Scheduling Dags under Uncertainty∗

Grzegorz Malewicz
Department of Engineering
Google, Inc.
Mountain View, CA 94043, USA
malewicz@google.com

February 1, 2008

Abstract

This paper introduces a parallel scheduling problem where a directed acyclic graph modeling t tasks and their dependencies needs to be executed on n unreliable workers. Worker i executes task j correctly with probability \( p_{i,j} \). The goal is to find a regimen \( \Sigma \), that dictates how workers get assigned to tasks (possibly in parallel and redundantly) throughout execution, so as to minimize the expected completion time. This fundamental parallel scheduling problem arises in grid computing and project management fields, and has several applications.

We show a polynomial time algorithm for the problem restricted to the case when dag width is at most a constant and the number of workers is also at most a constant. These two restrictions may appear to be too severe. However, they are fundamentally required. Specifically, we demonstrate that the problem is NP-hard with constant number of workers when dag width can grow, and is also NP-hard with constant dag width when the number of workers can grow. When both dag width and the number of workers are unconstrained, then the problem is inapproximable within factor less than \( 5/4 \), unless P=NP.

Keywords: algorithms and theory, combinatorial optimization, grid computing, Markov chains, probabilistic failures, project management, resource constraints, scheduling of dags.

1 Introduction

Grid computing infrastructures have been developed over the past several years to enable execution of computations on shared distributed resources [10]. The machines, disks and network often operate at a slower pace or stop operating, due to hardware and software failures and sharing. Nevertheless, there is significant demand to perform scientific computations with complex task dependencies on grids (cf. [1, 2, 6, 16]). Among the most important remaining challenges is to determine how to quickly execute large-scale, sophisticated computations using unreliable resources. When a task fails to get computed correctly, then the progress of execution may be delayed because dependent tasks cannot be executed pending successful execution of the task. It is conceivable

∗Preliminary version of this work appeared in the Proceedings of the 17th ACM Symposium on Parallelism in Algorithms and Architectures (SPAA’05). Research performed in part during a visit to the Division of Mathematics and Computer Science, Argonne National Laboratory, Argonne, IL 60439 supported by NSF grant ITR-800864, and a stay with the Department of Computer Science, University of Alabama, Tuscaloosa, AL 35487.
that task dependencies and resource reliabilities play a significant role in the ability to execute computations quickly. Therefore, one would like to determine relationships among these factors, and develop algorithms for quick execution of complex computations using unreliable resources.

A similar problem arises when managing projects such as production planning or software development. Here a collection of activities and precedence constraints are given. Workers can be assigned to perform the activities. In practice, a worker assigned to an activity may fail to perform it. For example, if an activity consists of writing a piece of code and testing it, it could happen that the test fails. The manager of the project may be able to estimate the success probability of a worker assigned to an activity based on prior experience with the worker. The manager may be able to redundantly assign workers to an activity. For example, two workers may independently write a piece of code and test it; if at least one test succeeds, the activity is completed. Thus the manager faces a problem of how to assign workers to activities, possibly in parallel and redundantly, over the course of the project, so as to minimize the total time of conducting the project.

These two application areas motivate the study of the following fundamental parallel computing scheduling problem. A directed acyclic graph (dag) is given representing tasks and their dependencies. There are workers. At any given unit of time workers are assigned in some way to the tasks that are “eligible” based on precedence constraints and tasks executed thus far (any worker is assigned to at most one task at a time; more than one task may get assigned; more than one worker can be assigned to a task; workers can idle). The workers then attempt to execute the assigned tasks. The attempt of worker to execute task succeeds with probability . In the next unit of time workers are again assigned to tasks. The execution proceeds in this manner until the tasks have been executed. The goal is to determine a regimen that dictates how workers get assigned to tasks throughout execution, that minimizes the expected completion time.

1.1 Contributions

In this paper we propose and investigate a parallel scheduling problem of executing dags using unreliable workers. Our specific contributions are as follows:

(i) We introduce a new combinatorial optimization problem: given a dag with tasks and workers such that is the probability that worker executes task correctly, find a regimen that minimizes the expected completion time.

We show that the ability to solve this problem in polynomial time depends in a crucial way on two natural parameters of the problem: the number of workers and the width of the dag . The latter parameter denotes the maximum cardinality of an antichain (all technical terms are defined in Section 2).

(ii) We develop a polynomial time algorithm that finds an optimal regimen for a restricted version of the problem. Specifically, we assume that the width of the dag is at most a constant, and also that the number of workers is at most a constant. Note that our algorithm allows the dag to have complex structure and dag size to grow, however the dag must be “narrow”. The algorithm uses a dynamic programming approach. Given dag , we construct a directed graph called admissible evolution of execution that contains all possible sets of tasks that a regimen can have executed at a point of time, and how one set could result form another within one unit of time. This graph turns out to be a dag. We then formulate a recurrence to define a regimen for a node of based on definitions for the children of the node.
We sharply contrast our algorithmic results with complexity lower bounds, by demonstrating that our restrictions are fundamentally necessary. Specifically, we show that the optimization problem is NP-hard when dag width is constant while the number of workers can grow, and is also NP-hard when the number of workers is constant while dag width can grow. Moreover, if both dag width and the number of workers are unconstrained, then we show that the problem cannot be approximated with a factor less than $5/4$, unless $P=NP$.

Our reductions demonstrate that the complexity of the problem is quite steep. First, the problem is trivial when the dag has just one task. Simply then an optimal regimen assigns all workers to that task. However, the problem becomes NP-hard when dag has just two independent tasks! (Arguably this is the “second simplest” dag.) Second, the problem is easy when there is just one worker. Simply then an optimal regimen follows any topological sort and assigns the (only) worker to the task at hand. However, the problem becomes NP-hard when there are just two workers!

### 1.2 Related work

There are several studies that deal with scheduling under constrained resources \[30, 28\]. Many such problems are NP-hard \[5\], and the studies typically focus on heuristic approaches. Our work differs from the studies because we also consider resource failures. More closely related are results \[27\] on scheduling where each task can be executed in one of a few ways. Each way has certain resource requirements, duration, and failure probability. After execution failure, the task is reexecuted fault-free. The goal is to minimize the expected completion time. Several heuristics are proposed. A similar model is studied \[37\] with the goal of maximizing the probability of successful completion using heuristics.

In project scheduling under uncertainty \[15, 8, 9\] each task has a duration that is a random variable and a requirement on the amount of different resources needed to perform the task. There is a fixed amount of each resource available. The goal is to determine how to assign resources to tasks over the execution of the project to minimize the expected completion time. Our problem is similar because workers can be treated as a resource. However, our problem is different because we allow multiple workers to be assigned to a task, which modifies the distribution of task duration. A model is studied \[38\] where the execution time of each task in a dag is a random variable with a certain distribution that depends on the amount of resources assigned to the task. Execution may fail if it takes longer than a threshold. The goal is to maximize the probability of completion. Several heuristics have been proposed.

One of the goals of stochastic scheduling is to minimize the expected completion time when task durations are random variables and tasks need to be scheduled on parallel machines. Such problem was studied for independent tasks \[19, 13\] and for dependent tasks \[35\], however, under a different setting, because we assume that two or more machines may be assigned to the same task, which may modify the probability distribution of task duration. Other related problems are the Network Reliability Problem and the Network Survivability Problem \[12\] that model failures probabilistically, but have different optimization goals.

Computing tasks over the Internet poses distinct challenges. It is known \[11\] which dependent tasks should be executed by a reliable server and which by workers whose unreliability is modeled probabilistically, so as to maximize the expected number of correct tasks, under a constraint on completion time. A similar probabilistic model is studied \[34\] for independent tasks. Also related
is the problem of scheduling tasks so as to render tasks eligible for allocation to workers (hence for execution) at the maximum possible rate \[32, 33, 25, 24, 23, 3\].

There are several systems in grid computing and project management fields that are related to our work. Condor [36], for example, executes computations with complex task dependencies. The clients to which tasks are sent are commonly unreliable. Condor assigns any task to one computer at a time (no redundancy), and uses a “FIFO” topological sort to sequence task submissions. This may sometimes lead to an ineffective use of computing resources. One example of a project management system is the Microsoft Project 2003 [26]. The system can estimate project duration using a PERT algorithm based on a probabilistic model of duration of a task, but does not take into account resource constraints.

In a companion to this paper the author explores [22] implementation details and experiments with the algorithm. Specifically, a range of theoretical and practical approaches are proposed to craft an efficient implementation of the algorithm. The benefits of the approaches and scalability of the implementation are evaluated experimentally.

Approximation algorithms for several cases of the problem studied in this paper were recently given [20, 21]. The case of independent tasks was shown to admit an \(O(\log t)\)-approximation. Richer dependency structures also admit approximations. For disjoint chains, an \(O(\log n \log t \log(n + t)/\log \log(n + t))\)-approximation was given. A collection of directed out- or in-trees admits an \(O(\log n \log^2 t)\)-approximation, while a directed forest admits an \(O(\log n \log^2 t \log(n + t)/\log \log(n + t))\)-approximation.

