Stochastic Task Networks
Trading Performance for Stability

Kiriakos Simon Mountakis[1], Tomas Klos[2], and Cees Witteveen[1(✉)]

[1] Delft University of Technology, Delft, The Netherlands
C.Witteveen@tudelft.nl
[2] Utrecht University, Utrecht, The Netherlands

Abstract. This paper concerns networks of precedence constraints between tasks with random durations, known as stochastic task networks, often used to model uncertainty in real-world applications. In some applications, we must associate tasks with reliable start-times from which realized start-times will (most likely) not deviate too far. We examine a dispatching strategy according to which a task starts as early as precedence constraints allow, but not earlier than its corresponding *planned release-time*. As these release-times are spread farther apart on the time-axis, the randomness of realized start-times diminishes (i.e. *stability* increases). Effectively, task start-times becomes less sensitive to the outcome durations of their network predecessors. With increasing stability, however, performance deteriorates (e.g. expected makespan increases). Assuming a sample of the durations is given, we define an LP for finding release-times that minimize the performance penalty of reaching a desired level of stability. The resulting LP is costly to solve, so, targeting a specific part of the solution-space, we define an associated Simple Temporal Problem (STP) and show how optimal release-times can be constructed from its earliest-start-time solution. Exploiting the special structure of this STP, we present our main result, a dynamic programming algorithm that finds optimal release-times with considerable efficiency gains.

Keywords: Activity network · Stochastic scheduling · Solution robustness

1 Introduction

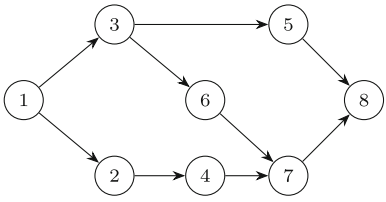
A *stochastic task network* is a directed acyclic graph $G(V, E)$ with each node in $V = \{1, \ldots, n\}$ representing a task with a random duration and each arc $(i, j) \in E$ representing a precedence-constraint between tasks i and j, specifying that task j cannot start unless task i has finished. Such networks appear in several domains like project scheduling [16], parallel computing [22], or even digital circuit design [4], where there is a need to model a partial order of events with uncertain durations. Postulating that a model of uncertainty is known, task durations are described by a random vector $D = (D_1, \ldots, D_n)$ with a known

D. Salvagnin and M. Lombardi (Eds.): CPAIOR 2017, LNCS 10335, pp. 302–311, 2017.
DOI: 10.1007/978-3-319-59776-8_25

probability distribution. In project scheduling, for example, the duration D_i of task i may turn out to be shorter or longer than a nominal value according to a certain distribution.

A given task network is typically mapped to a *realized schedule* (i.e. an assignment of start-times to tasks) via *earliest-start dispatching*; i.e. observing outcome durations and starting a task immediately when precedence-constraints allow (i.e. not later than the maximum finish-time of its network predecessors). Random durations make the realized start-time of a task (and the overall realized schedule makespan) also random. Since *PERT networks* [17], a large body of literature focused on the problem of determining the makespan distribution [1], eventually shown to be a hard problem [11]. A variety of efficient heuristics have been developed so far (see [4]), among which Monte Carlo sampling remains, perhaps, the most practical.

Consider, for example, the stochastic task network in Fig. 1, detailing the plan of a house construction project, assuming task durations are random variables that follow the uniform distribution within respective intervals. With earliest-start dispatching, the overall duration of the project (i.e. the realized schedule makespan) will range between 12 and 20 days with an expected value of a little over 16 days.



(a) Example construction plan.

Tasks	Durations (days)
1. Erect walls	2-4
2. Finish walls	3-5
3. Finish roof	2-6
4. Install plumbing	3-5
5. Finish exterior	6-8
6. Install electricity	3-5
7. Paint interior	2-4
8. Finishing touches	1-1

(b) Estimated task durations.

Fig. 1. A motivating example.

This paper addresses a problem which, to our knowledge, has not been addressed in existing literature. To motivate our problem, let us return to the earlier example and suppose task 7 ("Paint interior") is assigned to a painting crew charging $100 per day. Assume we are willing to hire them for at least 4 days (the maximum number of days they will need) and for at most 6 days; i.e. we have a budget of $600 for painting. With earliest-start dispatching, 7 may start within 8 to 15 days from the project start (the start-date of task 1). A challenge that arises in this situation is deciding when to hire the painting crew, because to allow for an expected makespan of a little over 16 days (as mentioned earlier), we must book the painting crew from the 8-th day and until the 19-th day, at the excessive cost of $1100. The solution we examine here, is to use a different

dispatching strategy, associating task 7 with a *planned release-time*, t_7, before which it may not start even if installing plumbing and electricity are finished earlier than t_7. If we choose that 7 may not start earlier than, e.g., $t_7 = 13$ days from the project start, we only need to book the painting crew on the 13-th day until the 19-th day, for an acceptable cost of \$600. However, the price to pay for this stability is an expected makespan increase to a little over 17 days.

