Investigating Constraint Programming for Real World Industrial Test Laboratory Scheduling

Tobias Geibinger, Florian Mischek, and Nysret Musliu

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

Abstract. In this paper we deal with a complex real world scheduling problem closely related to the well-known Resource-Constrained Project Scheduling Problem (RCPSP). The problem concerns industrial test laboratories in which a large number of tests has to be performed by qualified personnel using specialised equipment, while respecting deadlines and other constraints. We present different constraint programming models and search strategies for this problem. Our approaches are evaluated using CP and MIP solvers on a set of generated instances of different sizes. With our best approach we could find feasible solutions for instances that are generated based on real-world test laboratory problems.

1 Introduction

Project scheduling includes various problems of high practical relevance. Such problems arise in many areas and include different constraints and objectives. Usually project scheduling problems require scheduling of a set of project activities over a period of time and assignment of resources to these activities. Typical constraints include the time windows for activities, precedence constraints between the activities, assignment of appropriate resources etc. The aim is to find feasible schedules that optimize several criteria such as the minimization of total completion time.

In this paper we investigate solving a real-world project scheduling problem that arises in an industrial test laboratory of a large company. This problem, Industrial Test Laboratory Scheduling (TLSP), which is an extension of the well known Resource-Constrained Project Scheduling Problem (RCPSP), was originally described in [14, 13]. It consists of a grouping stage, where smaller activities (tasks) are joined into larger jobs, and a scheduling stage, where those jobs are scheduled and have resources assigned to them. In this work, we deal with the second stage and assume that a grouping of tasks into jobs is already provided. Since we focus on the scheduling part, we denote the resulting problem TLSP-S.

The investigated problem has several features of previous project scheduling problems in the literature, but also includes some specific features imposed by

the real-world situation, which have rarely been studied before. Among others, these include heterogeneous resources, with availability restrictions on the activities each unit of a resource can perform. While work using similar restrictions exists ([5, 23]), most problem formulations either assume homogeneous, identical units of each resource or introduce additional activity modes for each feasible assignment, which quickly becomes impractical for higher resource requirements. Another specific feature of TLSP(-S) is that of linked activities, which require identical assignments on a subset of the resources. To the best of our knowledge, a similar concept appears only in [18], where modes should be identical over subsets of all activities. We also deal with several non-standard objectives instead of the usual makespan minimization, which arise from various business objectives of our industrial partner. Most notably, we try to minimize the total completion time of each project, i.e. the time between the start of the first and the end of the last job in the project.

In practice, exact solutions for this problem are desired especially in situations where it is necessary to check if a feasible solution exists at all. In the application that we consider, checking quickly if activities of additional projects can be added on top of an existing schedule is very important. In this paper we investigate exact methods for solving this problem. Although it is known from previous papers [21, 23] that constraint programming techniques can give good results for similar project scheduling problems, it is an interesting question if Constraint Programming (CP) techniques can also solve TLSP-S that includes additional features and larger instances.

We provide a constraint model for our problem by exploiting some previous ideas for a similar problem from [21, 23] and extend it to model the additional features of TLSP-S. This includes, for example, the handling of the problem specific differences discussed above but also new redundant constraints as well as search procedures tailored to the problem. Using the MiniZinc [15] constraint programming language we experiment with various strategies involving the formulation of resource constraints, the reduction of the search space, and search procedures based on heuristics.

Our final experiments show that constraint programming techniques can reach very good results for realistic instances and outperform MIP solvers on the same model. Our results strengthen the conclusion of previous studies and show that CP technology can be applied successfully for solving large project scheduling problems.

The rest of the paper is organised as follows. In the next Section we give the related work. Section 3 introduces the problem that we investigate in this paper. A constraint model is given in Section 4. Experimental results are presented in Section 5 and the last section gives conclusions.

2 Literature Overview

The Resource-Constrained Project Scheduling Problem (RCPSP) has seen vast amounts of work over the last decades. For a comprehensive overview over pub-

lications dealing with this problem and its many variants, we refer to surveys e.g. by Brucker et al. [3], Hartmann and Briskorn [9], or Mika et al. [12].