1.3 Paper organization

The remainder of paper is structured as follows. In Section 2 we define a model of executing dags where workers can fail with certain probabilities, and formulate an optimization problem of minimizing the expected completion time. In Section 3 we give a polynomial time algorithm for the problem with dags that have constant width and where the number of workers is constant. Finally, in Section 4 we explain why restricting dag width and the number of workers is fundamentally required.

2 Model and background

A directed graph \(\mathcal{G}\) is given by a set of nodes \(N_\mathcal{G}\) and a set of arcs (or, directed edges) \(A_\mathcal{G}\), each having the form \((u \rightarrow v)\), where \(u, v \in N_\mathcal{G}\). A path in \(\mathcal{G}\) is a sequence of arcs that share adjacent endpoints, as in the following path from node \(u_1\) to node \(u_k\): \((u_1 \rightarrow u_2), (u_2 \rightarrow u_3), \ldots, (u_{k-2} \rightarrow u_{k-1}), (u_{k-1} \rightarrow u_k)\). A dag (directed acyclic graph) \(\mathcal{G}\) is a directed graph that has no cycles; i.e., in a dag, no path of the preceding form has \(u_1 = u_k\). Given an arc \((u \rightarrow v) \in A_\mathcal{G}\), \(u\) is a parent of \(v\), and \(v\) is a child of \(u\) in \(\mathcal{G}\). Each parentless node of \(\mathcal{G}\) is a source (node), and each childless node is a sink (node); all other nodes are internal.

Given a dag, an antichain is a set of its nodes such that no two are “comparable” i.e., for any two distinct nodes \(u\) and \(v\) from the set, there is no path from \(u\) to \(v\) nor from \(v\) to \(u\). The largest cardinality of an antichain is called width of the dag. A chain is a path. A set of chains is said to cover the dag if every node of the dag is a node in at least one of the chains (chains may “overlap”). A Dilworth’s Theorem [7] states that dag width is equal to the minimum number of chains that cover the dag.
A computation is modeled by a dag. Then the nodes are called tasks and for concreteness we assume that \( N_G = \{1, \ldots, t\} \). We denote a set \( \{1, \ldots, t\} \) by \([t]\). Arcs specify dependencies among tasks: given an arc \((u \rightarrow v)\), \(v\) cannot be executed until \(u\) has been. A set of tasks satisfies precedence constraints if, for every task in the set, every parent of the task is also in the set. Given such set \(X\), we denote by \(E(X)\) the set of tasks not in \(X\) every of whose parent is in \(X\); tasks in this set are called eligible when tasks \(X\) have been executed. (So any source not in \(X\) is eligible.)

The execution of tasks is modeled by the following game. There are \(n\) workers identified with elements of \([n]\). Let \(X\) be a set of tasks that satisfies precedence constraints. The game starts with \(Y = X\), and proceeds in rounds. During a round, workers are assigned to tasks in \(E(Y)\) according to a regimen \(\Sigma\). The regimen specifies an assignment \(\Sigma(Y)\) that maps each worker to an element of the set \(E(Y) \cup \{\perp\}\) i.e., either to a task that is eligible in this round, or to a distinguished element \(\perp\). Note that the assignment is determined by the set of tasks \(Y\). The assignment enables directing multiple workers to the same task, or to different tasks; workers can also idle. Then each worker that was assigned to a task attempts to execute the task. The attempt of worker \(i\) assigned to task \(j\) succeeds with probability \(0 \leq p_{i,j} \leq 1\) independently of any other attempts. We assume that there is at least one worker that has non-zero probability of success, for any given task. A task is executed in this round if, and only if, at least one worker assigned to the task has succeeded. Every executed task is added to \(Y\), and the game proceeds to the next round. It could be the case that every attempt has failed; then the set \(Y\) remains unchanged, and in the next round worker assignment remains unchanged, too. Formally, a regimen \(\Sigma\) is a function \(\Sigma : 2^{N_G} \rightarrow ([n] \rightarrow (N_G \cup \{\perp\}))\), such that for any subset \(Z\) of tasks that satisfies precedence constraints, the value \(\Sigma(Z)\) is a function from \([n]\) to the set \(E(Z) \cup \{\perp\}\). The game proceeds until a round when every sink of \(G\) is in \(Y\). We say that the game ends in such round.

The quality of the game is determined by how quickly the game ends. Specifically, the number of rounds of the game, beyond the first round, until the round when the game ends is called time to completion of regimen \(\Sigma\) starting with tasks \(X\) already executed. This time is a random variable. When \(X\) is empty, we call the time simply completion time. Our goal is to find a regimen \(\Sigma_{OPT}\) that minimizes the expected completion time. We call this goal the Recomputation and Overbooking allowed Probabilistic dAg Scheduling Problem (ROPAS).

**ROPAS**

**Instance:** A dag \(G\) describing dependencies among \(t\) tasks, \(n\) workers such that worker \(i\) succeeds in executing task \(j\) with probability \(0 \leq p_{i,j} \leq 1\), and that for any task \(j\) there is worker \(i\) with \(p_{i,j} > 0\).

**Objective:** Find a regimen \(\Sigma_{OPT}\) that minimizes the expected completion time.

We observe that the optimization problem yields a finite expectation. Let \(X\) be a subset of tasks that satisfies precedence constraints. Denote by \(B_X\) the minimum expected time to completion across regimens that start with tasks \(X\) already executed. The subsequent lemma states that \(B_X\) is finite. The proof, while simple, is presented here in detail to help the reader gain familiarity with our notation.

**Lemma 2.1.** For any set \(X\) of tasks that satisfies precedence constraints, \(B_X\) is finite.

**Proof.** We define a regimen that has finite expectation; thus minimum expectation must be finite, too. We take a topological sort \(t_1, \ldots, t_m\) of the subdag of \(G\) induced by tasks not in \(X\). Note that it is possible to execute tasks in the order of the sort because once tasks in \(X\) and \(t_1, \ldots, t_{j-1}\) have been executed, task \(t_j\) is eligible. Our regimen will follow this order, each time assigning every worker
to the task at hand. The probability that every worker fails to execute task $t_j$ is $\prod_{i=1}^n (1 - p_{i,t_j})$, which by assumption is strictly smaller than 1. Thus the expected time to execute the task is $1/(1 - \prod_{i=1}^n (1 - p_{i,t_j}))$, because execution time follows geometric distribution. We can use linearity of expectation to conclude that the expected time to completion for the regimen is just the sum of expectations for each individual task in the sort. Thus $B_X$ is at most $\sum_{j=1}^t 1/(1 - \prod_{i=1}^n (1 - p_{i,j}))$, as desired.

### 3 Scheduling algorithm for a restricted problem

This section presents a polynomial time algorithm that finds an optimal regimen for a restricted version of the ROPAS Problem. Specifically, we make two assumptions: (1) the dag $G$ has at most a constant width, and (2) the number of workers is at most a constant. These two restrictions may appear to be quite severe, and perhaps too restrictive! However, they are fundamentally necessary, as we shall see in Section 4.

We construct a directed graph that models how computation can evolve. The graph $A = (N_A, A_A)$ called admissible evolution of execution for $G$ (an example is in Figure 2) is constructed inductively. Each node of $A$ will be a subset of nodes of $G$. We begin with a set $N_A = \{\emptyset\}$. For any node $X \in N_A$ that does not contain every sink of $G$, we calculate the set of eligible tasks $E(X)$ in $G$. We then take the non-empty subsets $D \subseteq E(X)$, add to $N_A$ a node $X \cup D$, if it is not there already, and add to $A_A$ an arc $(X, X \cup D)$, if it is not there already. Since $G$ is finite, the inductive process clearly defines a unique directed graph $A$. The structure of this graph is explained in the subsequent lemma.

**Lemma 3.1.** Let $A$ be the admissible evolution of execution for a dag $G$. Then $A$ is dag. Its nodes are exactly the sets of tasks that satisfy precedence constraints. It has a single source $\emptyset$ and a single sink $N_G$.

**Proof.** We verify the assertions in turn. The graph $A$ cannot have any cycle, because any arc points from a node $X$ to a node $X \cup D$, but $X \cup D$ has larger cardinality than the set $X$.

Nodes of $A$ are exactly the sets of tasks that satisfy precedence constraints. Indeed, if $X$ satisfies precedence constraints, then clearly so does its union with any subset of $E(X)$. Thus any node of $A$ satisfies precedence constraints. Now pick any subset $Y$ of tasks that satisfies precedence constraints. Let $t_1, \ldots, t_{|Y|}$ be a topological sort of the subdag of $G$ induced by $Y$. Clearly, $t_j$ belongs to the set of tasks eligible when tasks $t_1, \ldots, t_{j-1}$ have been executed, for any $j$. So if $\{t_1, \ldots, t_{j-1}\}$ is a node of $A$, so is $\{t_1, \ldots, t_j\}$. Since $\emptyset$ is a node of $A$, $Y$ must also be. A corollary to this is that $N_G$ is a node of $A$.

We add a node $Y$ to $N_A$ only if there is an arc leading to $Y$ from some other node. So $Y$ cannot be a source. However, $\emptyset$ is a source because no arc leads to a set with the same number or fewer elements.

