A time-based schedule-generation scheme for project scheduling with storage resources

Mario C. Sillus and Christoph Schwindt

Clausthal University of Technology, Germany
{mario.christian.sillus,christoph.schwindt}@tu-clausthal.de

Keywords: schedule-generation scheme, storage resources, dynamic release dates, generalized precedence relations.

1 Introduction

Storage resources model material stocks or liquid funds, which are depleted and replenished at the occurrence of certain events during the execution of a project. Both renewable and nonrenewable resources are special cases of storage resources. Project scheduling subject to storage-resource constraints and generalized precedence relations consists in sequencing the events in such a way that the inventories of the storage resources evolve within given bounds and prescribed minimum and maximum time lags between the events are met.

The problem was introduced by Neumann and Schwindt (2002), who addressed structural issues, generated benchmark data sets, and devised a branch-and-bound algorithm enumerating disjunctions of precedence relations. Laborie (2003) presents consistency tests and an effective constrained-based branch-and-bound algorithm solving all remaining open instances. Carlier *et al.* (2009) consider a special case with a single resource of infinite capacity. They report on complexity results and propose polynomial-time algorithms to compute an optimal schedule for a given sequence of events. Carlier *et al.* (2018) propose tight lower bounds for the instances of Neumann and Schwindt.

In this paper, we present a time-based schedule-generation scheme that is intended to serve as a building block for metaheuristic schedule-improvement procedures for large problem instances. The method decodes an event list into a feasible schedule by iteratively resolving resource conflicts.

The remainder of this paper is organized as follows. In Sect. 2 we provide a conceptual model formulation and briefly review structural properties of the problem. The basic schedule-generation scheme and several enhancements are developed in Sect. 3. In Sect. 4 we report on computational results obtained on a standard data set from literature.

2 Problem statement

We consider a project employing several storage resources $k \in \mathcal{R}$. During project execution, the inventory level of each resource k must remain within a given nonempty interval $[\underline{R}_k, \overline{R}_k]$ with $\underline{R}_k \in \mathbb{Z} \cup \{-\infty\}$ and $\overline{R}_k \in \mathbb{Z} \cup \{\infty\}$. The inventory levels of resources $k \in \mathcal{R}$ change upon occurrences of events $i = 1, \ldots, n$, which typically coincide with project milestones or starts and completions of project activities. Upon occurrence of event i, the inventory levels of resources k change by $r_{ik} \in \mathbb{Z}$ units. We say that i replenishes the inventory of k if $r_{ik} > 0$ and i depletes the inventory of k if $r_{ik} < 0$. The set V of all events also contains two fictitious events $i = 0$ and $i = n + 1$ standing for the project start and termination, respectively, where r_{0k} is the opening stock level of resource k and without loss of generality $r_{(n+1)k} = 0$. For certain pairs $(i, j) \in A$ of events $i, j \in V$, a minimum time lag $\delta_{ij} \in \mathbb{Z}$ between the occurrences of i and j is prescribed. If $\delta_{ij} < 0$, the value $-\delta_{ij}$ can be viewed as maximum time lag between events j and i. The project scheduling problem $(PSc|temp|C_{max})$ under consideration consists in assigning occurrence times $t_i \geq 0$ to all events $i \in V$ in such a way that (1) a given regular (*i. e.*, componentwise nondecreasing)

objective function f in vector $\boldsymbol{t} = (t_i)_{i \in V}$ is minimized, (2) the resource constraints arising from the storage resources and (3) the generalized precedence relations defined by the time lags are satisfied, and (4) the project is started at time 0.

$$(PSc|temp|f) \quad \begin{cases} \text{Min.} & f(\boldsymbol{t}) & & (1) \\ \text{s. t.} & \underline{R}_k \leq \sum\limits_{i \in V: t_i \leq t_j} r_{ik} \leq \overline{R}_k & (j \in V; k \in \mathcal{R}) & (2) \\ & t_j - t_i \geq \delta_{ij} & ((i,j) \in A) & (3) \\ & t_0 = 0 & & (4) \end{cases}$$

Problem $PSc|temp|f$ generalizes the classical problem $PS|temp|f$ with renewable resources, whose feasibility variant is known to be strongly NP-hard. In difference to the case of renewable resources, finding a feasible schedule for the more general problem with storage resources remains NP-hard even if $|\mathcal{R}| = 1$ and $\delta_{ij} > 0$ for all $(i,j) \in A$.

