European Journal of Operational Research 256 (2017) 729–741

Contents lists available at ScienceDirect

European Journal of Operational Research

journal homepage: www.elsevier.com/locate/ejor

Discrete Optimization

A multi-threaded local search algorithm and computer implementation for the multi-mode, resource-constrained multi-project scheduling problem

Martin Josef Geiger

Helmut-Schmidt-University/University of the Federal Armed Forces Hamburg, Holstenhofweg 85, 22043 Hamburg, Germany

ARTICLE INFO

Article history:
Received 15 November 2014
Accepted 11 July 2016
Available online 16 July 2016

Keywords:
Multi-mode resource-constrained multi-project scheduling
Local search
Variable neighborhood search
Iterated local search

ABSTRACT

The article describes a solution approach for the multi-mode, resource-constrained multi-project scheduling problem. Our key ideas are based on the concepts of Variable Neighborhood Search, together with Iterated Local Search. A particular focus of this work is devoted towards the parallel implementation of such local search solution techniques. With the increasing availability of parallel (multi-core) computer hardware, we believe that such techniques are about to gain attractiveness in the future.

Experiments have been conducted on benchmark instances of the *MISTA 2013 Challenge*, as well as on the well-known MMLIB-datasets. The approach described in this article ranked second in the Challenge, and found new best results to 1371 of the 4320 MMLIB-instances.

Note that the source-code of our implementation has been made available under http://dx.doi.org/10.17632/cw95t56hjv.1. The software may be freely used for non-profit (research/education) purposes.

1. Introduction

Resource-constrained project scheduling problems describe a class of rather well-understood and actively investigated problems from combinatorial optimization. Common to those problems is the existence of a set of limited resources, used by activities. In this sense, activities do not only consume time when being performed (scheduled), but also certain resource capacities. Therefore, and in addition to the usual constraints known from scheduling, such as precedence constraints, release times, and similarly, the limited availability of the relevant resources creates conflicts among the activities using them. Thus, the starting times of activities can be delayed due to the unavailability of certain resources for some activities.

In the classical variant of the resource-constrained scheduling problem, each activity has exactly one execution mode that defines the consumption of resources when being scheduled. The multi-mode formulation generalizes this by defining a set of potential execution modes for each activity. It is common to assume that the durations of the modes, as well as the resource consumptions differ, so that a real choice problem, i. e. the problem of choosing the set of execution modes arises.

In multi-project scheduling, an additional aspect becomes apparent: A set of several projects, each of which is, with respect to its activities, independent from the others, has to be scheduled at once. The combination of the multi-project setting and the resource-constrained variant then assumes that at least some resources are commonly used/shared among the project.

Research in project scheduling is foremost of academic nature. An impressive number of publications are available on the subject. Important textbooks on the wider field of project scheduling include (Demeulemeester & Herroelen, 2002). A particular focus on the issue of constrained resources is shown in Neumann, Schwindt, and Zimmermann (2003), and the classical textbook (Sprecher, 1994) investigates the multi-mode case. A recent literature review on project scheduling problem variants is provided in Hartmann and Briskorn (2010), and the more recent work of Węglarz, Józefowska, Mika, and Waligóra (2011) focuses on the particular aspect of multi-mode project scheduling.

Due to both the relevance and complexity of most project scheduling problems, a wide range of solution techniques has been put forward, which is already documented in an early review (Brucker, Drexl, Möhring, Neumann, & Pesch, 2001). Exact optimization algorithms are applicable to smaller data sets or special problem variants, but still provide useful insights into the particular problem structures and properties. For larger data sets, suboptimal methods have been proposed, mostly stemming from modern heuristics/metaheuristics (Kolisch & Hartmann, 2006).

E-mail address: m.j.geiger@hsu-hh.de

http://dx.doi.org/10.1016/j.ejor.2016.07.024
0377-2217/

Important examples include classical local search techniques (Kolisch & Drexl, 1997), Simulated Annealing (Bouleimen & Lecocq, 2003; Józefowska, Mika, Różycki, Waligóra, & Węglarz, 2001), Genetic Algorithms (Alcaraz, Maroto, & Ruiz, 2003; Hartmann, 2001), and, more recently, adaptive/learning heuristics (Wauters, Verstichel, Verbeeck, & Vanden Berghe, 2009).

Besides the apparent scientific progress in the field, numerous industrial applications are known, underlining the practical value of the research carried out. Of particular practical value are software systems that bridge the academic field of project scheduling (optimization) towards the practical application of the proposed techniques (Herroelen, 2005).

Recently, the *MISTA 2013 Challenge* (Wauters et al., 2016) stipulated research in the above sketched field by creating a research and implementation competition. Novel datasets of multi-mode, resource-constrained multi-project scheduling problems had to be solved under running time restrictions. Besides, new and more challenging multi-mode resource-constrained project scheduling datasets have been recently made available in the scientific literature (Van Peteghem & Vanhoucke, 2014). This article describes in detail our ideas we developed for the above introduced problem.

2. Problem description

In the multi-mode, resource-constrained multi-project scheduling problem, several projects, each of which comprises a set of activities, have to be integrated into a single overall plan. While precedence constraints among the activities of a single (sub-)project exist, there are no precedence relations between projects. A release date of each project has to be respected, and resources that are jointly used by several projects are present. Two types of resources are defined: renewable and non-renewable. The latter are always specific to each project, i. e. they are never shared among the projects, while some renewable resources apply to several projects at once.

In addition to this multi-project characteristic, multiple execution modes of the activities are provided, which implies that exactly one of them has to be chosen when placing the activity into a schedule.

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*), i. e. the completion time of the last activity over all projects/activities. The two objectives are considered in a lexicographical order with *TPD* over *TMS*. In detail, a "project delay" arises when a project is completed after its fictive lower bound on the completion time. This lower bound is derived by executing each activity in its fastest possible way while neglecting the resource constraints at the same time, thus computing a critical path duration *CPD*. Then, *TPD* sums up all delays of the sub-projects.

The problem can be formally described as follows (note that we here mostly adopt the notations of MIS (2013)). Let us identify each project by an index $i \in \mathcal{P}, \mathcal{P} = \{0, \ldots, n-1\}$. Each project consists of a set of non-preemptive activities J_i. A schedule then defines starting times s_{ij} for the activities ($j \in \{1, \ldots, |J_i|\}$). Because each project $i \in \mathcal{P}$ comes with a release date r_i, the starting times have to be $s_{ij} \geq r_i \; \forall i, j$. It is quite common to ensure that the first ($j = 0$) and the last activity ($j = |J_i| + 1$) of each project are "dummy activities", i. e. an activity with a duration of 0 and no resource consumptions.

Each project $i \in \mathcal{P}$ uses a set of local renewable resources $L_i^\rho = \{1, \ldots, |L_i^\rho|\}$ and a set of local non-renewable resources $L_i^\nu = \{|L_i^\rho| + 1, \ldots, |L_i^\rho| + |L_i^\nu|\}$. Each renewable resource has a capacity per time unit. The non-renewable resources come with a total capacity c_{il}, $l \in L_i^\nu$.

Aside from these resources, a set of global renewable resources G^ρ exists that is jointly used by all projects, limited by the capacity c_g, $g \in G^\rho$.

Let us define the set of execution modes for each activity j of project i as $M_{ij} = \{1, \ldots, |M_{ij}|\}$. Each mode m comes with an execution time d_{ijm} and resource consumption r^ν_{ijml}, r^ρ_{ijml}, and r^ρ_{ijmg} (for the local non-renewable, the local renewable, and the global renewable resources).

Any feasible schedule therefore has to respect the following constraints

$$\sum_{j \in J_i} \sum_{m \in M_{ij}} x_{ijt} y_{ijm} r^\rho_{ijml} \leq c_{il} \quad \forall i \in \mathcal{P}, l \in L_i^\rho, t \in [0, T] \tag{1}$$

$$\sum_{j \in J_i} \sum_{m \in M_{ij}} y_{ijm} r^\nu_{ijml} \leq c_{il} \quad \forall i \in \mathcal{P}, l \in L_i^\nu \tag{2}$$

$$\sum_{i \in \mathcal{P}} \sum_{j \in J_i} \sum_{m \in M_{ij}} x_{ijt} y_{ijm} r^\rho_{ijmg} \leq c_g \quad \forall g \in G^\rho, t \in [0, T] \tag{3}$$

$$\sum_{m \in M_{ij}} y_{ijm} = 1 \quad \forall i \in \mathcal{P}, j \in J_i \tag{4}$$

$$s_{ij} + \sum_{m \in M_{ij}} y_{ijm} d_{ijm} \leq s_{ij'} \quad \forall i \in \mathcal{P}, \; j, j' : j \prec j' \tag{5}$$

$$x_{ijt} \in \{0, 1\} \quad \forall i \in \mathcal{P}, j \in J_i, t \in [0, T] \tag{6}$$

$$y_{ijm} \in \{0, 1\} \quad \forall i \in \mathcal{P}, j \in J_i, m \in M_{ij} \tag{7}$$

Obviously, the binary variables x_{ijt} define whether an activity j of project i is executed at time t or not. Besides, the binary variables y_{ijm} define whether activity j of project i runs in mode m or not. Precedence constraints are given in expression (5).