Now suppose that after assessing our budget carefully it turns out that each task may deviate at most, say w days, from its respective planned release-time. The emerging question addressed in this paper is:

> Which planned release-times reach the desired level of stability[1] while minimizing the incurred performance penalty?

This problem does not involve resource-constraints. However, task networks are often used in the area of resource-constrained scheduling under uncertainty (see [2,12]) to represent solutions (e.g. the *earliest-start policy* [13], the *partial-order schedule* [5,10,20]). Thus, our work is expected to be useful in dealing with associated problems, such as distributing slack in a resource-feasible schedule to make it insensitive to durational variability [8].

Organization. A formal problem statement and its LP formulation are presented in Sect. 2. As the resulting LP can be quite costly to solve, Sect. 3 presents our main result, an efficient dynamic programming algorithm. Section 4 concludes the paper and outlines issues to be addressed in future work.

2 Problem Definition

We are given a task network $G(V, E)$ and a stochastic vector $D = (D_1, \ldots, D_n)$ describing task durations. Let $\mathcal{Q}$ index the space of all possible realization scenarios for D such that d_{ip} denotes the realized duration of task i in scenario $p \in \mathcal{Q}$. We assume to know the probability distribution of D; i.e. the probability $\mathbb{P}[D = (d_{1p}, \ldots, d_{np})]$ for all $p \in \mathcal{Q}$. To limit the unpredictability of the realized schedule, we want to associate tasks with respective *planned release-times* $t = (t_1, \ldots, t_n)$ such that the realized schedule is formed by starting a task as early as permitted by precedence-constraints, but not earlier than its release-time. That is, the start-time s_{jp} of task j in scenario p will be determined as:

$$s_{jp} = \max[\max_{(i,j)\in E}(s_{ip} + d_{ip}), t_j] \tag{1}$$

Given a sample $\mathcal{P} \subseteq \mathcal{Q}$ of size m of the stochastic durations vector, this paper is devoted to the following problem:

[1] As in Bidot et al. [3], stability here refers to the extent that a predictive schedule (planned release-times in our case) is expected to remain close to the realized schedule.

$$\min_{t \geq 0} \quad F := \sum_{j \in V, p \in \mathcal{P}} s_{jp} \tag{P}$$

$$\text{subject to} \quad s_{jp} = \max[\max_{(i,j) \in E} (s_i + d_i), t_j] \qquad \forall j \in V, p \in \mathcal{P} \tag{2}$$

$$s_{jp} - t_j \leq w \qquad \forall j \in V, p \in \mathcal{P} \tag{3}$$

This problem tries to optimize a trade-off between stability and performance: release-times are sparsely spread in time in order to form a stable schedule, i.e. such that in every considered scenario a realized start-time will stay within w time-units from the corresponding release-time.

Since the whole space of possible duration realizations, $\mathcal{Q}$, may be too large, or even infinite, we only consider a manageable sample $\mathcal{P} \subseteq \mathcal{Q}$ during optimization.[2] At the same time, we want to ensure a minimal performance penalty $F - F^*$ where F^* denotes the throughput of earliest-start dispatching with no release-times.[3]

Instead of minimizing a standard performance criterion like expected makespan, we choose to maximize *expected throughput*, $\frac{1}{m} \frac{n}{\sum_{j,p} s_{jp}}$, which equals the average rate at which tasks finish over all scenarios. It can be shown that a schedule of maximum throughput is one of minimum makespan and/or tardiness (in case tasks are associated with deadlines). We maximize throughput indirectly by minimizing its inverse, with the constant $\frac{m}{n}$ omitted for simplicity.

