Iterated Variable Neighborhood Search for the resource constrained multi-mode multi-project scheduling problem

Some comments on our contribution to the MISTA 2013 Challenge

Martin Josef Geiger

1 Introduction

The resource constrained multi-mode multi-project scheduling problem (RCMMMPSP) is a notoriously difficult combinatorial optimization problem. For a given set of activities, feasible execution mode assignments and execution starting times must be found such that some optimization function, e.g., the makespan, is optimized. When determining an optimal (or at least feasible) assignment of decision variable values, a set of side constraints, such as resource availabilities, precedence constraints, etc., has to be respected.

In 2013, the MISTA 2013 Challenge stipulated research in the RCMMMPSP. It's goal was the solution of a given set of instances under running time restrictions. We have contributed to this challenge with the here presented approach.

2 Problem description and implications

In the RCMMMPSP of the MISTA 2013 Challenge, several small projects, each of which comprises a set of activities, have to be integrated into a single overall plan. A release date of each project has to be respected, and resources that are jointly used by several projects may exist. Two types of resources exist: renewable and non-renewable. The latter are always specific to each project, i.e., they are never shared among the projects.

The quality of the obtained schedule is evaluated by two objective functions, namely the minimization of the total project delay (TPD), and the total makespan (TMS), which are considered in a lexicographical ordering with TPD over TMS.

For a more detailed description of the RCMMMPSP, we refer to http://allserv.kahosl.be/mista2013challenge/.

Martin Josef Geiger
Helmut-Schmidt-University, University of the Federal Armed Forces Hamburg
E-mail: m.j.geiger@hsu-hh.de
2.1 A remark on the multi-project characteristic

It is possible to combine the set of projects into an overall super-project by relabeling the numbering of the activities. Quite intuitively, this polynomial-time transformation allows for a rather straight-forward representation of alternatives, in which each activity is represented by a unique index $i, i = 1, \ldots, n$ (see section 3.1 below). However, we cannot rule out the possibility that a separate treatment/optimization of the different small sub-projects is not of any potential value.

2.2 A note on the mode assignment

In the general case, i.e. the case of at least two non-renewable resources, the determination of a feasible mode assignment is $\mathcal{NP}$-complete [1]. This means that, depending on the datasets, this sub-aspect of the problem may pose a problem on its own. Any heuristic approach for selecting mode assignments is therefore risky, especially under running time restrictions.

2.3 Two remarks on the objective functions

It can be shown that both objectives are regular functions. It is therefore possible to restrict the computations to active plans only, without excluding the optimum. This is an important observation, that allows the use of a serial scheduling scheme (SSS) [2,3]. In such a scheme, a schedule is constructed by sequentially assigning activities into a schedule, one activity at a time. An earliest-possible-('left-shifted-')scheduling-rule may here be employed for obtaining an active schedule. This implies that an optimization approach can then search the sequence in which the activities are placed in the final schedule, instead of searching the activities' starting times themselves.

Note that, in the general case, a combination of the two criteria into a single evaluation function is not possible. If however we may safely assume that, for a particular set of problem instances (datasets), $\text{TMS} < \alpha$, then a combination of TPD and TMS is possible such that: $F = \alpha \cdot \text{TPD} + \text{TMS}$. In the context of the MISTA 2013 Challenge, the organizers guarantee a value of $\alpha = 100,000$.

3 Solution approach

3.1 Representation of alternatives

An alternative $X$ is associated with two vectors $M = (m_1, \ldots, m_n)$ and $S = (s_1, \ldots, s_n)$. $M$ encodes for each activity $i$ the choice of the execution mode $m_i$, and $S$ is a permutation of the activity indexes. In the following, let $\mathcal{X}$ denote the set of feasible alternatives.