The binary variables x_{ijt} are linked to the starting times s_{ij} as given in expression (8) and (9). Intuitively, all binary variables x_{ijt} assume values of 1, beginning with the starting time $t = s_{ij}$ up to the end of the execution of the activity j. They are, otherwise, 0.

$$x_{ijt} = 1 \quad \forall i \in \mathcal{P}, j \in J_i, t \in [s_{ij}, s_{ij} - 1 + \sum_{m \in M_{ij}} y_{ijm} d_{ijm}] \tag{8}$$

$$x_{ijt} = 0 \quad \forall i \in \mathcal{P}, j \in J_i, t \notin [s_{ij}, s_{ij} - 1 + \sum_{m \in M_{ij}} y_{ijm} d_{ijm}] \tag{9}$$

On the basis of the above introduced notations, an earliest finish time EF_{ij} can be computed for each activity j of project i as given in expressions (10) and (11), with $pred(j)$ denoting the set of predecessors of activity j.

$$EF_{i0} = 0 \quad \forall i \in \mathcal{P} \tag{10}$$

$$EF_{ij} = \max_{j' \in pred(j)} EF_{ij'} + \min_{m \in M_{ij}} d_{ijm} \quad \forall i \in \mathcal{P}, j > 0 \tag{11}$$

The critical path duration CPD_i is then given as $CPD_i = EF_{i(|J_i|+1)}$, and the total project delay *TPD* as:

$$TPD = \sum_{i \in \mathcal{P}} s_{i(|J_i|+1)} - r_i - EF_{i(|J_i|+1)} \tag{12}$$

The second objective, the total makespan *TMS* is thus given as stated in the following expression:

$$TMS = \max_{i \in \mathcal{P}} s_{i(|J_i|+1)} - \min_{i \in \mathcal{P}} r_i \tag{13}$$

In the general case, a combination of the two, lexicographically ordered criteria into a single evaluation function is not possible. If

however we may safely assume that, for a particular set of problem instances, $TMS < \alpha$, then a combination of *TPD* and *TMS* into a single, overall evaluation function is possible:

$$F = \alpha\ TPD + TMS \tag{14}$$

3. Solution approach

3.1. Targeted research goals

In the light of the specific characteristics of the above presented problem, the following research goals have been formulated:

1. Fast construction of a first feasible solution, and ensuring feasibility of the computed solutions at all times.
   This is of importance as the algorithm should be able to return a solution after any given running time.
2. Fast convergence towards high quality solutions.
   On top of returning a feasible solution at all times, the quality of the alternatives under consideration should be improved as fast as possible. This presents a typical goal for any competitive algorithm.
3. Appropriate exploitation of the nowadays available multi-core architecture.
   Clearly, the formulation of a competitive approach requires the usage of the availability of parallel computing hardware. This comes with the question of how to manage multiple threads in parallel, how to implement the communication between them, etc.
4. Proposition of a concept that is not tied to specific problem characteristics, and therefore can address a range of datasets in the problem domain.
   On the one hand, this goal reflects the fact that the challenge organizers are testing the program on private ("hidden") instances. On the other hand, we are simply interested in how far a rather general local search can go in terms of the obtained solution quality. As we have seen in our previous work (Geiger, 2010, 2012), general local search concepts can yield satisfying results when implemented properly.

Unfortunately, in the general case, i. e. the case of at least two non-renewable resources, the determination of a feasible execution mode choice is $\mathcal{NP}$-complete (Kolisch & Drexl, 1997). This means that, depending on the datasets, this sub-aspect of the problem already presents a problem of its own, especially in the light of the running time restrictions. Existing approaches therefore often employ exact methods for the mode selection (Coelho & Vanhoucke, 2011).

3.2. Solution representation and preprocessing

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 can be put to good use in a serial scheduling scheme (Demeulemeester & Herroelen, 2002; Kelley, 1963). In such a scheme, a schedule is constructed by sequentially assigning activities into a schedule, one activity at a time. In the light of the special property of the objective functions, 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.

A further simplification is achieved by combining the set of projects into an overall super-project. In this linear-time transformation, each activity is renumbered and given a unique index $k, k = 1, \ldots, n-2$. Besides, two artificial "dummy"-activities are added: One representing the start of the super-project ($k = 0$), the other the end of it ($k = n-1$).

Alternatives $\vec{X}$ are represented by means of two vectors, $\vec{X} = (\vec{M}, \vec{S})$:

- A mode vector $\vec{M}, \vec{M} = (y'_0, \ldots, y'_k, \ldots, y'_n)$ stores for each activity k the mode in which it runs. As an example, $\vec{M} = (1, 3, 2, 1)$ encodes that activities $k = 0$ and $k = 3$ run in their individual mode 1, $k = 1$ runs in mode 3, and $k = 2$ runs in its mode 2. This representation directly reflects the binary decision variables y_{ijm}.
- A sequence vector $\vec{S}$ encodes, in the spirit of a permutation of jobs, the positions of the activities. As an example, $\vec{S} = (0, 2, 1, 3)$ suggests that activity 0 precedes 2, which precedes 1, which is performed before 3. Obviously, not all $n!$ possible permutations necessarily respect the precedence constraints (5), so the above mentioned serial scheduling scheme must be employed to 'decode' $\vec{S}$ into a feasible choice of starting times.
  In the implemented serial scheduling schemes, scheduling conflicts are resolved by giving priority to activities depending on their position in $\vec{S}$. At the same time, the serial scheduling scheme can build and return a new sequence vector $\vec{S}'$ which resolves any existing inconsistencies of $\vec{S}$. With respect to a neighborhood search procedure, this means that any permutation can be decoded into a feasible schedule, and when doing so, is, at the same time, mapped back into the space of precedence-constraint-respecting permutations.

3.3. Construction phase

The generation of initial alternatives is composed of two steps: a mode assignment phase and a scheduling phase. Algorithm 1 describes the procedure. The mode assignment phase, which is preceded by a preprocessing phase that removed non-executable modes (Sprecher, Hartmann, & Drexl, 1997), is found in lines 4–18, and the scheduling phase is described in lines 19–24. It can be seen that rep^{max} alternatives are created in the proposed fashion. The best alternative is kept for further improvements by the subsequent local search.

Mode assignment. In a first attempt, an entirely random mode assignment $\vec{M}$ is computed. If $\vec{M}$ is infeasible, i. e. by exceeding the capacity of the non-renewable resources, a repair procedure is triggered, which randomly alters values y'_k in $\vec{M}$ until the feasibility of the mode choices is restored. After a maximum number of unsuccessful repair attempts, $\vec{M}$ 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. The following expression measures this violation V of constraints (2), and assumes a value of $V = 0$ for a feasible mode vector $\vec{M}$.

$$V = \sum_{i \in \mathcal{P}} \sum_{l \in L_i^v} \max \left\{ 0, -c_{il} + \sum_{j \in J_i} \sum_{m \in M_{ij}} y_{ijm} r_{ijml}^v \right\} \tag{15}$$

The intuition of expression (15) has been previously described in Van Peteghem and Vanhoucke (2010) and referred to as an "excess of resource requests". Alternatively, an "excess ratio expressed in percentage" is given in Elloumi and Fortemps (2010).

Due to the above mentioned complexity of the mode choice problem, such a procedure could turn out to be problematic, as it does not guarantee the identification of the feasible mode assignment within finite time. In this sense, this particular design choice of the algorithm presents a risky and possibly disadvantageous element. It has therefore been thoroughly tested on all multi-mode,

Algorithm 1 Construction phase.

```
Preprocessing: Removal of infeasible modes
rep = 0
repeat
  repeat
    Create random mode vector M⃗, compute V
    repair = 0
    while V > 0 or repair < repair^max do
      Randomly select an element y'_k of M⃗ with more 2 or more possible execution modes
      Randomly change mode assignment of y'_k
      Compute the new violation of constraints V^changed
      if V^changed ≥ V then
        Restore old mode assignment of y'_k
      else
        V ← V^changed
      end if
      repair ← repair + 1
    end while
  until a feasible mode vector M⃗ has been found
  repeat
    Compute for all activities with scheduled predecessors the possible start and end
    Determine the activity with the smallest possible end, and compute the set of activities that overlap with this one ("conflict set")
    Sort the conflict set, in nondecreasing order of the lower bounds of the activities starting times
    Take the first l elements of the ordered conflict set and randomly select and schedule one of these l elements
  until all activities are scheduled
  Update of global best solution
  rep ← rep + 1
until rep = rep^max
```

resource-constrained project scheduling datasets which were delivered in the *MISTA 2013 Challenge*, and further tests on single-project MMRCPSP instances (Van Peteghem & Vanhoucke, 2014) confirmed our findings. In conclusion, the procedure repeatedly and consistently managed to create feasible mode vectors for all instances. The running times for the repair procedure were in the order of milliseconds. The maximum number of repair attempts has been set to $repair^{max} = 500$, a sufficiently large value for the problem and datasets at hand.

Initial scheduling. Second, a first schedule is constructed by sequentially scheduling all activities, one at a time. Activities can be placed into the schedule, if and only if their set of predecessors ($pred(k)$) has been already scheduled. This ensures that the obtained solution is an active schedule, a rather classical concept that can be traced back to Giffler and Thompson (1960). Starting with the activity that could end the earliest, the set of overlapping activities, i. e. the set of activities that have time-wise parallel/overlapping execution times, is computed. Then, the activity with the smallest lower bound on their possible starting times (computed as $\max_{k' \in pred(k)} EF_{k'}$) is chosen from this "conflict set".

A straight-forward randomization of this procedure is possible by computing a subset of l activities with small lower bounds of possible starting times. Then, one of the activities of this subset is chosen randomly. Such repetitive construction procedures are employed in concepts known from Greedy Randomized Adaptive Search (Resende & Ribeiro, 2003).

