10th International Conference of the Practice and Theory of Automated Timetabling
PATAT 2014, 26-29 August 2014, York, United Kingdom

# Hybrid Local Search for The Multi-Mode Resource-Constrained Multi-Project Scheduling Problem

**Haroldo Gambini Santos · Janniele Soares · Tùlio Toffolo**

**Abstract** In this work we present a multi-neighborhood, parallel local search approach for the Multi-Mode Resource-Constrained Multi-Project Scheduling Problem (MMRCMPSP). The search in multiple neighborhoods is conducted in parallel with dynamic load balancing among processors. Our solver works with an indirect solution representation and navigates through space of the feasible solutions by combining heuristics and mathematical programming. A perturbation procedure which alters a subset of job modes and still keeps the solution feasible is introduced. Very encouraging results were obtained, improving several best known solutions published at the MISTA Challenge.

**Keywords** Multi-Mode Resource-Constrained Multi-Project Scheduling Problem · Scheduling · Local Search

## 1 Introduction

A *Project Scheduling Problem* (PSP), in its general form, consists in scheduling the processing times of *jobs* (or activities) contained in a project. These jobs are interrelated by precedence constraints, that is, a job may require another job to be finished before its start. This class of problems models many situations of practical interest in engineering and management sciences in general.

Santos, H.G.
Computer Science Department, Federal University of Ouro Preto, Brazil
E-mail: haroldo@iceb.ufop.br

Soares, J.A.
Department of Computing and Information Systems, Federal University of Ouro Preto, Brazil
E-mail: janniele@decsi.ufop.br

Toffolo, T.A.M.
Computer Science Department, Federal University of Ouro Preto, Brazil
E-mail: tulio@toffolo.com.br

One well known application of the PSP is the area of Construction Scheduling [4]. For a broad review of this area we refer the reader to [5, 8, 10, 13, 14] and [2].

Recently, with the objective of bridging the gap between theory and practice, the MISTA 2013 [1] challenge was organized. In this challenge, a generalization of the PSP with resource constraints which takes into account several aspects of real world applications was considered: The Multi-Mode Resource-Constrained Multi-Project Scheduling Problem (MMRCMPSP). Several instances with different characteristics and sizes were proposed. Competing methods were evaluated in a controlled experimental environment. We participated as part of the GOAL team, which proposed an Integer Programming based approach for the MMRCMPSP. Our team was ranked third in this competition.

In this paper we explore the hybridization of Integer Programming and local search heuristics, incorporating some of the best characteristics of our method with some of the efficient local search procedures proposed by other teams.

The paper is organized as follows: Section 2 presents the MMRCMPSP; Section 3 introduces our algorithm and finally, Sections 4 and 5 present, respectively, conclusions and future works.

## 2 The Multi-Mode Resource-Constrained Multi-Project Scheduling Problem

The MMRCMPSP is stated as follows: a set $P$ of projects, each project $p \in P$ consisting of a set $J_p = \{1, \ldots, |J_p|\}$ of non-preemptive jobs, has to be scheduled. Each project $p$ has also a *release time*, that is, a time when its jobs processing may be started. The start and end of a project are delimited by *dummy* jobs 0 and $|J_p| + 1$, respectively the first and last jobs of each project.

To schedule a project means to determine the starting time of all of its jobs, subject to the precedence constraints among them and also their *resource* consumption in face of the available resources. Jobs may consume *local* resources – exclusive resources of a project – and *global resources* – resources shared among all projects. These resources can be either *renewable* – with capacity fixed per time unit during the project duration – or *non-renewable* – with capacity fixed per project duration. Each job may be executed in one or more *execution modes*, each requiring a specific amount of resources consumption and resulting in different durations for a job completion. Note that dummy jobs do not have any resource consumption and their duration is always zero.

