Memory and feasibility indicators in GRASP for Multi-Skill Project Scheduling with Partial Preemption

Oliver Polo Mejia, Christian Artigues, Pierre Lopez, Lars Mönch

Memory and feasibility indicators in GRASP for Multi-Skill Project Scheduling with Partial Preemption

Oliver Polo-Mejía[1,2], Christian Artigues[2], Pierre Lopez[2], Lars Mönch[3]

[1] CEA, DEN, DEC, SETC
St Paul lez Durance, France
oliver.polomejia@cea.fr

[2] LAAS-CNRS,Université de Toulouse, CNRS
Toulouse, France

[3] Department of Mathematics and Computer Science, University of Hagen
Hagen, Germany

Abstract

This paper describes a GRASP algorithm aiming to solve a new scheduling problem known as the Multi-Skill Project Scheduling Problem with Partial Preemption, in which not all resources are released during preemption periods. We use a self-adaptive strategy for fixing the cardinality of the restricted candidate list in the greedy phase of the GRASP. We also propose an adaptive evaluation function that includes memory-based intensification, exploiting the characteristics of the best solutions, and a feasibility element for increasing the number of feasible solutions visited. Numerical experiments show the interest of the proposed approach.

1 Introduction

We propose a variant of the Multi-Skill Project Scheduling Problem (MSPSP), aiming to represent the real scheduling process of activities within a nuclear research laboratory: the MSPSP with Partial Preemption (MSPSP-PP). The MSPSP is a generalisation of the well-known Resource-Constrained Project Scheduling Problem, where resources are characterised by the skills they master, and non-preemptive tasks require a certain amount of resources with a specific skill. Determining a solution consists in computing the periods in which each activity is executed and also which resources are assigned to the activity at each period, while satisfying activity and resource constraints: a resource can execute only those skills it masters and must cover only one skill per activity.

The main characteristic of the variant we consider, the MSPSP-PP, is that we allow activities preemption, but we handle the resource release during preemption periods in an innovative way: not all resources can be released, and the possibility of releasing them depends on the characteristics of the activity. We can classify activities within three types: *Non-preemptive*, for which no resource can be released; *partially-preemptive*, for which a subset of resources can be released; and *preemptive*, for which all resources can be released. These activities are scheduled over renewable resources with limited capacity; they can be cumulative mono-skilled resources (compound machines) or disjunctive multi-skilled resources (technicians) mastering Nb_j skills. Multi-skilled resources can respond to more than one skill requirement per activity and may partially execute it (except for non-preemptive activities where technicians must perform the whole activity). An activity is defined by its duration (d_i), its precedence relationships, its resource requirements k ($Br_{i,k}$), its skill requirements c ($b_{i,c}$) and the subset of preemptive resources. Also, activities might or not have either a deadline (dl_i) or a release date (r_i). The MSPSP-PP is NP-hard, and the use of exact methods can be prohibitive for large-sized industrial instances. That is why, in the remainder of this paper, we describe a Greedy Randomized Adaptive Search Procedure (GRASP) aiming to get near-optimal solutions in a reasonable amount of time.

2 GRASP for the MSPSP-PP

A GRASP is an iterative multi-start algorithm, in which each iteration consists of two phases: generation and local search. In the generation phase, a feasible solution is generated; then its neighbourhood is

explored by the local search algorithm. The best solution found over all GRASP iterations is kept as the results. In the following of this section, we describe a GRASP algorithm for the MSPSP-PP.

2.1 Generation and local search phase

In each greedy iteration, using a serial generation scheme, we select an activity randomly from the restricted candidate list (RCL), we then identify the earliest periods when this activity can be scheduled (according to its preemption type and resources and technicians availability), and we finally allocate the technicians to the activity using a Minimum-Cost Maximum-Flow (MCMF) problem.

Let $F(A_i)$ be an adaptive evaluation function, which indicates the degree of relevance of planning activity A_i in the current greedy iteration. Let also define M as the number of not yet scheduled activities after the current greedy iteration. The RCL is made up of the $1 + \alpha * M$, $\alpha \in [0, 1]$, activities having the best F values. Each element within the RCL has a probability of being chosen (π_i) defined as follows:

$$\pi_i = \frac{F(A_i)}{\sum_{j \in RCL} F(A_j)} \,. \tag{1}$$

For choosing the value of α, we propose to use the reactive strategy proposed by Praias and Ribeiro [2], where the value of α is randomly selected from a discrete set $\Psi = \{\alpha_1, ..., \alpha_n\}$ of possible α values. The probabilities associated with the choice of each value are all initially uniformly distributed. After a few iterations, they are periodically reevaluated taking into consideration the quality of the obtained solution for each $\alpha_k \in \Psi$.

The allocation problem is formulated as a MCMF problem over a bipartite graph $G_i = (X, F)$, $X = S \cup P$, where S represents the set of skills required by activity A_i and P is the subset of available technicians who master at least one of the skills within S. In this graph, there is an edge between the source vertex and each vertex $S_c \in S$ whose maximum capacity is equal to $b_{i,c}$ (need of the skill c for executing the activity A_i). There is also an edge between a vertex S_c and a vertex $P_j \in P$, iff the technician P_j masters the skill S_c. The maximum capacity of this arc is fixed to 1 since a technician can only respond to one unit of need per skill. Similarly, there is an edge between each vertex P_j and the sink of the graph, with a maximum capacity equal to the number of skills mastered by the technician P_j (Nb_j). We associate a cost ($CT_{i,j}$) that is related to the criticality of the technician P_j for not yet scheduled activities. To determine the technicians to allocate to the activity, we solve the MCMF problem, and we look at the vertices $P_j \in P$ through which the flow passes.

