A serial schedule-generation scheme for preemptive project scheduling problems with generalized precedence relations and regular min-max criteria

Christoph Schwindt and Tobias Paetz

Clausthal University of Technology, Germany
{christoph.schwindt,tobias.paetz}@tu-clausthal.de

Keywords: project scheduling, activity preemptions, generalized precedence relations, min-max criteria, schedule-generation scheme.

1 Introduction

In this paper we devise an extension of the serial schedule-generation scheme that is adapted to a general class of preemptive project scheduling problems. We suppose that each activity of the project can be interrupted at any point in time, that generalized precedence relations between the activities have to be taken into account, and that the objective function f_{max} to be minimized can be expressed as the maximum of regular functions $f_j(C_j)$ in the completion times C_j of activities j.

Approaches for preemptive project scheduling that allow for continuous activity interruptions are generally based on problem formulations where feasible sets of activities are associated with decision variables representing the time during which these activities are processed in parallel (see, *e. g.*, Damay *et al.* 2007 or Schwindt and Paetz 2015). The main difficulty in developing schedule-construction procedures for continuous preemption problems arises from the fact that activities may have to be interrupted at points in time when no activity is released and no scheduled activity is started or completed. We resolve this difficulty by performing a binary search over the set of potential optimum objective function values v, translating them into deadlines for the completion of the activities, and considering the resulting *latest* start and completion times of the activities as additional decision points at which activities may be suspended or resumed.

2 Problem statement

Consider a project that consists of a set V of activities j with deterministic durations p_j. To perform the activities, a set $\mathcal{R}$ of renewable resources k with capacities R_k is available. When being in progress, each activity j requires r_{jk} units of resources $k \in \mathcal{R}$. The processing of an activity may be stopped at any point in time and resumed later on using an arbitrary set of r_{jk} available units of each resource k. As it has been shown by Baptiste *et al.* (2011) for a more general preemptive scheduling problem, each feasible instance of the problem admits a solution with a finite number of activity interruptions. Hence, a schedule may be encoded as a finite set Σ of triples (j, s_j, c_j) providing the time intervals $[s_j, c_j[$ during which activities j are in progress. Equivalently, a solution to the problem can be specified as a trajectory $x : t \mapsto (x_j(t))_{j \in V}$, where $x_j(t) \in [0, 1]$ denotes the percentage of activity j that has been processed by time $t \geq 0$.

The generalized precedence relations between the activities can be stated in the following way. Let $t_j^-(\xi) := \min\{t \geq 0 \mid x_j(t) \geq \xi\}$ with $0 < \xi \leq 1$ and $t_j^+(\xi) := \sup\{t \geq 0 \mid x_j(t) \leq \xi\}$ with $0 \leq \xi < 1$ be the earliest and latest times t, respectively, at which activity j has been executed to portion ξ. The precedence relations $\Delta_{ij} = (\xi_i, \xi_j, \delta_{ij})$ for pairs $(i, j) \in E \subseteq V \times V$ require that the final portion $1 - \xi_j$ of activity j can be started δ_{ij} time units after activity i attained relative progress ξ_i at the earliest, *i. e.*, $t_j^+(\xi_j) \geq t_i^-(\xi_i) + \delta_{ij}$ for $(i, j) \in E$. In Schwindt and Paetz (2015) it is shown that by appropriately splitting

activities, all precedence relations can be transformed into completion-to-start relations $\Delta_{ij} = (1, 0, \delta_{ij}^{cs})$ between the completion time C_i of activity i and the start time S_j of activity j. Note that (i) negative time lags $\delta_{ij}^{cs} < 0$ can be interpreted as positive maximum start-to-completion time lags $-\delta_{ij}^{cs}$ between the start of j and the completion of i and that (ii) set E may contain pairs (j, j) specifying maximum time spans for the execution of activities j. In sum, we obtain the following formulation of the preemptive project scheduling problem (P) under consideration, where $y_j(t) := p_j \cdot \frac{\mathrm{d}^+ x_j}{\mathrm{d}t}(t)$ equals one if activity j is in progress at time t, and zero, otherwise, and $r_k(\cdot)$ denotes the loading profile of resource k.