While the information in $M$ is obvious, $S$ can be decoded into a schedule by a SSS. In this transformation, the feasibility of $X$ with respect to $S$ is maintained at all times as we adhere to the precedence constraints of activities while carrying out the procedure. Scheduling conflicts are solved such that priority is given in accordance with the sequence of activities in $S$. Using such a SSS implies that any permutation of activity indexes leads to a feasible schedule.
3.2 Generation of first feasible alternatives

The generation of initial alternatives is twofold:

1. First, a random mode assignment $M$ is computed. If $M$ is infeasible, i.e. by exceeding the capacity of the non-renewable resources, a repair procedure is triggered, which randomly alters values $m_i$ in $M$ until the feasibility of the mode choices is restored. After a maximum number of unsuccessful repair attempts, $M$ is randomly reconstructed from scratch. Note that the repair procedure is guided towards a feasible mode assignment by minimizing the overall excessive use of the non-renewable resources (this assumes a value of 0 for a feasible $M$).

In the light of section 2.2, such a procedure could turn out to be problematic, as it does not guarantee the identification of the feasible mode assignment. It has therefore been tested on all 1,642 resource constrained multi-mode project scheduling datasets which were delivered in the MISTA 2013 Challenge. In conclusion, the procedure repeatedly and consistently managed to create feasible mode vectors for all instances.

2. Second, a scheduling sequence $S$ is constructed. The SSS described above is used here. For the first alternative, scheduling conflicts are solved by giving priority to activities with smaller lower bounds on their possible starting times.

3.3 Neighborhood search

The initially constructed alternative $X$ is then improved by means of local search. The proposed concept is based on the ideas of Variable Neighborhood Search [4] (VNS) and Iterated Local Search [5]. Starting from an initial (feasible) alternative, neighboring alternatives are generated by means of several neighborhood operators and tested for acceptance. Once an alternative is reached that cannot be improved by any operator (a local optimum), a (possibly worsening) perturbation move is performed and search continues from this alternative.

Four different neighborhood operators have been implemented:

1. Exchange (EX), which exchanges the position of two activities in $S$. We denote the set of the neighbors of $X$ by the operator EX as EX($X$). A neighboring solution $X' \in$ EX($X$) is accepted iff $F(X') < F(X)$.

2. Inversion (INV), which inverts a subsequence of activities within $S$. We accept $X' \in$ INV($X$) iff $F(X') < F(X)$.

3. Single mode change (SMC), which changes the mode assignment of a single element in $M$. $X' \in$ SMC($X$) is accepted iff $F(X') < F(X) \land X' \in \mathcal{X}$.

4. Two modes change (TMC), which changes the mode assignments of two elements in $M$. We accept $X' \in$ TMC($X$) iff $F(X') \leq F(X) \land X' \in \mathcal{X}$. Note that this acceptance rule differs from the ones of EX, INV, and SMC: Accepting alternatives of equal quality appears to have a beneficial effect for this operator, possibly due to some diversification effect that results from it.

In case of smaller instances ($n < 307$; this value has been determined by a series of experiments), each neighborhood is repeatedly searched until no improvement can be identified. Then, the next neighborhood operator is taken. For larger instances, this does not appear to be an appropriate strategy. Here, we search each neighborhood,
and then move directly to the next one, thus iterating through the set of neighborhood operators more quickly.

Escaping local optima: The perturbation move consists in randomly changing the execution mode of a single activity. This move is followed by calling the randomized mode-repair-procedure, which, if required, alters several other mode assignments and thus implicitly provides some sort of diversification.

3.4 A note on the parallelization of the approach

In case of smaller instances ($n < 307$), the above presented local search is simply executed in parallel (one master thread, four worker threads). Once a worker thread reaches a local optimum, the currently best found alternative is updated in the master thread and search continues with this best known alternative (perturbation + VNS).

Larger instances come with the difficulty of obtaining a local optimum in the first place. Therefore, all available CPU-cores are concentrated on a single solution by dividing up the investigation of a neighborhood among them.

4 Experiments and results

4.1 Implementation details and hardware

Our algorithm is coded using Microsoft Visual Studio 2012. The program therefore requires a Microsoft “Windows” operating system and a current version of the “.NET Framework” (version 4.5). We have used an Intel X5550 processor, running at 2.67 GHz for the test runs. Despite the availability of eight cores on our machine, only four threads of the local search algorithm are executed in parallel. The target platform for the implemented code optimizations is 64-bit (OS/CPU).