The outcome of the second step of the construction phase is a first feasible alternative $\vec{X} \in \mathcal{X}$ (with $\mathcal{X}$ denoting the set of feasible solutions), consisting of a feasible mode choice $\vec{M}$ and a permutation of activities $\vec{S}$. The permutation $\vec{S}$ stems from the sequence in which the serial scheduling scheme placed the activities into the schedule.

3.4. Iterative improvement search phase

The initially constructed alternative $\vec{X}$ is then improved by means of local search. The proposed concept is based on the ideas of Variable Neighborhood Search (Hansen & Mladenović, 2001) (VNS) and Iterated Local Search (Lourenço, Martin, & Stützle, 2003). 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.

Neighborhoods. We proposed four different neighborhoods, two of which work on the mode vector $\vec{M}$, two altering the activity sequence $\vec{S}$, plus a perturbation move that uses one of the neighborhoods:

- The proposed two mode modification neighborhoods:
  1. Single mode change (SMC).
     SMC changes the mode choice of a single element in $\vec{M}$. Note that this mode change may lead to an infeasible $\vec{M}$, in which case the move is discarded.
     The cardinality of solutions created by SMC is bounded by the number of activities (n) and the number of modes ($|M_k|$) for each activity: $|\text{SMC}| \leq \sum_{k=1}^{n} (|M_k| - 1)$.
  2. Two modes change (TMC).
     TMC changes the mode assignments of exactly two elements in $\vec{M}$. Identical to the strategy of SMC, only feasible modifications of $\vec{M}$ are permitted, infeasible transformations are not investigated further. In comparison to SMC, much more solutions are created, as pairs of activities k, k' are considered: $|\text{TMC}| \leq \frac{1}{2} \sum_{k=1}^{n} \sum_{k',k' \neq k}^{n} (|M_k| - 1)(|M_{k'} - 1|)$.
- The proposed two sequence modification neighborhoods:
  1. Exchange (EX).
     EX exchanges the positions of two activities in $\vec{S}$, and therefore can create $|\text{EX}| = \frac{n(n-1)}{2}$ neighboring alternatives.
  2. Inversion (INV).
     INV inverts a subsequence of activities within $\vec{S}$. A subsequence within $\vec{S}$ is defined by a first activity position and a last activity position, and includes all activities in between those positions. Thus, and identical to EX, $|\text{INV}| = \frac{n(n-1)}{2}$ inversions exist.
- Perturbation, escaping local optima:
  The perturbation move starts with randomly changing the execution mode of a single activity. In case this change yields a feasible $\vec{M}$, the perturbation ends. Otherwise, the above described repair procedure is called, and feasibility of $\vec{M}$ is restored. In the latter case, several activities are affected by the perturbation. This provides a certain degree of diversification, without modifying the current solution too much.

Acceptance criterion. In general, neighboring solutions are accepted if they improve the current alternative $\vec{X}$, i. e. $F(\vec{X}') < F(\vec{X}) \wedge \vec{X}' \in \mathcal{X}$.

After some initial experiments with the neighborhoods described above, some alteration was introduced to this acceptance rule for the TMC neighborhood. In the TMC, we accept $\vec{X}' \in \text{TMC}(\vec{X})$ if $F(\vec{X}') \leq F(\vec{X}) \wedge \vec{X}' \in \mathcal{X}$. Accepting alternatives of equal quality appears to have a beneficial effect for this operator, possibly due to some diversification that results from it.

Sequence of neighborhoods. It has been shown that in Variable Neighborhood Search, the sequence in which the neighborhoods are searched may have an influence on the outcome (Geiger, 2010). Some caution is therefore advisable when determining the sequence in which the neighborhoods are searched.

The above presented construction phase comes with the clear disadvantage of possibly selecting a suboptimal set of execution modes. It is therefore beneficial to quickly apply the SMC operator before exhaustively searching other neighborhoods. Then, the four neighborhoods are applied in a repetitive fashion until no improvement is possible. We have chosen to start with the sequence modification neighborhoods, followed by the mode modification neighborhoods. This follows the fact that the sequences of the activities are, when starting the local search, rather poorly chosen. EX and INV then improve the situation quickly. Note that this is also the case when returning from a perturbation move, as the perturbation only works on $\vec{M}$.

The following Algorithm 2 describes the logic of the proposed Iterated Variable Neighborhood Search approach.

Algorithm 2 Iterated Variable Neighborhood Search.

1: Create initial feasible alternative $\vec{X} = (\vec{M}, \vec{S})$, $\vec{X} \in \mathcal{X}$ using Algorithm 1
2: Apply SMC to $\vec{S}$ of $\vec{X}$ until no improvement
3: Update of global best solution
4: **repeat**
5: **repeat**
6: Apply EX to $\vec{S}$ of $\vec{X}$ until no improvement
7: Apply INV to $\vec{S}$ of $\vec{X}$ until no improvement
8: Apply SMC to $\vec{M}$ of $\vec{X}$ until no improvement
9: Apply TMC to $\vec{M}$ of $\vec{X}$
10: **until** $\vec{X}$ locally optimal with respect to EX, INV, and SMC, and TMC has been applied
11: Update of global best solution
12: Perturbation of $\vec{X}$
13: **if** $\vec{X} \notin \mathcal{X}$: **then**
14: restore feasibility
15: **end if**
16: **until** termination criterion is met

It can be seen in Algorithm 2, that EX, INV, and SMC each are applied on the current solution until the operator cannot yield an improvement. This is not true for the two modes change neighborhood TMC, which presents a noticeable difference. As we have shown above, TMC is much more computationally demanding than EX, INV, and SMC, in particular. Concentrating on this comparable larger neighborhood therefore comes with the disadvantage of consuming a lot of computing time. In the light of research goal 2 of Section 3.1, i. e. the fast converge towards a good solution, this adjustment to the algorithm has been made.

3.5. Multi-threaded local search variants

Algorithm 2 describes the mechanism used in a single thread of the program. As we have access to multi-core processors, such procedures should be integrated into a framework that allows the control and usage of multiple local search runs in parallel.

Two different variants of multi-threaded local search have been implemented, depicted in Figs. 1 and 2.

Fig. 1 shows version (a), in which a master thread controls several (four) worker threads, running in parallel, independent from each other. Exchange of information with the master thread is only done once a worker thread reaches a local optimum. Then, the global best solution is updated, if possible, and the search within the worker thread continues with the perturbation move, followed by the Variable Neighborhood Search. Those steps are shown in lines 3 and 11 of Algorithm 2. Obviously, the communications between threads are kept to a minimum in this setting.

Preliminary experiments have shown that this strategy (a) is beneficial for smaller instances. In those datasets, the VNS quickly reaches a local optimum, and an update of the global best solution appears to be sufficient once this happens. Larger instances come with the difficulty of obtaining a local optimum in the first place. Therefore, a second parallel local search variant (b) has been proposed and implemented, as shown in Fig. 2. In this setting, all available threads/CPU-cores are concentrated on a single solution by dividing up the investigation of a neighborhood among them. This means that each thread takes up the computation of around $\frac{1}{4}$ of the neighboring solutions. Note that the usage of more than four threads makes sense on platforms with more than 4 cores. Besides, technologies such as Intel's Hyper-Threading (Marr et al., 2002) may call for more worker threads than available cores.

The assignment of the computational load to the worker threads presents a parallel machine scheduling problem itself. As we cannot easily control the exact assignment of threads to processor cores (this is done by the operating system), some other strategy must be implemented to utilize the CPU to a maximum, and to minimize the makespan of the computations for a single neighborhood. An appropriate concept is the division of the set of neighboring solutions into a larger number of smaller subsets (in the case of Fig. 2: nine). Each subset is subsequently assigned to a worker thread, which then quickly responds to the master thread. We may describe this as an (each worker thread gets a) 'slice of the pie'-strategy, and refer to the number of slices (subsets) in the following as s.

This concept allows a fast update of the currently best solution even for larger instances. Unfortunately, the communications between the threads are increased in this setting, so it is important to mention that it is disadvantageous for smaller instances. The effort for communicating is directly influenced by the value of s, and an appropriate value must be found by experimental investigations.

4. Experimental investigation

4.1. The case of the MISTA 2013 Challenge

We have tested the solution approach of Section 3 on the novel datasets of the *MISTA 2013 Challenge*. The implementation also participated in the finals of the competition, ranking first in competition phase 1 (qualification), and second in competition phase 2/overall (finals). Details on the experiments are reported in the following subsections.

4.1.1. Implementation details and hardware

Our algorithm is coded in Microsoft Visual Basic programming language using Microsoft Visual Studio 2012. Despite all traditional deliberation on the language, the current Microsoft JiT-compiler creates an extremely convincing result. Some other implementations of heuristics for complex planning problems point in this direction: Recently, our approach for the *EURO/ROADEF Challenge 2014* won the qualification phase a ranked third in the finals. In the same year, our algorithm for the *VeRoLog Solver Challenge*, organized by the *EURO Working Group on Vehicle Routing and Logistics*, was awarded the third price.

We have used an Intel X5550 and a set of Intel X5650 processors, running at 2.67 Gigahertz for the test runs. Despite the availability of eight/twelve 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).