LP Formulation. The resulting problem is not easy to handle due to the equality constraint, but using a standard trick it can be rewritten as the following linear program (LP):

$$\min_{s,t \geq 0} \quad F := \sum_{j \in V, p \in \mathcal{P}} s_{jp} \tag{P}$$

$$\text{subject to} \quad s_{jp} \geq s_{ip} + d_{ip} \qquad (i,j) \in E, p \in \mathcal{P} \tag{4}$$

$$s_{jp} \geq t_j \qquad j \in V, p \in \mathcal{P} \tag{5}$$

$$s_{jp} - t_j \leq w \qquad j \in V, p \in \mathcal{P} \tag{6}$$

Note that the solution-space of the resulting LP encompasses that of the original formulation. However, it is easy to show that both problems have the same set of optimal solutions, because a solution (s, t) for the LP cannot be optimal unless it satisfies (2).

Currently, the best (interior-point) LP solvers have a complexity of $O(N^3 M)$ where N is the number of variables and M the input complexity [21]. Thus, letting $\delta \leq n$ denote the max in-degree in $G(V, E)$, the cost of solving (P) as an LP with nm variables and $O(n\delta m)$ constraints can be bounded by $O(n^4 m^4 \delta) \subseteq$

[2] Knowing the distribution of D, we assume to be able to draw $\mathcal{P}$.

[3] The reader can easily recognize the similarity of the proposed LP with a so-called Sample Average Approximation (SAA) of a stochastic optimization problem [14].

$O(n^5m^4)$, which can be daunting even for small instances. Fortunately, as shown in the following section, we manage to obtain the substantially tighter bound of $O(n^2m)$ for solving (P), by exploiting its simple structure to devise a dynamic programming algorithm.

3 Fast Computation of Planned Release-Times

We first show that a fixed relationship between variables s_{jp} and t_j can be assumed while looking for an optimal (s,t). Based on this, a problem (P') is defined which can be solved instead of (P).

A Tighter Formulation. Begin by rewriting (6) as $t_j \geq \max_p s_{jp} - w\,, \forall j \in V$. Now, let Λ denote the set of all feasible (s,t) for problem (P) and let $\Lambda^* \subseteq \Lambda$ be that part of the solution-space that only contains (s,t) for which $t_j = \max_p s_{jp} - w$ for all j.

Lemma 1. *For every feasible $(s,t) \in \Lambda \setminus \Lambda^*$ there exists $(s',t') \in \Lambda^*$ with equal objective value.*

Proof. Consider feasible (s,t) with $t_j = \max_p s_{j^*p} - w + c$ with $c > 0$ for some j^*. Construct t' by letting $t'_j = t_j$ for all $j \neq j^*$ and $t'_{j^*} = t_{j^*} - c = \max_p s_{j^*p} - w$. Trivially, if (s,t) is feasible, so is (s,t'), with the same objective value. Keeping s fixed, we may repeat this construction to enforce that $t_j = \max_p s_{jp} - w$ for all j and have $(s,t') \in \Lambda^*$.

The previous result allows us to consider the following problem, obtained by substituting $\max_{p' \in \mathcal{P}} s_{jp'} - w$ for t_j in (P):

$$\min_{s \geq 0} \quad \sum_p s_{np} \tag{P'}$$

$$\begin{aligned}
\text{subject to} \quad & s_{jp} \geq s_{ip} + d_{ip} && (i,j) \in E, p \in \mathcal{P} && (7)\\
& s_{jp} \geq \max_{p' \in \mathcal{P}} s_{jp'} - w && j \in V, p \in \mathcal{P} && (8)\\
& s_{jp} - (\max_{p' \in \mathcal{P}} s_{jp'} - w) \leq w && j \in V, p \in \mathcal{P} && (9)
\end{aligned}$$

Clearly, $(s,t) \in \Lambda^*$ iff s is feasible for (P').[4] In other words, the solution-space of P' comprises only those s that can be paired with t by letting $t_j = \max_p s_{jp} - w$ to form a feasible (s,t) for (P). By Lemma 1, if s is optimal for (P'), then (s,t) is optimal for (P). Also, if (P) has a solution (i.e. if $G(V,E)$ is acyclic), then (P') also has a solution.

[4] Since $(s,t) \in \Lambda^*$ implies $\max_{p'} s_{jp'} - w = t_j$ for all j.

The Resulting STP. Formulation (P') is useful because it can be cast as a certain type of Temporal Constraint Satisfaction Problem (TCSP) [9]. We start by noting that (9) is always true and can be omitted. Moreover, (8) can be rewritten as (11), to obtain the following reformulation:

$$\min_{s \geq 0} \quad \sum_{p} s_{np} \tag{P'}$$

$$\text{subject to} \quad s_{ip} - s_{jp} \leq -d_{ip} \qquad (i,j) \in E, p \in \mathcal{P} \tag{10}$$

$$s_{jp} - s_{jp'} \leq w \qquad (p,p') \in \mathcal{P}^2, j \in V \tag{11}$$

Constraints (10) and (11) effectively represent the solution-space of a Simple Temporal Problem (STP) [9] with temporal variables $\{s_{jp} : j \in V, p \in \mathcal{P}\}$. The structure of the resulting STP (specifically, of its *distance graph* [9]) is demonstrated in Fig. 2.

