An *any-order* SGS for project scheduling with scarce resources and precedence constraints

Cyril Briand and Jérome Bezanger
LAAS-CNRS, 7 avenue du Colonel Roche, 31077 Toulouse, France
{briand, jbezange}@laas.fr

1 Introduction

This paper describes a new *Solution Generation Scheme* (SGS) for project scheduling problems with resource and precedence constraints. That problem class is denoted as $PS|prec|C_{\max}$ in the literature (according to Brucker's notation [3]). A set of n interrelated activities has to be processed in order to achieve the project. Precedence constraints stipulate that an activity can not start before all its predecessors have finished. In order to perform an activity, several limited-capacity resources are used. During the processing window $[s_j, s_j + p_j]$ of an activity j (s_j being the starting time of the activity and p_j its duration), $r_{j,k}$ resource units of resource k are used. A schedule is resource feasible if, at any time t, for any resource k, the sum $\sum r_{j,k}$ of the jobs j in progress at time t never exceeds the maximum resource capacity R_k. A schedule is said *feasible* if it is both precedence-feasible (i.e. it satisfies the precedence constraints) and resource-feasible. The objective is to find a feasible schedule that minimizes the total project duration.

That optimisation problem is known as NP-hard [4]. Exact solution procedures have been proposed which are quite efficient on medium-sized problem instances. A large collection of heuristics (X-pass methods, Genetic Algorithms, Tabu Search, ...) is also available to find optimal or near-optimal solutions for larger problem instances within a reasonable amount of time. For a description and a comparison of those heuristics, it is recommended to refer to the Hartmann and Kolisch's survey [5], as well as to its recent update [6].

A frequent ingredient used inside most heuristics is a SGS. Combined with a priority rule, a SGS determines for a given problem a single near-optimal feasible solution. Since a SGS can be called very often, it has to run very fast in particular on large problem instances. Two classic SGSs are mainly referred in the literature : the *serial-SGS* and the *parallel-SGS*.

A serial-SGS consists of n iterations. In each iteration, an activity is selected according to its priority and inserted inside a partial schedule at the earliest (respecting the precedence and resource constraints), while keeping unchanged the starting time of the already scheduled activities. Only an *eligible* activity can be selected at each iteration. An activity is eligible if all its predecessors have already been scheduled. The activity priorities are determined according to a given rule (minimum Latest Starting Time, Earliest Starting Time, . . .).

On the other hand, a parallel-SGS is time oriented and requires, at most, n iterations. In each iteration, several eligible activities can be scheduled. A time t_k is associated with one iteration k : t_k equals the minimum finishing time of the activities scheduled during the $k-1$ previous stages (provided that $t_k > t_{k-1}$). The activities having their earliest starting time equals t_k are scheduled at time t_k, one by one, with respect of the priority order. If starting activity i at time t_k avoids the possibility to start activity j at the same time, then j is considered later, in further iterations.

A serial-SGS produces active schedules (no activity can be started earlier without scheduling another activity later), the class of the active schedules being dominant with regards to the $C_{\max}$ criterion. In contrast, a parallel-SGS produces no-delay schedules, that class being not dominant. The time complexity of both SGSs is $O(mn^2)$, m being the number of resources and n the number of activities.

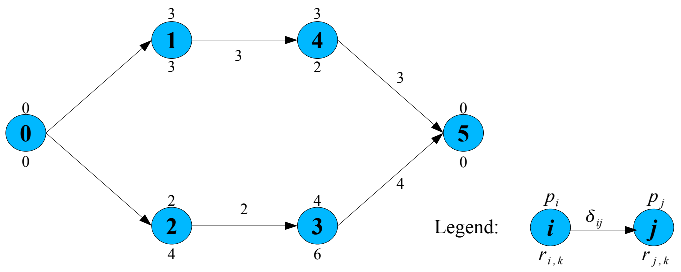
2 Motivations

A brief analysis of the parallel and serial SGSs brings the following remark. The solution space that the SGSs potentially allow to explore is restricted because, while inserting an activity, both SGSs do not allow to right-shift any activity that is already scheduled. As a consequence, the considered activity is often inserted at the end of the partial schedule, after all the other activities. Therefore, if the selection order of the activities is not optimal (in the sense it is not compatible with any optimal schedule), the probability for the produced schedule to be optimal becomes thinner.