Pick any node $X$ of $A$ and suppose that it does not contain the sinks of $G$. By looking at a topological sort of $G$ we notice that there is a task of $G$ not in $X$ such that every parent in $G$ of the task is in $X$. Thus $E(X)$ is not empty, and so $X$ has a child in $A$. Pick any node $X$ of $A$ that contains every sink of $G$. Since $X$ satisfies precedence constraints, it contains every task, and so $X = N_G$. But $E(N_G)$ is empty, so $X$ has no children in $A$. 

Given a regimen $\Sigma$, we can convert $A$ into a Markov chain that models how execution can evolve for this particular regimen. Specifically, for any node $X$ of $A$, $\Sigma$ defines the assignment of workers
Let $X$ be a topological sort of $\mathcal{A}$ such that $X_i$ is the expected time to completion for regimen $\Sigma$ of tasks executed by workers in the assignment $\Sigma_{OPT}(X_i)$ be the assigned tasks. Then

$$T_{X_0} = \frac{1}{a_1 + \ldots + a_k} \left( 1 + \sum_{i=1}^{k} a_i \cdot T_{X_i} \right),$$

where $T_{X_i}$ is the expected time to completion for regimen $\Sigma$ starting with tasks $X_i$ already executed, and by convention $1/0 = \infty$ and $0 \cdot \infty = 0$.

Our goal, however, is not to compute the expected completion time for a given regimen, but rather find a regimen that minimizes the expectation. For this purpose, we give a dynamic programming algorithm called OPT that defines a regimen called $\Sigma_{OPT}$. Since $\mathcal{A}$ has no cycles, we can hope to apply the recurrence of Theorem 3.2 starting from the sink and backtracking towards the source. Specifically, we initialize the regimen arbitrarily. Then we take a topological sort $Y_1, \ldots, Y_m$ of $\mathcal{A}$ and process it in the reverse order of the sort. See Figure 1 for a pseudocode. When we process a node $X$ of $\mathcal{A}$, we define two values: a number $S_X$ and an assignment $\Sigma_{OPT}(X)$. We begin by setting $S_{Y_m}$ to 0 and $\Sigma_{OPT}(Y_m)$ so that each worker is assigned to $\perp$. Now let $1 \leq h < m$, and let us discuss how $S_{Y_h}$ and $\Sigma_{OPT}(Y_h)$ are defined. Let $D_0, \ldots, D_k$ be the distinct subsets of $E(Y_h)$, such that $D_0 = \emptyset$. We consider the distinct $|E(Y_h)|^n$ assignments of the $n$ workers to the tasks of $E(Y_h)$, but not to $\perp$. For any assignment, we calculate the probability $a_i$ that $D_i$ is exactly the set of tasks executed by workers in the assignment. If $a_1 + \ldots + a_k > 0$, then we compute the weighted sum $1/(a_1 + \ldots + a_k) \cdot (1 + \sum_{i=1}^{k} a_i S_{Y_h\cup D_i})$. We pick an assignment that minimizes the sum. We set $S_{Y_h}$ to the minimum, and $\Sigma_{OPT}(Y_h)$ to the assignment that achieves the minimum. Then we move back in the topological sort to process another node, by decreasing $h$. After the entire sort has been processed, the regimen $\Sigma_{OPT}$ has been determined.

Our goal is twofold. First, we show that the regimen $\Sigma_{OPT}$ indeed minimizes the expected completion time. We do this by considering an extended algorithm OPT* where workers can idle.
This freedom matches that defined in ROPAS and we can use the recurrence to demonstrate that OPT* solves ROPAS. We then argue that idling is not needed, because an optimum can be found among regimens where every worker always gets assigned to a task. We, hence, prove that OPT solves ROPSA, too. We conclude by showing that OPT runs in polynomial time when dag width and the number of workers are at most a constant.

It is convenient to consider an extended version OPT* of the algorithm where workers are allowed to idle. Specifically, when processing $Y_h$, we consider the $|E(Y_h)| + 1)^n$ distinct assignments of the $n$ workers to the tasks of $E(Y_h)$ or to $\bot$. The regimen resulting from processing the topological sort is denoted by $\Sigma_{OPT^*}$. The following proposition explains why the extended algorithm finds best regimen. (Note a “strong” sequence of quantifiers—a single regimen that is optimal for all $X$ of $\mathcal{A}$ when competing with any regimen that start with $X$.)

**Proposition 3.3.** When the algorithm OPT* halts, thus computing a regimen $\Sigma_{OPT^*}$, for any node $X$ of $\mathcal{A}$, the expected time to completion for the regimen starting with tasks $X$ already executed is equal to the minimum expected time to completion $B_X$ that any regimen can achieve starting with tasks $X$ already executed.

**Proof.** Let $Y_1, \ldots, Y_m$ be the topological sort of $\mathcal{A}$ which was used by the algorithm OPT*. We argue that the following invariant parameterized by $h$ holds during the reverse processing of the sort:

For all $i$ such that $h \leq i \leq m$, $S_{Y_i} = B_{Y_i}$ and $S_{Y_i}$ is equal to the expected time to completion of $\Sigma_{OPT^*}$ that starts with tasks $Y_i$ already executed.

This invariant is clearly true for $h = m$. Indeed, by Lemma 3.1, $\mathcal{A}$ has just one sink which is equal to $N_G$, and so $Y_m = N_G$. If every task is already executed, the minimum expected time to completion is zero. The algorithm, thus, correctly assigns zero to $S_{Y_m}$. The expectation of any regimen that starts with every task already executed is zero, and so the value of $\Sigma_{OPT^*}(Y_m)$ is satisfactory.

Now pick any $2 \leq h \leq m$. We shall see that after OPT* has processed $Y_{h-1}$, the invariant is true for $h$ decreased by one. Let $X = Y_{h-1}$. Since $X$ is not a sink of $\mathcal{A}$, it has at least one child. Let $X_1, \ldots, X_k$ be the children.

We first argue that the value assigned to $S_X$ is at most the minimum expected time to completion $B_X$. Pick a regimen $\Sigma$ that achieves the minimum when starting with tasks $X$ already executed. Some workers may get assigned to $\bot$ in $\Sigma(X)$. Lemma 2.4 ensures that $B_X$ is finite. We can use Theorem 3.2 to express the expectation of the regimen as a weighted sum

$$B_X = 1/(a_1 + \ldots + a_k) \cdot (1 + \sum_{i=1}^{k} a_i T_{X_i})$$

of expected times $T_{X_i}$ to completion for $\Sigma$ starting with tasks from the sets $X_i$ already executed. By the invariant $S_{X_i} = B_{X_i}$, and so the expectation $T_{X_i}$ is at least $S_{X_i}$. The dynamic program considers every assignment as a candidate for $\Sigma_{OPT^*}(X)$, and the assignment $\Sigma(X)$ in particular. For this assignment the algorithm calculates the weighted sum $1/(a_1 + \ldots + a_k) \cdot (1 + \sum_{i=1}^{k} a_i S_{X_i})$. Hence the weighted sum is at most $B_X$. The algorithm selects an assignment that minimizes the weighted sum, so the value of $S_X$ is at most $B_X$, as desired.

After values of $S_X$ and $\Sigma_{OPT^*}(X)$ have been selected, the value of $S_X$ is clearly equal to the expectation of $\Sigma_{OPT^*}$ starting with $X$ already executed. Indeed, for any assignment considered by
the algorithm, the weighted sum is, by Theorem 3.2, equal to the expected time to completion of 
$\Sigma_{OPT^*}$ that starts with tasks $X$ already executed and uses this assignment in place of $\Sigma_{OPT^*}(X)$. 

Since $\Sigma_{OPT^*}$ has expectation $S_X$ and $S_X$ is at most $B_X$, the expected time to completion for $\Sigma_{OPT^*}$ starting with $X$ already executed is actually equal to $B_X$. Hence the invariant holds when $h$ gets decreased by one. This completes the proof.

We will use this result to show that it is not necessary for workers to idle. We begin by observing that, roughly speaking, “more done, less time remaining to complete”, as detailed in the subsequent lemma.

**Lemma 3.4.** For any subsets $X \subseteq Y$ of tasks that satisfy precedence constraints, $B_X \geq B_Y$.

**Proof.** It is sufficient to prove the lemma with an additional restriction on $Y$. Specifically, the proof considers any $X$ as stated, any $u \in E(X)$, and $Y = X \cup \{u\}$. The proof is by the reverse induction on the cardinality of $X$. The theorem is obvious when $|X| = t - 1$, because then $B_Y = 0$.

For the inductive step, take $X$ of cardinality at most $t - 2$. Consider the regimen $\Sigma_{OPT^*}$. By Proposition 3.3 the expected time to completion when $\Sigma_{OPT^*}$ starts with tasks $Z$ already executed is $B_Z$, for any $Z$ that satisfies precedence constraints. Let $t_1, \ldots, t_k$ be the tasks to which $\Sigma_{OPT^*}(X)$ assigns workers. Since $B_X$ is finite, $k \geq 1$. Let $p_i$ be the probability that at least one worker assigned to task $t_i$ succeeds. Let $p_C = \prod_{i \in C} p_i \prod_{i \in [k] \setminus C} (1 - p_i)$ be the probability that exactly the set of tasks indexed by $C$ gets executed in $\Sigma_{OPT^*}(X)$ among tasks indexed by $[k]$. We can use Theorem 3.2 to express the expected time to completion for $\Sigma_{OPT^*}$ as

$$B_X = \frac{1 + \sum_{\emptyset \subseteq C \subseteq [k]} p_C \cdot B_{X \cup C}}{1 - p_0}$$

with $p_0 < 1$.