$$(P)\ \begin{cases} \text{Minimize} & f_{max}(C) = \max_{j \in V} f_j(C_j) & \\ \text{subject to} & r_k(t) := \sum_{i \in V} r_{ik} y_i(t) \leq R_k & (k \in \mathcal{R};\ t \geq 0) \\ & S_j \geq C_i + \delta_{ij}^{cs} & ((i,j) \in E) \\ & S_j \geq 0,\ C_j \geq S_j + p_j & (j \in V) \end{cases}$$

3 Serial schedule-generation scheme

The serial schedule-generation scheme (SGS) is a basic building block of most heuristic procedures for non-preemptive resource-constrained project scheduling problems with regular objective functions. SGS schedules the activities one after another at their earliest time- and resource-feasible start times. The basic SGS was enhanced with an unscheduling method by Franck *et al.* (2001) to cope with generalized precedence relations. In what follows we adapt the latter scheme to problems with continuous activity preemptions and regular min-max criteria f_{max}.

3.1 Binary search procedure

By $D^{cs} = (d_{ij}^{cs})_{i,j \in V}$ we denote the matrix of transitive completion-to-start time lags that are implied by time lags δ_{ij}^{cs} with $(i, j) \in E$. Matrix D^{cs} can be computed efficiently by an adapted version of the Floyd-Warshall algorithm for longest path calculations in networks. Moreover, we assume that set V contains a dummy activity $j = 0$ with $p_j = 0$ representing the project beginning at time $t = 0$. Then, $ES_j = d_{0j}^{cs}$ and $LC_j = -d_{j0}^{cs}$ are the earliest time-feasible start and the latest time-feasible completion time of activity j.

Now assume that we are given some upper bound v on the objective function value $f_{max}(C)$. Upper bound v can easily be translated into a set of latest completion times LC_j of activities $j \in V$ by taking advantage of the following equivalences:

$$f_{max}(C) = \max_{j \in V} f_j(C_j) \leq v \Leftrightarrow f_j(C_j) \leq v\ (j \in V) \Leftrightarrow C_j \leq f_j^{-1}(v)\ (j \in V)$$

where $f_j^{-1}(v) := \sup\{C_j \mid f_j(C_j) \leq v\}$ and $\sup \emptyset := -\infty$. The second equivalence holds because we assume that functions f_j are regular, *i. e.*, nondecreasing in C_j. Hence, by putting the latest completion times LC_j to $\min\{-d_{j0}^{cs}, f_j^{-1}(v)\}$ for all $j \in V$ we ensure that $f_{max}(C) \leq v$ for any time-feasible schedule. Given the optimum objective function value $v = f_{max}^*$, the latest completion times LC_j then provide the latest start times $LC_j - p_j$ at which the activities must be started in order to obtain an optimal schedule.

The basic principle of our schedule-construction procedure is as follows. Assuming that the problem is feasible, we compute some lower and upper bounds lb and ub on f_{max}^*. We then perform a binary search for f_{max}^* on interval $[lb, ub]$ starting with $v = \frac{lb+ub}{2}$. In each iteration, we apply the preemptive SGS to the instance where the latest completion times LC_j are set according to current upper bound v. If a feasible schedule is found, we know that $f_{max}^* \leq f_{max}(C) \leq v$, which leads to the tighter upper bound $ub := f_{max}(C)$. Otherwise, we were not able to generate a feasible schedule with $f_{max}(C) \leq v$, and we put $lb := v$. With $\varepsilon > 0$ denoting a given accuracy tolerance, we recursively continue the search process on the updated interval $[lb, ub]$ until the width of the interval has been reduced to $ub - lb \leq \varepsilon$. The number of iterations of the binary search is of order $O(\log(ub - lb) - \log \varepsilon)$, where lb and ub stand for the original lower and upper bounds.

