Noname manuscript No.
(will be inserted by the editor)

A Local Search Framework for Industrial Test Laboratory Scheduling

Florian Mischek · Nysret Musliu

Received: date / Accepted: date

Abstract In this paper we deal with a complex scheduling problem that arises in a real-world industrial test laboratory, where a large number of activities has to be performed using qualified personnel and specialized equipment, subject to time windows and several other constraints. The problem is an extension of the well-known Resource-Constrained Project Scheduling Problem (RCPSP) and features multiple heterogeneous resources with very general availability restrictions. We present and evaluate different metaheuristic approaches to solve this problem and show that Simulated Annealing can be used to achieve very good results over a wide range of instances generated based on real-world data. In particular for large instances, the heuristic is able to find better solutions than a state-of-the-art constraint programming solver within reasonable time.

Keywords RCPSP · Local Search · Real-world · Simulated Annealing

1 Introduction

Project scheduling problems appear in countless variations wherever multiple activities have to be scheduled and assigned resources of some kind, subject to various constraints. Examples include production and manufacturing environments, event management, software development, and many more. Since these problems can become quite large and include complex constraints in practical settings, there is an ever increasing need for automated solution approaches to produce high-quality solutions in acceptable time.

In this paper, we introduce a real-world scheduling problem that arises in an industrial test laboratory of a large company. It is related to the well-known Resource-Constrained Project Scheduling Problem (RCPSP), on which it builds by adding various additional extensions, both traditional and new, to capture the specific requirements of this, and other similar laboratories.

Christian Doppler Laboratory for Artificial Intelligence and Optimization for Planning and Scheduling,
DBAI, TU Wien
Favoritenstraße 9-11, 1040 Vienna, Austria E-mail: {fmischek,musliu}@dbai.tuwien.ac.at

In the Test Laboratory Scheduling Problem (TLSP), first described in a technical report by Mischek and Musliu (2018), tasks have to be grouped into larger units called jobs, which derive their properties from the tasks they contain. The jobs then have to be scheduled, subject to various constraints.

In this paper, we focus on the scheduling part of the problem, which we call TLSP-S. We assume that an acceptable grouping is already provided, which will not be changed in the course of the scheduling procedure. As a result, the properties of the jobs are static and the resulting problem is a direct extension of (multi-modal) RCPSP.

Besides several well-known features from the literature, such as time windows, TLSP(-S) also features additional constraints imposed by the real-world problem setting. In particular, RCPSP and its variants usually assume that units of a resource are identical and homogeneous and therefore can be used to cover the demand of all tasks. Due to this, an assignment of individual units to tasks is not necessary. Exceptions exist, e.g. in Dauzère-Pérès et al. (1998) or Bellenguez and Néron (2005), and sometimes an equivalent effect is achieved by introducing additional modes for each possible resource assignment (e.g. in Schwindt and Trautmann (2000) and Bartels and Zimmermann (2009)). However, this is practical only for a single resource with very few available values per task. In contrast, TLSP(-S) features multiple heterogeneous resources, with very general restrictions on which units can be used to perform a job, and potentially large demands.

In addition, jobs in TLSP(-S) can be *linked* to each other, indicating that these jobs must be performed by the same employees[1]. To the best of our knowledge, a similar concept exists only in Salewski et al. (1997) and Drexl et al. (2000), where the mode of several jobs is required to be the same.

This article is an extension of work presented at the conference on Practice and Theory in Automated Timetabling (PATAT) 2018, where we introduced TLSP as a new problem. With this work, we provide an instance generator for TLSP, which is capable of randomly generating instances based on real-world data with a wide variety of configuration options. We also introduce a local search framework for TLSP-S, and evaluate the performance of several different meta-heuristics on the problem. We show that Simulated Annealing can be used to produce high-quality results, that rival those of a state-of-the-art constraint programming model and even outperforms it on larger instances.

The rest of the paper is structured as follows: Section 2 gives a short overview over related literature and variants of RCPSP that are particularly relevant for TLSP. Section 3 formally introduces the problem definition, input data and constraints. Section 4 provides information about our instance generator and available data sets. Section 5 describes the local search framework and the algorithms used, followed by experimental results in Section 6. Finally, concluding remarks and future work are given in Section 7.

[1] This can be used to model multi-step procedures that require knowledge from previous steps, such as ensuring that documentation of completed tasks is prepared by the employees who actually performed the tasks.

2 Related literature

As the standard problem in the area of project scheduling, RCPSP has seen vast amounts of work over several decades. For surveys on literature regarding this problem and its many variants, we refer to surveys e.g. by Mika et al. (2015), Hartmann and Briskorn (2010) and Brucker et al. (1999).

In particular, Hartmann and Briskorn (2010) provide a comprehensive overview regarding work dealing with extensions to RCPSP. In the following, we provide some references for more recent literature showcasing particular features, where available.

One particular variant that is employed in lots of publications, and is also included in TLSP, is multi-mode RCPSP (MRCPSP), originally formulated in Elmaghraby (1977). It allows each activity to be performed in one of several modes which can affect duration and resource requirements. A survey focused on MRCPSP formulations is provided by Węglarz et al. (2011). A more recent example, including a Constraint Programming (CP) model for the problem, is given by Szeredi and Schutt (2016).

Time windows for activities, in the form of release dates, due dates and deadlines, appear in various combinations in many works, e.g. in Polo Mejia et al. (2017) and Avar and Najafi (2017), although typically either due dates or deadlines are used. Wauters et al. (2016) introduce the Multi-Mode Resource Constrained Multi-Project Scheduling Problem (MMRCMPSP), where activities are partitioned into multiple projects, with project-specific constraints and objective functions. This problem formulation was used for the MISTA 2013 challenge, which was won by Asta et al. (2016), using a hybrid of memetic and hyperheuristic methods with Monte-Carlo tree search. The same problem is also treated in Ahmeti and Musliu (2018), who provide a local search algorithm based on a combination of the min-conflict heuristic and tabu search, which bears some similarities to the heuristics used in this paper.

The objective to be minimized in RCPSP and most variants is the makespan, but also other objective functions, including multi-objective formulations have been considered. For example, Gonçalves et al. (2008) employ a weighted linear combination of several performance measures, including the total duration of each project, similar to the approach used in TLSP.