We begin with a few special cases that are simpler to analyze and that simplify the follow-up analysis. Suppose first that $\Sigma_{OPT^*}(X)$ does not assign any worker to $u$. Then each assigned task is still eligible when tasks $X \cup \{u\}$ have been executed. Hence we can define a new regimen $\Sigma'$ equal to $\Sigma_{OPT^*}$ except that $\Sigma'(X \cup \{u\}) = \Sigma_{OPT^*}(X)$. But then the expected time to completion for $\Sigma'$ starting with tasks $X \cup \{u\}$ already executed is

$$T'_{X \cup \{u\}} = \frac{1 + \sum_{\emptyset \subseteq C \subseteq [k]} p_C \cdot B_{X \cup C \cup \{u\}}}{1 - p_0}.$$ 

Clearly, by the inductive assumption, $B_{X \cup C} \geq B_{X \cup C \cup \{u\}}$. So $B_X \geq T'_{X \cup \{u\}}$. Since $T'_{X \cup \{u\}} \geq B_{X \cup \{u\}}$, we have that $B_X \geq B_{X \cup \{u\}}$, as desired. Hence in the remainder we can assume that $t_k = u$. Suppose now that $k = 1$. Then $B_X$ is equal to $1/(1 - p_0) + B_{X \cup \{t_k\}}$ (because $p_0$ cannot be 1), and so clearly the latter is less than $B_X$. A similar conclusion follows when $k \geq 2$ but $0 = p_1 = \ldots = p_{k-1}$.

After having tackled the simpler special cases, we proceed to the final, most interesting, case when $k \geq 2$, $t_k = u$, and at least one of $p_1, \ldots, p_{k-1}$ is strictly greater than 0. The Equation (11) for $B_X$ contains probabilities of events, and it is convenient to partition the events into three groups:

(i) only task $t_k$ gets executed, which occurs with probability $p_k \cdot (1 - p_1) \cdot \ldots \cdot (1 - p_{k-1})$,

(ii) precisely the tasks indexed by a non-empty set $C \subseteq [k - 1]$ get executed and also task $t_k$, which occurs with probability $p_k \cdot \prod_{i \in C} p_i \prod_{i \in [k-1] \setminus C} (1 - p_i)$,
We define $\Sigma$ to be equal to $\Sigma^{\text{OPT}}_U$ to task 
 of group (ii) with the corresponding summand of group (iii), and obtain
and the resulting expression will not increase. But then we can pairwise combine each summand
of group (ii) with the corresponding summand of group (iii), and obtain

$$B_X \geq \frac{1 + p_k q_0 \cdot B_{X \cup \{t_k\}} + \sum_{\emptyset \subset C \subseteq [k-1]} q_C \cdot B_{X \cup C \cup \{t_k\}}}{1 - (1 - p_k) q_0},$$

where $q_C = \prod_{i \in C} p_i \prod_{i \in [k-1] \setminus C} (1 - p_i)$ is the probability that exactly the set of tasks indexed by $C$ gets executed among tasks from $[k - 1]$. Let us denote the big summation by $S$. It is now enough to show that

$$\frac{1 + p_k q_0 \cdot B_{X \cup \{t_k\}} + S}{1 - (1 - p_k) q_0} \geq B_{X \cup \{t_k\}}.$$

In order to verify this inequality, we transform it to an equivalent form by multiplying both sides 
by the denominator and grouping for $B_{X \cup \{t_k\}}$

$$\frac{1 + S}{1 - q_0} \geq B_{X \cup \{t_k\}},$$

because $q_0 < 1$. We notice that the left hand side is the expected time to completion of a regimen
$\Sigma''$ that starts with tasks $X \cup \{t_k\}$ already executed. This regimen $\Sigma''$ is equal to $\Sigma^{\text{OPT}}$, except that 
when tasks $X \cup \{t_k\}$ need to be executed, the regimen assigns workers to tasks the way $\Sigma^{\text{OPT}}(X)$ 
does, omitting these workers assigned to $t_k$ (such workers do not get assigned then at all). Since 
$B_{X \cup \{t_k\}}$ is the minimum expectation, it must indeed be smaller or equal to the expectation of that 
regimen $\Sigma''$.

This completes the inductive step and the proof.

We use the lemma to argue that when processing $Y_h$, it is sufficient to consider assignments $S$ 
that map every worker to an eligible task, i.e., we do not need to assign any worker to $\perp$.

**Proposition 3.5.** For any set $X$ of tasks that satisfies precedence constraints and does not contain 
every sink of $G$, there is a regimen $\Sigma$ such that $\Sigma(X)$ assigns every worker to a task of $E(X)$ and 
the expected time to completion of $\Sigma$ that starts with tasks $X$ already executed is $B_X$ (i.e., minimum).

**Proof.** We shall construct the regimen $\Sigma$ using the regimen $\Sigma^{\text{OPT}}$, of Proposition 3.3. We will show that the expectation of $\Sigma$ is at most that of $\Sigma^{\text{OPT}}$, and since the latter is minimum, the expectations will have to be equal.

Let us first make a few observations about $\Sigma^{\text{OPT}}$. Let $t_1, \ldots, t_k$ be the tasks to which $\Sigma^{\text{OPT}}(X)$ 
assigns workers. We know that $k \geq 1$. If every worker has been assigned by $\Sigma^{\text{OPT}}(X)$ to a task, 
then the claim follows. We, therefore, assume that there is at least one worker not assigned by $\Sigma^{\text{OPT}}(X)$ to any task.

We construct a regimen $\Sigma$. Let $U$ be the set of workers not assigned by $\Sigma^{\text{OPT}}(X)$ to any task. 
We define $\Sigma$ to be equal to $\Sigma^{\text{OPT}}$ except that the assignment $\Sigma(X)$ also assigns every worker from $U$ to task $t_1$. Let $q$ be the probability that at least one worker from $U$ succeeds when assigned to
Let us compare equations for $B_X$ and $T_X$. As earlier, let $p_C$ be the probability that exactly the tasks indexed by $C$ get executed. We can express $B_X$ using Theorem 3.2 as

$$B_X = \frac{1 + \sum_{\emptyset \subseteq C \subseteq [k]} p_C \cdot B_{X \cup C}}{1 - p_\emptyset},$$

and we know that $p_\emptyset < 1$. Similarly, we can express $T_X$. We notice that for any $C$, $\emptyset \subseteq C \subseteq [k]$, the event that exactly the tasks in $C$ are executed in $\Sigma_{OPT^*}(X)$ can be decomposed into two mutually exclusive “subevents”: at least one worker from $U$ succeeds, and every worker from $U$ fails. Hence

$$T_X = \frac{1 + q \cdot p_\emptyset \cdot B_{X \cup \{t_1\}} + \sum_{\emptyset \subseteq C \subseteq [k]} q \cdot p_C \cdot B_{X \cup C \cup \{t_1\}} + \sum_{\emptyset \subseteq C \subseteq [k]} (1 - q) \cdot p_C \cdot B_{X \cup C}}{1 - (1 - q) \cdot p_\emptyset},$$

even if $t_1 \in C$ for some $C$.

We make two simplifying assumptions. Notice that if $q = 0$, then workers of $U$ always fail, and so $T_X$ is then obviously equal to $B_X$. Notice also that if $p_\emptyset = 0$ then there must be a task that is certainly executed. We can then reassign workers of $U$ to that task instead of to $t_1$, and the resulting $T_X$ will also be equal to $B_X$. Therefore, we can assume without loss of generality that $q > 0$ and $p_\emptyset > 0$.

We will modify the expression for $T_X$ to make it look more like $B_X$. By Lemma 3.4 we know that $B_{X \cup C \cup \{t_1\}} \leq B_{X \cup C}$. Hence we can replace the former with the latter in the expression for $T_X$, pairwise combine terms, and obtain a bound

$$T_X \leq \frac{1 + q \cdot p_\emptyset \cdot B_{X \cup \{t_1\}} + \sum_{\emptyset \subseteq C \subseteq [k]} p_C \cdot B_{X \cup C}}{1 - (1 - q) \cdot p_\emptyset},$$

that begins to resemble the expression for $B_X$.

We shall conclude that the bound in Equation (2) is at most $B_X$, which will mean that $T_X \leq B_X$, as desired, because then of course $T_X = B_X$. Let $S$ denote the summation in the enumerator. We restate our goal as that of showing that

$$\frac{1 + q \cdot p_\emptyset \cdot B_{X \cup \{t_1\}} + S}{1 - (1 - q) \cdot p_\emptyset} \leq B_X = \frac{1 + S}{1 - p_\emptyset}.$$ 

But this inequality can be equivalently expressed as

$$B_{X \cup \{t_1\}} \leq \frac{B_X (1 - p_\emptyset + q \cdot p_\emptyset) - 1 - S}{q \cdot p_\emptyset} = \frac{1 + S}{1 - p_\emptyset} \left( \frac{1 - p_\emptyset}{q \cdot p_\emptyset} + 1 \right) - \frac{1 + S}{q \cdot p_\emptyset} = \frac{1 + S}{1 - p_\emptyset} = B_X,$$

because $q, p_\emptyset > 0$. However, by Lemma 3.4, $B_{X \cup \{t_1\}}$ is indeed at most $B_X$, and thus the proof is completed.