Of particular interest for the problem treated in this work are various extensions to the classical RCPSP.

Multi-Mode RCPSP (MRCPSP) formulations allow for activities that can be scheduled in one of several modes. This variant has been extensively studied since 1977 [7], we refer to the surveys by Węglarz et al. [22] and Hartmann and Briskorn [9]. A good example of a CP-Model for the MRCPSP was given by Szeredi and Schutt [21].

Many formulations, including TLSP, make use of release dates, due dates, or both. An example of this can be found in [6]. Further relevant extensions deal with multi-project formulations, including alternative objective functions (e.g. [17]). Usually, the objective in (variants of) RCPSP is the minimization of the total makespan [9]. However, also other objective values have been considered. Of particular relevance to TLSP are objectives based on total completion time and multi-objective formulations (both appear in e.g. [16]). Salewski et al.[18] include constraints that require several activities to be performed in the same mode. This is similar to the concept of linked jobs introduced in the TLSP.

RCPSP itself and most variants assume that individual units of each resource are identical and interchangeable. A problem closely related to TLSP-S is Multi-Skill RCPSP (MSPSP), first introduced by Bellenguez and Néron [2]. In this problem, each resource unit possesses certain skills, and an activity can only have those resources with the required skills assigned to it. This is similar to the availability restrictions on resources that appear in TLSP. Just like for our problem, they also deal with the problem that while availability restrictions could be modeled via additional activity modes corresponding to feasible resource assignments (e.g. in [17, 20, 1]), this is intractable due to the large number of modes that would have to be generated [2]. To the best of our knowledge, the best results for the MSPSP problem have been achieved by Young et al. [23], who use a CP model to solve the problem.

Bartels and Zimmermann [1] describe a problem for scheduling tests of experimental vehicles. It contains several constraints that also appear in similar form in TLSP-S, but includes a different resource model and minimises the number of experimental vehicles used.

3 Problem Description

As mentioned before, we deal with a variant of TLSP [14], 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 introduce the TLSP-S problem.

Each instance consists of a scheduling period of h discrete *time slots*. Further, it lists 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. These each represent 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 j has several properties[1]:

- A time window, given via a *release date* α_j and a *due date* ω_j. In addition, it has a *target date* $\bar{\omega}_j$, which is similar to the due date, 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 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* $E_j^{Pr} \subseteq E_j$.
  - The number of *required workbenches* $r_j^{Wb} \in \{0, 1\}$. If a workbench is required, it must be chosen from the *available workbenches* $B_j \subseteq B$.
  - 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*.
  - An *initial mode* $\dot{m}_j$.
  - An *initial starting time slot* $\dot{s}_j$.
  - *Initial resource assignments*: For each employee $e \in E$, the boolean parameter $\dot{a}_{ej}^{Em}$ indicates whether e is initially assigned to j. Analogously, $\dot{a}_{bj}^{Wb}$ and $\dot{a}_{dj}^{Eq}$ perform the same function for each workbench $b \in B$ and each device $d \in G_g, g \in G^*$, 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 fulfill the following conditions:

- It must have a preassigned mode.
- Its start time must be set to 1.

[1] 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.

– It must have initial resource assignments fulfilling all requirements.

The initial assignments of a started job must not be changed in the solution.

A complete description of all constraints of the original model can be found in [14]. The hard and soft constraints that we consider for the TLSP-S will be described in the next section, where we will introduce the CP model.

The aim for this problem is to find an assignment of a mode, time slot and resources to each (given) job, such that all hard constraints are fulfilled and the violation of soft constraints is minimized.

4 Constraint Programming Model

We developed our model using the solver-independent modeling language MiniZinc [15]. Using MiniZinc we can easy compare different solvers. Furthermore, previous studies have shown that CP gives very good results for similar project scheduling problems. Most notably, the Multi-Skill and Multi-Mode RCPSP approaches by Young et al. [23] and Szeredi et al. [21] respectively. MiniZinc also enables the use of user defined search strategies, which were shown to be very effective for Multi-Skill RCPSP (MSPSP) [23]. For both scheduling problems, the LCG solver Chuffed [4] was able to achieve very good results.