Regarding the heterogeneous resources, Bellenguez and Néron (2005) introduced the Multi-Skill RCPSP (MSPSP). In MSPSP, each resource unit has a number of skills, and resource requirements are given as a multi-set of required skills. Despite this different resource model, a distinct objective function and several other discrepancies, MSPSP is still quite closely related to TLSP-S. It also faces a similar problem as TLSP-S in that the number of possible assignments for each job is far too large to explicitly translate into separate modes. Instances for MSPSP contain up to 90 activities, compared to 300+ jobs for instances of practical size for TLSP-S. To the best of our knowledge, the best results so far for MSPSP have been achieved by Young et al. (2017), who used a CP approach to solve the problem.

Two problems dealing with scheduling activities in laboratories are given below: Bartels and Zimmermann (2009) deal with scheduling tests of experimental vehicles. The described problem contains several aspects and constraints similar to TLSP. However, it uses a different resource model and uses the number of employed

vehicles as the main optimization criterion. Polo Mejia et al. (2017) developed an integer linear program for scheduling research activities for a nuclear laboratory, using a problem formulation derived from MSPSP, but with (limited) preemption of activities.

3 Problem description

As mentioned before, we deal with a variant of TLSP (defined in the technical report Mischek and Musliu (2018)), where we assume that a grouping of tasks into jobs is already provided for each project, and focus on the scheduling part of the problem instead (TLSP-S). Thus, the goal is to find an assignment of a mode, time slot and resources to each (given) job, such that all constraints are fulfilled and the objective function is minimized.

In the following, we describe the TLSP-S problem.

Each instance consists of a scheduling period of h discrete *time slots* and resources of different kinds:

- *Employees* $E = \{1, \ldots, |E|\}$ who are qualified for different types of jobs.
- A number of *workbenches* $B = \{1, \ldots, |B|\}$ with different facilities.
- Various auxiliary lab *equipment* groups $G_g = \{1, \ldots, |G_g|\}$, where g is the group index. Each group represents a set of similar devices. The set of all equipment groups is denoted G^*.

Further we have given the set of *projects* labeled $P = \{1, \ldots, |P|\}$, and the set of *jobs* to be scheduled $J = \{1, \ldots, |J|\}$. For a project p, the jobs of this project are given as $J_p \subseteq J$. Each job belongs to exactly one project.

Each job j has several properties[2]:

- A time window, given via a *release date* α_j and a *deadline* ω_j. In addition, it has a *due date* $\bar{\omega}_j$, which is similar to the deadline, except that exceeding it is only a soft constraint violation.
- A set of *available modes* $M_j \subseteq M$, where M is the set of all modes.
- A *duration* d_{mj} for each available mode $m \in M_j$.
- The resource requirements for the job:
  - The number of *required workbenches* (at most one) $r_j^{Wb} \in \{0, 1\}$. If a workbench is required, it must be chosen from the *available workbenches* $B_j \subseteq B$.
  - The number of *required employees* r_m^{Em} depends on the mode $m \in M_j$. Each of these employees must be chosen from the set of *qualified employees* $E_j \subseteq E$. Additionally, there is also a set of *preferred employees* $E_j^{Pr} \subseteq E_j$.
  - For each equipment group $g \in G^*$, the job requires r_{gj}^{Eq} devices, which must be taken from the set of *available devices* $G_{gj} \subseteq G_g$ for the group.
- The *predecessors* $\mathcal{P}_j$ of the job, which must be completed before the job can start. Precedence relations will only occur between jobs of the same project.
- *Linked jobs* L_j of this job. All linked jobs must be performed by the same employee(s). As before, such links only occur between jobs of the same project.
- Optionally, the job may contain *initial assignments*:

[2] In TLSP, these are derived from the tasks contained within a job. Since we assume the distribution of tasks into jobs to be fixed, they can be given directly as part of the input for TLSP-S.

- An *initial mode* m_j^0.
- An *initial starting time slot* t_j^{s0} and an *initial completion time slot* t_j^{c0}.
- *Initial resource assignments*: $a_j^{Wb0} \in B$ and the sets $A_j^{Em0} \subseteq E$ and $A_{gj}^{Eq0} \subseteq G_g \; \forall g \in G*$ are the assigned workbench, employees and devices per group, respectively.

Some or all of these assignments may be present for any given job.

Out of all jobs, a subset are *started jobs* $J^S \subseteq J$. A started job will always have a start time of 0 and initial mode and resource assignments fulfilling all requirements. Usually, the available resources for started jobs are also restricted to the assigned values, to avoid interruptions of ongoing work due to reassignments. This is the case in all instances used in this article.

A solution for TLSP-S is a schedule, which must contain, for each job j, the following information:

- A *mode* $\dot{m}_j$.
- The job's *starting* and *completion time slots* $\dot{t}_j^s$ and $\dot{t}_j^c$.
- *Resource assignments*: $\dot{a}_j^{Wb} \in B \cup \{\epsilon\}$ and the sets $\dot{A}_j^{Em} \subseteq E$ and $\dot{A}_{gj}^{Eq} \subseteq G_g \; \forall g \in G*$ for workbench, employees and equipment assignment.

3.1 Constraints

A solution is evaluated in terms of constraints that it should fulfill. *Hard constraints* must all be satisfied in any feasible schedule, while the number and degree of violations of *soft constraints* in a solution give a measure for its quality.

Note that this is a direct translation of the constraints in Mischek and Musliu (2018), with adaptations only to account for the already provided grouping. We have preserved the original constraint numbering, resulting in missing numbers (H1 - H3) where constraints pertain to the grouping itself.

For the purpose of modeling, we introduce additional notation: The set of *active jobs* at time t is defined as $\mathcal{J}_t := \{j \in J : \dot{t}_j^s \leq t \wedge \dot{t}_j^c > t\}$.