The benchmark program of the 2011 International Timetabling Competition gave, on the organizers machine, which is an Intel Core i7-2600 (3.4 GHz), an execution time of 619 seconds. On our Intel X5550, the very same program reported a running time of 804 seconds (average of 5 trials). We have concluded that the Intel X5550 should be given 29.9% more computing time, resulting in a permitted computing time of 389 seconds for each run. Note that the comparison of the ITC 2011 benchmark program pretty much reflects the different clock speeds of the CPUs.

4.2 Results and execution of the program

Table 1 reports the results of 20 test runs, i.e. the quality of the best found alternatives and the median values. We can observe a considerable deviation of the best value from the median. This indicates that the approach is not consistently solving the datasets each and every time it is executed. While this is typical for a stochastic solver, especially under, for the larger datasets, tight running time restrictions, we suspect that there is still some room for improving the approach.

Reproduction of the reported results should be possible by re-running the program. The command line works as follows:

```
npuScheduler.exe A-1.txt A-1_sol_13264537.txt 300 13264537
```
Table 1 Best results and median values over 20 test runs

<table>
<thead>
<tr>
<th>Dataset</th>
<th>Best results</th>
<th>Median</th>
<th>Best results</th>
<th>Median</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>TPD</td>
<td>TMS</td>
<td>TPD</td>
<td>TMS</td>
</tr>
<tr>
<td>A-1.txt</td>
<td>1</td>
<td>23</td>
<td>1</td>
<td>23</td>
</tr>
<tr>
<td>A-2.txt</td>
<td>2</td>
<td>41</td>
<td>2</td>
<td>41</td>
</tr>
<tr>
<td>A-3.txt</td>
<td>0</td>
<td>50</td>
<td>0</td>
<td>50</td>
</tr>
<tr>
<td>A-4.txt</td>
<td>65</td>
<td>42</td>
<td>65</td>
<td>42</td>
</tr>
<tr>
<td>A-5.txt</td>
<td>153</td>
<td>104</td>
<td>155</td>
<td>106</td>
</tr>
<tr>
<td>A-6.txt</td>
<td>144</td>
<td>94</td>
<td>150</td>
<td>97</td>
</tr>
<tr>
<td>A-7.txt</td>
<td>601</td>
<td>206</td>
<td>620</td>
<td>204</td>
</tr>
<tr>
<td>A-8.txt</td>
<td>319</td>
<td>162</td>
<td>340</td>
<td>161</td>
</tr>
<tr>
<td>A-9.txt</td>
<td>225</td>
<td>128</td>
<td>240</td>
<td>131</td>
</tr>
<tr>
<td>A-10.txt</td>
<td>920</td>
<td>313</td>
<td>955</td>
<td>321</td>
</tr>
</tbody>
</table>

The program `npuScheduler.exe` is run on instance `A-1.txt` (which has to be stored in the very same directory as the executable). A solution file, `A-1_sol_13264537.txt`, is produced after 300 seconds (in the same directory as the application `npuScheduler.exe`), using a random number seed of 13264537.

5 Limitations and future research

The proposition of the solution approach assumes that the datasets provided by the organizers of the MISTA 2013 Challenge are a representative subset for the problem that should be solved. This implies the following limitations of our concept:

- The generation of feasible mode assignments by means of a randomized algorithm can, as described above, be a problematic issue in case of very limited non-renewable resources. In such situations, infeasible mode selections could be (temporarily) accepted in the search procedure, and evaluated by means of a penalty function [6].

- The implemented neighborhood operators are, with the exception of the SMC, of complexity $O(n^2)$. In can be expected that larger instances will call for adaptations of the neighborhood search.

In the MISTA 2013 Challenge, the largest test instance was B-9 with $n = 642$ (incl. dummy-activities). For this particular dataset, no local optimum was found within the given time limit, and thus, no perturbation move was performed here.

Naturally, future research should address the above mentioned limitations.

References