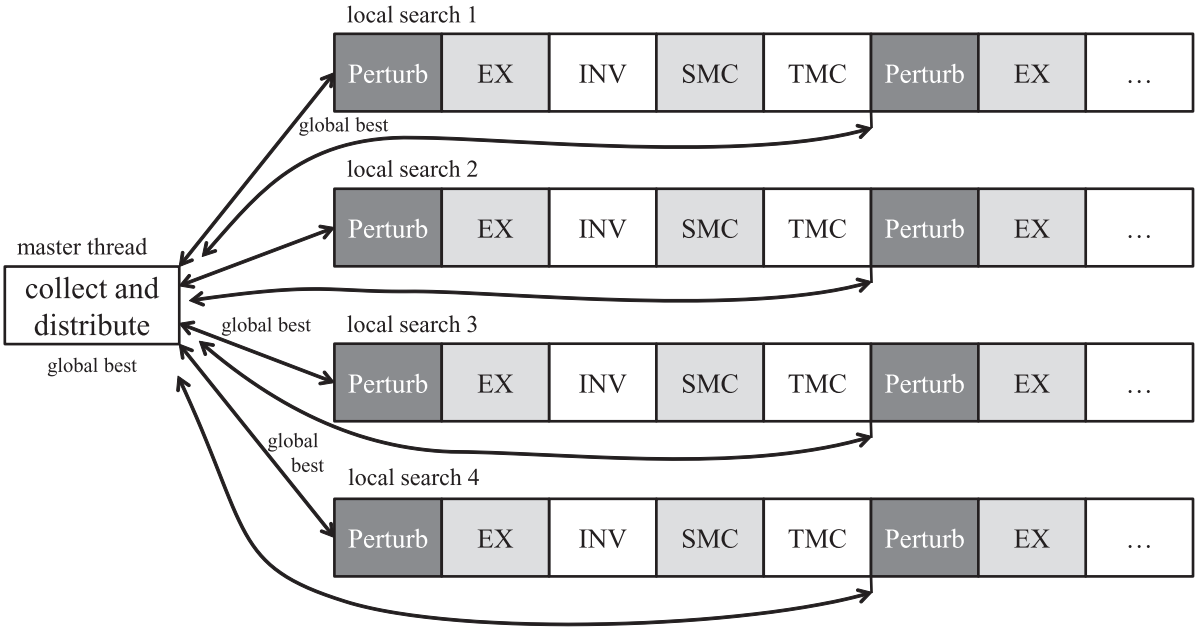

Fig. 1. Variant (a) of multi-threaded local search: independent runs with global archive.

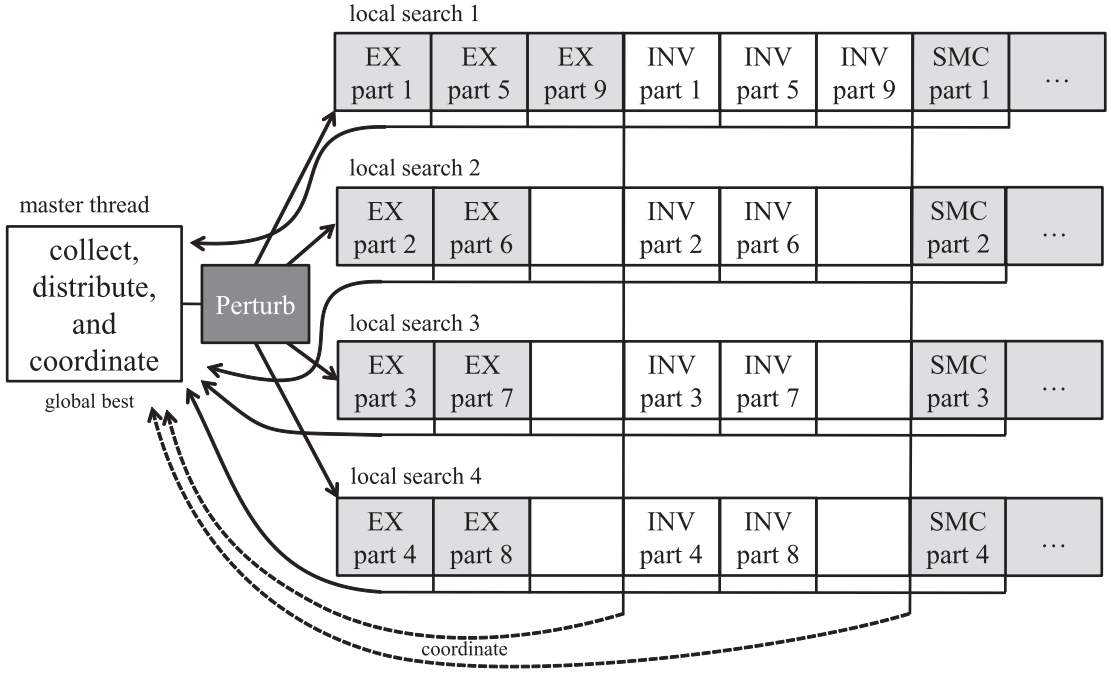

Fig. 2. Variant (b) of multi-threaded local search: coordinated runs, each working on a subset of the currently investigated neighborhood.

The provided benchmark program of the 2011 International Timetabling Competition gave, on the organizers machine, which is an Intel Core i7-2600 (3.4 Gigahertz), 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 percent 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. The derived computing time of 389 seconds on an Intel X5550/5650 has been used as the termination criterion for all experiments.

4.1.2. Instances

The MISTA 2013 Challenge brought three sets of instances. Datasets 'A' were released first and used in the qualification phase. Datasets 'B' were made available later in the final phase, and some additional instances ('X') were privately used by the organizers to verify and compare the submitted programs.

Table 1 gives an overview about the datasets used in the experiments and their key characteristics.

The datasets (multi-projects) combine several single projects, taken from the J10, J20, and J30-instances. In each project, there are always two local non-renewable resources. Each project also has at most two renewable resources. Global resources are

Table 1
Overview about the datasets.

Dataset	projects	all activities	dummy activities	global resources	local renewable resources	local non-ren. resources
A-1	2	26	6	1	2	4
A-2	2	46	6	1	2	4
A-3	2	66	6	1	2	4
A-4	5	62	12	1	5	10
A-5	5	112	12	1	5	10
A-6	5	162	12	1	5	10
A-7	10	122	22	2	0	20
A-8	10	222	22	2	0	20
A-9	10	322	22	1	10	20
A-10	10	322	22	1	10	20
B-1	10	122	22	1	10	20
B-2	10	222	22	2	0	20
B-3	10	322	22	1	10	20
B-4	10	182	32	1	10	20
B-5	15	332	32	1	15	30
B-6	15	482	32	1	15	30
B-7	20	242	42	2	0	40
B-8	20	442	42	2	0	40
B-9	20	642	42	1	20	40
B-10	20	462	42	2	0	40

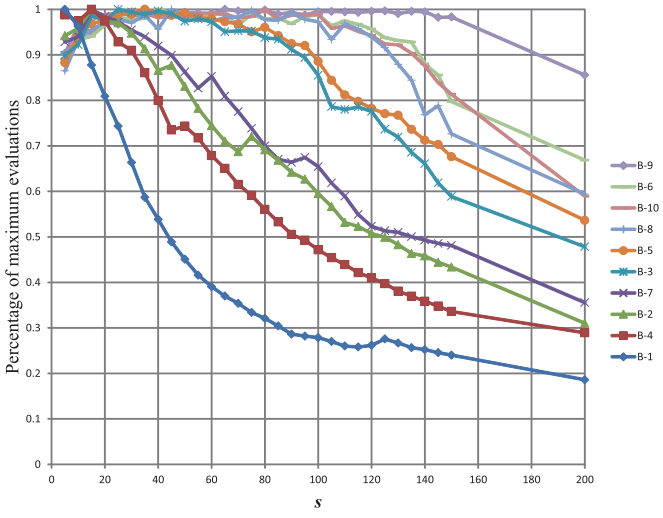

Fig. 3. Influence of s on the number of evaluations in parallel approach (b). Average values over 20 test runs.

introduced as follows: They replace a single or both local renewable local resources. This means that instances with two global renewable resources do not have additional local renewable resources, see Table 1.

Clearly, the B-instances are more challenging, due to their size in terms of number of activities and increased number of resources. In comparison to instances reported in the literature (Browning & Yassine, 2010; Van Peteghem & Vanhoucke, 2014), the *MISTA 2013 Challenge*- datasets are significantly larger. Any further comparison is difficult, mostly because the objective function of this study differs from the commonly considered makespan minimization.

4.1.3. Results and comparison

Comparison of parallel processing variants. We first study the effect of the parameter s on the number of evaluations. Values of $s = 5, 10, 15, 20, \ldots, 150$ and $s = 200$ have been tested. Fig. 3 depicts the obtained results of this analysis. In order to better compare the instances, the maximum number of evaluations reached by the parallel implementation (b) has been set to 100 percent. The figure then plots the relative performance of any value of s to this maximum.

In case of the smaller instances B-1, B-4, B-2, and B-7, the communication overhead climbs quickly with increasing s. Clearly, the neighborhoods, here being of relatively small cardinality, are, with an increasing value of s, divided into tiny subsets which cannot be handled efficiently by the implementation. This is not true for the medium sized instances B-3, B-5, and the larger ones, B-8, B-10, B6, and B-9. Conversely, and especially in case of the larger datasets, the maximum number of evaluations is reached at medium values of s, before declining again.

It follows that the value of s should be set accordingly. For the finals of the *MISTA 2013 Challenge*, we have chosen a value of $s = 70$.

Some further comparisons of the parallel concept (b), using $s = 70$, with the parallel local search variant (a) have been done in order to better understand the computational overhead of the different threading models. Table 2 reports the obtained number of evaluations (i. e. the number of generated schedules), and the number of perturbation moves executed by the local search algorithms. The latter directly corresponds to the number of local optima found during search.

Variant (b) cannot compete with (a) in the smaller instances. Both the number of evaluations and the number of perturbations are significantly smaller. The picture changes starting with the medium sized instances, B-3 and B-5. Here, more local optima are visited by local search variant (b), and this tendency develops further for the larger datasets.