3.2 Serial schedule-generation scheme

To generate a schedule for a given upper bound v, we developed a preemptive version of the enhanced SGS of Franck *et al.* (see Algorithms 1 and 2). All quantities referring to sub-activities executed in time intervals $[s_j, c_j[$ are designated with lowercase symbols, whereas the quantities that apply to the entire activities j are written in uppercase letters.

Algorithm 1 Preemptive serial schedule-generation scheme

Input: Upper bound v on f^*_{max}, maximum number u_{max} of unscheduling steps
Output: Feasible schedule Σ with $f_{max}(C) \leq v$

1: **for** $j \in V$ **do** $d^{cs}_{j0} := \max\{d^{cs}_{j0}, -f_j^{-1}(v)\}$;
2: **for** $i \in V$ **do** $d^{cs}_{i0} := \max\{d^{cs}_{i0}, \max_{j \in V}(d^{cs}_{ij} + p_j + d^{cs}_{j0})\}$;
3: **for** $i, j \in V$ **do** $d^{cs}_{ij} := \max\{d^{cs}_{ij}, d^{cs}_{i0} + d^{cs}_{0j}\}$;
4: **for** $j \in V$ **do** $es_j := d^{cs}_{0j}$, $LC_j := -d^{cs}_{j0}$, $Pred^{\prec}(j) := \{i \in V \mid d^{cs}_{ij} + p_i \geq 0 \wedge d^{cs}_{ji} + p_j < 0\}$;
5: $\mathcal{S} := \mathcal{C} := \{0\}$, $\Sigma := \{(0,0,0)\}$, $\Theta := \{0\}$, $u := 0$;
6: **while** $\mathcal{C} \neq V$ **do** (∗ not all activities have entirely been processed ∗)
7: $\quad \mathcal{E} := \{j \in V \setminus \mathcal{C} \mid Pred^{\prec}(j) \subseteq \mathcal{C}\}$;
8: $\quad$ select activity $j^* \in \arg\min_{j \in \mathcal{E}} \pi(j)$;
9: $\quad s_{j^*} := \min\{t \geq es_{j^*} \mid r_k(t) + r_{j^*k} \leq R_k \text{ for all } k \in \mathcal{R}\}$;
10: $\quad$ **if** $s_{j^*} + p_{j^*} > LC_{j^*}$ **then** (∗ activity j^* cannot be completed on time ∗)
11: $\quad\quad$ **if** $u \geq u_{max}$ **then** terminate;
12: $\quad\quad$ **else** $u := u + 1$, ***unschedule***$(j^*, s_{j^*} + p_{j^*} - LC_{j^*})$;
13: $\quad$ **else** (∗ process activity j^* from s_{j^*} to c_{j^*} ∗)
14: $\quad\quad$ **if** $j^* \notin \mathcal{S}$ **then**
15: $\quad\quad\quad \mathcal{S} := \mathcal{S} \cup \{j^*\}$, $S_{j^*} := s_{j^*}$;
16: $\quad\quad\quad$ **for** $j \in V \setminus \mathcal{C}$ **do** $LC_j := \min\{LC_j, S_{j^*} - d^{cs}_{jj^*}\}$;
17: $\quad\quad$ **for** $j \in V \setminus (\mathcal{C} \cup \{j^*\})$ **do** $es_j := \max\{es_j, s_{j^*} + p_{j^*} + d^{cs}_{j^*j}\}$;
18: $\quad\quad c_{j^*} := \min\{t > s_{j^*} \mid t \in \Theta \cup \{s_{j^*} + p_{j^*}\} \cup (\cup_{j \in V \setminus (\mathcal{C} \cup \{j^*\})}\{es_j, es_j + p_j, LC_j - p_j, LC_j\})\}$;
19: $\quad\quad \Sigma := \Sigma \cup \{(j^*, s_{j^*}, c_{j^*})\}$, $\Theta := \Theta \cup \{s_{j^*}, c_{j^*}\}$;
20: $\quad\quad p_{j^*} := p_{j^*} - (c_{j^*} - s_{j^*})$;
21: $\quad\quad$ **if** $p_{j^*} > 0$ **then** $es_{j^*} := c_{j^*}$;
22: $\quad\quad$ **else** $\mathcal{C} := \mathcal{C} \cup \{j^*\}$, $C_{j^*} := c_{j^*}$;
23: **return** Σ;

