldsymbol{v}$ we denote by $\sqsubseteq$ a total order on $\Gamma$ such that whenever $X \sqsubseteq Y$ then $\boldsymbol{v}_X \leq \boldsymbol{v}_Y$. If $X \sqsubseteq Y$, then we say that $X$ is lighter than $Y$. The *$\boldsymbol{v}$-light-first scheduler* is an online scheduler that, in each step, picks a task of the lightest type available in the pool according to $\boldsymbol{v}$. Theorem 4.3 below strengthens the upper bound of Theorem 4.1 for light-first schedulers. For the second part of Theorem 4.3 we use the notion of *$\boldsymbol{v}$-accumulating types*. A type $X \in \Gamma$ is $\boldsymbol{v}$-accumulating if for every $k \geq 0$ the $\boldsymbol{v}$-light-first scheduler has a nonzero probability of reaching a state with at least $k$ tasks of type $X$ in the pool.

**Theorem 4.3.** *Let $\Delta$ be subcritical and $\boldsymbol{v} \in (1,\infty)^\Gamma$ with $\boldsymbol{f}(\boldsymbol{v}) \leq \boldsymbol{v}$. Let $\sigma$ be a $\boldsymbol{v}$-light-first scheduler. Let $\boldsymbol{v}_{minmax} := \min_{X \hookrightarrow \langle Y,Z\rangle} \max\{\boldsymbol{v}_Y, \boldsymbol{v}_Z\}$ (here the minimum is taken over all transition rules with two types on the right hand side). Then $\boldsymbol{v}_{minmax} \geq \boldsymbol{v}_{min}$ and for all $k \geq 1$*

$$\Pr[S^\sigma \geq k] \leq \frac{\boldsymbol{v}_{X_0} - 1}{\boldsymbol{v}_{min}\boldsymbol{v}_{minmax}^{k-1} - 1}\,.$$

*Moreover, let $\boldsymbol{v}_{minacc} := \min\{\boldsymbol{v}_X \mid X \in \Gamma,\ X \text{is } \boldsymbol{v}\text{-accumulating}\}$. Then $\boldsymbol{v}_{minacc} \geq \boldsymbol{v}_{minmax}$, $\boldsymbol{v}_{minacc}$ can be computed in polynomial time, and there is an integer $\ell$ such that for all $k \geq \ell$*

$$\Pr[S^\sigma \geq k] \leq \frac{\boldsymbol{v}_{X_0} - 1}{\boldsymbol{v}_{min}^{\ell}\boldsymbol{v}_{minacc}^{k-\ell} - 1}\,.$$

*Proof sketch.* Recall the proof sketch of Theorem 4.1 where we used that $S^\sigma \geq k$ implies $\sup_i m^{(i)} \geq k\boldsymbol{u}_{min}$, as each type has at least weight $\boldsymbol{u}_{min}$. Let $\ell$ be such that no more than $\ell$ tasks of non-accumulating type can be in the pool at the same time. Then $S^\sigma \geq k$ implies $\sup_i m^{(i)} \geq \ell\boldsymbol{u}_{min} + (k-\ell)\boldsymbol{u}_{minacc}$ which leads to the final inequality of Theorem 4.3 in a way analogous to the proof sketch of Theorem 4.1. $\square$

Intuitively, a light-first scheduler "works against" light tasks by picking them as soon as possible. In this way it may be able to avoid the accumulation of some light types, so it may achieve $\boldsymbol{v}_{minacc} > \boldsymbol{v}_{min}$. This is illustrated in the following example.

*Example 4.4.* Consider the task system with 2 task types and pgfs $x = a_2xy + a_1y + a_0$ and $y = b_2xy + b_1y + b_0$, where $a_2 + a_1 + a_0 = 1 = b_2 + b_1 + b_0 = 1$. The system is subcritical if $a_1b_2 < a_2b_1 - a_2 + b_0$. The pgfs have a greatest fixed point $\boldsymbol{v}$ with $\boldsymbol{v}_X = (1-a_2-b_1-a_1b_2+a_2b_1)/b_2$ and $\boldsymbol{v}_Y = (1-b_1-b_2)/(a_2+a_1b_2-a_2b_1)$. We have $\boldsymbol{v}_X \leq \boldsymbol{v}_Y$ iff $a_2 - b_2 \leq a_2b_1 - a_1b_2$, and so the light-first scheduler chooses $X$ before $Y$