The values obtained for B-3 and B-5 are of particular interest: while fewer function evaluations are counted, more local optima are visited. At a first glance this appears to be counter-intuitive. However, due to the more frequent communication between the threads, improvements are propagated faster from one worker thread to another (by the in-between master thread), and as a corollary, local optima are reached faster.

It is further worth to mention that in case of instance B-9, with a size of $n = 642$, variant (b) very rarely reaches a local optimum, while variant (a) never did in the experiments.

Table 3 compares the average values of F of the two parallel approaches (a) and (b). Average values of F have been computed over 20 test runs. It is possible to observe that variant (b) provides a clear advantage for medium-sized and large instances B-3, B-5, B-8, B-10, and B-9, whereas variant (a) is superior for the smaller datasets.

The investigations clearly show that, in case of the *MISTA 2013 Challenge* problem, the application of local search variant (a) versus (b) can be guided by the size of the instances. Our findings indicate a transition from variant (a) to (b) at around $n = 300$. The final submission of our code/implementation has been set accordingly.

Relative comparison to other participants. Table 4 reports the quality of the best found alternatives and the median values (Geiger, 2013). 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 and/or the implementation.

This observation also holds for the results obtained by our colleagues (Asta, Karapetya, Kheiri, Özcan, & Parkes, 2013; López-Ibáñez, Mascia, Marmion, & Stützle, 2013; Toffolo, Santos, Carvalho, & Soares, 2013), which we repeat in Tables 5–7.

The direct comparison to Asta et al. (2013) (i. e. comparing Tables 4 and 5) reveals that our definition of an Iterated Variable Neighborhood Search algorithm for the problem at hand is relatively outperformed. It can be suspected that this is partially due to the considerable larger set of neighborhood operators used in

Table 2
Comparison of parallel variant (a) and (b): evaluations and perturbations. 389 seconds on an Intel X5550/5650, 4 worker threads in parallel, average values of 20 test runs.

		Evaluations			Perturbations	
Dataset	n	version (a)	version (b)	gap	version (a)	version (b)
B-1	122	48,992,407.9	5,212,610.4	−89 percent	879.5	103.9
B-4	182	26,923,069.4	9,476,406.8	−65 percent	187.7	75.1
B-2	222	25,580,780.4	12,168,442.2	−52 percent	80.8	47.2
B-7	242	21,114,348.3	12,451,055.3	−41 percent	66.7	50.9
B-3	322	15,230,758.2	12,942,977.5	−15 percent	16.0	19.5
B-5	332	14,172,206.0	12,323,139.7	−13 percent	13.4	18.6
B-8	442	9,676,301.9	9,416,647.0	−3 percent	1.2	3.6
B-10	462	8,624,003.1	8,834,494.8	+2 percent	0.2	3.6
B-6	482	9,169,916.6	8,770,202.0	−4 percent	0.9	3.3
B-9	642	5,532,794.2	5,766,109.1	+4 percent	0.0	0.1

Table 3
Comparison of parallel variant (a) and (b): Average results of F ($\alpha = 100,000$) for instances B. 389 seconds on an Intel X5550/5650, 4 worker threads in parallel, average values of 20 test runs.

		Average value of F		
Dataset	n	version (a)	version (b)	diff.
B-1	122	35,800,131	36,886,798	+3.0 percent
B-4	182	133,390,287	136,836,957	+2.6 percent
B-2	222	52,320,176	56,020,180	+7.1 percent
B-7	242	84,950,235	87,776,910	+3.3 percent
B-3	322	64,170,219	61,776,883	−3.7 percent
B-5	332	93,690,264	89,356,930	−4.6 percent
B-8	442	377,110,574	369,423,906	−2.0 percent
B-10	462	386,310,486	372,547,143	−3.6 percent
B-6	482	111,260,249	110,926,915	−0.3 percent
B-9	642	533,480,846	477,477,470	−10.5 percent

Asta et al. (2013). On contrast to our approach, the authors present and implement 13 “low level heuristics” (neighborhoods), which is significantly more. It follows as an observation, that more operators are beneficial, something to learn and keep in mind for further studies.

Also, in the work of Asta et al. (2013), a preprocessing phase in terms of ordering the projects is done, which we do not adopt. This element can be seen as a problem-specific ingredient. In contrast, the above presented concept lacks such an element.

On the other hand, the outcomes of the elaborate integer programming formulation of Toffolo et al. (2013) are, in most cases, surpassed, which underlines the applicability of our rather problem-independent solution approach (comparison of Tables 4 to 6).

An interesting further approach to the problem has been introduced by López-Ibáñez et al. (2013), ranking, together with other teams, fifth in the competition. The authors employed a system that automatically searched the space of algorithm designs for the problem at hand, and thus lift the research question onto an even higher level. Important questions, such as the sequence in which neighborhoods are investigated, are here searched in an automated fashion. In this light, we believe that a comparison to such an automated design process is of relevance. Table 7 reports the obtained values of López-Ibáñez et al. (2013).

In conclusion, and also in comparison to López-Ibáñez et al. (2013), we are able to show that the presented approach encapsulated some sensible design choices. Despite the simplicity of the neighborhood operators, solutions of competitive quality are found. Besides, the insights gained when experimenting with different parallel implementations of local search strategies can be of benefit for other applications as well.

4.2. *MMLIB experiments*

In order to better understand the performance and capabilities of the presented approach, additional experiments were conducted on recent (single-project) multi-mode, resource-constrained project scheduling problems. Clearly, those datasets are more deeply studied, and extensive computational results are available that allow a thorough comparison.

4.2.1. *Adjustments and implementation details*

The commonly considered single-project variant of the MMR-CPSP can be solved with above presented approach by introducing some minor adjustments. For once, only a single project is subsumed under the super-project (see Section 3.2, which means that effectively two additional “dummy”-activities are introduced. Then, the objective function (14) is modified by assuming of value of $\alpha = 0$, so that only the total makespan *TMS* remains for the further experiments.

Table 4
Our best results and median values over 20 test runs (Geiger, 2013). 389 seconds, executing four worker threads on an Intel X5550/X5650.

	Best results		Median			Best results		Median	
Dataset	*TPD*	*TMS*	*TPD*	*TMS*	Dataset	*TPD*	*TMS*	*TPD*	*TMS*
A-1	1	23	1	23	B-1	349	130	357	131
A-2	2	41	2	41	B-2	481	171	515	178
A-3	0	50	0	50	B-3	604	214	624	216
A-4	65	42	65	42	B-4	1283	287	1328	286
A-5	153	104	155	106	B-5	866	252	909	265
A-6	144	94	150	97	B-6	1067	246	1098	250
A-7	601	206	620	204	B-7	827	232	844	241
A-8	319	162	340	161	B-8	3618	565	3729	574
A-9	225	128	240	131	B-9	4606	783	4825	806
A-10	920	313	955	321	B-10	3541	473	3751	474

Table 5
Results of Ahmed Kheiri, Shahriar Asta, Ender Özcan, Daniel Karapetyan, and Andrew J. Parkes, who ranked first in the *MISTA 2013 Challenge*. Values reported in Asta et al. (2013), best results over a "large number" of test runs.

	Best results		Mean			Best results		Mean	
Dataset	*TPD*	*TMS*	*TPD*	*TMS*	Dataset	*TPD*	*TMS*	*TPD*	*TMS*
A-1	1	23	1	23	B-1	345	125	352	128
A-2	2	41	2	41	B-2	424	165	441	169
A-3	0	50	0	50	B-3	526	208	547	211
A-4	65	42	65	42	B-4	1254	277	1279	285
A-5	150	104	152	104	B-5	809	251	831	247
A-6	133	91	136	91	B-6	897	224	936	233
A-7	589	201	602	202	B-7	782	229	796	231
A-8	272	151	280	151	B-8	3088	533	3315	534
A-9	196	125	203	126	B-9	4097	745	4197	755
A-10	840	306	850	308	B-10	3136	445	3238	454

Table 6
Results of Túlio A.M. Toffolo, Haroldo G. Santos, Marco A.M. Carvalho, and Janniele A. Soares, who ranked third in the *MISTA 2013 Challenge*. Values reported in Toffolo et al. (2013), based on 50 test runs.

	Best results		Mean			Best results		Mean	
Dataset	*TPD*	*TMS*	*TPD*	*TMS*	Dataset	*TPD*	*TMS*	*TPD*	*TMS*
A-1	1	23	1	23	B-1	360	131	360	131
A-2	2	41	2	41	B-2	431	155	437	160
A-3	0	50	0	50	B-3	585	199	619	207
A-4	68	46	68	46	B-4	1435	290	1504	291
A-5	159	103	191	106	B-5	867	250	938	251
A-6	170	93	189	102	B-6	970	224	1157	233
A-7	639	193	771	212	B-7	902	239	911	245
A-8	281	145	348	147	B-8	3001	521	3244	533
A-9	212	122	269	126	B-9	4807	746	5339	769
A-10	983	303	1182	332	B-10	3123	425	3198	437

Table 7
Results of Manuel López-Ibáñez, Marie-Eléonore Marmion, Franco Mascia, and Thomas Stützle, who ranked fifth in the *MISTA 2013 Challenge*. Values of F ($\alpha = 100{,}000$) reported in López-Ibáñez et al. (2013), based on 15 test runs. Final configuration of the algorithm found after 50,000 test runs.