A lower bound on a project earliest finish time is the *critical path duration*. The *Critical Path Method* [12] is a tool for general project management that represents the precedence constraints as a network, where each job is a node and arcs connect jobs to its successors and predecessors, and calculates the earliest and latest start and finish times for each job such that the project is not delayed, while observing the precedence constraints. The critical path

itself is the sequence of related jobs that cannot be delayed without delaying the whole project, denoted by a path between the two dummy jobs in the network. Thus, the critical path duration is the sum of these jobs durations. To compute a valid critical path based bound for a MMRCMPSP instance one can set job durations to the minimum among all possible execution modes.

Once a project is scheduled, its *makespan* is defined as the difference between the project finish and release times, and the *project delay*, is defined as the difference between the critical path based bound and the actual project duration.

In order to measure the quality of the solutions submitted to the MISTA challenge, an objective function with two components was proposed: to minimize the *total project delay* (TPD) and the *total makespan* (TMS). The TPD is defined as the sum of all projects delays, and the TMS is defined as the time required to finish all projects, i.e., the difference between the maximum finish time of a project and the minimum release time of a project. TPD is the main objective, while TMS is a tie-breaker.

## 3 The Proposed Algorithm : Overall Working

One fundamental characteristic of our method is that it always navigates in the feasible search space of solutions. To accomplish this we employ an indirect solution representation abstracting starting times of tasks: valid topological orderings are decoded by a constructive algorithm which allocates each task in the sequence as soon as possible, this approach was also used by [3, 6].

The selection of an initial set of valid execution modes for tasks is also computationally challenging: [3, 6] relax this constraint and move it to the objective function. In our approach, the definition of an initial set of modes is carried out by the solution of a binary programming model.

Thus, our solution representation consists in an ordered pair $(\pi, \mathbb{M})$ where $\pi$ is valid topological sorting of $J$ and $\mathbb{M}$ is a valid set of modes. Since the latter is much harder to determine, the next subsection will describe our approach for this part.

Subsequently, local search is performed in a Variable Neighborhood Descent [7] fashion, iterated with a perturbation of modes. One novel feature of our algorithm is a controlled perturbation in the set of modes, which also involves a binary programming model.

### 3.1 Initial Feasible Solution

The initial set of selected modes must respect resources constraints. Since the selection of processing modes also determines the duration of tasks, one greedy strategy is to prioritize the selection of fast processing modes. Considering this latter criterion, it is important to observe that it does not guarantees a smaller TPD, since renewable resources constraints can increase the starting times of

tasks. Considering non-renewable resources, a greedy strategy can have worse consequences: many combinations of modes may not respect the usage of these resources.

The binary program to select this initial set of modes considers: $J$ jobs with respective processing times $p_{jm}$ and $N$ non renewable resources. Each job has a set $M_j$ of possible modes and the non-renewable resource $n$ consumption of job $j$ in mode $m$ is denoted as $r_{jmn}$. Thus, the following binary program is solved to select which mode $m$ job $j$ will be allocated, considering its respective decision variables $x_{jm}$ and resource availability $q_n$ for each non renewable resource:

$$min.: \quad \sum_{j \in J} p_{jm}.x_{jm} \tag{1}$$

$$s.t.: \quad \sum_{j \in J} \sum_{m \in M_j} r_{jmn} x_{jm} \leq q_n \ \ \forall n \in N \tag{2}$$

$$x_{jm} \in \{0,1\} \ \ \forall j \in J, m \in M_j \tag{3}$$

Which corresponds to the NP-Hard problem of the 0-1 multidimensional knapsack problem. Fortunately, modern integer programming solvers [9, 11] consistently solve this problem to the optimality considering all instances of the competition, always in a fraction of a second.

## 3.2 Neighborhoods

In this section we present the neighborhoods used in our search methods. Most of them were proposed or were inspired by the works which won the first two places of the MISTA 2013 challenge [3, 6].

These neighborhoods can be classified in two types: the ones which change modes and the ones which change sequence. The search only operates in the feasible search space: neighbors which disrespect non-renewable resources constraints are ignored and neighbors which disrespect processing dependencies are repaired using a topological order constructed using the invalid order to define priorities.

### *3.2.1 Change One Mode (COM)*

This first neighborhood is based on [3, 6], and receives as a parameter a vector of modes initially allocated to jobs. Since we concentrate in the feasible search space this is a quite restricted neighborhood, since many single mode changes do not produce feasible solutions.