3.1.1 Hard Constraints

H4: Job duration. The interval between start and completion of a job must match the job's duration.

$$\begin{aligned} &\forall j \in J : \\ &\quad \dot{t}_j^c - \dot{t}_j^s = d_{\dot{m}_j j} \end{aligned}$$

H5: Time Window. Each job must lie completely within the time window from the release date to the deadline.

$$\begin{aligned} &\forall j \in J : \\ &\quad \dot{t}_j^s \geq \alpha_j \\ &\quad \dot{t}_j^c \leq \omega_j \end{aligned}$$

H6: Task precedence. A job can start only after all prerequisite jobs have been completed.

$$\forall j \in J, k \in \mathcal{P}_j : \\ \dot{t}^c_k \leq \dot{t}^s_j$$

H7: Started jobs. A started job must start at time 0.

$$\forall j \in J^S : \\ \dot{t}^s_j = 0$$

H8: Single assignment. At any one time, each workbench, employee and device can be assigned to at most one job.

$$\forall b \in B, t \in \{0 \dots h-1\} : \\ |\{j \in \mathcal{J}_t : \dot{a}^{Wb}_j = b\}| \leq 1 \\ \forall e \in E, t \in \{0 \dots h-1\} : \\ |\{j \in \mathcal{J}_t : e \in \dot{A}^{Em}_j\}| \leq 1 \\ \forall g \in G^*, d \in G_g, t \in \{0 \dots h-1\} : \\ |\{j \in \mathcal{J}_t : d \in \dot{A}^{Eq}_{gj}\}| \leq 1$$

H9a: Workbench requirements. Each job requiring a workbench must have a workbench assigned.

$$\forall j \in J : \\ \dot{b}_j = \epsilon \iff r^{Wb}_j = 0$$

H9b: Employee requirements. Each job must have enough employees assigned to cover the demand given by the selected mode.

$$\forall j \in J : \\ |\dot{A}^{Em}_j| = r^{Em}_{\dot{m}_j}$$

H9c: Equipment requirements. Each job must have enough devices of each equipment group assigned to cover the demand for that group.

$$\forall j \in J, g \in G^* : \\ |\dot{A}^{Eq}_{gj}| = r^{Eq}_{gj}$$

H10a: Workbench suitability. The workbench assigned to a job must be suitable for it.

$$\forall j \in J : \\ \dot{a}^{Wb}_j = \epsilon \vee \dot{a}^{Wb}_j \in B_j$$

H10b: Employee qualification. All employees assigned to a job must be qualified for it.

$$\forall j \in J : \\ \dot{A}^{Em}_j \subseteq E_j$$

H10c: Equipment availability. The devices assigned to a job must be taken from the set of available devices for each group.

$$\forall j \in J, g \in G^* : \\ \dot{A}^{Eq}_{gj} \subseteq G_{gj}$$

H11: Linked jobs. Linked jobs must be assigned exactly the same employees.

$$\forall j \in J, k \in L_j : \\ \dot{A}^{Em}_{j} = \dot{A}^{Em}_{k}$$

3.1.2 Soft Constraints

The following constraints can be used to evaluate the quality of a feasible solution. They arise from the business requirements of our industrial partner.

Each soft constraint violation induces a penalty on the solution quality, denoted as C^i, where i is the soft constraint violated.

S1: Number of jobs[3]. The number of jobs should be kept as low as possible.

$$\forall j \in J : \\ C^{S1} \coloneqq |J|$$

S2: Employee project preferences. The employees assigned to a job should be taken from the set of preferred employees.

$$\forall j \in J : \\ C^{S2}_j \coloneqq |\{e \in \dot{A}^{Em}_j : e \notin E^{Pr}_j\}|$$

S3: Number of employees. The number of employees assigned to each project should be minimized.

$$\forall p \in P : \\ C^{S3}_p \coloneqq |\bigcup_{j \in J_p} \dot{A}^{Em}_j|$$

S4: Due date. The due date for each job should be observed.

$$\forall j \in J : \\ C^{S4}_j \coloneqq \max(\dot{t}^c_j - \bar{\omega}_j, 0)$$

S5: Project completion time. The total completion time (start of the first job to end of the last) of each project should be minimized.

$$\forall p \in P : \\ C^{S5}_p \coloneqq \max_{j \in J_p} \dot{t}^c_j \; - \; \min_{j \in J_p} \dot{t}^s_j$$

[3] This constraint is obviously constant in TLSP-S, but is included for comparability of results with instances of TLSP.

Constraint S2 allows defining "auxiliary" employees, which should only be used if necessary. Typically, these employees usually have other duties, but also possess the required qualifications to perform (some) tasks in the laboratory.

Constraints S3 and S5 avoid overheads by reducing the need for communication (both internal and external), (re-)familiarization with project-specific test procedures and storage space.

Constraint S4 makes the schedule more robust by encouraging jobs to be completed earlier than absolutely required, so they can still be finished on time in case of delays or other disturbances.

The overall solution quality will be determined as the weighted sum over all soft constraint violations.

The relative importance of these constraints (i.e. their weights) still needs to be defined. In practical applications, we expect them to be chosen interactively according to the current situation. For our evaluations, we have assumed a uniform weight of 1 for all soft constraints.

4 New instances for TLSP

In this section, we introduce an instance generator, which can be used to randomly generate instances for TLSP(-S) based on real-world data. We also propose two sets of new and publicly available instances for TLSP-S assembled using our instance generator.

4.1 Instance generator

We have developed an instance generator which can generate random instances for TLSP which exhibit a similar structure as the real-world data they are based on.

This instance generator works as follows: First, a reference solution is built by iteratively and greedily adding jobs to the schedule. Each such job is grown outward from a randomly selected point until the desired duration is reached, taking care to guarantee that a feasible resource assignment for the job exists at that location. During this process, job and task durations and their allocation among the projects are chosen using a very similar distribution as in the real-world laboratory. The other properties of the jobs and tasks, such as precedence constraints and available resources, are then generated based on the reference schedule, such that it is a feasible solution for the resulting instance. Finally, the initial assignments are also extracted and possibly adapted from the reference solution.

Several aspects of this generation process, such as the extraction of initial assignments or the structure of precedence constraints, can be configured to create instances with different properties. For a more in-depth description of the instance generator and possible options, we refer to the technical report by Mischek and Musliu (2018).