Dataset	Mean	Standard dev.	Instance	Mean	Standard dev.
A-1	100,023.0	0.0	B-1	47,513,470.0	993,455.0
A-2	200,041.0	0.0	B-2	62,533,505.2	1,478,739.0
A-3	6,716.7	25,820.9	B-3	77,833,562.7	2,689,041.3
A-4	7,733,384.6	308,609.9	B-4	171,953,646.1	2,329,583.1
A-5	19,326,779.4	471,271.0	B-5	125,606,943.5	2,904,055.1
A-6	18,906,768.8	517,503.1	B-6	147,626,938.3	3,769,328.2
A-7	65,733,541.5	952,942.3	B-7	117,420,271.2	1,729,658.8
A-8	41,460,167.7	1,202,855.6	B-8	43,0687,283.1	9,350,930.5
A-9	34,213,480.4	720,980.5	B-9	641,294,225.6	6,006,603.6
A-10	121,873,675.7	2,599,030.5	B-10	456,520,505.2	6,424,312.1

With regard to the typical instance sizes of the MMRCPSP datasets ($n = 50$, $n = 100$), only variant (a) of the multi-threaded local search is tested. Note that this directly reflects the findings of Section 4.1.3.

For a better comparison to other approaches found in the literature, the program was also executed with a single thread only, limiting its' computational capabilities to what was, not long ago, the state-of-the-art on 'regular' computer hardware. Besides, we have adopted the way of counting evaluations in the resource-constrained project scheduling literature. This allows us to report results obtained after 1000, 5000, and 50,000 evaluations as commonly done in the literature.

4.2.2. Benchmark instances

Van Peteghem and Vanhoucke recently introduced new benchmark instances for the MMRCPSP, which are referred to as the MMLIB50, MMLIB100 and MMLIB+ datasets Van Peteghem and Vanhoucke (2014). Based on their extensive studies, those instances possess favorable attributes which are not equally present in the older datasets. For instance, no execution mode can be excluded based on simple dominance considerations. Moreover, the MMLIB+ instances come with up to four renewable and four non-renewable resources, as well as up to nine execution modes per activity. Table 8 gives a brief overview about the most basic attributes of the MMLIB-datasets.

4.2.3. Results and comparison

In some first tests, each MMLIB instance was solved ten times with variant (a) of our approach, testing one versus four local search threads. We were surprised to see that, especially for the MMLIB+ datasets, numerous improvements were found, sometimes even within the first second. In the end, the four-threaded algorithm improved 1134 of the 3240 MMLIB+ instances after 389 seconds of computing time (927 in case of the single-threaded version; computer hardware as specified above). Quite obviously, longer running times and more threads in parallel lead to better

Table 8
Basic properties of the MMLIB-instances (Van Peteghem & Vanhoucke, 2014).

	MMLIB50	MMLIB100	MMLIB+
number of instances	540	540	3240
known optima from Van Peteghem and Vanhoucke (2014)	212	236	356
known optima including our results	*216*	*236*	*361*
activities	50	100	50, 100
renewable resources	2	2	2, 4
non-renewable resources	2	2	2, 4
modes per activity	3	3	3, 6, 9

Table 9
Number of improved instances (new best known solutions) after different running times, comparison of 1 versus 4 threads.

	1 thread			4 threads		
Seconds	MMLIB50	MMLIB100	MMLIB+	MMLIB50	MMLIB100	MMLIB+
1	2	0	69	7	0	134
10	8	1	258	15	6	429
30	13	4	441	17	10	665
60	18	8	587	22	16	818
120	20	11	714	25	19	946
180	20	13	793	28	23	1014
240	20	14	841	29	24	1064
300	20	15	877	31	24	1098
389	23	16	927	31	28	1134

Table 10
Number of improved instances (new best known solutions) after different number of evaluations.

Number of schedules	MMLIB50	MMLIB100	MMLIB+
1000	1	0	5
5000	1	2	20
50,000	8	8	189

results. While this was to be expected, the amount of new best known solutions is still an interesting finding. Table 9 reports the number of instances for which new best known solutions were found, recorded after different running times.

The program was re-run, limiting the number of evaluations to 1000, 5000, and 50,000 evaluations, as commonly done in the literature on resource-constrained project scheduling problems. Table 10 reports the number of improved instances from these experiments (30 independent runs in total).

Limiting the search to 50,000 schedules (or fewer) was comparable less successful. We believe that this is understandable, as the initial design of the algorithm was not done with the idea of strictly limiting the number of schedules. Nevertheless, new best known results are found, even in those restricted experiments.

The above presented test runs led so far to the identification of new best known solutions for 1371 of the 4320 MMLIB-datasets (note that the single-threaded and the four-threaded runs interestingly not always improved the same instances). As an additional outcome of our experiments, nine instances have been solved to optimality for the first time (see Table 8).

However, the comparison to other approaches cannot be based on this observation only. The following Table 11 repeats, for MMLIB50, MMLIB100, and MMLIB+, the best five approaches as identified in Van Peteghem and Vanhoucke (2014). Average deviations from the known lower bounds are given for different amounts of function evaluations, including the ones of our approach (counting the number of generated schedules). In contrast, our average deviations from the lower bounds when recording the running time are reported in Table 12.

Table 11
Solutions for the MMLIB50, MMLIB100 and MMLIB+ datasets after 1000, 5000, and 50,000 schedules, values reported in Van Peteghem and Vanhoucke (2014). The numbers are the average percentage deviation from the lower bounds.

Instances:	Number of schedules		
Author, reference	1000	5000	50,000
MMLIB50:			
HART01, (Hartmann, 2001)	35.40	30.61	26.81
LOVA09, (Lova, Tormos, Cervantes, & Barber, 2009)	34.16	28.59	26.69
DAMA09, (Damak, Jarboui, Siarry, & Loukil, 2009)	46.19	32.46	26.27
VANP10, (Van Peteghem & Vanhoucke, 2010)	34.07	27.12	24.93
VANP11, (Van Peteghem & Vanhoucke, 2011)	28.17	25.45	23.79
This approach	42.96	33.02	28.35
MMLIB100:			
HART01, (Hartmann, 2001)	39.69	33.98	29.04
DAMA09, (Damak et al., 2009)	52.31	36.87	28.92
LOVA09, (Lova et al., 2009)	36.29	31.01	27.89
VANP10, (Van Peteghem & Vanhoucke, 2010)	37.58	29.55	25.63
VANP11, (Van Peteghem & Vanhoucke, 2011)	29.77	26.51	24.02
This approach	53.26	44.11	32.91
MMLIB+:			
ELLO10, (Elloumi & Fortemps, 2010)		130.06	106.35
DAMA09, (Damak et al., 2009)		126.69	103.75
JOZE01, (Józefowska et al., 2001)		121.09	103.19
LOVA09, (Lova et al., 2009)		114.07	102.73
VANP11, (Van Peteghem & Vanhoucke, 2011)		101.45	92.76
This approach	170.67	149.50	114.22

Table 12
Results of the new approach after different running times, comparison of 1 versus 4 threads. The numbers are the average percentage deviation from the lower bounds.

	1 thread			4 threads		
Seconds	MMLIB50	MMLIB100	MMLIB+	MMLIB50	MMLIB100	MMLIB+
1	29.64	36.89	127.97	27.83	33.90	119.85
10	27.41	32.72	108.91	26.37	30.72	103.70
30	26.57	31.16	103.87	25.78	29.61	99.68
60	26.15	30.37	101.67	25.49	29.02	97.99
120	25.79	29.73	99.91	25.23	28.56	96.67
180	25.60	29.37	99.02	25.10	28.33	96.02
240	25.48	29.15	98.45	25.00	28.16	95.61
300	25.40	28.99	98.05	24.94	28.03	95.30
389	25.30	28.82	97.59	24.87	27.89	94.95

At this points, the experiments reveal a somehow mixed performance of our approach. On the one hand, several best known solutions have been improved. But on the other hand, when measuring the average deviation from the lower bounds, the results are not outperforming existing approaches. Especially when limiting the number of generated schedules to 1000, 5000, or 50,000 shows this situation. We believe that the deviation from the best results as documented in the literature is due to the design of the algorithm, which is more towards fast (full) evaluations of schedules. As a matter of fact, this deviation shrinks when regarding the running time, as documented in Table 12.

The large number of new best known solutions is intriguing, especially in the light of the overall mixed performance of the algorithm. The following Fig. 4 sheds some light into this by analyzing each of the 4320 instances with respect to two attributes: The average running time of the 4-threaded local search for *finding* the best known solution (BKS), and the average running time for *improving* the BKS (maximum running time was 389 seconds; each run which does not find or improve the BKS assumes 400 seconds in the plot). Obviously, the latter cannot assume a smaller value than the former.

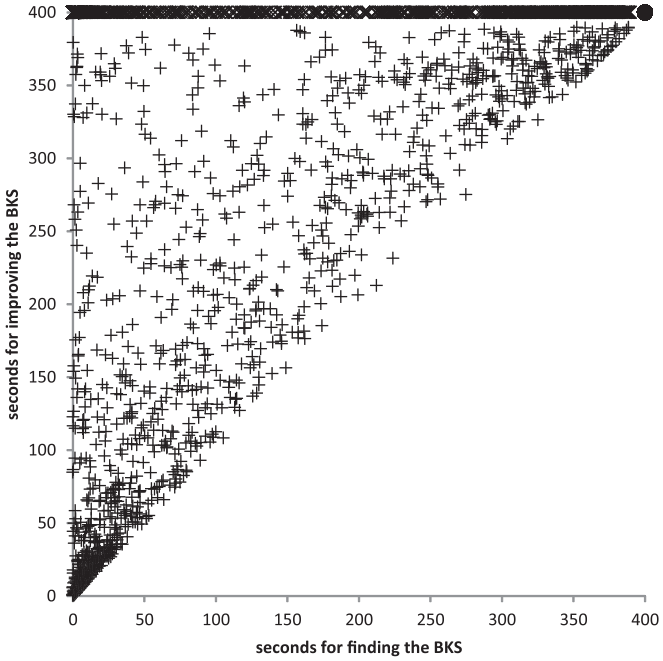

Fig. 4. Average running times for finding or improving the BKS. Experiments in which the BKS is not found or improved are plotted with a value of 400 seconds for comparison reasons.