*3.2.2 Change Two Modes (CTM)*

Similar to the movement change one mode, but explores changes on pairs of modes. This is a significantly larger neighborhood since it has a quadratic size with respect to its input and because valid two mode changes are more common.

*3.2.3 Change Three Modes (CTRM)*

We observed that an unrestricted search considering all valid change mode triples would be too expensive. At the same time, jobs which are closer in the dependency graph tend to be much more sensible to changes in the mode of their neighbors than changes in modes of distant jobs. So we restricted the involved jobs $j_1, j_2$ an $j_3$ to be consecutive in the dependency graph: $(j_1 \rightarrow j_2 \rightarrow j_3)$.

*3.2.4 Change Four Modes (CFM)*

Just like three mode change this neighborhood restricts neighbors by imposing the same dependency relationships in the dependency graph.

*3.2.5 Invert Subsequence (INV)*

This movement is based on [6], and receives as a parameters a sequence of jobs and an integer $k$ that determines the size of the subsequences to be inverted. For each position $i$ of the sequence the next $k-1$ jobs are included in a subsequence and inverted.

Figure 1 shows an example of this movement using subsequence with size $k = 4$ and starting position 2.

$s'$=INV($s$,2,4)

| $s$ | 2 | 3 | 5 | 8 | 14 | 10 | 7 | 12 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| $s'$ | 2 | 14 | 8 | 5 | 3 | 10 | 7 | 12 | 9 |

**Fig. 1** Invert Subsequence

*3.2.6 Shift Jobs (SJ)*

This movement was proposed by [3], and is responsible for systematically moving forward or backward a job in the sequence in $k$ neighbor positions. The method receives as a parameter a sequence of jobs and a parameter $k$ that determines the maximum distance which job $j$ can be displaced.

Figure 2 shows an example of this movement, shifting the job allocated at position 4 3 positions ahead.

$s'$=SJ($s$,4,3)

| $s$ | 2 | 3 | 5 | 8 | 14 | 10 | 7 | 12 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| $s'$ | 2 | 3 | 5 | 14 | 10 | 7 | 8 | 12 | 9 |

**Fig. 2** Shift Jobs

### *3.2.7 Swap Jobs (SWJ)*

In this neighborhood, which it was based on [3, 6], two jobs in the sequence are swapped. This method receives as a parameter a sequence of jobs and a parameter $k$ which restricts the maximum distance between the two jobs to be swapped.

Figure 3 shows an example of this movement, swapping the job at position 5 with the job currently occupying position 5.

$s'$=SWJ($s$,3,5)

| $s$ | 2 | 3 | 5 | 8 | 14 | 10 | 7 | 12 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| $s'$ | 2 | 3 | 14 | 8 | 5 | 10 | 7 | 12 | 9 |

**Fig. 3** Swap Jobs

### *3.2.8 Compact Project (CP)*

This movement is based on the proposal of [3] and tries to accelerate the completion time of a project by shifting tasks of other projects which appear in the mid the project sequence for later processing. The objective is to *shrink* the later portion of the project.

In our implementation a parameter $perc \in (0, 1]$ determines percentage of tasks which will be compressed. These tasks are selected starting from the end of project, i.e: $perc = 0.5$ means that the second half of the tasks in the project sequence will be compressed.

Figure 4 shows an example of this movement, compacting 100% the jobs of project $p_1$, which is shown in gray.

$s'$=CP($s$,1,1)

| $s$ | 2 | 3 | 5 | 8 | 14 | 10 | 7 | 12 | 9 |
|---|---|---|---|---|---|---|---|---|---|
| $s'$ | 2 | 3 | 5 | 14 | 10 | 9 | 8 | 7 | 12 |

**Fig. 4** Compact projects

### *3.2.9 Shift Project (SP)*

This movement is based on the proposed by [3] and is similar to the shift jobs, but moves forward or backward all jobs of a project $p$ on $k$ positions. The movement receives as parameters a maximum shifting distance $k$ and a parameter $p$ which determines the involved.