4.2 Data sets

Currently, there are two sets of test data available, *General* and *LabStructure*, both with 60 instances each. These were generated for TLSP, but include a grouping of tasks into jobs that is guaranteed to have at least one feasible solution. Thus, they can be directly translated to TLSP-S by assuming that grouping to be fixed and calculating the properties of the jobs from the tasks they contain. Feasible solutions for TLSP-S are guaranteed to also be feasible for TLSP under that grouping and their objective values differ only by a constant amount (due to constraint S1, which counts the number of jobs and is trivially useless for TLSP-S).

Both data sets feature instances of various sizes, starting at 5 projects over a period of 88 time slots, up to 90 projects over 782 time slots. A single project contains an average of close to 4 jobs. There are no initial assignments except for the started jobs at the beginning of the scheduling period and a small number ($\approx 5\%$) of jobs whose assignments are fixed.

The difference between the two data sets is that for the LabStructure set, the instance generator was configured to produce instances that are as close as possible to the actual real-world data. In contrast, the General set uses the same distribution of work across projects and tasks, but is more flexible regarding several other structural features, such as equipment groups and job precedence.

On average, jobs have 5 available workbenches and 6 qualified employees (the different modes each require between 0 and 2 employees). The demand and availability of equipment is more varied and differs a lot between data sets and instances. Three different types of equipment demands appear: Of any given group, jobs require either a single specific device (these are usually project-specific), one out of a small list of options (< 10) or several (12 on average, but with a large variance) out of a large list (depends on the instance size, up to several hundred). The distribution of these types is heavily skewed towards the first two options. Further, the huge number of possible assignments for equipment demands of the third type is offset by the fact that most jobs do not require equipment of these groups and the number of devices per group is large enough that feasible equipment assignments can be found quite easily. These resource distributions were adapted from real-world data in our partner laboratory.

Both data sets are available for download at `https://www.dbai.tuwien.ac.at/staff/fmischek/TLSP`. Once additional data sets become available, they will also be added there. In particular, we plan to also provide (anonymized) real-world data sets.

5 Local Search

For easy implementation of and comparison between different solution techniques, we have developed a solution framework that provides a unified workflow, common data structures and utility functionality for TLSP solver implementations.

While theoretically also applicable for other solution approaches, this framework is mainly intended for use with local search. Here, the basic building block is that of a *move*, which is a small change to a given solution, such as a replacement of a time slot or a resource assignment for a single job. Each move contains the necessary information to be applied to a schedule and to efficiently evaluate its

effects on the solution quality. A basic set of move implementations are provided, which can be combined to form more complex changes. *Neighborhoods* define a set of moves available from a current schedule, and provide functionality to access and iterate over these moves. In addition they also allow for the selection of a random or the best move among those they contain. These neighborhoods are employed by *search heuristics*, which implement the strategies to choose a move that should be applied from among several neighborhoods in each step of the search.

5.1 Neighborhoods

For TLSP-S, we developed two different neighborhoods. Both are the combination of several smaller neighborhoods focused on specific types of moves, and contain the union of all these moves.

The first set, called *Simple*, contains neighborhoods that each affect a single aspect of a job's assignments. It consists of the following neighborhoods:

- *Mode Change*: Switches the assigned mode of the job to a different value. The start time of the job is kept constant for all moves in this neighborhood, except where the new duration would conflict with time windows or precedence constraints. In these cases, the start time is adjusted to ensure that those constraints are satisfied. Also, the number of assigned employees is adjusted to match the new requirements.
- *Time Slot Change*: Moves the job to a new position in the schedule. As before, time windows and precedence constraints are respected by all possible moves.
- *Resource change*: Switches out a single assigned resource unit (workbench, employee or device) by a different unit of the same type (and group, for equipment).
- *Resource Swap*: Swaps a unit of a resource assigned to this job with a different unit of the same type assigned to an overlapping job. A resource unit is considered for a swap only if it is suitable for its new job.

While moves from the first three neighborhoods are theoretically sufficient to reach an optimal solution from any schedule already satisfying time window and precedence constraints, the addition of resource swaps adds more options in situations where a single change would result in a prohibitively large number of conflicts.

The second variant, called *JobOpt* completely removes a job from the schedule and then finds all possible combinations of mode, time slot and resource assignments for that job. As before, time windows and precedence constraints are respected, wherever possible.

However, in our experiments it turned out that the enormous number of potential equipment assignments per job in some instances made this approach infeasible in practice. Instead, we employed a reduced version of the JobOpt neighborhood, which keeps equipment assignments intact, combined with a Resource Change neighborhood limited to equipment changes. This results in much more manageable, but still potentially large neighborhood sizes. To further increase the efficiency when the best move in the neighborhood is required, JobOpt utilizes independencies between assigned resources to precompute and cache the effects of assigning individual units to the job. In the rest of the article, 'JobOpt' refers to

this combination of neighborhoods (the reduced JobOpt procedure and Resource Change limited to equipment).

5.2 Search heuristics

We have implemented three well-known metaheuristic search algorithms and adapted them for the use in TLSP-S: Tabu search (TS), Min-Conflict (MC) and Simulated Annealing (SA). These implementations are problem-agnostic and could also be used for TLSP and related variants, as long as the employed neighborhoods are adjusted to fit.

5.2.1 Tabu Search

Tabu search (Glover (1989)) is similar to hill climbing in that the best move from the neighborhood is chosen at each step. However, it contains a mechanism to escape from local optima in the form of a *tabu list* - a list of the last l applied moves. These moves must not be used again unless doing so results in a solution better than all solutions found so far.

This encourages the algorithm to explore new regions of the search space by avoiding cycles shorter than l moves in length. On the other hand, too large values for l can lead to scenarios where promising opportunities are not detected because they are still tabu from when the heuristic was still exploring another part of the search space.

5.2.2 Min-Conflict

The min-conflicts heuristic (Minton et al. (1992)) was originally developed to solve constraint satisfaction problems (including scheduling applications). It works by randomly selecting a variable appearing in at least one conflict (violated constraint) and choosing a value for it that minimizes the number of conflicts remaining.

This strategy can be adapted as follows for TLSP(-S) and our solver framework: Choose a job at random that violates at least one constraint, and pick a move from the neighborhood involving the chosen job that minimizes constraint violations (i.e. the best move).