**Corollary 3.6.** The algorithm $OPT$ finds a regimen $\Sigma_{OPT}$ that minimizes the expected completion time.

We now tackle the second subgoal of showing that the algorithm runs in polynomial time for dags of at most a constant width and when the number of workers is at most a constant. Let $w$ be
Figure 2: Depicted is the dag $A$, called admissible evolution of execution, constructed for the given dag $G$. The dashed boxes and dashed arrows represent nodes and arcs of $A$. Depicted are also optimal regimens for the case of two workers, where each has the same probability $0 < p < 1$ of executing successfully any given task. Each box contains: a subset of executed tasks of $G$ (black), and tasks that are eligible then (green); the minimum expected time to completion (upper left hand corner); and how the two workers should be assigned to eligible tasks so as to yield the expectation (same-color “$\lor$” signs)—every assignment that yields this expectation is shown, except for symmetric assignments where workers could be swapped. Note that in the first round the two workers must be assigned to distinct tasks, if not then the expected completion time is not minimized.

the width of $G$. By Dilworth’s Theorem, the dag $G$ can be covered by $w$ chains. If any task of the chain is executed, so must be every preceding task in the chain. Each chain can have length at most $t$. Hence, there are at most $(t + 1)^w$ distinct subsets of tasks that satisfy precedence constraints.
When the algorithm processes a node of \( A \), it considers assignments of workers to eligible tasks. Any worker can, therefore, be assigned in at most \( t \) ways, and so there are at most \( t^n \) assignments. We notice that for a given assignment, we actually only need to evaluate probabilities \( a_i \) for at most \( 2^n \) of the sets \( D_i \), because the workers can get assigned to at most \( n \) different tasks, and so probabilities associated with sets that are not subsets of these tasks must be zero, and so can be omitted from the weighted sum. This yields the main result of this section.

**Theorem 3.7.** The algorithm \( OPT \) solves the ROPAS Problem in polynomial time when dag width is bounded by a constant and the number of workers is also bounded by a constant.

An example of an application of the algorithm to a dag is given in Figure 2.

4 Complexity of scheduling

This section shows fundamental reasons why the algorithm presented in the previous section is so restricted. Recall that the algorithm restricts both the number of workers and dag width to at most a constant. It turns out that ROPAS is NP-hard if the number of workers can grow while dag width is at most a constant, or if dag width can grow while the number of workers is at most a constant. Therefore, a polynomial time algorithm for the problem must restrict both dag width and the number of workers, unless \( P \neq NP \). We also show that if we allow the number of workers to grow and at the same time dag width to grow, then the problem is inapproximable within factor less than \( 5/4 \), unless \( P=NP \).

Our reductions demonstrate that the complexity of the ROPAS Problem is quite steep. First, the problem is trivial when the dag has just one task. Simply then an optimal regimen assigns every worker to that task. However, the problem becomes NP-hard when dag has just two independent tasks! (Arguably this is the “second simplest” dag.) Intuitively, the hardness results from the difficulty to perfectly load balance unreliable workers across eligible tasks. Second, the problem is easy when there is just one worker. Simply then an optimal regimen follows any topological sort and assigns the (only) worker to the task at hand. However, the problem becomes NP-hard when there are just two workers! Intuitively, here hardness is a consequence of possible complex task dependencies which make it difficult to determine how to pick tasks for execution so that sufficiently many tasks become eligible thus preventing workers from compulsory idling.

4.1 Scheduling narrow dags with many workers is NP-hard

In this section we show that ROPAS restricted to the dag with two independent tasks, but where the number of workers can grow, is NP-hard. For this purpose we introduce two auxiliary problems and determine their complexities. The first of them is the Multiplicative Partition Problem.

**Multiplicative Partition (MP)**

*Instance:* Number \( n \), integer sizes \( s_1, \ldots, s_n \geq 2 \).

*Question:* Is there a set \( P \) such that \( \prod_{i \in P} s_i = \prod_{i \notin P} s_i \) ?

It is tempting to try to show NP-completeness of MP by a reduction from the well-known Partition Problem. Here we could hope to use the fact that \( \sum_{i \in P} r_i = \sum_{i \notin P} r_i \) if, and only if, \( \prod_{i \in P} 2^{s_i} = \prod_{i \notin P} 2^{s_i} \). Unfortunately, an \( s_i = 2^{r_i} \) takes space exponential with respect to the space taken by \( r_i \). The issue of space explosion can be avoided by carrying out a more subtle reduction.
from the Exact Cover by 3-Sets (X3C) Problem that uses certain fundamental facts from Number Theory.

**Lemma 4.1.** The MP Problem is NP-complete.

*Proof.* We will show a reduction from the Exact Cover by 3-Sets Problem to the Multiplicative Partition Problem.

Pick any instance of X3C—a collection $C = \{S_1, \ldots, S_n\}$ of 3-element subsets of the set $U = \{1, \ldots, 3m\}$, $S_i = \{a_i, b_i, c_i\}$. Recall that the instance is positive if, and only if, there is a subcollection of $C$ such that every element of $U$ occurs in exactly one subset of the subcollection. We can assume that $m \leq n$ and that the union of every subset of $C$ is $U$, as otherwise the instance of X3C is definitely negative. We build an instance of MP as follows. We begin by enumerating the first consecutive primes $p_1, \ldots, p_{3m+1}$. By the Prime Number Theorem (see e.g., [17]), the prime $p_{3m+1}$ has value $O(m \ln m)$. Therefore, we can enumerate the primes in time $O(m^2 \log^2 m)$ using the Eratosthenes sieve; this time is polynomial with respect to the size of the instance of X3C. The instance of the MP Problem will have $n + 2$ sizes. The first $n$ are defined as products of primes indexed by the elements of subsets i.e., $s_i = p_{a_i} \cdot p_{b_i} \cdot p_{c_i} \geq 2$. Note that $s_i$’s are fairly small, so that each $s_i$ is $O(n^6)$. We add two “dummy” sizes. Let $p$ be the product of sizes $s_1$ through $s_n$, and $q$ be the product of the first $3m$ primes; each is $O(n^6)$, so sufficiently small. Note that $p/q$ is an integer, because $\bigcup_i S_i = U$. We define $s_{n+1} = p_{3m+1} \cdot p/q$ and $s_{n+2} = p_{3m+1} \cdot q$. Therefore, the instance of the MP Problem can be represented in time and space polynomial with respect to $n$, as desired.

Suppose that the instance of X3C is positive, and let $Q$ be the indices that define the subcollection. But then $\prod_{i \in Q} s_i$ is the product of all primes from $p_1$ to $p_{3m}$. So then $\prod_{i \notin Q} s_i$ is equal to the product of the $s_i$’s divided by the product of these primes. Therefore,

$$s_{n+1} \prod_{i \in Q} s_i = s_{n+2} \prod_{i \notin Q} s_i,$$

and so the instance of MP is positive.

Suppose that the instance of MP is positive, and let $P$ be a subset such that $\prod_{i \in P} s_i = \prod_{i \notin P} s_i$. Note that there are only two sizes, $s_{n+1}$ and $s_{n+2}$, that have the prime $p_{3m+1}$ as a factor. But two natural numbers are equal if, and only if, they have the same factorization. Thus in order for the products to be equal, either $n + 1$ or $n + 2$ is in $P$, but not both. We can assume, without loss of generality, that $n + 1$ is in $P$. Let $L = P \setminus \{n + 1\}$, and $R$ be the elements not in $P$ and other than $n + 2$. But then

$$s_{n+1} \prod_{i \in L} s_i = s_{n+2} \prod_{i \in R} s_i,$$

which can be equivalently expressed as

$$p/q \cdot \prod_{i \in L} s_i = q \cdot \prod_{i \in R} s_i.$$

Of course, $L$ and $R$ partition $\{1, \ldots, n\}$, so $p = \prod_{i \in L} s_i \cdot \prod_{i \in R} s_i$. Thus $q^2 = (\prod_{i \in L} s_i)^2$, and since the numbers are positive, $q = \prod_{i \in L} s_i$. Therefore, there is a selection of the $s_i$’s whose product contains each prime from $p_1$ to $p_{3m}$ exactly once. This set $L$ defines a subcollection of $C$ such that every element of $U$ is contained in exactly one subset of the subcollection. So the instance of X3C is positive. 

\[\square\]
The MP Problem is used to demonstrate NP-hardness of the second auxiliary problem called Partition with Minimum Sum of Products.

Partition with Minimum Sum of Products (PMSP)

**Instance:** Number \( n \), rational numbers \( 0 < r_1, \ldots, r_n < 1 \).

**Objective:** Find a set \( P \) that minimizes \( \prod_{i \in P} r_i + \prod_{i \not\in P} r_i \).

**Lemma 4.2.** The PMSP Problem is NP-hard.

**Proof.** We give a polynomial time Turing reduction of MP to PMSP. Specifically, we show how to find an answer to a given instance of MP by inspecting the minimum value for an appropriately constructed instance of PMSP. The proof is based on the fact that for any \( d > 0 \), the function \( f(x) = e^x + e^{d-x} \) defined for \( 0 \leq x \leq d \) is minimized exactly when \( x = d/2 \). The difficulty is to ensure that we compute using rational numbers only.