Figure 5 shows an example of this movement to the project $p_1$ (shown in gray) and $k = -2$.

$s'$=SP($s$,1,-2)

| $s$ | 1 | 4 | 3 | 2 | 9 | 7 | 5 | 8 | 6 |
|---|---|---|---|---|---|---|---|---|---|
| $s'$ | 1 | 2 | 4 | 7 | 5 | 3 | 9 | 8 | 6 |

**Fig. 5** Shift a project

### *3.2.10 Swap Two Projects (SWP)*

This movement it is based on the proposals of [3, 6], and is similar to the idea of swapping jobs, but now the swap happens between two projects.

Considering the swap of two projects, $p_1$ and $p_2$, a new subsequence is generated as follows: firstly, one identifies the starting position $sp$ of the project which finishes earlier.

In a vector $R$ are stored all jobs before $sp$ that do not belong to projects $p_1$ or $p_2$. In a vector $D$ are stored all jobs after $sp$ that do not belong to any of these projects too.

The reconstruction of the sequence is made by allocating all jobs of the vector $R$. Subsequently, all jobs that belong to the project that finished later, followed by all jobs that belong to the project that finished earlier and finally all the jobs of the vector $D$.

Figure 6 shows an example of this movement, swapping the projects $p_1$, dark gray, and $p_2$, in light gray.

$s'$=SWP($s$,1,2)

| $s$ | 5 | 3 | 9 | 6 | 13 | 1 | 7 | 2 | 25 |
|---|---|---|---|---|---|---|---|---|---|
| $s'$ | 5 | 3 | 6 | 2 | 9 | 13 | 1 | 7 | 25 |

**Fig. 6** Swap two projects

### 3.3 Perturbation

As observed by [6], in the MMRCMPSP, changes in the mode set appear to have a much more profound effect than changes in the sequence. We developed a perturbation strategy where only a controlled number of modes is randomly changed, keeping the resulting mode set feasible. In our method, a parameter %MC determines the percentage of jobs which will have their modes changed at each application of the perturbation procedure.

This procedures solves exactly the same problem as in subsection 3.1, but with a different objective function. The main idea is to change the mode of some jobs while trying to minimize the *collateral effect* induced by the satisfaction of the non-renewable resources constraints. More specifically, the model allows any other task to have its mode changed too, but tries to minimize these cases. As input this method receives the current mode $m_j$ of each job and a boolean value $c_j$ indicating if one wants to perturb the current solution by changing this job mode. Coefficients $p_{jm}$ of the binary programming model are substituted as follows:

$$p_{jm} = \begin{cases} M & \text{if } c_j \text{ is true and } m \text{ is the current mode of job } j \\ & \text{ or if } c_j \text{ is false and } m \text{ is not the current mode of } j \\ \epsilon & \text{otherwise} \end{cases}$$

Where $M$ represents a sufficiently large constant (500 in our experiments) and $\epsilon$ a small one (1 in our experiments). Small random changes with value $r \in \{1, \ldots, 10\}$ are also introduced in $p_{jm}$ to further randomize the procedure.

The perturbation procedure is applied after a complete search on all previously presented neighborhoods is conducted and no improved solution was found.

### 3.4 Paralell Search

Many neighbors presented before contain a large number of solutions. One parallelization strategy would be an static neighborhood decomposition, where a fixed portion of the neighborhood is sent to a processor. This strategy can introduce a severe imbalance of load among processors, since the computing

time to to evaluate different solutions varies significantly, depending on several steps in the decoding of each solution. Thus, we opted for a dynamic load balancing strategy: a pool of neighbors to be evaluated is build and different thread continually request new neighbors to be evaluated until a better solution is found or all neighbors have been processed.

## 4 Computational Experiments

All algorithms were coded in C++ and the binary programming models were solved by CPLEX 12.6. The code was compiled with GCC 4.7.1 using flag -O3. All tests ran on a computer with an Intel Core i7 processor[1] and 24 Gb of RAM, running OpenSUSE Linux 12.1.