At first, we translate upper bound v into maximum time lags $f_j^{-1}(v)$ between project beginning $j = 0$ and the completion of activities $j \in V$ and update time lag matrix D^{cs} (lines 1–3). Next, we schedule $j = 0$ and initialize the earliest start times es_j, the latest completion times LC_j, the predecessor sets $Pred^{\prec}(j)$, the sets of started and completed activities $\mathcal{S}$ and $\mathcal{C}$, the schedule Σ, the set Θ of start and completion times of scheduled sub-activities, and the number u of unscheduling steps (lines 4 and 5). In each iteration, we then compute the set $\mathcal{E}$ of all activities whose predecessors with respect to precedence order $\prec$ were already completed and which consequently are eligible to be scheduled (line 6). By applying some priority rule π we choose one eligible activity j^* and determine the earliest point in time s_{j^*} at which it can be started or resumed (lines 6 and 7). If the resulting earliest completion time of j^* is larger than its latest completion time LC_{j^*}, we try to repair the schedule by calling the unscheduling procedure (lines 10–12). Otherwise, we process j^* from time s_{j^*} to the next decision time c_{j^*}, decrease the residual processing time of j^* by $c_{j^*} - s_{j^*}$, and update the earliest start and latest completion times es_j and LC_j, schedule Σ, as well as sets $\mathcal{S}$, $\mathcal{C}$, and Θ (lines 14–22). The set of all decision times contains the start and completion times $t \in \Theta$ of scheduled sub-activities, the earliest completion time $s_{j^*} + p_{j^*}$ of activity j^*, as well as the earliest and latest start and completion times of all (sub-)activities not yet scheduled. When all activities have been entirely added to the schedule, we return the generated schedule Σ (line 23).

The unscheduling procedure identifies the activities $j \in \mathcal{U}$ that determined the latest completion time LC_{j^*} of activity j^*. The current schedule Σ is then cleared from the minimum start time t^* of activities $j \in \mathcal{U}$. Next, the activities $j \in \mathcal{U}$ are delayed by the minimum amount of time $\Delta = s_{j^*} + p_{j^*} - LC_{j^*}$ that is necessary to get a latest completion time of $LC_{j^*} = s_{j^*} + p_{j^*}$ in the next pass of the SGS. Finally, the earliest start and latest completion times are updated.

Algorithm 2 ***unschedule***(j^*, Δ)