Different selection strategies are possible, due to the distinction between hard and soft constraints. We have experimented with three different variants of Min-Conflict. The first selects jobs from among those violating hard constraints. Once the solution is feasible, also soft constraint violations are considered. The second variant considers both hard and soft constraint violations from the start. Finally, the third variant chooses from all jobs, regardless of their presence in any kind of constraint violation.

In our experiments, it turned out that the second and third variants were virtually equivalent. This can be explained by soft constraints S3 (Number of employees) and S5 (Project completion time), whose presence entails that most, if not all, jobs are involved in at least one soft constraint violation[4]. Further, results

[4] This is exacerbated in TLSP, where soft constraint S1 (Number of jobs) trivially involves every single job in a soft constraint violation

did not differ in any meaningful way between the first and either of the latter variants.

For simplicity, and to avoid having to keep a running account of the jobs involved in constraint violations, the experiments in Section 6 have been done using variant three.

A weakness of MC is that it contains no mechanism to avoid repeating already visited solutions and thus might get stuck where several adjacent configurations for most or all jobs are locally optimal. A possible countermeasure is to inject additional randomness into the solution procedure. In our framework, we have included a random walk (RW) algorithm, that is called with a low probability at each step instead of MC. It randomly selects a job and performs a random move for the chosen job.

Also, the search automatically restarts from a new initial solution if no progress has been achieved within a certain number of moves.

5.2.3 Simulated Annealing

Simulated Annealing is a search heuristic introduced by Kirkpatrick et al. (1983) that is inspired by physical annealing processes in mechanical statistics. It has been employed successfully to solve many NP-complete problems, including RCPSP (e.g. Bouleimen and Lecocq (2003); Laurent et al. (2017)).

In SA, the search starts from a randomly generated solution. In each step, a candidate move is chosen randomly from the available neighborhoods. The difference in objective value Δ due to the chosen move is calculated. If $\Delta < 0$ (for minimization problems), the move is applied. Otherwise, it is still accepted with probability $e^{-\Delta/T}$. Thus, the acceptance probability depends on Δ (smaller values have a larger probability to be accepted) and a parameter T called *temperature* (higher values result in a larger acceptance probability). The temperature starts at an initial value T^0 and is iteratively reduced after a certain number of steps, until a minimum temperature T^{min} is reached. At this point, the search stops.

Cooling is usually done by multiplying the current temperature by a *cooling factor* α, with $0 < \alpha < 1$. We have evaluated two variants for the choice of α. In the first variant, α is constant for the whole search. If T^{min} is reached, the search restarts, either from a new initial solution, or from the current solution, but with a reset in temperature (reheating). In the second variant, α is dynamically adjusted to the number of moves applied per second, such that T^{min} is reached right at the end of the available time (dynamic cooling).

A schedule for TLSP-S can contain both hard and soft constraint violations. This has to be taken into account when calculating a value for Δ. In our implementation, we have weighted each hard constraint violation by a factor w^H.

6 Experimental results

For the experiments, a set of thirty instances of different sizes were chosen (15 each from the General and LabStructure sets described in Section 4.2 - one of each size, plus a second instance for the three smallest sizes). The instances and some important properties are listed in Table 1.

#	Data Set	ID	$\|P\|$	$\|J\|$	h	$\|E\|$	$\|B\|$	$\|G^*\|$	$\overline{\|E_j\|}$	$\overline{\|B_j\|}$	$\overline{\|G_{gj}\|}$
1	General	000	5	7	88	7	7	3	2.08	3.57	1.5
2	General	001	5	8	88	7	7	3	4.88	3.63	15.67
3	LabStructure	000	5	24	88	7	7	3	1.84	3.38	11.67
4	LabStructure	001	5	14	88	7	7	3	4.36	3.5	0.36
5	General	005	10	29	88	13	13	4	4.04	3.48	5.76
6	General	006	10	18	88	13	13	6	5.56	4.22	13.28
7	LabStructure	005	10	37	88	13	13	3	6.16	4.03	0.65
8	LabStructure	006	10	29	88	13	13	3	6.21	3.76	21.01
9	General	010	20	60	174	16	16	5	7.42	4.42	11.36
10	General	011	20	84	174	16	16	4	7.31	4.3	3.7
11	LabStructure	010	20	65	174	16	16	3	6.28	4.43	26.26
12	LabStructure	011	20	62	174	16	16	3	7.27	4.24	1.21
13	General	020	15	29	174	12	12	5	5.76	3.97	1.12
14	LabStructure	020	15	53	174	12	12	3	6.28	4.47	20.63
15	General	025	30	113	174	23	23	3	8.26	4.41	5.71
16	LabStructure	025	30	105	174	23	23	3	7.52	4.25	39.63
17	General	015	40	126	174	31	31	3	9.26	4.48	29.53
18	LabStructure	015	40	138	174	31	31	3	7.36	3.57	41.93
19	General	030	60	208	174	46	46	6	9.85	4.11	31.45
20	LabStructure	030	60	212	174	46	46	3	9.28	4.17	78.16
21	General	035	20	76	520	6	6	5	4.24	3.62	8.08
22	LabStructure	035	20	71	520	6	6	3	4.3	3.42	11.70
23	General	040	40	196	520	12	12	4	6.95	4.47	4.24
24	LabStructure	040	40	187	520	12	12	3	6.55	4.51	1.38
25	General	045	60	260	520	18	18	6	7.65	4.52	23.95
26	LabStructure	045	60	239	520	18	18	3	7.44	4.42	33.65
27	General	050	60	270	782	13	13	4	6.89	4.39	3.89
28	LabStructure	050	60	247	782	13	13	3	6.97	4.21	23.42
29	General	055	90	384	782	19	19	5	7.27	4.29	26.89
30	LabStructure	055	90	401	782	19	19	3	7.34	4.53	36.76

Table 1 The set of test instances used for the experiments. Shown are the data set the instance is taken from and the ID within that set. The following columns list the number of projects, jobs and the length of the scheduling period, followed by the number of employees, workbenches and equipment groups. The last columns contain the mean qualified employees and available workbenches per job, as well as the mean available devices per job and equipment group (only over those jobs that actually require at least one device of the group, about 10% of all jobs).