In order to represent a solution for the scheduling problem we use the following decision variables. The start time variable s_j assigns a start time to each job j. Similarly, for each job j mode variable m_j assigns it a mode. For resource assignments we need the following variables: For each job j, the variable a_{ej}^{Em} is set to 1 if employee e is assigned to j and 0 otherwise, the variable a_{bj}^{Wb} is 1 if j is performed on workbench b and 0 otherwise, and the variable a_{dj}^{Eq} is 1 if device d is used by j and 0 otherwise.

4.1 Basic Hard Constraints

The following constraints follow directly from the problem definition.

$$s_j \geq \alpha_j \ \wedge \ (s_j + d_{m_j j}) \leq \omega_j \qquad j \in J \tag{1}$$

$$s_j \geq (s_k + d_{m_k k}) \qquad j \in J,\ k \in \mathcal{P}_j \tag{2}$$

$$m_j \in M_j \qquad j \in J \tag{3}$$

$$a_{ej}^{Em} = 1 \ \rightarrow \ e \in E_j \qquad j \in J,\ e \in E \tag{4}$$

$$a_{bj}^{Wb} = 1 \ \rightarrow \ b \in B_j \qquad j \in J,\ b \in B \tag{5}$$

$$a_{dj}^{Eq} = 1 \ \rightarrow \ d \in G_{gj} \qquad j \in J,\ g \in G^*,\ d \in G_g \tag{6}$$

$$\sum_{e \in E} a_{ej}^{Em} \ = \ r_{m_j}^{Em} \qquad j \in J \tag{7}$$

$$\sum_{b \in B} a_{bj}^{Wb} \ = \ r_{j}^{Wb} \qquad j \in J \tag{8}$$

$$\sum_{d \in G_g} a_{dj}^{Eq} = r_{gj}^{Eq} \qquad j \in J,\ g \in G^* \tag{9}$$

$$a_{ej}^{Em} = a_{ek}^{Em} \qquad j \in J,\ k \in L_j,\ e \in E \tag{10}$$

$$\begin{aligned} s_j = 1 \ \wedge\ m_j = \dot{m}_j \ \wedge \qquad & j \in J^S,\ e \in E,\ b \in B, \\ a_{ej}^{Em} = \dot{a}_{ej}^{Em} \ \wedge\ a_{bj}^{Wb} = \dot{a}_{bj}^{Wb} \ \wedge\ a_{dj}^{Eq} = \dot{a}_{dj}^{Eq} \qquad & g \in G^*,\ d \in G_g \end{aligned} \tag{11}$$

Constraint (1) makes sure that each job is executed in its time window, (2) enforces that the prerequisite jobs of a job are always completed before it starts. Constraints (3–6) ensure that assigned modes, employees, workbenches, and devices are available for the respective job. In order to make sure that each job has exactly as many resources as required, we have constraints (7–9). Furthermore, we need constraint (10) to make sure that linked jobs are assigned to the same employees and constraint (11) to fix the resource assignments of jobs which are already started.

The above set of constraints is however not enough to ensure a valid solution. Additionally, we have to consider constraints which enforce that no resource (employee, workbench, or equipment) is assigned to two or more jobs at the same time. Like it was the case with MSPSP [23], the constraints used for modeling those *unary resource requirements* have a tremendous impact on the practicability of the model and in the next subsection we will present different options for modeling such constraints.

4.2 Unary Resource Constraints

We will now present three different approaches for modeling unary resource constraints, each of which is designed with CP solvers in mind. Two of those three quickly proved to be impractical for our problem.

Time-indexed approach The probably most straight forward way to model the non-overuse of any resource at any given time is captured by the following constraints.

$$\sum_{j \in J, s_j \le t < (s_j + d_{m_j j})} a_{ej}^{Em} \le 1 \qquad e \in E,\ 1 \le t \le h \tag{12}$$

$$\sum_{j \in J, s_j \le t < (s_j + d_{m_j j})} a_{bj}^{Wb} \le 1 \qquad b \in B,\ 1 \le t \le h \tag{13}$$

$$\sum_{j \in J, s_j \le t < (s_j + d_{m_j j})} a_{dj}^{Eq} \le 1 \qquad g \in G^*,\ d \in G_g,\ 1 \le t \le h \tag{14}$$

The number of constraints generated by MiniZinc based on (12–14) is of course directly dependent on the planning horizon h and the total number of resources. Because of the long compilation time and the high computer resource consumption, it quickly became immanent that for our larger instances the time-indexed

approach is not efficient. This is of course not surprising since Young et al. came to a similar conclusion for MSPSP [23]. Hence, we discarded this option after some preliminary testing.