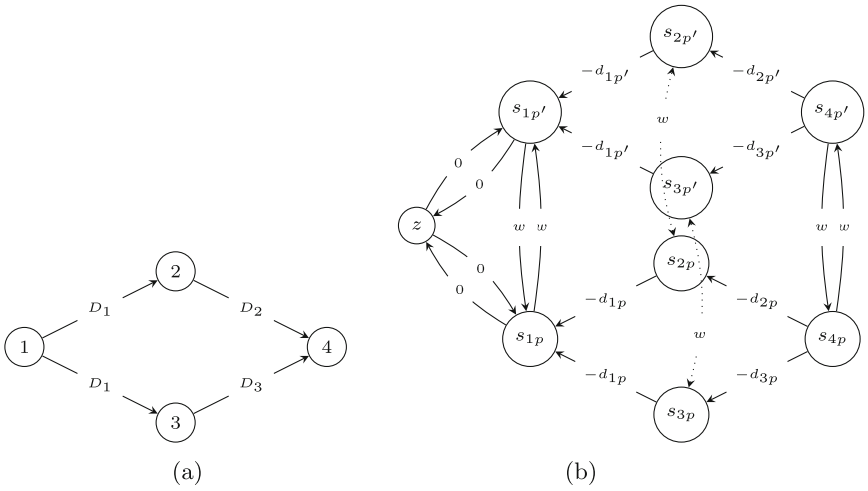


Fig. 2. Example task network (a) and resulting STP (b) for a sample $P = \{p, p''\}$.

The *earliest start time* (est) solution of any given STP (assuming it is consistent) assigns to each variable the smallest value it may take over the set of feasible solutions. Therefore, the est solution of the resulting STP optimally solves (P'), leading us to the following observation.

Observation 1. *By Lemma 1, an optimal solution (s,t) for (P) can be formed by finding the earliest start time solution s of the resulting STP and pairing it with t formed by letting $t_j = \max_{p \in \mathcal{P}} s_{jp} - w$ for all j.*

Algorithm 1

$l(s_{1p}) \leftarrow 0$ for all $p \in \mathcal{P}$
for each tier j in a topological sort of $G(V, E)$ **do**
  $k_{jp} \leftarrow \min\{l(s_{ip}) - d_{ip} : (i, j) \in E\}$ for all $p \in \mathcal{P}$
  $p^* \leftarrow \arg\min\{k_{jp} : p \in \mathcal{P}\}$
  $l(s_{jp}) \leftarrow \max\{k_{jp}, k_{jp^*} + w\}$ for all $p \in \mathcal{P}$
end for
$s_{jp} \leftarrow l(s_{jp})$ for all $j \in V, p \in \mathcal{P}$
$t_j \leftarrow \max_{p \in P} s_{jp} - w$ for each $j \in V$

The est value of s_{jp} is the length of the shortest-path (in the distance graph) from (the node corresponding to) s_{jp} to the special-purpose variable z which is fixed to zero. Those values can be found with a single-source shortest-path algorithm (e.g. Bellman-Ford [19]) in time $O(NM)$ where N is the number of nodes and M the number of arcs. In our case, $N = nm$ and $M = O(nm\delta)$, yielding $O(n^3m^2)$; already a better bound than that of solving (P) as an LP. However, in the following we obtain an even better bound with a dynamic programming algorithm.

Computing the est Solution by Dynamic Programming. Let us associate each task $j \in V$ with a corresponding *tier* including all nodes $\{s_{jp} : p \in \mathcal{P}\}$ of the STP distance graph. A few remarks on the structure of the STP are in order. First, due to (11) the resulting STP is not acyclic, but each cycle only includes nodes that belong to the same tier. Second, due to (10) there is a path from each node in tier j to each node in tier i if and only if there is a path from task i to j in $G(V, E)$.