Pick any instance of the MP Problem and let \( s_1, \ldots, s_n \) be the positive integer sizes, each at least 2. We define an instance of the PMSP Problem by taking \( r_i = h^2_i \), where \( h_i = 1/s_i \). Then \( 0 < r_i < 1 \), as desired. We shall see that we can decide exactly when the instance of MP is positive, just by checking if the minimum for the instance of PMSP is equal to the value \( 2 \prod_{i=1}^n h_i \).

Let us see when the product \( \prod_{i \in P} r_i + \prod_{i \not\in P} r_i \) achieves that value. Let \( d = \sum_{i=1}^n \ln r_i \). Then
\[
\prod_{i \in P} r_i + \prod_{i \not\in P} r_i = e^{\sum_{i \in P} \ln r_i} + e^{d - \sum_{i \in P} \ln r_i} \geq 2e^{d/2}
\]
with equality only when \( d/2 = \sum_{i \in P} \ln r_i \). The crucial observation is that \( 2e^{d/2} \) is just \( 2 \prod_{i=1}^n h_i \), which is a rational number. Thus we can invoke an oracle that solves PMSP on the instance that we have constructed, then inspect the minimum returned by the oracle, and determine whether or not there exists \( P \) such that \( d/2 = \sum_{i \in P} 2 \ln h_i \). But the equation can be equivalently written as \( 1/2 \sum_{i=1}^n \ln h_i = \sum_{i \in P} \ln h_i \), which is equivalent to \( \sum_{i \in P} \ln h_i = \sum_{i \not\in P} \ln h_i \). This in turn is equivalent to \( \prod_{i \in P} h_i = \prod_{i \not\in P} h_i \). But this is equivalent to \( \prod_{i \in P} s_i = \prod_{i \not\in P} s_i \). So the instance of MP is positive if, and only if, the minimum for the instance of PMSP is equal to \( 2 \prod_{i=1}^n h_i \).

We have prepared a toolset, and now we are ready to show the main hardness result of this subsection.

**Theorem 4.3.** The ROPAS Problem restricted to the dag with two independent tasks (where the number of workers may grow) is NP-hard.

**Proof.** We give a polynomial time Turing reduction of PMSP to ROPAS. Pick any \( 0 < r_1, \ldots, r_n < 1 \), and let us consider a dag with just two isolated nodes, call them left task and right task. There are \( n \) workers. We define the probability of success of worker \( i \), \( p_{i, \text{left}} \) and \( p_{i, \text{right}} \), to be \( 1 - r_i \), and denote it simply by \( p_i \). Pick a regimen \( \Sigma \) that minimizes the expected completion time. Let \( L \) be the set of workers assigned by \( \Sigma(\emptyset) \) to the left task i.e., assigned when both tasks are eligible. We shall argue that the expected completion time is minimized if, and only if, \( \prod_{i \in L} r_i + \prod_{i \not\in L} r_i \) is minimized. Therefore, a best regimen can be transformed to a solution to the instance of PMSP.

In order to show the equivalence, we make a sequence of observations about the assignments that the regimen must make. Let \( R \) be the workers assigned by \( \Sigma(\emptyset) \) to the right task. Hence \( L \) and \( R \) are disjoint and included in \( \{1, \ldots, n\} \). Let \( \ell = 1 - \prod_{i \in L} (1 - p_i) \) be the probability that at
least one left worker succeeds, and \( r = 1 - \prod_{i \in R} (1 - p_i) \) that at least one right succeeds. We can use Theorem 3.2 to calculate the expected completion time \( T_0 \)

\[
T_0 = \frac{1 + \ell (1 - r) \cdot T_{\{\text{left}\}} + (1 - \ell) r \cdot T_{\{\text{right}\}}}{1 - (1 - \ell) (1 - r)},
\]

where \( T_{\{\text{left}\}} \) is the expected time to completion by the regimen starting with the left task already executed, and \( T_{\{\text{right}\}} \) starting with the right task already executed.

We can simplify this expression by observing that \( T_{\{\text{left}\}} \) and \( T_{\{\text{right}\}} \) are equal. Indeed, treating sets \( L \) and \( R \) as constants, we notice that \( T_0 \) is minimized when \( T_{\{\text{left}\}} \) and \( T_{\{\text{right}\}} \) are minimized.

Let \( K \subseteq \{1, \ldots, n\} \) be the set of workers assigned to the left task in \( \Sigma(\{\text{right}\}) \) i.e., when the right task has been executed and the left one is the only eligible task. Then the expected time to completion \( T_{\{\text{right}\}} \) is \( 1/\left(1 - \prod_{i \in K} (1 - p_i)\right) \). Since \( 0 < p_i < 1 \), this expectation is clearly minimized if, and only if, \( K = \{1, \ldots, n\} \). We can apply the same argument to conclude that the regimen assigns every worker to the right task when the task is the only one remaining to be executed. Therefore, \( T_{\{\text{left}\}} = T_{\{\text{right}\}} \). Let us denote this expectation by \( T \). Observe that \( T > 0 \). Hence

\[
T_0 = \frac{1 + T (\ell (1 - r) + (1 - \ell) r)}{1 - (1 - \ell) (1 - r)} = T + \frac{1 - T \ell r}{\ell + r - \ell r}.
\]

We notice that \( \Sigma(\emptyset) \) assigns every worker to a task. In order to show this, we demonstrate that the expectation decreases as \( \ell \) or \( r \) increase. We notice that the formula is symmetric with respect to \( \ell \) and \( r \), so we only calculate its partial derivative with respect to \( r \)

\[
\frac{\partial}{\partial r} = -\frac{1 + \ell - T \ell^2}{(\ell + r - \ell r)^2}.
\]

This derivative is strictly negative for any \( 0 < \ell < 1 \). So the expectation is minimized when \( r \) and \( \ell \) are maximized. This is equivalent to the condition that every worker must be assigned; in other words, the sets \( L \) and \( R \) partition the set \( \{1, \ldots, n\} \).

Let us summarize our observations so far: the regimen initially assigns every worker to a task, and when one task remains to be executed, the regimen assigns every worker to the task.

With these observations, we turn back to determining under what circumstances \( T_0 \) is minimized. Since \( L \) and \( R \) partition \( \{1, \ldots, n\} \), then \( \ell + r - \ell r \) is just the probability that at least one of the \( n \) workers succeeds, the value of which is independent of how workers are assigned to left and right task. Thus the expectation is minimized if, and only if, \( \ell r \) is maximized across partitions \( L \) and \( R \) of \( \{1, \ldots, n\} \), since \( T \) is now a constant. But

\[
\ell r = 1 - \left( \prod_{i \in L} (1 - p_i) \right) - \left( \prod_{i \in R} (1 - p_i) \right) + \left( \prod_{i \in L} (1 - p_i) \prod_{i \in R} (1 - p_i) \right),
\]

and since \( L \) and \( R \) is a partition, the expression in the third big parenthesis does not depend on the choice of \( L \) and \( R \). So the expression is maximized, for a partition \( L \) and \( R \) that minimizes \( \prod_{i \in L} r_i + \prod_{i \in R} r_i \).

We summarize the observations that we have made so far. A regimen minimizes expectation if, and only if, its set \( L \) minimizes \( \prod_{i \in L} r_i + \prod_{i \notin L} r_i \) and the set \( K = \{1, \ldots, n\} \). Thus the theorem is proven. \( \square \)
4.2 Scheduling wide dags with few workers is NP-hard

We show that the ROPAS Problem restricted to two workers, but where dag width can grow, is NP-hard. Toward this end, we first establish NP-completeness of an auxiliary problem. Using this problem, we establish NP-completeness of other problem. Finally, that other problem is used in a polynomial time Turing reduction that yields NP-hardness of the restricted ROPAS Problem.

We first demonstrate how to modify a reduction of Johnson [18] to fit our purpose. The reduction is from a variant of Clique to the Balanced Complete Bipartite Subgraph Problem (cf. [31]). The reduction takes a graph on $2g$ nodes and $e$ edges, and produces a bipartite graph with $(g^2 + g)$ nodes on the left and $e \leq (2g^2)$ nodes on the right. The question of finding a $g$-clique in the original graph is show to be equivalent to that of finding a balanced complete bipartite subgraph with $(g^2 + g)$ nodes on the left and the same number of nodes on the right. Since $(g^2 + g) \leq (2g^2)$, we can pad the bipartite graph with isolated nodes for uniformity, and then the resulting problem is NP-complete.

**Fixed Ratio Balanced Complete Bipartite Subgraph (FIRBCBS)**

*Instance:* Number $g$, bipartite graph $G$ with $(g^2)$ nodes on the left and $(g^2)$ on the right.

*Question:* Does $G$ contain a balanced complete bipartite subgraph with $(g^2)$ nodes on the left and $(g^2)$ on the right?