if this condition holds, and $Y$ before $X$ otherwise. We show that the light-first scheduler is asymptotically optimal. Assume w.l.o.g. $\boldsymbol{v}_X \leq \boldsymbol{v}_Y$. Then $X$ is not accumulating (because $X$-tasks are picked as soon as they are created), and so $\boldsymbol{v}_{minacc} = \boldsymbol{v}_Y$. So the upper bound for the light-weight scheduler yields a constant $c_2$ such that $\Pr[S^\sigma \geq k] \leq c_2/\boldsymbol{v}_Y^k$. But the general lower bound for arbitrary online schedulers states that there is a constant $c_1$ such that $\Pr[S^\sigma \geq k] \geq c_1/\boldsymbol{v}_Y^k$, so we are done.

### 4.3 Tail Bounds for Depth-first Schedulers

Space-efficient scheduling of multithreaded computations has received considerable attention [21, 6, 2, 1]. The setting of these papers is slightly different from ours, because they assume data dependencies among the threads, which may cause a thread to wait for a result from another thread. In this sense our setting is similar to that of [18], where, in thread terminology, the threads can execute independently.

These papers focus on *depth-first* computations, in which if thread $A$ has to wait for thread $B$, then $B$ was spawned by $A$ or by a descendant of $A$. The optimal scheduler is the one that, when $A$ spawns $B$, interrupts the execution of $A$ and continues with $B$; this online scheduler produces the familiar stack-based execution [6, 21].

We study the performance of this *depth-first* scheduler. Formally, a depth-first scheduler $\sigma_\lambda$ is determined by a function $\lambda$ that assigns to each rule $r = X \hookrightarrow \langle Y, Z\rangle$ either $YZ$ or $ZY$. If $\lambda(r) = YZ$, then $Z$ models the continuation of the thread $X$, while $Y$ models a new thread for whose termination $Z$ waits. The depth-first scheduler $\sigma_\lambda$ keeps as an internal data structure a word $w \in \Gamma^*$, a "stack", such that the Parikh image of $w$ is the multiset of the task types in the pool. If $w = Xw'$ for some $w' \in \Gamma^*$, then $\sigma$ picks $X$. If a transition rule $X \hookrightarrow \alpha$ "fires", then $\sigma_\lambda$ replaces $Xw'$ by $\beta w'$ where $\beta = \lambda(X \hookrightarrow \alpha)$.
Using techniques of [8] for *probabilistic pushdown systems*, we obtain the following:

**Theorem 4.5.** *Let $\Delta$ be subcritical and $\sigma$ be any depth-first scheduler. Then $\Pr[S^\sigma = k]$ can be computed in time $\mathcal{O}(k \cdot |\Gamma|^3)$ in the unit-cost model. Moreover, there is $0 < \rho < 1$ such that $\Pr[S^\sigma \geq k] \in \Theta(\rho^k)$, i.e, there are $c, C > 0$ such that $c\rho^k \leq \Pr[S^\sigma \geq k] \leq C\rho^k$ for all $k$. Furthermore, $\rho$ is the spectral radius of a nonnegative matrix $B \in \mathbb{R}^{\Gamma\times\Gamma}$, where $B$ can be computed in polynomial time.*

While the proof of Theorem 4.5 does not conceptually require much more than the results of [8], the technical details are delicate. The proof can be found in [9].

## 5 Expectations

In this section we study the expected completion space, i.e., the expectation $\mathbb{E}[S^\sigma]$ for both offline and online schedulers. Fix a task system $\Delta = (\Gamma, \hookrightarrow, Prob, X_0)$.

*Optimal (Offline) Schedulers.* The results of Section 3 allow to efficiently approximate the expectation $\mathbb{E}[S^{op}]$. Recall that for any random variable $R$ with values in the natural numbers we have $\mathbb{E}[R] = \sum_{i=1}^{\infty} \Pr[R \geq i]$. So we can (under-) approximate $\mathbb{E}[R]$ by $\sum_{i=1}^{k} \Pr[R \geq i]$ for finite $k$. We say that $k$ *terms compute $b$ bits of* $\mathbb{E}[S^{op}]$ if $\mathbb{E}[S^{op}] - \sum_{i=0}^{k-1}(1 - \boldsymbol{\nu}_{X_0}^{(i)}) \leq 2^{-b}$.