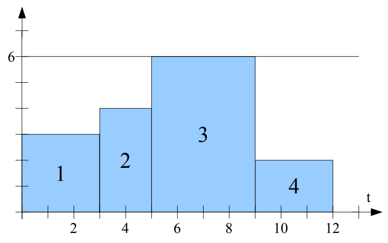
This is illustrated below on the example of Figure 1.a-b). The activities $1 \ldots 4$ have to be processed on a single cumulative resource of capacity $R = 6$. Activities 0 and 5 are dummy and represent the origin of time and the end of the project respectively. Figure 1.c) shows the schedule that is obtained using a serial-SGS and a LST priority rule. The selection order is $1 \prec 2 \prec 3 \prec 4$ which is not compatible with the optimal sequence represented on Figure 1.d) since Activity 3 cannot be delayed.

Activities	0	1	2	3	4	5
p_i	0	3	2	4	3	0
r_i	0	3	4	6	2	0
lst_i	0	0	0	2	3	6

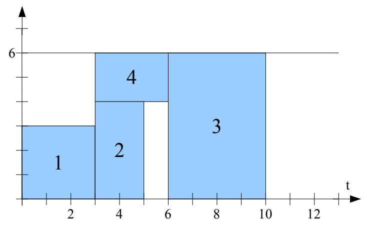
a) Problem data



b) Precedence graph



c) Solution using a serial-SGS



d) An optimal schedule

Figure 1: A problem and two feasible schedules

In accordance with the previous remark, an alternative SGS is proposed below. The activities are considered any order (hence the name *any-order-SGS*). As shown in the next section, in that scheme, the eligibility of an activity is never considered. Moreover, the insertion of an activity can delay some other ones.

3 Description of any-order-SGS

The basic component of the any-order-SGS is the insertion procedure proposed by C. Artigues [1, 2]. This procedure aims at solving the problem of inserting an activity inside a partial schedule. In that procedure, a transportation network is associated with the partial schedule.

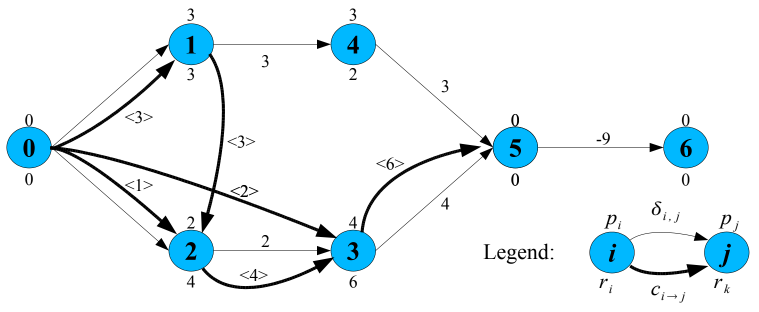


Figure 2: Graph G associated with a partial schedule

An arc of that network, linking the activities i and j, is associated with a resource flow with a capacity vector $(c_{1,i\to j}, \dots, c_{m,i\to j})$ (where $c_{k,i\to j}$ is the number of resource units k which are transfered from i to j). The insertion procedure enumerates a finite set of maximal insertion cuts (a maximal cut corresponds to a set of arcs having the sum of their capacity vectors equals $(R_1, \dots, R_m)$). Each maximal cut characterises a set of valid cut regarding the insertion of a given activity j (a valid cut corresponds to a set of arcs having the sum of their capacity vectors equals $(r_{j,1}, \dots, r_{j,m})$). Given an activity j to insert and its execution window $[est_j, lft_j]$, the insertion procedure visits all the valid cuts and selects an optimal one according to the minimisation of the makespan increase. The time complexity of this procedure is $O(nKm^2)$ where n and m are the number of activities and resources, K is the maximum resource capacity.