The algorithms described in Section 5 were implemented in Java 8. All experiments were performed on a Lenovo ThinkPad University T480s with an Intel Core i7-8550U (1,8 GHz), using a single thread and a timeout of ten minutes.

6.1 Parameter configuration and tuning

For parameter tuning, we used SMAC3 (Hutter et al. (2011)), version 0.10.0. Tuning was performed on a set of 30 instances chosen in the same way as, but distinct from, the test data set. In each case, we allocated a budget of 500 target algorithm runs to SMAC.

6.1.1 Tabu Search

Despite considerable effort to improve its efficiency, the JobOpt set of neighborhoods remained too large to compute the best move for each job at each step, in particular in combination with checking for tabu moves. Even the Simple set of neighborhoods resulted in quite slow search progress and correspondingly bad results, especially due to the large time windows of many jobs. For our final experiments, we used a reduced version of the latter, Simple$^-$, which only considers time slots within 5 slots of the current assignment.

The initial solution is created using a greedy construction heuristic, which iterates over the jobs in order of ascending deadline and assigns to each job the currently best values. This helps to efficiently resolve "easy" assignments already in the initial solution and focus the main part of the search on areas where good solutions are hard to find.

Regarding the choice of the length of the tabu list l, we used SMAC to find appropriate candidates. There is one additional boolean parameter, TL^{rel}. If set, the actual length of the tabu list is scaled with the number of projects in the instance. Table 2a shows the parameters and their domain, together with the best configuration found.

6.1.2 Min-Conflict

For the Min-Conflict heuristic, the restriction to a single job at each step means that both the Simple and the JobOpt neighborhoods can be explored in reasonable time. Over various solver runs, both neighborhoods achieved comparable results on the training set over several different configurations. For parameter tuning and the final evaluations, we decided to use the JobOpt neighborhood.

While MC itself does not include any parameters, there are still several possibilities for configuration. The parameters submitted to SMAC for tuning are listed on Table 2b and include the following: The initial solution - parameter *Init* - can either be generated with the same greedy algorithm employed for Tabu Search (*Greedy*), or use random values for all assignments, respecting time windows, job precedence and resource availability constraints (*Random*). Another parameter is the maximum number of moves without improvement ($MMWI$) before the search restarts from a new initial solution. Finally, p^{RW} denotes the probability at each step that a single move of random walk is performed instead of the min-conflict heuristic.

6.1.3 Simulated Annealing

During our experiments, we noted that both the number of feasible solutions found and the overall solution quality increased with longer cooling cycles (and fewer restarts or reheatings). For this reason, we used the dynamic cooling variant for tuning and the final evaluations.

The remaining parameters to tune are the initial temperature T^0, the minimum temperature T^{min} and the weight factor for hard constraint violations w^H. The parameters for SMAC and tuning results are listed in Table 2c.

Parameter	Value range	Best configuration
l	$\{50 \dots 5000\}$	1283
TL^{rel}	$\{true, false\}$	false

(a) Tabu Search

Parameter	Value range	Best configuration
Init	{Random, Greedy}	Greedy
MMWI	$\{100 \dots 10000\}$	131
p^{RW}	$\{0, 0.05, 0.1, 0.2\}$	0.1

(b) Min-Conflict + Random Walk

Parameter	Value range	Best configuration
T^0	$\{10 \dots 100\}$	69
T^{min}	$\{0.001, 0.01, 0.02, 0.05, 0.1, 0.2, 0.5, 1, 2, 5, 10\}$	0.1
w^H	$\{10, 20, 50, 100, 200\}$	10

(c) Simulated Annealing

Table 2 Tuning parameters for Tabu Search (a), Min-Conflict with Random Walk (b) and Simulated Annealing (c). The last column lists the parameter values for the best configuration found.

As for the Min-Conflict heuristic, we saw comparable results for both the Simple and JobOpt neighborhoods and opted to continue the experiments with the JobOpt neighborhood.

6.2 Evaluation

Table 3 shows a comparison of results for Tabu Search (TS), Min-Conflict with Random Walk (MC+RW) and Simulated Annealing (SA), with the configuration described in the previous section. These results are compared to those of a Constraint Programming model (CP), written in MiniZinc (Nethercote et al. (2007)) and using the solver Chuffed (Chu (2011). Details for this CP model can be found in Geibinger et al. (2019).

Due to their non-deterministic behavior, both MC+RW and SA were run 10 times on each instance, with different seeds for the (pseudo-) random number generator. The table shows the best solution found over all runs, the number of feasible solutions found, and the average penalty among all feasible solutions.

From these results, it can be seen that instances can be split into two groups: For small instances with 20 projects or less (instances 1-14 and 21,22), good solutions could be found in most cases. This was much more difficult for larger instances with more than 20 projects.

TS was unable to find feasible solutions for most large instances and also for a few small instances within the given time. Moreover, even where it does find solutions without conflicts, the resulting penalty is often only slightly better

#	TS	MC+RW			SA			CP	
		Best	#Feas.	Avg	Best	#Feas.	Avg		
1	98	98	10	98	98	10	98	98	*
2	75	73	10	73	73	10	73	73	*
3	152	149	10	149.3	152	10	156.4	149	*
4	116	105	10	105.3	105	10	105	105	*
5	305	285	10	286.9	287	10	300.1	283	*
6	174	162	10	162.4	177	10	192.2	162	*
7	386	327	10	331.7	307	10	307.4	307	*
8	361	314	10	323.5	310	10	312	310	*
9	714	625	10	648.8	501	10	502.7	501	*
10	721	725	10	751	564	10	565.3	892	
11	1041	993	10	1049.9	874	10	879	856	*
12	-	749	10	768.9	663	10	668	713	
13	388	340	10	341	352	10	352.1	340	*
14	483	450	10	457.9	422	10	425.7	420	*
15	-	1800	4	1841	1087	10	1090.6	1653	
16	1426	1357	8	1429.8	1143	10	1155.2	1561	
17	1418	1381	10	1410.9	1195	10	1234	1382	
18	1772	1688	10	1760.7	1364	10	1375.3	1822	
19	-	2675	6	2735.3	2277	10	2337	2650	
20	-	2853	2	2898.5	2312	10	2360.6	2892	
21	1053	908	10	1007.7	683	10	686.6	930	
22	1160	1033	10	1079.7	767	10	771.9	839	
23	-	-	0	-	2393	6	2476.3	3531	
24	-	2484	2	2664	1808	10	1852.5	2454	
25	-	-	0	-	2908	8	3050.9	3281	
26	-	-	0	-	2724	10	2805	3899	
27	3356	3372	3	3405.7	2176	10	2191.3	3146	
28	2820	2585	10	2617.2	2367	10	2375.8	2569	
29	-	5334	2	5406.5	4208	9	4428.4	4548	
30	-	6453	10	6646.8	4828	9	4896.8	5905	