Overlap constraint For MSPSP, Young et al. [23] achieved their best results using a so-called *order constraint*. This constraint basically enforces that two activities cannot overlap in their execution when they use a common resource. During the initial modeling phase we tried a very similar approach. First, we introduced a predicate `overlap`:

$$\texttt{overlap}(j,k) := s_k < (s_j + d_{m_j j}) \ \wedge \ (s_k + d_{m_k k}) > s_j$$

In MSPSP, resources are assigned to activities with respect to the needed skill of the activity. For the overlap constraint it is not important which skill requirement the resource contributes to, so Young et al. [23] had to introduce an auxiliary variable to express that a resource is used by an activity. We on the other hand assign the resources directly and thus can model our *overlap constraint* without any new variables.

$$\begin{aligned}\texttt{overlap}(j,k) \rightarrow (\ & \bigwedge_{e \in E} (\neg a_{ej}^{Em} \vee \neg a_{ek}^{Em}) \ \wedge \\ & \bigwedge_{b \in B} (\neg a_{bj}^{Wb} \vee \neg a_{bk}^{Wb}) \ \wedge \\ & \bigwedge_{g \in G^*, d \in G_g} (\neg a_{dj}^{Eq} \vee \neg a_{dk}^{Eq}) \ ) \qquad j,k \in J, \ j \neq k, \\ & \qquad\qquad \alpha_k < \omega_j \wedge \omega_k > \alpha_j \end{aligned} \tag{15}$$

Just like with the time-indexed approach, it turned out that the overlap constraint produced too many constraints and was thus impractical for larger instances. This is interesting because Young et al. had no such problems, but their biggest instances only had 60 resources and 42 activities, whereas we have instances with more than 300 resources and jobs respectively. It should however be noted that Young et al. [23] reduced the number of generated constraints by considering only *unrelated* activity pairs, i.e. activities which do not depend on the execution of each other (related activities can obviously never overlap). We on the other hand generate constraints for all pairs of jobs which are allowed to overlap based on their release and due dates. Comparing only unrelated jobs requires the computation of the transitive closure of the job precedence relation and because our instances have a lot of unrelated jobs, we don't expect any significant improvement.

Cumulative constraints Another way to model the unary resource constraints is to use a global constraint like `cumulative`. The `cumulative` constraint takes as input the start times, durations and resource requirements of a list of jobs and ensures that their resource assignments never exceed a given bound. This

is of course a perfect way to enforce non overload of any resource and both Multi-Skill as well as Multi-Mode RCPSP have efficient models which make use of `cumulative` in some way [23][21]. In order to enforce the non-overload of any resource we need three constraints (one for each resource type).

$$\texttt{cumulative}((s_j)_{j\in J},\ (d_{m_j j})_{j\in J},\ (a_{ej}^{Em})_{j\in J},\ 1) \qquad e \in E \tag{16}$$

$$\texttt{cumulative}((s_j)_{j\in J},\ (d_{m_j j})_{j\in J},\ (a_{bj}^{Wb})_{j\in J},\ 1) \qquad b \in B \tag{17}$$

$$\texttt{cumulative}((s_j)_{j\in J},\ (d_{m_j j})_{j\in J},\ (a_{dj}^{Eq})_{j\in J},\ 1) \qquad g \in G^*, d \in G_g \tag{18}$$

In difference to our first two modeling approaches, this one turned out to scale pretty well. Since the others performed so poorly on large instances, the rest of our experiments were performed with the `cumulative` unary resource constraints.

4.3 Soft Constraints

There are several soft constraints in our problem definition. Since MiniZinc has no direct support for soft constraints, we define them as sums which should be minimised. Those sums are given as follows.