3 Schedule-generation scheme

Let Π be the set of all precedence-feasible permutations π on set V, *i. e.*, the set of all event lists $\pi = (0, i_1, \ldots, i_n, n+1)$ with $\lambda < \mu$ if $d_{ij} \geq 0$ and $d_{ji} < 0$ for $i = i_\lambda$ and $j = i_\mu$. By $\mathcal{S}'$ we denote the set $\mathcal{S}$ of all feasible schedules $\boldsymbol{t}$ plus an infeasible schedule $\boldsymbol{t}^\infty$ serving to indicate that no feasible schedule could be found. Basically, a schedule-generation scheme (SGS) is a mapping $\sigma : \Pi \to \mathcal{S}'$ assigning a schedule $\boldsymbol{t}' \in \mathcal{S}'$ to each permutation $\pi \in \Pi$.

3.1 Basic scheme

Traditional schedule-generation schemes, like the serial SGS for problems with renewable resources, schedule the activities one by one in the order given by permutation π. In each iteration, the respective partial schedule represents a feasible solution to the problem defined on the set of activities scheduled thus far. Deadlocks caused by maximum time lags are resolved using unscheduling techniques. Given that preserving the feasibility of a partial schedule would generally require the simultaneous addition of several events and that finding such a set constitutes an NP-hard problem, we opt for a different approach, which draws from an enumeration scheme with dynamic activity release dates devised by Fest *et al.* (1998) for the project duration problem $PS|temp|C_{max}$. The basic idea consists in first relaxing the resource constraints and then iteratively resolving inventory shortfalls or excesses by defining release dates δ_{0j} for appropriate events j, which are determined by their position in π. The principle to (pre-)select the events to be postponed via a permutation can be interpreted as a linear preselective strategy (Stork 2001).

Our time-based SGS is displayed in Algorithm 1. Let $D = (d_{ij})_{i,j \in V}$ denote the distance matrix containing the transitive time lags d_{ij} implied by prescribed time lags δ_{ij}. In each iteration of the SGS, we consider the current earliest schedule $\boldsymbol{et} = (d_{0i})_{i \in V}$ with $t_0 = 0$ and satisfying all time lags δ_{ij} for $(i,j) \in A$ and release dates δ_{0j} introduced so far. Schedule $\boldsymbol{et}$ is then scanned for the earliest time t at which a resource conflict occurs, *i. e.*, $r_k(\boldsymbol{et}, t) := \sum_{i \in V: et_i \leq t} r_{ik} \notin [\underline{R}_k, \overline{R}_k]$ for some k. If $\underline{R}_k \leq r_k(\boldsymbol{et}, t) \leq \overline{R}_k$ for all $k \in \mathcal{R}$ and all $t \geq 0$, the SGS is terminated by returning feasible schedule $\boldsymbol{et}$. Otherwise, we compute the set $\mathcal{C}$ of all events $j \in V$ that contributed to the resource conflict on k. To resolve the conflict, at least one event $j \in \mathcal{C}$ has to be delayed to the earliest occurrence time $et_i > t$ of an event i with the opposite resource requirement, *i. e.*, $r_{ik} \cdot r_{jk} < 0$. Event j can be synchronized with event i precisely if $d_{ji} \leq 0$. The event that is actually deferred is the last element $j \in \mathcal{C}$ on list π possessing such a partner event i. If no such event j exists, the SGS is stopped and infeasible schedule $\boldsymbol{t}^\infty$ is returned. Otherwise, the new release date δ_{0j} is set to $t_j^* := \min\{et_i \mid i \in V,\ et_i > t,\ r_{ik} \cdot r_{jk} < 0,\ d_{ji} \leq 0\}$. Finally, δ_{0j} is added to distance matrix D by updating $d_{0l} := \max\{d_{0l}, t_j^* + d_{jl}\}$ for all events $l \in V$.

Algorithm 1 Basic schedule-generation scheme SGS(π)

1: compute distance matrix $D = (d_{ij})_{i,j \in V}$ of time lags $(\delta_{ij})_{(i,j) \in A}$;
2: **loop**
3: set earliest schedule $\boldsymbol{et} := (d_{0i})_{i \in V}$;
4: determine earliest time t at which a resource conflict occurs on some resource $k \in \mathcal{R}$;
5: **if** $t = \infty$ **then return** $\boldsymbol{et}$; ($*$ feasible schedule has been generated $*$)
6: **if** $r_k(\boldsymbol{et}, t) < \underline{R}_k$ **then** set conflict set $\mathcal{C} := \{j \in V \mid et_j \leq t,\ r_{jk} < 0\}$;
7: **else** set conflict set $\mathcal{C} := \{j \in V \mid et_j \leq t,\ r_{jk} > 0\}$;
8: determine last event $j \in \mathcal{C}$ on π with $t_j^* := \min_{i \in V}\{et_i \mid et_i > t,\ r_{ik} \cdot r_{jk} < 0,\ d_{ji} \leq 0\} < \infty$;
9: **if** there is no such event j **then return** $\boldsymbol{t}^\infty$; ($*$ no feasible schedule could be found $*$)
10: **else** put $d_{0l} := \max\{d_{0l}, t_j^* + d_{jl}\}$ for all $l \in V$; ($*$ add $\delta_{0j} = t_j^*$ and update D $*$)