**Lemma 4.4 ([18]).** The FIRBCBS Problem is NP-complete.

The FIRBCBS Problem can be reduced to an auxiliary problem of finding many subsets whose union is small.

**Fixed Ratio Many Subsets with Small Union (FIRMSSU)**

*Instance:* Number $k$, nonempty subsets $S_1, \ldots, S_{3k}$ of the set $\{1, \ldots, 3k\}$ whose union is $\{1, \ldots, 3k\}$.

*Question:* Are there $2k$ of these subsets whose union has cardinality at most $2k$?

The following lemma generalizes an earlier result of [11]. We can prove the lemma using FIRBCBS and padding.

**Lemma 4.5.** The FIRMSSU Problem is NP-complete.

*Proof.* Take any instance of FIRBCBS. We consider the $(2g)$ by $(2g)$ matrix $A = (a_{i,j})$ that is the “complement” of the adjacency-matrix of $G$ i.e., $a_{i,j} = 0$ if left node $i$ is linked to right node $j$, and $1$ if they are not linked. Its rows and columns can be (independently) rearranged so that the $(g^2)$ by $(g^2)$ submatrix in the upper left hand corner contains only zeros if, and only if, the instance is positive. Now we will pad $A$ by adding “strips” of certain widths (a sketch is in Figure 3). Pick the smallest $k$ such that $(\frac{2g}{2}) < k$. Notice that for such $k$, $(\frac{g}{2}) < k$. Let $a = k - (\frac{g}{2})$ and $b = 3k - (\frac{2g}{2}) - a - k$. This implies that $b > (\frac{g}{2}) \geq 1$. We then add $a$ rows and $a + k$ columns, every filled with zeros, then $k + b$ rows and $b$ columns, every filled with ones. The resulting matrix $A'$ has the property that its rows and columns can be (independently) rearranged so that the $k$ by $2k$ rectangle in the upper left hand corner contains only zeros if, and only if, the instance is positive. The matrix also has at least one row and one column containing only ones, because $b \geq 1$.

We define an instance of FIRMSSU such that each column of $A'$ is a characteristic vector of a subset. This clearly yields a desired instance and completes the reduction. 

We are now ready to prove the main hardness result of the subsection.
Theorem 4.6. The ROPAS Problem restricted to two workers (where the dag width may grow) is NP-hard.

Proof. We give a polynomial time Turing reduction from FIRMSSU to ROPAS—by checking if the minimum expectation is small enough we can determine exactly when a given instance of FIRMSSU is positive.

Take any instance of FIRMSSU with \( n = 3k \) nonempty subsets \( S_1, \ldots, S_{3k} \) on \( \{1, \ldots, 3k\} \). We construct an instance of ROPAS with two workers. The dag will have three levels (an example is in Figure 4). We start with a bipartite dag with \( n^2 \) sources \( A_1 \) and \( n \) sinks \( B_2 \). Sources are partitioned into \( n \) groups of cardinality \( n \) each. There is an arc leading from a source of group \( i \) to a sink \( j \) exactly when \( i \) is in the set \( S_j \). There are \( 2/3n^2 \) extra sources \( A_2 \). Arcs lead from every extra source to every sink \( B_2 \). There are \( 2/3n \) internal tasks \( B_1 \). Arcs lead from every source of \( A_2 \) to every internal task of \( B_1 \). There are \( 1/3n^2 \) extra sinks \( C_2 \). Arcs lead from every internal task of \( B_1 \) to every extra sink of \( C_2 \). The first worker is certain to execute every task from \( A_1 \) and \( B_1 \), while the second worker fails on any. The second worker is certain to execute every task from \( A_2 \), \( B_2 \) and \( C_2 \), while the first worker fails on any.

Suppose that the instance of FIRMSSU is positive i.e., there are \( 2k \) subsets whose union is at most \( 2k \). Let \( U \) be the union of the \( 2k \) subsets—if \( U \) has cardinality less than \( 2k \), add to \( U \) arbitrary elements of \( \{1, \ldots, 3k\} \) so as to increase the cardinality of \( U \) to \( 2k \). We construct a regimen. The second worker executes sources \( A_2 \) during the first \( 2/3n^2 \) rounds. Hence the \( 2/3n \) internal tasks of \( B_1 \) become eligible. Meanwhile, the first worker executes the \( 2/3n^2 \) sources of \( A_1 \) from the groups corresponding to \( U \). As a result, at least \( 2/3n \) sinks of \( B_2 \) become eligible. Following round \( 2/3n^2 \), the second worker begins executing the eligible sinks of \( B_2 \), and computes \( 2/3n \) of them by the end of round \( 2/3n^2 + 2/3n \). Meanwhile, the first worker executes the \( 2/3n \) internal tasks \( B_1 \), which makes every of the \( 1/3n^2 \) sinks of \( C_2 \) eligible. Following round \( 2/3n^2 + 2/3n \), the second worker
executes $1/3n^2$ sinks of $C_2$, while the first worker resumes executing the remaining $1/3n^2$ sources of $A_1$. Thus by the end of round $n^2 + 2/3n$, the remaining $1/3n$ unexecuted sinks of $B_2$ are eligible. Finally, the second worker executes these sinks. Hence the computation is completed at the end of round $n^2 + n$.

Suppose now that the instance of FIRMSSU is negative. Pick any regimen. For the reason that the instance is negative, the union of any $2k$ subsets has cardinality at least $2k + 1$. But the cardinality of any group is $n$, so at least $2/3n^2 + n$ sources of $A_1$ must be executed in order to have $2/3n$ sinks of $B_2$ eligible. Therefore, at most $2/3n - 1$ sinks of $B_2$ can get executed by the end of round $2/3n^2 + 2/3n$. By the end of this round, no sinks of $C_2$ can be executed, because each depends on the execution of $2/3n^2 + 2/3n$ other tasks. Hence, by the end of that round, there are at least $1/3n + 1 + 1/3n^2$ unexecuted tasks that can only be executed by the second worker. As a result the regimen takes at least $n^2 + n + 1$ rounds to complete computation.

4.3 Scheduling wide dags with many workers is inapproximable

We have seen that ROPAS is NP-hard either when dag can have large width or when the number of workers can be large. It is, therefore, interesting to see how hard the problem is when not only dag width but also the number of workers can be large. Here we show that then ROPAS is inapproximable with a factor less than $5/4$, unless P=NP.

**Theorem 4.7.** The ROPAS Problem cannot be approximated with factor less than $5/4$, unless $P=NP$.

**Proof.** The proof constructs a dag similar to that in the proof of Theorem 4.6. Since we can afford
many workers, the execution of the dag can be completed in as few as four rounds under favorable circumstances. When circumstances are not favorable, any regimen needs five rounds or more because some workers will necessarily idle. This yields a desired gap.

Specifically, we show a gap creating reduction from FIRMSSU to ROPAS. Take any instance of FIRMSSU with \( n = 3k \) nonempty subsets \( S_1, \ldots, S_{3k} \) on \( \{1, \ldots, 3k\} \). We construct an instance of the ROPAS Problem as follows. There are \( 4k \) workers. For any task and any worker, the probability that the worker succeeds in executing the task is one. The dag has three levels (an example is in Figure 5). There are \( 6k \) sources, denoted by \( A \), partitioned into \( 3k \) groups, associated with numbers 1 through 3\( k \), of two tasks each. There are \( 3k \) internal tasks, denoted by \( B_a \), associated with numbers 1 through 3\( k \). Any source from \( A \) from group \( i \) is linked to the internal task \( i \) from \( B_a \). In addition, there are \( k \) extra internal tasks, denoted by \( B_b \). Each source from \( A \) is linked to every internal task from \( B_b \). There are \( 3k \) sinks, denoted by \( C_a \), associated with subsets \( S_1 \) through \( S_{3k} \). Any internal task \( i \) from \( B_a \) is linked to sink \( S_j \) if, and only if, \( i \) is in the subset \( S_j \). In addition, there are \( 3k \) extra sinks, denoted by \( C_b \). Each internal task from \( B_b \) is linked to every sink from \( C_b \). We shall see that there is a gap in the minimum expected time to completion for the instance of ROPAS, that differentiates the case when the instance of FIRMSSU is positive from the case when it is negative.

Suppose that the instance of FIRMSSU is positive i.e., there are \( 2k \) of the subsets whose union has cardinality at most \( 2k \). We will argue that the minimum expected time to completion is at most 4. Let \( U \) be the union of the \( 2k \) subsets—if \( U \) has cardinality less than \( 2k \), add to \( U \) arbitrary elements of \( \{1, \ldots, 3k\} \) so as to increase the cardinality of \( U \) to \( 2k \). The set \( U \) represents \( 2k \) internal tasks from \( B_a \). We describe a regimen. In the first round, we execute the \( 4k \) sources from \( A \) the children of which are the \( 2k \) internal tasks from \( B_a \) associated with tasks of \( U \). This makes the tasks associated with \( U \) eligible. In the second round, we execute the \( 2k \) eligible tasks from \( B_a \). Because the instance is positive, \( 2k \) sinks from \( C_a \) become eligible. In addition, we execute the remaining \( 2k \) sources from \( A \). Because now every source is executed, the remaining \( 2k \) internal tasks are eligible at the end of the second round. In the third round, we execute the \( 2k \) eligible sinks from \( C_a \) and the \( 2k \) eligible internal tasks from \( B_a \) and \( B_b \). Because now every internal task is executed, any non-executed sink is eligible. In the fourth round, we executed the remaining \( 4k \) sinks.