For each job, violating its target date should be avoided:

$$s_1 = w_1 \cdot \sum_{j\in J} max(s_j + d_{m_j j} - \bar{\omega}_j, 0)$$

Furthermore, we want to prefer solutions where the assigned employees of a job are taken from the set of preferred employees:

$$s_2 = w_2 \cdot \sum_{j\in J} \sum_{e\in(E\setminus E_j^{Pr})} a_{ej}^{Em}$$

Aside from soft constraints for jobs, we also have ones regarding projects. For each project, the total number of employees assigned to it should be minimised:

$$s_3 = w_3 \cdot \sum_{p\in P} \sum_{e\in E} (\sum_{j\in J_p} a_{ej}^{Em}) > 0$$

Project durations should be as small as possible:

$$s_4 = w_4 \cdot \sum_{p\in P} max_{j\in J_p}(s_j + d_{m_j j}) - min_{j\in J_p}(s_j)$$

Our last soft constraint compares how the solution changes the base schedule with regard to the additional employees assigned to a project:

$$s_5 = w_5 \cdot \sum_{p\in P} \sum_{e\in E} \sum_{j\in J_p} a_{ej}^{Em} - \dot{a}_{ej}^{Em}$$

The objective of the search is then given by $min\ \sum_{1\le i\le 5} s_i$.

At the moment, the values of the weights w_i $(1 \le i \le 5)$ are being determined in correspondence with a real-world laboratory. Currently all these weights are set to 1.

4.4 Redundant Constraints

Finding good redundant constraints for our problem proved to be very hard since the search space is usually very big and at the beginning of the search there is little knowledge about the final duration of the jobs. To combat this we introduced a relaxed `cumulative` constraint enforcing a global resource bound.

$$\begin{aligned}\texttt{cumulative}((s_j)_{j\in J},\\ &(min_{m\in M}(d_{m_j j}))_{j\in J},\\ &(max_{m\in M_j}(r_m^{Em}) + r_j^{Wb} + \sum_{g\in G^*} r_{gj}^{Eq}\ )_{j\in J},\\ &|E| + |B| + \sum_{g\in G^*} |G_g| \qquad)\end{aligned} \tag{19}$$

This enables the search to discard scheduling options which are impossible regardless of the chosen modes early on.

Nonetheless, we can of course also formulate more straightforward `cumulative` constraints which ensure the global resource bounds at any point in time.

$$\texttt{cumulative}((s_j)_{j\in J},\ (d_{m_j j})_{j\in J},\ (r_m^{Em})_{j\in J},\ |E|) \tag{20}$$

$$\texttt{cumulative}((s_j)_{j\in J},\ (d_{m_j j})_{j\in J},\ (r_j^{Wb})_{j\in J},\ |B|) \tag{21}$$

$$\texttt{cumulative}((s_j)_{j\in J},\ (d_{m_j j})_{j\in J},\ (r_{gj}^{Eq})_{j\in J},\ |G|) \qquad g \in G^* \tag{22}$$

Given the large search space, trying to restrict the scope of the decision variables seems like a worthwhile idea. We achieve this by using *global cardinality constraints*. Those constraints allow us to give tight bounds for the total number of resources which should be used.

$$\begin{aligned}\texttt{global\_cardinality\_low\_up}((a_{ej}^{Em})_{e\in E, j\in J},\ 1,\ &\sum_{j\in J} min_{m\in M_j}(r_m^{Em}),\\ &\sum_{j\in J} max_{m\in M_j}(r_m^{Em}))\end{aligned} \tag{23}$$

$$\texttt{global\_cardinality\_low\_up}((a_{bj}^{Wb})_{b\in B, j\in J},\ 1,\ \sum_{j\in J} r_j^{Wb}, \sum_{j\in J} r_j^{Wb}) \tag{24}$$

$$\begin{aligned}\texttt{global\_cardinality\_low\_up}((a_{dj}^{Eq})_{g\in G^*, d\in G_g, j\in J},\ 1,\ &\sum_{j\in J}\sum_{g\in G^*} r_{gj}^{Eq},\\ &\sum_{j\in J}\sum_{g\in G^*} r_{gj}^{Eq})\end{aligned} \tag{25}$$

Constraint (23) enforces that no more employees can be assigned than the sum of the highest possible employee requirements and no less than the sum of the

minimum requirements. The other two constraints ensure that the number of assigned workbenches and equipment is tightly bounded by the cumulative requirement of all jobs.