3.2 Expansions

E1: Randomization. Given that we add release dates $\delta_{0j} = t_j^*$ and not precedence relations $\delta_{ij} = 0$, the SGS can encounter a so-called leapfrogging phenomenon that may cause cycling in an infinite loop of mutual shifting among three or more events. To get out of leapfrogging, it proves expedient to introduce a pinch of randomness when selecting event j. More precisely, let $\mathcal{C}' := \{j \in \mathcal{C} \mid t_j^* < \infty,\ (d_{jj'} < 0) \vee (d_{j'j} \geq 0) \text{ for all } j' \in \mathcal{C}\}$ be the set of candidates to deferment. Starting with the last $j \in \mathcal{C}'$ in π, we accept j as the event to be delayed with probability $p < 1$. If j was rejected, we recursively proceed with the preceding $j \in \mathcal{C}'$ in π until some $j \in \mathcal{C}'$ was accepted, where we return to the last $j \in \mathcal{C}'$ in π if the first event $j \in \mathcal{C}'$ was rejected. The number of iterations follows geometric distribution $Geo(p)$. Consequently, it suffices to draw a random number z from distribution $Geo(p)$ and to select the mth element $j \in \mathcal{C}'$ in π with $m := |\mathcal{C}'| - (z-1) \mod |\mathcal{C}'|$.

E2: Preservation of feasible initial inventories. If the problem instance is feasible and no deadlines $-d_{j0} < \infty$ are imposed on the occurrence of events $j \in V$, Algorithm 1 can only be quitted without feasible schedule if the initial inventory level $r_k(\boldsymbol{et}, 0)$ is not within the bounds $\underline{R}_k$ and $\overline{R}_k$. The reason is that due to constraint $t_0 = 0$, conflicts at time $t = 0$ may become unsolvable if $r_{0k} \notin [\underline{R}_k, \overline{R}_k]$. If such a resource k with infeasible opening stock exists, we should avoid right-shifting any event l with $et_l = 0$ while removing an infeasibility at time $t > 0$. When performing the update of distance matrix D, delaying event j to time t_j^* leads to an increase of d_{0l} exactly if $t_j^* + d_{jl} > et_l$. Accordingly, we impose the additional condition $t_j^* \leq \min\{-d_{jl} \mid l \in V : et_l = 0\}$ on the selection of event j, provided that at least one event from set $\mathcal{C}'$ satisfies this condition.

E3: Schedule contraction. Despite the additional condition introduced in expansion E2, we may still encounter situations in which there does not exist any $j \in \mathcal{C}'$ when dealing with a conflict at time $t = 0$. Since in general $r_{0k} \neq 0$, it might be useful to synchronize $j = 0$ with the next appropriate event i for solving the conflict. In the basic SGS, however, condition $d_{ji} \leq 0$ prevents $j = 0$ from being moved because $et_i = d_{0i} > 0$ contradicts $d_{ji} = d_{0i} \leq 0$. Instead of delaying $j = 0$, the schedule contraction technique performs an equivalent relative movement between events by left-shifting event i and further events l. To this end, we put $d_{0l} := \max\{\hat{d}_{0l}, d_{0l} - et_i\}$ for all $l \in V$, where $\hat{d}_{0l}$ stands for the initial earliest occurrence times computed on line 1 of Algorithm 1. Schedule contraction allows us to entirely avoid premature terminations of the SGS.