Suppose that the instance of FIRMSSU is negative i.e., the union of any \( 2k \) of the subsets has cardinality \( 2k + 1 \) or more. Take an arbitrary regimen. We will argue that any execution must take at least five rounds. At the end of the first round, at most \( 2k \) internal tasks can be eligible, and these must be tasks from \( B_a \). Indeed, each task from \( B_b \) is connected to each of the \( 6k \) sources from \( A \), while at most \( 4k \) of them have been executed; also any internal task from \( B_a \) is connected to two distinct sources. Therefore, in round two, the only internal tasks that can get executed are at most \( 2k \) tasks from \( B_a \). Because the instance is negative, any \( 2k \) subsets have cardinality \( 2k + 1 \) or more, and so one would have to execute at least \( 2k + 1 \) tasks from \( B_a \) to make \( 2k \) sinks from \( C_a \) eligible. Thus at most \( 2k - 1 \) sinks from \( C_a \) can become eligible at the end of the second round. But there are \( 6k \) sinks in total, so at least \( 4k + 1 \) of them are not eligible then. In round three, some nodes get executed, but none of the \( 4k + 1 \) sinks. Even if every of these sinks is eligible at the end of round three, then in round four at most \( 4k \) of them can get executed, leaving one unexecuted at the end of round four. So the regimen requires at least five rounds to complete computation.
Acknowledgements. The author thanks Bruce R. Barkstrom of NASA Langley Research Center and Jennifer M. Schopf of Argonne National Laboratory for motivating this study, and Arnold L. Rosenberg of University of Massachusetts Amherst for discussions on dag scheduling. One missing part of “the square” was proven by inspiration of Xiuning Le.

References

[1] Annis, J., Zhao, Y., Voeckler, J., Wilde, M., Kent, S., Foster, I.: Applying Chimera Virtual Data Concepts to Cluster Finding in the Sloan Sky Survey. ACM/IEEE Conference on Supercomputing (2002) 1–14

[2] Berriman, B., Bergou, A., Deelman, E., Good, J., Jacob, J., Katz, D., Kesselman, C., Laity, A., Singh, G., Su, M.-H., Williams, R.: Montage: A Grid-Enabled Image Mosaic Service for the NVO. Astronomical Data Analysis Software & Systems (2003)

[3] Cordasco, G., Malewicz, G., Rosenberg, A.: Advances in IC-Scheduling Theory: Scheduling Expansive and Reductive Dags and Scheduling Dags via Duality. IEEE Transactions on Parallel and Distributed Systems (2007) to appear

[4] Cormen, T.H., Leiserson, C.E., Rivest, R.L., Stein, C.: Introduction to Algorithms (2nd Edition). MIT Press (2001)

[5] Crescenzi, P., Kann, V. (eds.): A compendium of NP optimization problems. http://www.nada.kth.se/~viggo/wwwcompendium/node173.html

[6] Deelman, E., Kesselman, C., Mehta, G., Meshkat, L., Pearlman, L., Blackburn, K., Ehrens, P., Lazzarini, A., Williams, R., Koranda, S.: GriPhyN and LIGO, building a virtual data grid
for gravitational wave scientists. High Performance Distributed Computing (HPDC’02) (2002) 225–234.

[7] Dilworth, R.P.: A decomposition theorem for partially ordered sets. Annals of Mathematics, Vol. 51 (1950) 161–166

[8] Fernandez, A., Armacost, R., Pet-Edwards, J.: Understanding Simulation Solutions to Resource constrained Project Scheduling Problems with Stochastic task Durations. Engineering Management Journal, Vol. 10(4) (1998) 5–13

[9] Fernandez, A., Armacost, R.L., Pet-Edwards, J.: A Model for the Resource Constrained Project Scheduling Problem with Stochastic Task Durations. 7th Industrial Engineering Research Conference Proceedings (1998)

[10] Foster, I., Kesselman, C. [eds.]: The Grid: Blueprint for a New Computing Infrastructure, 2nd ed. Morgan-Kaufmann, San Francisco, CA (2004)

[11] Gao, L., Malewicz, G.; Toward Maximizing the Quality of Results of Dependent Tasks Computed Unreliably. Theory of Computing Systems (2007) available online

[12] Garey, M.R., Johnson, D.S.: Computers and Intractability. Freeman, New York (1979)

[13] Goel, A., Indyk, P.: Stochastic load balancing and related problems. 40th Annual Symposium on Foundations of Computer Science (FOCS) (1999) 579–586

[14] Hillier, F.S., Lieberman, G.J.: Introduction to Operations Research, 8th ed. McGraw-Hill (2004)

[15] Herroelen, W., Leus, R.: Project scheduling under uncertainty: Survey and research potentials. European Journal of Operational Research, Vol. 165(2) (2005) 289–306

[16] Horn, J.V., Dobson, J., Woodward, J., Wilde, M., Zhao, Y., Voeckler, J., Foster, I.: Grid-Based Computing and the Future of Neuroscience Computation. Methods in Mind, MIT Press, (2005).

[17] Ireland, K., Rosen, M.: A Classical Introduction to Modern Number Theory, 2nd ed. Springer-Verlag (1990)

[18] Johnson, D.S.: The NP-completeness column: An ongoing guide. Journal of Algorithms, Vol. 8(3) (1987) 438–448

[19] Kleinberg, J., Rabani, Y., Tardos, E.: Allocating Bandwidth for Bursty Connections. SIAM Journal on Computing, Vol. 30(1) (2000) 191–217

[20] Lin, G., Rajaraman, R.: Approximation Algorithms for Multiprocessor Scheduling under Uncertainty. 19th ACM Symposium on Parallelism in Algorithms and Architectures (2007) to appear

[21] Lin, G.: New paradigms and approximation algorithms for optimization under uncertainty. Ph.D. Dissertation, College of Computer and Information Science, Northeastern University (2006)
[22] Malewicz, G.: Implementation and Experiments with an Algorithm for Parallel Scheduling of Complex Dags under Uncertainty. Submitted for publication. Preliminary version 8th Workshop on Algorithm Engineering and Experiments (ALENEX’06) (2006) 66–74

[23] Malewicz, G., Foster, I., Rosenberg, A., Wilde, M.: A Tool for Prioritizing DAGMan Jobs and Its Evaluation. Journal of Grid Computing Vol. 5(2) (2007) 197–212

[24] Malewicz, G., Rosenberg, A.L.: On batch-scheduling dags for Internet-based computing. 11th European Conference on Parallel Processing (Euro-Par) (2005) 262–271

[25] Malewicz, G., Rosenberg, A., Yurkewych, M.: Toward a Theory for Scheduling Dags in Internet-Based Computing. IEEE Transactions on Computers, Vol. 55(6) (2006) 757–768

[26] Microsoft Project 2003 [http://www.microsoft.com/office/project/default.asp]

[27] Mori, M., Tseng, C.: A Resource Constrained Project Scheduling Problem with Reattempt at Failure: A Heuristic Approach. Journal of the Operations Research Society of Japan, Vol. 40(1) (1997) 33–44

[28] Narasimhan, M., Ramanujam, J.: A fast approach to computing exact solutions to the resource-constrained scheduling problem. ACM Transactions on Design Automation of Electronic Systems, Vol. 6(4) (2001) 490–500

[29] Norris, J.R.: Markov Chain. Cambridge University Press (1997)

[30] Özdamar, L., Ulusoy, G.: A survey on the resource-constrained project scheduling problem. IIE Transactions, Vol. 27 (1995) 574–586

[31] Peeters, R.: The maximum edge biclique problem is NP-complete. Discrete Applied Mathematics, Vol. 131(3) (2003) 651–654

[32] Rosenberg, A.L.: On scheduling mesh-structured computations for Internet-based computing. IEEE Transactions on Computers, Vol. 53 (2004) 1176–1186

[33] Rosenberg, A.L., Yurkewych, M.: Guidelines for scheduling some common computation-dags for Internet-based computing. IEEE Transactions on Computers, Vol. 54(4) (2005) 428–438

[34] Sarmenta, L.F.G.: Sabotage-tolerance mechanisms for volunteer computing systems. Future Generation Computer Systems, Vol. 18(4) (2002) 561–572

[35] Skutella, M., Uetz, M.: Scheduling precedence-constrained jobs with stochastic processing times on parallel machines. 12th ACM-SIAM Symposium on Discrete Algorithms (SODA) (2001) 589–590

[36] Thain D., Tannenbaum, T., Livny, M.: Distributed computing in practice: the Condor experience. Concurrency and Computation: Practice and Experience, Vol. 17 (2005) 323–356

[37] Tseng, C.C., Mori, M., Yajima, Y.: A project scheduling model considering the success probability. Proc. of the Association of Asian Pacific Operational Research Societies (APROS) (1994) 399–406
[38] Turnquist, M.A., Nozick, L.K.: Allocating Time and Resources in Project Management Under Uncertainty. 36th Annual Hawaii International Conference on System Sciences (HICSS) (2003) 250c