$\mathcal{U} := \{j \in \mathcal{S} \mid LC_{j^*} = S_j - d^{cs}_{j^*j}\}$, $t^* := \min_{j \in \mathcal{U}} S_j$;
for $j \in \mathcal{S}$ **do** ($*$ clear schedule from time t^* to end $*$)
  $p_j := p_j + \sum_{(j,s_j,c_j) \in \Sigma : s_j \geq t^*} (c_j - s_j)$;
  $\Sigma := \Sigma \setminus \{(j, s_j, c_j) \mid s_j \geq t^*\}$, $\Theta := \Theta \setminus \{s_j, c_j\}$;
  if $\{(j, s_j, c_j) \mid (j, s_j, c_j) \in \Sigma\} = \emptyset$ **then** $\mathcal{S} := \mathcal{S} \setminus \{j\}$;
  if $j \in \mathcal{C}$ **and** $p_j > 0$ **then** $\mathcal{C} := \mathcal{C} \setminus \{j\}$;
  if $j \in \mathcal{S} \setminus \mathcal{C}$ **then** $es_j := \max\{c_j \mid (j, s_j, c_j) \in \Sigma\}$;
for $j \in \mathcal{U}$ **do** ($*$ delay activities $j \in \mathcal{U}$ $*$)
  $es_j := S_j + \Delta$;
  if $es_j + p_j > -d^{cs}_{j0}$ **then** terminate;
($*$ update earliest start and latest completion times $*$)
for $j \in V \setminus (\mathcal{S} \cup \mathcal{U})$ **do** $es_j := \max\{d^{cs}_{0j}, \max_{i \in \mathcal{S} \cup \mathcal{U}}(es_i + p_i + d^{cs}_{ij})\}$;
for $j \in V \setminus \mathcal{C}$ **do** $LC_j := \min\{-d^{cs}_{j0}, \min_{i \in \mathcal{S}}(S_i - d^{cs}_{ji})\}$;

4 Experimental performance analysis

We tested the performance of the preemptive SGS by solving the preemptive versions of 360 instances of the resource-constrained project duration problem with generalized precedence relations $PS|temp|C_{max}$. The respective UBO-10, UBO-20, UBO-50, and UBO-100 test sets each contain 90 projects with 5 resources and $n = 10$, 20, 50, or 100 activities (see Franck *et al.* 2001). In our randomized multi-start implementation, we repeated the binary search 100 times and selected the best schedule found. The priority indices for choosing activity j^* to be scheduled next in the SGS were calculated according to

$$\pi(j) = (LC_j - p_j) \cdot (1 - 2\sigma(\mu) \cdot rnd) \text{ with } \sigma(\mu) = [1 + e^{-\alpha \cdot (\mu - \mu_{max}/2)}]^{-1}$$

where $\mu = 1, \ldots, \mu_{max}$ is the number of the respective run of the binary search, rnd is a $(0, 1)$-uniformly distributed pseudo-random number, and parameter α was set to 0.1. This multi-start heuristic was applied to the original UBO instances and the set of mirrored instances that were created using the transformation rules described in Hanzálek and Šůcha (2009). The algorithm was coded in C# under Visual Studio Express 2015 for Win Desktop. We performed the experiments on an Intel i5 PC with 3,4 GHz clock pulse and 8 GB RAM, running Win 7 Professional 64 Bit as operating system. The accuracy tolerance of the binary search was set to $\varepsilon = 10^{-4}$.

Tables 1 and 2 show the results obtained when the multi-start algorithm with was applied to the original instances ("forward scheduling" with $\mu_{max} = 100$) and to the original and the mirrored instances ("forward and backward scheduling" with $\mu_{max} = 50$). For each of the four test sets, the tables display (all percentages are specified in percent)

- the percentages p^{pmtn}_{inf} and p^{npmtn}_{inf} of instances for which infeasibility is known when preemption is allowed or not allowed,
- the relative deviations Δ_{MILP} and Δ_{npmtn} of the best project durations yielded by SGS from the best known solutions for the preemptive problem obtained with the MILP formulation of Schwindt and Paetz (2015) and for the the non-preemptive problem,
- the percentages p_{opt} and p_{feas} of optimal and feasible but not necessarily optimal schedules found as well as the percentages p_{nfeas} of instances for which no feasible schedule was generated,

– the percentages p_{nfnd} of feasible instances for which no solution was found, the percentages p_{fnd} of infeasible instances of the non-preemptive problem for which a feasible preemptive schedule was found, and the percentages p_{imp} of instances for which SGS improved the project duration compared to the best known non-preemptive solution,
– the mean CPU times t_{cpu} in seconds and the mean total number $\#\,it$ of SGS runs in the binary search.