E4: Postprocessing. Randomizing the selection of event j reliably prevents long leapfrogging phases. Nevertheless, leapfrogging cannot be avoided completely, and thus the schedule $\boldsymbol{et}$ yielded by the SGS often fails to be quasiactive (Neumann *et al.* 2003, Sect. 2.4). In this case, $\boldsymbol{et}$ can be further compressed by applying the following postprocessing procedure. For each pair (i, j) with $et_j \geq et_i$ and $r_{ik} \cdot r_{jk} < 0$ for some $k \in \mathcal{R}$ we put $\hat{d}_{ij} := \max\{\hat{d}_{ij}, 0\}$ and restore the transitivity of matrix $\hat{D}$ by applying the Floyd-Warshall algorithm. Since $\boldsymbol{et}$ is resource-feasible, any schedule $\boldsymbol{t}$ satisfying the temporal constraints $t_j \geq t_i + \hat{d}_{ij}$ is feasible as well. In particular, this holds true for the compressed earliest schedule $\boldsymbol{et}' = (\hat{d}_{0j})_{j \in V} \leq \boldsymbol{et}$.

4 Computational experiments

We tested the performance of different versions of the SGS using the solvable instances of the project duration problem with $n = 100$ events and 5 storage resources from Neumann and Schwindt (2002). The test set contains 90 randomly generated instances, 57 of which possess a feasible solution. We considered three types of precedence-feasible permutations π: the ***et***-list, which orders the events according to nondecreasing earliest occurrence times, the ***lt***-list, where events are arranged in order of nondecreasing latest occurrence times, and randomly generated lists. In total, we carried out the four experiments listed in Table 1.

Experiments A to D stepwise introduce extensions E1 to E3 into the basic SGS with postprocessing E4. For each instance, 100 executions of the SGS are distributed over the list types as shown in the second column of the table. The CPU time limit was set to 10 ms per execution, and the smallest project duration found in previous runs was defined as a deadline. While the randomization in experiment A comes from outside the SGS via the event lists, the randomization in B to D is implemented inside the SGS via acceptance probability $p < 1$ (we chose $p = 0.4$). The SGS was implemented in Java 11 under Eclipse and ran on a PC with one core and 4 GHz clock pulse operating under Windows 10.

In the right part of Table 1 we list the mean percentages p_{opt}, p_{feas}, and p_{no} of optimally solved, feasibly, but not optimally solved, and unsolved instances averaged over ten replications for each experiment. Δ_{opt} stands for the relative optimality gap of the instances solved to feasibility by the respective SGS version. All percentages refer to the best schedules found for each instance. Symbol t_{cpu} denotes the mean CPU time for 100 runs per instance in seconds.

Table 1. Experimental design and computational results

Experiment	`#et/#lt/#rand`	Expansions	p_{opt} [%]	p_{feas} [%]	p_{no} [%]	Δ_{opt} [%]	t_{cpu} [s]
A	1/1/98	E4	64.4	23.1	12.4	1.12	0.34
B	50/50/0	E1, E4	80.0	10.9	9.1	0.32	0.12
C	50/50/0	E1, E2, E4	91.4	5.8	2.8	0.29	0.16
D	50/50/0	E1 – E4	93.0	5.6	1.5	0.41	0.25

The successive additions of extensions consistently lead to higher percentages of optimally and of feasibly solved instances. In particular, the fully equipped SGS achieves good results with more than 90 % of instances solved to optimality and a mean optimality gap of 0.24 %, while the computational effort remains in reasonable order of magnitude.

In a further experiment we ran the SGS with expansions E1, E3 and the ***et*** and ***lt*** lists without time limit. As expected, the method delivered a feasible solution to each instance, which averaged over ten replications took 2.77 s for the ***et*** and 0.9 s for the ***lt*** list.

References

Carlier J., A. Moukrim, H. Xu, 2009, "The project scheduling problem with production and consumption of resources: A list-scheduling based algorithm", *Discrete Appl Math*, Vol. 157, pp. 3631–3642.

Carlier J., A. Moukrim, A. Sahli, 2018, "Lower bounds for the event scheduling problem with consumption and production of resources", *Discrete Appl Math*, Vol. 234, pp. 178–194.

Fest A., R. H. Möhring, F. Stork, M. Uetz, 1998, "Resource-constrained project scheduling with time windows: A branching scheme with dynamic release dates", *Working Paper 596, Fachbereich Mathematik, TU Berlin*.

Laborie P., 2003, "Algorithms for propagating resource constraints in AI planning and scheduling: Existing approaches and new results", *Artif Intell*, Vol. 143, pp. 151–188.

Neumann, K., C. Schwindt, 2002, "Project scheduling with inventory constraints", *Math Method Oper Res*, Vol. 56, pp. 513–533.

Neumann, K., C. Schwindt, J. Zimmermann, 2003, "Project Scheduling with Time Windows and Scarce Resources". Springer, Berlin

Stork, F., 2001, "Stochastic Resource-Constrained Project Scheduling". PhD Thesis, TU Berlin