Let $l(s_{jp})$ denote the shortest-path length from s_{jp} to z (i.e. the value of variable s_{jp} in an optimal solution of (P')). From the structure of the resulting STP, we have:

$$l(s_{jp}) = \min\left\{\min_{(i,j)\in E}(l(s_{ip}) - d_{ip}), \min_{p' \neq p} l(s_{jp'}) + w\right\} \tag{12}$$

The existence of cycles complicates solving subproblem $l(s_{jp})$ as it depends on (and is a dependency of) other subproblems $l(s_{jp'})$ in the same tier. However, we can "break" dependencies between subproblems in the same tier as shown below.

Define $k_{jp} := \min_{(i,j)\in E}(l(s_{ip}) - d_{ip})$ and $p^* := \arg\min_{p\in\mathcal{P}} k_{jp}$.

Lemma 2. $l(s_{jp}) = \min\{k_{jp}, k_{jp^*} + w\}$

Proof. Begin by noting that the shortest-path from s_{jp} to z visits at most one node $s_{jp'}$ from the same tier. As such, for every s_{jp} we have that: either $l(s_{jp}) = k_{jp}$, or $l(s_{jp}) = k_{jp'} + w < k_{jp}$ for some $p' \neq p$.

Now, note that $l(s_{jp^*}) = k_{jp^*}$*, since if not (i.e. if* $l(s_{jp^*}) \neq k_{jp^*}$*), then* $l(s_{jp^*}) = k_{jp'} + w < k_{jp^*}$ *with* $p' \neq p^*$*, which contradicts the definition of* p^**.*

Last, we show that if $l(s_{jp}) \neq k_{jp}$ *then* $l(s_{jp}) = k_{jp^*} + w$*. Suppose not. Since* $l(s_{jp}) \neq k_{jp}$ *then according to (12),* $l(s_{jp}) = l(s_{jp'}) + w$ *but with* $p' \neq p^*$*. Expanding* $l(s_{jp'})$ *according to (12),*

$$\min\{k_{jp'}, \min_{p'' \neq p'} l(s_{jp''}) + w\} + w < l(s_{jp^*}) + w = k_{jp^*} + w$$

and since $k_{jp'} \geq k_{jp^*}$*,*

$$\begin{aligned} &\min_{p'' \neq p'} l(s_{jp''}) + w < k_{jp^*} \\ &\Leftrightarrow l(s_{jp^*}) + w < k_{jp^*} \end{aligned}$$

which contradicts that $l(s_{jp^*}) = k_{jp^*}$*.*

The resulting recursion suggests a dynamic programming approach, summarized in Algorithm 1. It involves solving the subproblems of one tier at a time, visiting tiers according to a topological sort of $G(V, E)$ (recall that tiers correspond to tasks $j \in V$). Finding a topological sort takes $O(n\delta)$ [23], recalling that δ denotes the max in-degree of a task in the network. The overall complexity of Algorithm 1 is therefore $O(nm\delta) \subseteq O(n^2 m)$.

4 Conclusion

Given a stochastic task network with n tasks we consider dispatching the tasks as early as possible, subject to (planned) release-times. Assuming a sample with m realizations of the stochastic durations vector is drawn, we defined an LP for finding optimal release-times; i.e. that minimize the performance penalty of reaching a desired level of stability. The resulting LP is costly to solve, so pursuing a more efficient solution method we managed to show that optimal release-times can be expressed as a function of the earliest start time solution of an associated Simple Temporal Problem. Exploiting the structure of this STP, we were able to define a dynamic programming algorithm for finding optimal release-times with considerable efficiency, in time $O(n^2 m)$.

Future Work. Since we optimize according to a manageable sample $\mathcal{P}$, there is a (potentially non-zero) probability $\mathbb{P}_v$ that the realized start-time of a task deviates further than w time-units from its planned release-time. The question of how $\mathbb{P}_v$ (or $\mathbb{E}[\mathbb{P}_v]$ as in [6]) depends on m (the size of $\mathcal{P}$) should be addressed in future work. Furthermore, in an earlier paper [18], an LP similar to (P) was used it in a two-step heuristic for a flavor of the *stochastic resource constrained project scheduling problem* (stochastic RCPSP) [15,24]. Given a resource allocation determined in a first step, in a second step a LP was used to find planned release-times that minimize the total expected deviation of the realized schedule from those release-times. This heuristic was found to outperform the state-of-the-art in the area of *proactive project scheduling*. In future work, we shall

investigate using the algorithm presented here in order to stabilize the given resource-allocation, expecting gains in both efficiency and effectiveness. Finally, a potentially related problem, namely PERTCONVG, is studied by Chrétienne and Sourd in [7], which involves finding start-times for a task network so as to minimize the sum of convex cost functions. In fact, their algorithm bears structural similarities to ours, since subproblems are solved in a topological order. It would be worth investigating if their analysis can be extended in order to enable casting the problem studied here as an instance of that problem.