2.2 Local Search

To explore the neighbourhood, we use an incomplete binary search tree partially inspired on the "Limited Discrepancy Search". For each sequence used in the generation phase, there is a big amount of possible schedules that are defined by the technician allocations we made. The objective of the algorithm is to visit some of these possible schedules. For developing the search tree, we use the sequence obtained in the generation phase within a serial greedy algorithm (similar to the one used in the generation phase, but forced always to follow a given sequence). Every time we must effectuate a technician allocation, we generate a node having in the left-hand branch the best allocation we get solving the MCMF, while in the right-hand branch we have the second best allocation (if such a solution exists). Visiting the whole binary tree can be still prohibitive for industrial instances. From the way schedules are generated, we can expect that heuristic?s chance of making poor decisions decreases as we add more activities to the partial schedule (going deep in the search tree). We exploit this characteristic by giving to each node a probability, that decreases with its depth in the tree, to examine the right branch; reducing the number of explored branches. Moreover, we limit the maximal number of "discrepancies" (number of times we choose the second best solution for the MCMF) of the branch we visit.

2.3 Adaptive greedy evaluation function

The proposed adaptive greedy evaluation function has three components: priority rule ($L(A_i)$), intensification ($I(A_i)$) and feasibility ($G(A_i)$); and it is defined as follows:

$$F(A_i) = \beta * L(A_i) + \delta * I(A_i) + \gamma * G(A_i) \ . \tag{2}$$

Priority due to priority rule ($L(A_i)$): Computational experiments, presented in [1], suggest that using priority rules "Most Successors" and "Greatest Rank" provide smaller optimality gaps. Let Sc_i be the set of successors of activity A_i; we can define the priority function as follows:

$$L(A_i) = d_i + \sum_{j \in Sc_i} d_j \ . \tag{3}$$

Intensification component ($I(A_i)$): The idea is to use the characteristics of a set ε of q elite solutions to influence the construction phase. In our algorithm, the quality of the solution is highly dependent on the order (Seq_k) in which activities have been treated to obtain solution k. Let define $Bef(k, i, l)$ as a binary function taking the value of 1 is activity i was treated before activity l in the Seq_k, 0 otherwise. Let L be the set of not yet scheduled activities. The intensification component is defined as follows:

$$I(A_i) = \sum_{k \in \varepsilon} \sum_{l \in L} Bef(k, i, l) \ . \tag{4}$$

Feasibility factor ($G(A_i)$): Time windows makes it difficult to find feasible solutions with the greedy algorithm. We introduce a component giving priority to activities with a short slack time. Slack time ($Slack_i$) refers to the margin that an activity A_i has in its planning window. It is a function of the deadline (dl_i), the earliest start time ($\underset{\sim}{r_i}$), and the activity duration. The feasibility factor is defined as follows:

$$G(A_i) = \frac{1}{dl_i - \underset{\sim}{r_i} - d_i} \ . \tag{5}$$

Note that $L(A_i)$, $I(A_i)$ and $G(A_i)$ must be normalised before taking the weighted sum. Moreover, we have β, δ, $\gamma \in [0, 1]$ and $\beta + \delta + \gamma = 1$. Parameters δ and γ are self-adaptive, and their values are periodically updated. If after K iterations the number of infeasible solutions increases, we must increase the value of γ; on the contrary, if this number decreases, we decrease γ. On the other hand, the parameter δ decreases when the diversity of obtained solutions is too low and increases when the variability is high.

3 Results and conclusions

The GRASP algorithm was tested over a set of 200 instances. The instances have an average makespan of 80 time units, 30 activities with a duration between 5 to 15 time units, up to 5 skills per activity, 8 cumulative resources, 8 technicians (multi-skilled resources). 20% of the activities have release date and deadline (all other characteristics are random). The algorithm allows to obtaining an average gap to optimality of 2.12% within an average time of 35.5 seconds with the size of the elite solution set equal to 20, and the maximal number of GRASP iterations with feasible solutions equal to 550. To analyse the impact of the intensification component, we tested a version of the algorithm without this factor with the same stopping criteria. After a statistical test, we cannot prove that the intensification component significantly improved the average optimality gap (2.16% for the version without intensification). However, the time required by the algorithm is lower when the intensification component is used (44.87 sec if not used). This is because the intensification factors also help the algorithm to generate less unfeasible solutions.

References

[1] Oliver Polo-Mejía, Christian Artigues, and Pierre Lopez. A heuristic method for the multi-skill project scheduling problem with partial preemption. In *8th International Conference on Operations Research and Enterprise Systems*, pages 111–120, Prague, Czech Republic, February 2019.

[2] Marcelo Prais and Celso C. Ribeiro. Reactive GRASP: An application to a matrix decomposition problem in TDMA traffic assignment. *INFORMS Journal on Computing*, 12(3):164–176, 2000.