4.5 Search Strategies

During initial testing it quickly became immanent that the default search strategy of Chuffed (or Gecode) was not even able to find feasible solutions for most instances. This was not surprising since Young et al. [23] already had a similar issue with MSPSP. However, they were able to improve their results drastically by employing a new MiniZinc search annotation called `priority_search` which is supported by Chuffed [8]. Based on their research we have experimented with four slightly different versions of `priority_search`:

(i) `ps_startTimeFirst_aff`
(ii) `ps_startTimeFirst_ff`
(iii) `ps_modeFirst_aff`
(iv) `ps_modeFirst_ff`

All four search strategies branch over the jobs and their resource assignments. The order of the branching is the same for all strategies and is determined by the smallest possible start times of the jobs in ascending order. For each branch, searches (i) and (ii) assign the smallest start time to the selected job followed by assigning it the mode which minimises the job duration. Search procedures (iii) and (iv) start with the mode assignment and then assign the start time. Once the start time and the mode have been assigned for the selected job, all of the search strategies make resource assignments for the job. Searches (i) and (iii) start by assigning available resources to job, whereas (ii) and (iv) start by setting assignments which are not available to the job to zero.

5 Experiments and Comparison

We ran our experiments on a benchmark server with two Intel Xeon CPU E5345 each with max. 2.33GHz and 48GB RAM. Since all of the solvers we experimented with are single threaded, we usually ran two sets of benchmarks in parallel. We used MiniZinc 2.2.1 [15] with Chuffed 0.10 [4] and CPLEX 12.8.0 [10]. We have also experimented with Gecode [19] as an alternative CP solver, but it quickly proved to be inferior to Chuffed even when run with multiple threads. Furthermore, we also tested our best model with Gurobi 8.1 [11], but the solver crashed on larger instances. When we removed global cardinality constraints Gurobi worked, but since removing them would make any comparison impossible we decided to drop the solver completely. It can however be said that Gurobi did not offer any improvements over Chuffed or CPLEX on smaller instances. Regarding comparison to other approaches in the literature, to the best of our knowledge no solutions exist yet for the problem we consider in this paper.

5.1 Instances

We use a total of 30 randomly generated instances (based on real-life situations) of different sizes for our experiments. A summary of the instances grouped by their size is given in Table 1. The instances all have three modes: a *single* mode requiring only one employee, a *shift* mode which requires two employees but has a reduced duration, and an *external* mode that requires no employees at all. In general, jobs can be done in single mode or optionally in shift mode. Some instances however also include jobs which can only be performed in external mode. Also, in the test instances the initial assignments are restricted to jobs which are already started or are fixed to their current value via availability restrictions and time windows.

While all of the instances were generated randomly, they are still modelled after real-world scenarios. Half of the instances are modelled very closely to a real-world laboratory, whereas the other half is more general and makes full use of the problem features. The details of how this generation works as well as the exact differences between the laboratory instances and general instances are given in [14]. Furthermore, our 30 instances are a selection from a total of 120 instances given in the report. We chose the first two instances of each size (scheduling horizon and number of projects) and two additional instances for the 3 smallest sizes. This selection was necessary, because of the long time it would have taken to experiment with all 120 instances. Since those instances were generated with the full TLSP in mind and in TLSP-S we take the initial job grouping as fixed and unchangeable, the instances had to be converted. This is achieved by viewing the jobs as the smallest planning unit and assigning the job parameters – which are defined by the tasks contained in the job in TLSP – directly to the jobs.

	no. of instances	h	$\|P\|$	$\|J\|$	$\|E\|$	$\|B\|$	$\|G^*\|$	$\|\bigcup_{g\in G^*} G_g\|$
1	8	89	5-10	7-37	7-13	7-13	3-4	6-148
2	12	175	15-60	29-212	12-46	12-46	3-6	16-271
3	6	521	20-60	71-260	6-18	6-18	3-6	18-218
4	4	783	60-90	247-401	13-19	13-19	3-5	16-284

Table 1. Instances Summary

5.2 Results

Table 2 shows the comparison of search procedures described in Section 4.5. The column "♯ sat" shows for how many instances the model-search combination found feasible solutions, whereas the "♯ opt" is the number of instances solved to optimality. Furthermore, the values in the column "cum. obj" show the cumulative objective value of all instances, and "avg. rt sat" is the average time it took to find the first feasible solution.