For the algorithm, a graph $G =< V, U_\prec, U_r >$, representing a current partial schedule, is supposed to be available. An arc u between two vertices i and j of G corresponds either to a transportation arc associated with a capacity vector (i.e. $a \in U_R$) or to a potential constraint of length $\delta_{i,j} = p_i$ (i.e. $a \in U_\prec$). Each vertex $i \in V = \{0, \dots, n+1, n+2\}$ of G corresponds to a project activity. Classically, the dummy activities 0 and $n+1$ correspond to the beginning and the end of the project respectively. The additional dummy activity $n+2$ allows to quantify the increase of the $C_{\max}$ value after the insertion. There is a potential arc between $n+1$ and $n+2$ having the length $\delta_{n+1,n+2} = -est_{n+1}$ (this length is updated after each insertion if est_{n+1} has changed).

For instance, Figure 2 shows the graph G associated with the schedules of Figure 1.c-d), just before the insertion of Activity 4. The arcs of U_R are in bold.

The any-order-SGS is described by the algorithm below. At the beginning of the algorithm, U_R does not hold any arc, excepted the one between 0 and $n+1$ having the maximum capacity $(R_1, \dots, R_m)$. The arcs of $U_\prec$ all correspond to precedence constraints (excepted the arc between $n+1$ and $n+2$). The earliest starting time and the latest starting time of the activities are supposed to be known.

Algorithm 1 AO_SGS(G)

```
$\varepsilon \leftarrow \{1...n\}$;
for i ← 1 to $n$ do
    $j \leftarrow$ Select_Activity($\varepsilon, r$);
    $C^* \leftarrow$ Search_Optimal_Insertion_Cut($j, G$);
    Insert($j, C^*$,G);
    Forward_Backward_Propagation($G$);
    $\varepsilon \leftarrow \varepsilon - \{j\}$;
end for
```

In each iteration, an activity $j \in \varepsilon$ is selected according to the priority rule r. Then an optimal insertion cut C^* is determined for the insertion of j in U_R. In the next step, the insertion is performed leading to an update of U_R. The new earliest and latest starting time are computed. The algorithm stops after n iterations when all the activities have been scheduled. Its time complexity is $O(Kn^2m^2)$.

4 Experiments

The any-order-SGS has been tested on the problem instances of the PSPLIB library (480 instances with 30 activities - 480 instances with 60 activities and 600 instances with 120 activities). For each problem class, 11 priority rules have been used. For each problem instance and each priority rule, the results obtained using the any-order-SGS have been compared with those obtained by using the serial-SGS. The results are synthesised in Table 1. For each problem class, we count the number of problem instances for which the best solution found by the any-order-SGS (among the 11 ones which were generated) is better or equivalent to the best solution found with the serial-SGS (also among the 11 ones).

	30 activities			60 activities			120 activities		
Any-Order SGS is	Better	Equivalent	Worst	Better	Equivalent	Worst	Better	Equivalent	Worst
Percentage	33.54	61.25	5.21	31.04	61.46	7.5	50.33	19.83	29.83

Table 1: Comparison between the serial-SGS and the any-order SGS

The previous results are still temporary since some improvements concerning the *insert* procedure are currently in progress. Nevertheless, we think that they are significant enough to prove the interest of the any-order-SGS. A comparison with the parallel-SGS will be proposed during the workshop.

References

[1] Artigues, C., & Roubellat, F., A polynomial activity insertion algorithm in a multi-resource schedule with cumulative constraints and multiple mode, 2000, European Journal of Operational Research, 127, 297-316.

[2] Artigues, C., Michelon, P. & Reusser, S., Insertion Techniques for static and dynamic resource-constrained project scheduling, 2003, European Journal of Operational Research, 149, 249-267.

[3] Brucker, P., Drexl, A., Möhring, R.H., Neumann, K., & Pesh, E. Resource-constrained project scheduling: Notation, classification, models and methods, 1999, European Journal of Operational Research, 112, 3-41.

[4] Blazewicz, J., Lenstra, J., Rinnooy kan A., Scheduling subject to resource constraints: Classification and complexity, 1983, Discrete Applied Mathematics, 5, 11-24.

[5] Hartmann, S., & Kolisch, R., Experimental evaluation of the state-of-the-art heuristics for the resource-constrained project scheduling problem, 2000, European Journal of Operational Research, 127, 394-407.

[6] Kolisch, R., & Hartmann, S., Experimental Investigation of Heuristics for Resource-Constrained Project Scheduling: An Update, 2005, *to appear*, European Journal of Operational Research.