References

1. Adlakha, V., Kulkarni, V.G.: A classified bibliography of research on stochastic pert networks: 1966–1987. INFOR **27**, 272–296 (1989)
2. Beck, C., Davenport, A.: A survey of techniques for scheduling with uncertainty (2002)
3. Bidot, J., Vidal, T., Laborie, P., Beck, J.C.: A theoretic and practical framework for scheduling in a stochastic environment. J. Sched. **12**, 315–344 (2009)
4. Blaauw, D., Chopra, K., Srivastava, A., Scheffer, L.: Statistical timing analysis: from basic principles to state of the art. IEEE Trans. Comput. Aided Des. Integr. Circuits Syst. **27**, 589–607 (2008)
5. Bonfietti, A., Lombardi, M., Milano, M.: Disregarding duration uncertainty in partial order schedules? Yes, we can!. In: Simonis, H. (ed.) CPAIOR 2014. LNCS, vol. 8451, pp. 210–225. Springer, Cham (2014). doi:10.1007/978-3-319-07046-9_15
6. Calafiore, G., Campi, M.C.: Uncertain convex programs: randomized solutions and confidence levels. Math. Program. **102**, 25–46 (2005)
7. Chrétienne, P., Sourd, F.: Pert scheduling with convex cost functions. Theoret. Comput. Sci. **292**, 145–164 (2003)
8. Davenport, A., Gefflot, C., Beck, C.: Slack-based techniques for robust schedules. In: Sixth European Conference on Planning (2014)
9. Dechter, R., Meiri, I., Pearl, J.: Temporal constraint networks. Artif. Intell. **49**, 61–95 (1991)
10. Godard, D., Laborie, P., Nuijten, W.: Randomized large neighborhood search for cumulative scheduling. In: ICAPS. vol. 5 (2005)
11. Hagstrom, J.N.: Computing the probability distribution of project duration in a pert network. Networks **20**, 231–244 (1990)
12. Herroelen, W., Leus, R.: Project scheduling under uncertainty: survey and research potentials. EJOR **165**, 289–306 (2005)
13. Igelmund, G., Radermacher, F.J.: Preselective strategies for the optimization of stochastic project networks under resource constraints. Networks **13**, 1–28 (1983)
14. Kleywegt, A.J., Shapiro, A., Homem-de Mello, T.: The sample average approximation method for stochastic discrete optimization. SIAM J. Optim. **12**, 479–502 (2002)
15. Lamas, P., Demeulemeester, E.: A purely proactive scheduling procedure for the resource-constrained project scheduling problem with stochastic activity durations. J. Sched. **19**, 409–428 (2015)
16. Leus, R.: Resource allocation by means of project networks: dominance results. Networks **58**, 50–58 (2011)
17. Malcolm, D.G., Roseboom, J.H., Clark, C.E., Fazar, W.: Application of a technique for research and development program evaluation. Oper. Res. **7**, 646–669 (1959)

18. Mountakis, S., Klos, T., Witteveen, C., Huisman, B.: Exact and heuristic methods for trading-off makespan and stability in stochastic project scheduling. In: MISTA (2015)
19. Pallottino, S.: Shortest-path methods: complexity, interrelations and new propositions. Networks **14**, 257–267 (1984)
20. Policella, N., Oddi, A., Smith, S.F., Cesta, A.: Generating robust partial order schedules. In: Wallace, M. (ed.) CP 2004. LNCS, vol. 3258, pp. 496–511. Springer, Heidelberg (2004). doi:10.1007/978-3-540-30201-8_37
21. Potra, F.A., Wright, S.J.: Interior-point methods. J. Comput. Appl. Math. **124**, 281–302 (2000)
22. Shestak, V., Smith, J., Maciejewski, A.A., Siegel, H.J.: Stochastic robustness metric and its use for static resource allocations. J. Parallel Distrib. Comput. **68**, 1157–1173 (2008)
23. Tarjan, R.E.: Edge-disjoint spanning trees and depth-first search. Acta Informatica **6**, 171–185 (1976)
24. Van de Vonder, S., Demeulemeester, E., Herroelen, W.: Proactive heuristic procedures for robust project scheduling: an experimental analysis. EJOR **189**, 723–733 (2008)