When the multi-pass method was applied to the original instances, we obtained moderate mean deviations from the best known MILP solutions of 0.5 and 0.06 %. Allowing for activity preemptions reduced the project duration by 1.2 to 1.7 % on average. For ten of the 360 instances no feasible schedule could be generated even though the instances are known to be feasible. On the other hand, we obtained a feasible schedule for 18 instances that are infeasible when activity splitting is not allowed and got better project durations when considering activity interruptions in 140 cases. Ranging between 0.14 and 24.5 seconds, the mean CPU times appeared to be in a reasonable order of magnitude.

Table 1. Results for UBO test sets: forward scheduling

	p_{inf}^{pmtn}	p_{inf}^{npmtn}	Δ_{MILP}	Δ_{npmtn}	p_{opt}	p_{feas}	p_{nfeas}	p_{nfnd}	p_{fnd}	p_{imp}	t_{cpu}	$\#\,it$
$n = 10$	14.44	18.89	0.50	−1.71	48.89	36.67	14.44	0.00	4.44	27.78	0.14	1668
$n = 20$	10.00	22.22	0.06	−1.68	21.11	66.67	12.22	0.00	10.00	36.67	0.37	1746
$n = 50$	12.22	18.89	N/A	−1.35	22.22	61.11	16.67	1.11	3.33	44.44	3.15	1728
$n = 100$	5.56	13.33	N/A	−1.16	15.56	62.22	22.22	8.89	0.00	46.67	24.50	1837

Combining forward and backward scheduling further enhanced the performance of the multi-pass method. In particular, the improvement on the non-preemptive schedules was increased and more feasible schedules were found. In fact, there is only one instance known to be feasible for which no schedule could be generated. Given that the feasibility variant of the problem is strongly NP-complete, the performance seems promising.

Table 2. Results for UBO test sets: forward and backward scheduling

	p_{inf}^{pmtn}	p_{inf}^{npmtn}	Δ_{MILP}	Δ_{npmtn}	p_{opt}	p_{feas}	p_{nfeas}	p_{nfnd}	p_{fnd}	p_{imp}	t_{cpu}	$\#\,it$
$n = 10$	14.44	18.89	0.17	−2.04	52.22	33.33	14.44	0.00	4.44	31.11	0.14	1662
$n = 20$	10.00	22.22	−1.08	−2.66	26.67	62.22	11.11	0.00	11.11	46.67	0.34	1715
$n = 50$	12.22	18.89	N/A	−2.12	23.33	60.00	16.67	0.00	2.22	52.22	3.37	1732
$n = 100$	5.56	13.33	N/A	−1.41	16.67	70.00	13.33	1.11	1.11	57.78	25.16	1835

References

Baptiste P., J. Carlier, A. Kononov, M. Queyranne, S. Sevastyanov, and M. Sviridenko, 2011, "Properties of optimal schedules in preemptive shop scheduling". *Discrete Appl Math*, Vol. 159, pp. 272–280.

Damay, J., A. Quilliot, and E. Sanlaville, 2007, "Linear programming based algorithms for preemptive and non-preemptive RCPSP". *Eur J Oper Res*, Vol. 182, pp. 1012–1022.

Franck, B., K. Neumann, and C. Schwindt, 2001, "Truncated branch-and-bound, schedule-construction, and schedule-improvement procedures for resource-constrained project scheduling", *OR Spektrum*, Vol. 23, pp. 297–324.

Hanzálek Z., P. Šůcha, 2009, "Time Symmetry of Project Scheduling with Time Windows and Take-give Resources", In: *4th Multidisciplinary International Scheduling Conference: Theory and Applications*, pp. 239–253.

Schwindt C., T. Paetz, 2015, "Continuous Preemption Problems", Chapter 13, In: *Handbook on Project Management and Scheduling Vol. 1*, C. Schwindt, J. Zimmermann, Springer, Cham, Heidelberg, New York, Dordrecht, London, pp. 251–295.