**Theorem 5.1.** *The expectation $\mathbb{E}[S^{op}]$ is finite (no matter whether $\Delta$ is critical or subcritical). Moreover, $\mathcal{O}(b)$ terms compute $b$ bits of $\mathbb{E}[S^{op}]$. If the task system $\Delta$ is subcritical, then $\log_2 b + \mathcal{O}(1)$ terms compute $b$ bits of $\mathbb{E}[S^{op}]$. Finally, computing $k$ terms takes time $\mathcal{O}(k \cdot |\Gamma|^3)$ in the unit cost model.*

*Online Schedulers.* The main result for online schedulers states that the finiteness of $\mathbb{E}[S^\sigma]$ does not depend on the choice of the online scheduler $\sigma$.

**Theorem 5.2.** *If $\Delta$ is subcritical, then $\mathbb{E}[S^\sigma]$ is finite for every online scheduler $\sigma$. If $\Delta$ is critical, then $\mathbb{E}[S^\sigma]$ is infinite for every online scheduler $\sigma$.*

*Proof sketch.* The first assertion follows from Theorem 4.1. Let $\Delta$ be critical. For this sketch we focus on the case where $X_0$ is reachable from every type. By Proposition 2.5 the spectral radius of $\boldsymbol{f}'(\mathbf{1})$ equals 1. Then Perron-Frobenius theory guarantees the existence of a vector $\boldsymbol{u}$ with $\boldsymbol{f}'(\mathbf{1})\boldsymbol{u} = \boldsymbol{u}$ and $\boldsymbol{u}_X > 0$ for all $X$. Using a martingale argument, similar to the one of Theorem 4.1, one can show that the sequence $m^{(1)}, m^{(2)}, \ldots$ with $m^{(i)} := \boldsymbol{z}^{(i)} \bullet \boldsymbol{u}$ is a martingale for every scheduler $\sigma$, and, using the Optional-Stopping Theorem, that $\Pr[S^\sigma \geq k] \geq \boldsymbol{u}_{X_0}/(k+2)$. So we have $\mathbb{E}[S^\sigma] = \sum_{k=1}^{\infty} \Pr[S^\sigma \geq k] \geq \sum_{k=1}^{\infty} \boldsymbol{u}_{X_0}/(k+2) = \infty$. $\square$

Since we can decide in polynomial time whether a system is subcritical or critical, we can do the same to decide on the finiteness of the expected completion time.

*Depth-first Schedulers.* To approximate $\mathbb{E}[S^\sigma]$ for a given depth-first scheduler $\sigma$, we can employ the same technique as for optimal offline schedulers, i.e., we approximate $\mathbb{E}[S^\sigma]$ by $\sum_{i=1}^{k} \Pr[S^\sigma \geq i]$ for finite $k$. We say that *$k$ terms compute $b$ bits of* $\mathbb{E}[S^\sigma]$ if $\mathbb{E}[S^\sigma] - \sum_{i=1}^{k} \Pr[S^\sigma \geq i] \leq 2^{-b}$.

**Theorem 5.3 (see Theorem 19 of [8]).** *Let $\Delta$ be subcritical, and let $\sigma$ be a depth-first scheduler. Then $\mathcal{O}(b)$ terms compute $b$ bits of $\mathbb{E}[S^\sigma]$, and computing $k$ terms takes time $\mathcal{O}(k \cdot |\Gamma|^3)$ in the unit cost model.*

## 6 Conclusions

We have initiated the study of scheduling tasks that can stochastically generate other tasks. We have provided strong results on the performance of both online and offline schedulers for the case of one processor and task systems with completion probability 1. It is an open problem how to compute and analyze online schedulers which are optimal in a sense. While we profited from the theory of branching processes, the theory considers (in computer science terms) systems with an unbounded number of processors, and therefore many questions had not been addressed before or even posed.

*Acknowledgement.* We thank the referees for their helpful comments.