It becomes clear that a large number of best known solutions is identified during the running time. Some of them cannot be improved any further, as the BKS is the global optimum. Others are improved, and in several cases very quickly (see also the tables above). However, for 1799 instances, the BKS is never reached, and those map into a single point in the top right corner.

A thorough analysis has been conducted, trying to identify features that explain the different outcomes of the experiments on the instances. In detail, the following attributes of the data sets have been considered (Vanhoucke, 2013):

Network topology: Coefficient of Network Complexity (CNC), Order Strength (OS), Serial/Parallel indicator (SP), Activity Distribution indicator (AD), Length of Arcs indicator (LA), Length of Long Arcs indicator (I5), Topological Float (TF)

Renewable resource parameters: Number of resources (#RR), Resource Factor (RF), Resource Usage (RU), Resource Strength (RS), Resource Constrainedness (RC)

Nonrenewable resource parameters: Number of resources (#NR), Resource Factor (NRF), Resource Strength (NRS)

Other: Number of modes (#M)

Multiple regression analyses have been carried out, with the aim of explaining the average time for improving the BKS for a given instance as the dependent variable. Unfortunately, this did not turn out to be an easy endeavor. In fact, the data contains several cases which are difficult to treat.

- Some instances are already solved to optimality, and therefore cannot be improved any further, even if the BKS is identified.
- We may suspect that in some other cases, where the gap to the lower bound is already very small, the BKS is in fact the global optimum, without having definite proof of optimality. Again, those results cannot be improved by our method.
- Some instances are improved, but only in few runs and after longer running times. While our approach is successful in principle, the degree of success is relatively smaller. Effectively, the success rate is measured on an ordinal scale, with the time for finding an improved BKS as a cardinal part and another value for the instances which are not improved (in Fig. 4 this value is set to 400 seconds for visualization purposes).
- In some cases, the best known solution is found very quickly, but improvements take much longer. Fig. 4 illustrates this and the other cases.
- Several independent variables (characteristics of the data sets) assume only some few values. Distinguishing the cases based on those attributes is naturally difficult.
- The comparison of the observed improvement of the BKS is relative to the previously proposed optimization approaches and their results. Pretty much all of them are randomized algorithms/heuristics. As a corollary, the 'success' of such an algorithm has a stochastic component. We have observed this in several cases, in which some of the instances are improved, e. g. `Jall304_1`, while for others with the same characteristics, the BKS is never found, e. g. `Jall304_3`, `Jall304_4` (there are five instances for each attribute-combination in the `MMLIB` datasets).

Despite those problems, some factors that explain the selective performance of our approach have been identified. To do so, we concentrated on two groups of instances: datasets in which our approach was *not successful*, meaning that the BKS was never found, and datasets in which the BKS was improved *'quickly'*, i. e. below an average running time of 60 seconds. The former 1799 instances are shown in Fig. 4 as a point in the top right corner, while the latter 341 instances are within a bottom left triangle.

To our surprise, most resource-related attributes have no significant (#RR, RF, RU, RC, NRF, #M) or only a small impact on the observed behavior (#NR, NRS), and this also shows for CNC, AD, and I5. However, we find a meaningful relationship for SP, LA, and OS. Fig. 5 illustrates this finding by plotting the two groups of cases for the two most influential factors (SP and LA). We can see that our algorithm is particular successful with an increasing value of the Serial/Parallel indicator and the Length of Arcs indicator. Based on this analysis, we may conclude that more 'serial' networks are favorable for our approach, while the opposite holds for more 'parallel' networks.

5. Conclusions

The article presented a local search approach for the solution of the multi-mode, resource-constrained multi-project scheduling problem. The work was initially motivated by the *MISTA 2013 Challenge*, which provided an opportunity to implement novel ideas for the described application, but then applied to other newer datasets from the scientific literature, too.

Part of our work was the proposition of a rather problem-independent concept, with the goal of addressing a range of datasets without the need for problem-specific adaptations. We believe that with respect to this goal, the presented approach was successful. This finding is also supported by the application of our approach to multi-mode, resource-constrained (single-)project scheduling problem instances, for which we have been able to improve several benchmark instances.

However, some limitations arose. Especially for larger datasets, simple Variable Neighborhood Search is unable to identify local optima within reasonable/short time. Therefore, adaptations in the parallel processing framework have been proposed and implemented. This of course implies that the actual implementation of the algorithm is not entirely dataset-independent.

Nevertheless, we believe that our work underlines the potentials of rather straight-forward local search ideas for hard combinatorial planning problems. Our findings, put together with the recent work of Asta et al. (2013); López-Ibáñez et al. (2013); Toffolo et al. (2013), suggest that future research should address the

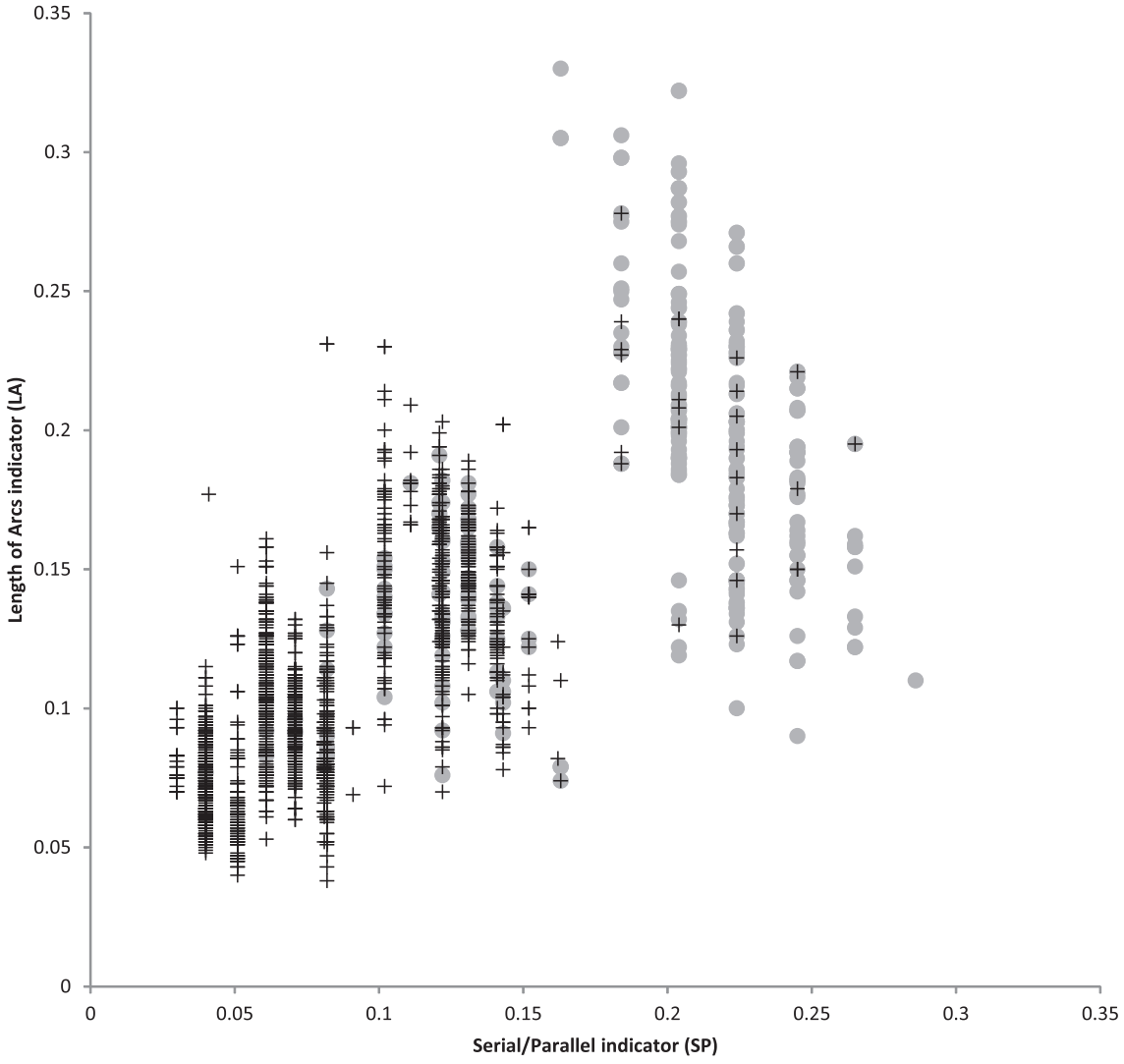

Fig. 5. Successful (grey dots) versus unsuccessful instances (crosses): Serial/Parallel indicator (SP) versus the Length of Arc (LA).

self-adaptiveness of algorithms to problem-specific and instance-specific properties.

Other limitations have been observed when applying the work to the recently proposed MMLIB instances. While we have been able to improve a considerable number of best known results, several instances are not solved in the best possible way. The approach put forward in this article turned out to be particularly strong for datasets with a more serial network structure.

In order to further disseminate the understanding of local search and its implementations, we share the source code of our algorithm with the scientific community, available for download from http://dx.doi.org/10.17632/cw95t56hjv.1.