Table 3 Comparison of results. Min-conflict with random walk and Simulated Annealing were both run 10 times each, with different seeds. Columns *Best* contain the best solution found, *#Feas.* the number of feasible solutions found (out of 10) and *Avg* the average penalty over all feasible solutions. Solutions for CP marked with "*" are optimal for the instance.

than what was already achieved with the greedy construction heuristic. Results for MC+RW are comparable, although feasible solutions were found for some additional instances.

SA performed much better and found feasible solutions in more than 97% of all runs. Also the solution quality is consistently better than with the other two heuristics, except for some small instances.

The CP model could find feasible solutions for all of the instances already within one minute. Within the full time limit, it could prove optimality for 12 of the 16 small instances.

Compared to the results for SA, CP slightly outperforms SA on those instances where it could find optimal solutions. However, SA produced better results for every single other instance, sometimes by more than 30%.

The differences between the results for small and large instances indicates that the number of projects (and thus jobs) is the main factor in determining the difficulty of an instance. In contrast, neither the number of time slots in the

#	SA			CP	
	Best	**#Feas.**	**Avg**		
1	98	5	98	98	*
2	73	5	73	73	*
3	151	5	152.4	149	*
4	105	5	105.4	105	*
5	292	5	300.8	283	*
6	180	5	191	162	*
7	307	5	307	307	*
8	310	5	310.2	310	*
9	501	5	501	501	*
10	564	5	564.6	740	
11	872	5	873.4	856	*
12	663	5	664.8	656	*
13	352	5	353.6	340	*
14	422	5	422.2	420	*
15	1086	5	1086.8	1647	
16	1141	5	1142.2	1561	
17	1195	5	1206	1284	
18	1360	5	1361	1820	
19	2196	5	2233	2650	
20	2261	5	2284.6	2888	
21	683	5	683.6	679	*
22	765	5	766.6	765	*
23	2200	5	2276	3487	
24	1782	5	1799.4	2452	
25	2598	5	2737.8	3278	
26	2605	5	2633.4	3894	
27	2155	5	2159.8	3130	
28	2333	5	2341.2	2569	
29	4038	5	4204.8	4539	
30	4601	5	4636.6	5904	

Table 4 Comparison of results with a timeout of one hour. Simulated Annealing was run 5 times per instance with different seeds. Column *Best* contains the best solution found, *#Feas.* the number of feasible solutions found (out of 5) and *Avg* the average penalty over all feasible solutions. Solutions for CP marked with "*" are optimal for the instance.

scheduling period, nor the number of resources available for each job seem to have a decisive impact on the difficulty of an instance.

6.2.1 Additional runtime

We also repeated our experiments with the same configuration for Simulated Annealing with a longer timeout of one hour. Table 4 shows the results of these experiments, again compared with the results for the CP model (also with a timeout of one hour).

With the increased time budget, feasible solutions could be found for all instances. As with the shorter time limit, the solutions produced by SA for all instances where CP could not find optimal solutions are better than those found by CP, in some cases by more than 30%. In particular, this includes all large instances.

For those small instances, where optimal solutions have been found, SA achieved solutions with the same or very close penalties.

Compared to the results with a shorter time limit, the penalty for large instances (>20 projects) has improved by nearly 3.8% on average with SA. CP could find optimal solutions for three of the four remaining small instances. However, the results for large instances improved by less than 1% on average.

7 Conclusions

In this article, we have described the TLSP-S problem, which is a complex extension to existing RCPSP variants based on real-world requirements.

We have introduced a flexible framework for solving this problem, which supports several metaheuristic solvers and provides multiple options for configuration and extensions. Using this framework, we have shown that Simulated Annealing can be used to provide high quality solutions, and outperforms a Constraint Programming model using a state-of-the-art exact solver for larger and practical instances both under strict time limits and with longer runtimes.

Regarding future work, we plan to investigate whether these results can also be transferred to the TLSP, which combines TLSP-S with an additional grouping stage. A promising direction of research also seems the combination of both local search and CP-based approaches, in the form of hybrid algorithms or large neighborhood search, to combine the advantages of both methods and further improve the results for TLSP-S.

Acknowledgements The financial support by the Austrian Federal Ministry for Digital and Economic Affairs and the National Foundation for Research, Technology and Development is gratefully acknowledged.

References

Ahmeti A, Musliu N (2018) Min-conflicts heuristic for multi-mode resource-constrained projects scheduling. In: Proceedings of the Genetic and Evolutionary Computation Conference, ACM, pp 237–244

Asta S, Karapetyan D, Kheiri A, Özcan E, Parkes AJ (2016) Combining monte-carlo and hyper-heuristic methods for the multi-mode resource-constrained multi-project scheduling problem. Information Sciences 373:476 – 498, DOI https://doi.org/10.1016/j.ins.2016.09.010, URL `http://www.sciencedirect.com/science/article/pii/S0020025516307502`

Avar M, Najafi BA (2017) Multi-mode resource-constrained project scheduling and soft and hard time windows for ending activities. QUID: Investigación, Ciencia y Tecnología (1):2342–2354

Bartels JH, Zimmermann J (2009) Scheduling tests in automotive r&d projects. European Journal of Operational Research 193(3):805 – 819, DOI 10.1016/j.ejor.2007.11.010

Bellenguez O, Néron E (2005) Lower bounds for the multi-skill project scheduling problem with hierarchical levels of skills. In: Burke E, Trick M (eds) Practice and Theory of Automated Timetabling V, Springer Berlin Heidelberg, Berlin, Heidelberg, pp 229–243, DOI 10.1007/11593577_14