## References

1. K. Agrawal, C.E. Leiserson, Y. He, and W.J. Hsu. Adaptive work-stealing with parallelism feedback. *ACM TOCS*, 26(3), 2008.

2. N.S. Arora, R.D. Blumofe, and C.G. Plaxton. Thread scheduling for multiprogrammed microprocessors. *Theory of Computing Systems*, 34:115–144, 2001.
3. K.B. Athreya. On the maximum sequence of a critical branching process. *Annals of Probability*, 16:502–507, 1988.
4. K.B. Athreya and P.E. Ney. *Branching Processes*. Springer, 1972.
5. L. Blum, F. Cucker, M. Shub, and S. Smale. *Complexity and Real Computation*. Springer-Verlag, 1998.
6. R.D. Blumofe and C.E. Leiserson. Scheduling multithreaded computations by work stealing. *Journal of the ACM*, 46(5):720–748, 1999.
7. K.A. Borovkov and V.A. Vatutin. On distribution tails and expectations of maxima in critical branching processes. *Journal of Applied Probability*, 33(3):614–622, 1996.
8. T. Brázdil, J. Esparza, and S. Kiefer. On the memory consumption of probabilistic pushdown automata. In *Proceedings of FSTTCS*, pages 49–60, 2009.
9. T. Brázdil, J. Esparza, S. Kiefer, and M. Luttenberger. Space-efficient scheduling of stochastically generated tasks. Technical report, 2010. Available at `http://arxiv.org/abs/1004.4286`.
10. J. Esparza, S. Kiefer, and M. Luttenberger. An extension of Newton's method to $\omega$-continuous semirings. In *DLT'07*, LNCS 4588, pages 157–168. Springer, 2007.
11. J. Esparza, S. Kiefer, and M. Luttenberger. On fixed point equations over commutative semirings. In *STACS'07*, LNCS 4397, pages 296–307. Springer, 2007.
12. J. Esparza, S. Kiefer, and M. Luttenberger. Convergence thresholds of Newton's method for monotone polynomial equations. In *STACS 2008*, pages 289–300, 2008.
13. J. Esparza, A. Kučera, and R. Mayr. Quantitative analysis of probabilistic pushdown automata: Expectations and variances. In *LICS 2005*, pages 117–126. IEEE, 2005.
14. J. Esparza, A. Kučera, and R. Mayr. Model checking probabilistic pushdown automata. In *LICS 2004*, pages 12–21. IEEE Computer Society, 2004.
15. K. Etessami and M. Yannakakis. Recursive markov chains, stochastic grammars, and monotone systems of nonlinear equations. *Journal of the ACM*, 56(1):1–66, 2009.
16. W. Feller. *An introduction to probability theory and its applications*, volume I. John Wiley & Sons, 1968.
17. T.E. Harris. *The Theory of Branching Processes*. Springer, 1963.
18. R.M. Karp and Y. Zhang. Randomized parallel algorithms for backtrack search and branch-and-bound computation. *Journal of the ACM*, 40(3):765–789, 1993.
19. S. Kiefer, M. Luttenberger, and J. Esparza. On the convergence of Newton's method for monotone systems of polynomial equations. In *STOC 2007*, pages 217–226. ACM, 2007.
20. T. Lindvall. On the maximum of a branching process. *Scandinavian Journal of Statistics*, 3:209–214, 1976.
21. G.J. Narlikar and G.E. Belloch. Space-efficient scheduling of nested parallelism. *ACM TOPLAS*, 21(1):138–173, 1999.
22. O. Nerman. On the maximal generation size of a non-critical galton-watson process. *Scandinavian Journal of Statistics*, 4(3):131–135, 1977.
23. J.M. Ortega and W.C. Rheinboldt. *Iterative solution of nonlinear equations in several variables*. Academic Press, 1970.
24. A.G. Pakes. A limit theorem for the maxima of the para-critical simple branching process. *Advances in Applied Probability*, 30:740–756, 1998.
25. F. Spitzer. *Principles of Random Walk*. Springer, 1976.
26. A. Spătaru. A maximum sequence in a critical multitype branching process. *Journal of Applied Probability*, 28(4):893–897, 1991.
27. D. Williams. *Probability with Martingales*. Cambridge University Press, 1995.