Besides, a solution-checker has been developed, which allows the fast and comfortable verification of solutions to the MMLIB-instances (Geiger, 2015). The source code of this checker, including a brief technical documentation, ara freely available through the library-website of our University (urn:nbn:de:gbv:705-opus-30976).

Acknowledgments

We acknowledge the excellent work of the *MISTA 2013 Challenge* organizers, namely of Tony Wauters (chair of the competition), Joris Kinable, Pieter Smet, Wim Vancroonenburg, Greet Vanden Berghe, and Jannes Verstichel (KAHO Sint-Lieven, Gent, Belgium). The organizers did create and publish novel data sets, and a working solution checker/evaluator that allowed the verification of the obtained results. They also impartially re-ran the submitted programs, verified their performance, and created a ranking of approaches. The competition website can be found under http://gent.cs.kuleuven.be/mista2013challenge/, and a scientific article is available here (Wauters et al., 2016).

We also would like to thank Mario Vanhoucke for providing us the detailed data/characteristics of the MMLIB instances (Vanhoucke, Coelho, & Batselier, 2016).

References

Alcaraz, J., Maroto, C., & Ruiz, R. (2003). Solving the multi-mode resource-constrained project scheduling problem with genetic algorithms. *Journal of the Operational Research Society, 54*, 614–626.

Asta, S., Karapetya, D., Kheiri, A., Özcan, E., & Parkes, A. J. (2013). Combining monte-carlo and hyper-heuristic methods for the multi-mode resource-constrained multi-project scheduling problem. In G. Kendall, G. Vanden Berghe, & B. McCollum (Eds.), *Proceedings of the 6th multidisciplinary international conference on scheduling: Theory and applications* (pp. 836–839). Belgium: Gent.

Bouleimen, K., & Lecocq, H. (2003). A new efficient simulated annealing algorithm for the resource-constrained project scheduling problem and its multiple mode version. *European Journal of Operational Research, 149*(2), 268–281.

Browning, T. R., & Yassine, A. A. (2010). A random generator of resource-constrained multi-project network problems. *Journal of Scheduling, 13*, 143–161.

Brucker, P., Drexl, A., Möhring, R., Neumann, K., & Pesch, E. (2001). Resource-constrained project scheduling: notation, classiffication, models, and methods. *European Journal of Operational Research, 130*, 449–467.

Coelho, J., & Vanhoucke, M. (2011). Multi-mode resource-constrained project scheduling using RCPSP and SAT solvers. *European Journal of Operational Research, 213*, 73–82.

Damak, N., Jarboui, B., Siarry, P., & Loukil, T. (2009). Differential evolution for solving multi-mode resource-constrained project scheduling problems. *Computers & Operations Research, 36*(9), 2653–2659.

Demeulemeester, E. L., & Herroelen, W. S. (2002). *Project Scheduling: A Research Handbook*. Kluwer Academic Publishers.

Elloumi, S., & Fortemps, P. (2010). A hybrid rank-based evolutionary algorithm applied to multi-mode resource-constrained project scheduling problem. *European Journal of Operational Research, 205*, 31–41.

Geiger, M. J. (2010). On heuristic search for the single machine total weighted tardiness problem – some theoretical insights and their empirical verification. *European Journal of Operational Research, 207*(3), 1235–1243.

Geiger, M. J. (2012). Applying the threshold accepting metaheuristic to curriculum based course timetabling – a contribution to the second international timetabling competition itc 2007. *Annals of Operations Research, 194*, 189–202.

Geiger, M. J. (2013). Iterated variable neighborhood search for the resource constrained multi-mode multi-project scheduling problem. In G. Kendall, G. Vanden Berghe, & B. McCollum (Eds.), *Proceedings of the 6th multidisciplinary international conference on scheduling: Theory and applications* (pp. 836–839). Gent, Belgium.

Geiger, M. J. (2015). MMLIBchecker – A checker software for multi-mode resource–constrained project scheduling problem (MRCPSP) solution files.. *Research Report RR-15-03-01*. Hamburg, Germany: Helmut-Schmidt-University/ University of the Federal Armed Forces Hamburg, Logistics Management Department.

Giffler, B., & Thompson, G. L. (1960). Algorithms for solving production-scheduling problems. *Operations Research, 8*, 487–503.

Hansen, P., & Mladenović, N. (2001). Variable neighborhood search: principles and applications. *European Journal of Operational Research, 130*, 449–467.

Hartmann, S. (2001). Project scheduling with multiple modes: a genetic algorithm. *Annals of Operations Research, 102*, 111–135.

Hartmann, S., & Briskorn, D. (2010). A survey of variants and extensions of the resource-constrained project scheduling problem. *European Journal of Operational Research, 207*, 1–14.

Herroelen, W. (2005). Project scheduling – theory and practice. *Production and Operations Management, 14*(4), 413–432.

Józefowska, J., Mika, M., Różycki, R., Waligóra, G., & Węglarz, J. (2001). Simulated annealing for multi-mode resource-constrained project scheduling. *Annals of Operations Research, 102*, 137–155.

Kelley, J. E. (1963). The critical-path method: Resources planning and scheduling. In J. F. Muth, & G. L. Thompson (Eds.), *Industrial scheduling* (pp. 347–365). Englewood Cliffs: Prentice Hall.

Kolisch, R., & Drexl, A. (1997). Local search for nonpreemptive multi-mode resource–constrained project scheduling. *IIE Transactions, 29*, 987–999.

Kolisch, R., & Hartmann, S. (2006). Experimental investigation of heuristics for resource constrained project scheduling: an update. *European Journal of Operational Research, 174*(1), 23–37.

López-Ibáñez, M., Mascia, F., Marmion, M.-E., & Stützle, T. (2013). Automatic design of a hybrid iterated local search for the multi-mode resource-constrained multi-project scheduling problem. In G. Kendall, G. Vanden Berghe, & B. McCollum (Eds.), *Proceedings of the 6th multidisciplinary international conference on scheduling: Theory and applications* (pp. 820–825). Gent, Belgium.

Lourenço, H. R., Martin, O., & Stützle, T. (2003). Iterated local search. In F. Glover, & G. A. Kochenberger (Eds.), *Handbook of metaheuristics chapter 11*. In *International Series in Operations Research & Management Science: vol. 57* (pp. 321–353). Boston, Dordrecht, London: Kluwer Academic Publishers.

Lova, A., Tormos, P., Cervantes, M., & Barber, F. (2009). An efficient hybrid genetic algorithm for scheduling projects with resource constraints and multiple execution modes. *International Journal of Production Economics, 117*(2), 302–316.

Marr, D. T., Binns, F., Hill, D. L., Hinton, G., Koufaty, D. A., Miller, J. A., & Upton, M. (2002). Hyper-Threading technology architecture and microarchitecture. *Intel Technology Journal, 6*(1), 4–15.

MISTA (2013). challenge: the multi-mode resource-constrained multi-project scheduling problem. http://gent.cs.kuleuven.be/mista2013challenge/files/problem-description.pdf.

Neumann, K., Schwindt, C., & Zimmermann, J. (2003). *Project Scheduling with Time Windows and Scarce Resources*. Springer Verlag.

Resende, M. G. C., & Ribeiro, C. C. (2003). Greedy randomized adaptive search procedures. In F. Glover, & G. A. Kochenberger (Eds.), *Handbook of metaheuristics chapter 8*. In *International Series in Operations Research & Management Science: vol.57* (pp. 219–249). Boston, Dordrecht, London: Kluwer Academic Publishers.

Sprecher, A. (1994). *Resource-constrained project scheduling*. Springer Verlag.

Sprecher, A., Hartmann, S., & Drexl, A. (1997). An exact algorithm for project scheduling with multiple modes. *Operations Research Spektrum, 19*(3), 195–203.

Toffolo, T. A., Santos, H. G., Carvalho, M. A., & Soares, J. A. (2013). An integer programming approach for the multi-mode resource-constrained multi-project scheduling problem. In G. Kendall, G. Vanden Berghe, & B. McCollum (Eds.), *Proceedings of the 6th multidisciplinary international conference on scheduling: Theory and applications* (pp. 840–847). Gent, Belgium

Van Peteghem, V., & Vanhoucke, M. (2010). A genetic algorithm for the preemptive and non-preemptive multi-mode resource-constrained project scheduling problem. *European Journal of Operational Research, 201*, 409–418.

Van Peteghem, V., & Vanhoucke, M. (2011). Using resource scarceness characteristics to solve the multi-mode resource-constrained project scheduling problem. *Journal of Heuristics, 17*, 705–728.

Van Peteghem, V., & Vanhoucke, M. (2014). An experimental investigation of metaheuristics for the multi-mode resource-constrained project scheduling problem on new dataset instances. *European Journal of Operational Research, 235*, 62–72.

Vanhoucke, M. (2013). *Project Management with Dynamic Scheduling*. Berlin, Heidelberg: Springer-Verlag.

Vanhoucke, M., Coelho, J., & Batselier, J. (2016). An overview of project data for integrated project management and control. *Journal of Modern Project Management, 3*(3).

Węglarz, J., Józefowska, J., Mika, M., & Waligóra, G. (2011). Project scheduling with finite or infinite number of activity processing modes – a survey. *European Journal of Operational Research, 208*, 177–205.

Wauters, T., Kinable, J., Smet, P., Vancroonenburg, W., Vanden Berghe, G., & Verstichel, J. (2016). The multi-mode resource-constrained multi-project scheduling problem. *Journal of Scheduling, 19*(3), 271–283.

Wauters, T., Verstichel, J., Verbeeck, K., & Vanden Berghe, G. (2009). A learning metaheuristic for the multi mode resource constrained project scheduling problem. In *Learning and intelligent optimization*.