The developed method ran in parallel using 4 threads. Parameter values were obtained after some preliminary empirical evaluation and are presented on Table 1. These parameters correspond respectively the limits used in Neighborhoods Invert Subsequence (INV), Shift Jobs (SJ), Swap Jobs (SWJ), Shift Project (SP),the percentage of Compact Project (CP) and the percentage of mode changes (MC) at each perturbation.

**Table 1** Parameters used for tests

| Local Search Parameters | | | | | |
|---|---|---|---|---|---|
| INV | SJ | SWJ | SP | %CP | %MC |
| 3 | 2 | 2 | 5 | 0.5 | 0.02 |

The results of the instance set A, used on the first stage of the competition, were announced during the qualification phase. Results of the instances of the set B and X, on the second and third phase of the competition, were announced during the conference.

Table 2 shows the best results found by the proposed approach, as well the mean and standard deviation, after 10 runs within 300 seconds of runtime. Instances that were better or equal to the results reported in the MISTA 2013 Challenge site are emphasized.

## 5 Conclusions and Future Works

In this work we presented a hybrid search method which combines a multi-neighborhood parallel local search with mathematical programming. Perturbation is executed in a controlled way, so that the search method always jumps from one feasible solution to another.

[1] the same of the MISTA 2013 Challenge

**Table 2** Best and average results after 10 runs of the algorithm sided with best results from MISTA

| Inst. | Best | | Average | | Std.Dev. | | MISTA | | ≤ MISTA? |
|---|---|---|---|---|---|---|---|---|---|
| | TPD | TMS | TPD | TMS | TPD | TMS | TPD | TMS | |
| A-1 | 1 | 23 | 1 | 23 | 0 | 0 | 1 | 23 | equal |
| A-2 | 2 | 41 | 2 | 41 | 0 | 0 | 2 | 41 | equal |
| A-3 | 0 | 50 | 0 | 50 | 0 | 0 | 0 | 50 | equal |
| A-4 | 65 | 42 | 65 | 42 | 0 | 0 | 65 | 42 | equal |
| A-5 | 157 | 107 | 164 | 109 | 4 | 2 | 153 | 105 | |
| A-6 | 153 | 98 | 161 | 102 | 5 | 3 | 147 | 96 | |
| A-7 | 620 | 205 | 630 | 204 | 8 | 4 | 596 | 196 | |
| A-8 | 300 | 160 | 323 | 161 | 6 | 2 | 302 | 155 | yes |
| A-9 | 221 | 130 | 227 | 133 | 8 | 3 | 223 | 119 | yes |
| A-10 | 926 | 324 | 950 | 327 | 18 | 3 | 969 | 314 | yes |
| B-1 | 294 | 118 | 298 | 121 | 4 | 2 | 349 | 127 | yes |
| B-2 | 474 | 177 | 482 | 179 | 9 | 2 | 434 | 160 | |
| B-3 | 573 | 215 | 595 | 218 | 16 | 2 | 545 | 210 | |
| B-4 | 1304 | 290 | 1333 | 291 | 19 | 5 | 1274 | 289 | |
| B-5 | 851 | 256 | 873 | 262 | 13 | 4 | 820 | 254 | |
| B-6 | 977 | 237 | 1035 | 244 | 31 | 4 | 912 | 227 | |
| B-7 | 817 | 237 | 838 | 239 | 18 | 4 | 792 | 228 | |
| B-8 | 3275 | 570 | 3361 | 576 | 74 | 6 | 3176 | 533 | |
| B-9 | 4633 | 812 | 4782 | 825 | 93 | 11 | 4192 | 746 | |
| B-10 | 3208 | 465 | 3284 | 470 | 40 | 4 | 3249 | 456 | yes |
| X-1 | 408 | 147 | 417 | 149 | 6 | 2 | 392 | 142 | |
| X-2 | 370 | 169 | 381 | 170 | 9 | 2 | 349 | 163 | |
| X-3 | 346 | 199 | 354 | 199 | 5 | 3 | 324 | 192 | |
| X-4 | 954 | 216 | 991 | 216 | 26 | 3 | 955 | 213 | yes |
| X-5 | 1858 | 390 | 1883 | 393 | 24 | 4 | 1768 | 374 | |
| X-6 | 779 | 249 | 810 | 254 | 22 | 5 | 719 | 232 | |
| X-7 | 890 | 237 | 893 | 242 | 3 | 4 | 861 | 237 | |
| X-8 | 1310 | 301 | 1369 | 306 | 41 | 5 | 1233 | 283 | |
| X-9 | 3529 | 689 | 3642 | 703 | 64 | 7 | 3268 | 643 | |
| X-10 | 1671 | 395 | 1719 | 404 | 27 | 7 | 1600 | 381 | |