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, DOI https://doi.org/10.1016/S0377-2217(02)00761-0, URL http://www.sciencedirect.com/science/article/pii/S0377221702007610, sequencing and Scheduling

Brucker P, Drexl A, Möhring R, Neumann K, Pesch E (1999) Resource-constrained project scheduling: Notation, classification, models, and methods. European Journal of Operational Research 112(1):3 – 41, DOI https://doi.org/10.1016/S0377-2217(98)00204-5, URL http://www.sciencedirect.com/science/article/pii/S0377221798002045

Chu G (2011) Improving combinatorial optimization. PhD thesis, University of Melbourne, Australia, URL http://hdl.handle.net/11343/36679

Dauzère-Pérès S, Roux W, Lasserre J (1998) Multi-resource shop scheduling with resource flexibility. European Journal of Operational Research 107(2):289 – 305, DOI https://doi.org/10.1016/S0377-2217(97)00341-X, URL http://www.sciencedirect.com/science/article/pii/S037722179700341X

Drexl A, Nissen R, Patterson JH, Salewski F (2000) Progen/πx – an instance generator for resource-constrained project scheduling problems with partially renewable resources and further extensions. European Journal of Operational Research 125(1):59 – 72, DOI https://doi.org/10.1016/S0377-2217(99)00205-2, URL http://www.sciencedirect.com/science/article/pii/S0377221799002052

Elmaghraby SE (1977) Activity networks: Project planning and control by network models. John Wiley & Sons

Geibinger T, Mischek F, Musliu N (2019) Investigating constraint programming for real world industrial test laboratory scheduling. In: to appear in Proceeding of the Sixteenth International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research (CPAIOR 2019)

Glover F (1989) Tabu search—part 1. ORSA Journal on Computing 1(3):190–206, DOI 10.1287/ijoc.1.3.190, URL http://dx.doi.org/10.1287/ijoc.1.3.190, http://dx.doi.org/10.1287/ijoc.1.3.190

Gonçalves J, Mendes J, Resende M (2008) A genetic algorithm for the resource constrained multi-project scheduling problem. European Journal of Operational Research 189(3):1171 – 1190, DOI https://doi.org/10.1016/j.ejor.2006.06.074, URL http://www.sciencedirect.com/science/article/pii/S0377221707005929

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):1 – 14, DOI https://doi.org/10.1016/j.ejor.2009.11.005, URL http://www.sciencedirect.com/science/article/pii/S0377221709008558

Hutter F, Hoos HH, Leyton-Brown K (2011) Sequential model-based optimization for general algorithm configuration. In: International Conference on Learning and Intelligent Optimization, Springer, pp 507–523

Kirkpatrick S, Gelatt CD, Vecchi MP (1983) Optimization by simulated annealing. Science 220(4598):671–680, DOI 10.1126/science.220.4598.671, URL https://science.sciencemag.org/content/220/4598/671, https://science.sciencemag.org/content/220/4598/671.full.pdf

Laurent A, Deroussi L, Grangeon N, Norre S (2017) A new extension of the rcpsp in a multi-site context: Mathematical model and metaheuristics. Computers & Industrial Engineering 112:634 – 644, DOI https://doi.org/10.1016/j.cie.2017.07.028, URL http://www.sciencedirect.com/science/article/pii/

S0360835217303303

Mika M, Waligóra G, Węglarz J (2015) Overview and state of the art. In: Schwindt C, Zimmermann J (eds) Handbook on Project Management and Scheduling Vol.1, Springer International Publishing, Cham, pp 445–490, DOI 10.1007/978-3-319-05443-8_21, URL https://doi.org/10.1007/978-3-319-05443-8_21

Minton S, Johnston MD, Philips AB, Laird P (1992) Minimizing conflicts: a heuristic repair method for constraint satisfaction and scheduling problems. Artificial Intelligence 58:161–205

Mischek F, Musliu N (2018) The test laboratory scheduling problem. Technical report, Christian Doppler Laboratory for Artificial Intelligence and Optimization for Planning and Scheduling, TU Wien, CD-TR 2018/1

Nethercote N, Stuckey PJ, Becket R, Brand S, Duck GJ, Tack G (2007) Minizinc: Towards a standard CP modelling language. In: Principles and Practice of Constraint Programming - CP 2007, 13th International Conference, CP 2007, Providence, RI, USA, September 23-27, 2007, Proceedings, pp 529–543, DOI 10.1007/978-3-540-74970-7_38

Polo Mejia O, Anselmet MC, Artigues C, Lopez P (2017) A new RCPSP variant for scheduling research activities in a nuclear laboratory. In: 47th International Conference on Computers & Industrial Engineering (CIE47), Lisbonne, Portugal, p 8p., URL https://hal.laas.fr/hal-01630977

Salewski F, Schirmer A, Drexl A (1997) Project scheduling under resource and mode identity constraints: Model, complexity, methods, and application. European Journal of Operational Research 102(1):88 – 110, DOI https://doi.org/10.1016/S0377-2217(96)00219-6, URL http://www.sciencedirect.com/science/article/pii/S0377221796002196

Schwindt C, Trautmann N (2000) Batch scheduling in process industries: an application of resource–constrained project scheduling. OR-Spektrum 22(4):501–524

Szeredi R, Schutt A (2016) Modelling and solving multi-mode resource-constrained project scheduling. In: Principles and Practice of Constraint Programming - 22nd International Conference, CP 2016, Toulouse, France, September 5-9, 2016, Proceedings, pp 483–492, DOI 10.1007/978-3-319-44953-1_31, URL https://doi.org/10.1007/978-3-319-44953-1_31

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, DOI 10.1007/s10951-014-0402-0, URL https://doi.org/10.1007/s10951-014-0402-0

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(3):177 – 205, DOI 10.1016/j.ejor.2010.03.037

Young KD, Feydy T, Schutt A (2017) Constraint programming applied to the multi-skill project scheduling problem. In: Beck JC (ed) Principles and Practice of Constraint Programming, Springer International Publishing, Cham, pp 308–317