There are several points in or algorithm which could be improved. Firstly, our implementation does not considers yet many optimizations described by competing teams of the MISTA Challenge in the serial schedule generation, this would speed up the entire algorithm. Secondly, the perturbation procedure could be improved to consider the history of the search process. As in some implementations of tabu search, the diversification process could consider some form of long term memory.

Nevertheless, the current implementation already outperformed our previous implementation which relied more on integer programming, showing the the combination of local search with some of our already proposed search methods can be very beneficial.

**Acknowledgements** The authors thank CNPq and FAPEMIG for supporting this research.

## References

1. Wauters, T. Kinable, J. Smet, P. Vancroonenburg, W. Berghe, G.V. and Verstichel, J. : MISTA - Multidisciplinary International Scheduling Conference (2013). URL `http://www.schedulingconference.org/`
2. Artigues, C., Demassey, S., Néron, E.: Resource-Constrained Project Scheduling: Models, Algorithms, Extensions and Applications. Scientific and Technical Publisher. Wiley (2013)
3. Asta, S., Karapetyan, D., Kheiri, A., Ozcan, E., Parkes, A.J.: Combining Monte-Carlo and Hyper-heuristic methods for the Multi-mode Resource-constrained Multi-project Scheduling Problem, technical report. Tech. rep., University of Nottingham, School of Computer Science (2014)
4. Callahan, M.T., Quackenbush, D.G., Rowings, J.E.: Construction Project Scheduling. McGraw-Hill (1992)
5. Demeulemeester, E.L., Herroelen, W.S.: Project Scheduling: A Research Handbook. Kluwer Academic Publishers, Leuven Belgium (2002)
6. Geiger, M.J.: Iterated Variable Neighborhood Search for the resource constrained multi-mode multi-project scheduling problem. In: Proceedings of the 6th Multidisciplinary International Scheduling Conference (MISTA) (2013)
7. Hansen, P., Mladenović, N.: Variable Neighborhood Search. Computers and Operations Research **24**(11), 1097–1100 (1997)
8. Hartmann, S.: A self-adapting genetic algorithm for project scheduling under resource constraints. Naval Research Logistics pp. 433–448 (2002)
9. Johnson, E., Nemhauser, G., Savelsbergh, W.: Progress in Linear Programming-Based Algorithms for Integer Programming: An Exposition. INFORMS Journal on Computing **12** (2000)
10. Józefowska, J., Weglarz, J.: Perspectives in modern project scheduling. International series in operations research & management science. Springer (2006)
11. Jünger, M., Liebling, T., Naddef, D., Nemhauser, G., Pulleyblank, W., Reinelt, G., Rinaldi, G., Wolsey, L.: 50 Years of Integer Programming 1958-2008. Springer (2010)
12. Kelley Jr, J.E., Walker, M.R.: Critical-path planning and scheduling. In: Papers Presented at the December 1-3, 1959, Eastern Joint IRE-AIEE-ACM Computer Conference, IRE-AIEE-ACM '59 (Eastern), pp. 160–173. ACM, New York, NY, USA (1959). DOI 10.1145/1460299.1460318
13. Klein, R.: Scheduling of Resource-Constrained Projects. Operations research/computer science interfaces series. Kluwer Academic (2000)
14. Weglarz, J.: Project Scheduling: Recent Models, Algorithms, and Applications. International series in operations research & management science. Kluwer (1999)