Constraints	Search	♯ sat	♯ opt	cum. obj.	avg. rt sat
(1-11),(16-18),(19-25)	Default	13	5	–	–
(1-11),(16-18),(19-25)	`ps_modeFirst_ff`	**30**	**7**	53183	35.22s
(1-11),(16-18),(19-25)	`ps_modeFirst_aff`	**30**	**7**	53388	35.15s
(1-11),(16-18),(19-25)	`ps_startTimeFirst_ff`	**30**	**7**	**50009**	30.14s
(1-11),(16-18),(19-25)	`ps_startTimeFirst_aff`	**30**	**7**	**50009**	30.11s
(1-11),(16-18),(20-25)	`ps_startTimeFirst_aff`	**30**	**7**	50036	**29.68s**

Table 2. Priority Search Experiments (Runtime 30m)

Each model was run using Chuffed with free search enabled. Free search alternates between user-defined and activity-based search on each restart. The time limit was set to 30 minutes for each instance. It can be easily seen that any version of `priority_search` is vastly superior to the default search of Chuffed. `priority_search` strategies solve more instances to optimality also found feasible solution for every instance. It should be noted that the seven optimally solved instances are the same over all search configurations. The seven optimally solved instances all have less than or equal to 15 projects (six of those instances are from class 1 and one is from class 2 (as decribed in Table 1). The searches `ps_startTimeFirst_aff` and `ps_startTimeFirst_ff` both achieved the best cumulative objective value, but `ps_startTimeFirst_aff` offered a slightly lower average time to the first feasible solution found.

While initial experiments showed that redundant constraints (20-25) have a high impact on the search, Table 2 shows that while constraint (19) also has a slightly positive impact on the solution quality, it does slow down the time to the first feasible solution. However, we decided to include the constraint in our final experiment since we were mainly interested in solution quality and not in quickly found feasible solutions.

Solver	Constraints	Search	♯ sat	♯ opt
CPLEX	(1-11),(16-18),(19-25)	Default	8	2
Chuffed	(1-11),(16-18),(19-25)	`ps_startTimeFirst_aff`	**30**	**9**

Table 3. CPLEX Comparison (Runtime 2h)

Table 3 shows our final experiment which is a comparison between Chuffed and the well-known MIP solver CPLEX. The time limit was set to 2 hours for each instance. CPLEX performed very poorly in comparison to Chuffed, which even found optimal solutions for two more instances with the longer time-limit (both of them from class 2). Of course it should be noted that our model was developed with CP solvers in mind and thus might not be a perfect fit for MIP solvers and that CPLEX was only run in single-threaded mode.

Table 4 lists the detailed results of Chuffed in the final experiment for all 30 test instances.

Instance	h	$\|P\|$	$\|J\|$	$\|E\|$	$\|B\|$	$\|G^*\|$	$\|\bigcup_{g\in G^*} G_g\|$	Objective	Optimal
1	89	5	7	7	7	3	6	96	**yes**
2	89	5	8	7	7	3	95	74	**yes**
3	89	5	24	7	7	3	48	153	**yes**
4	89	5	14	7	7	3	49	110	**yes**
5	89	10	29	13	13	4	100	275	**yes**
6	89	10	25	13	13	3	148	295	**yes**
7	89	10	37	13	13	3	93	337	?
8	89	10	29	13	13	3	98	313	?
9	175	15	29	12	12	5	115	345	**yes**
10	175	15	53	12	12	3	98	458	?
11	175	20	60	16	16	5	70	500	**yes**
12	175	20	84	16	16	4	16	830	?
13	175	20	65	16	16	3	134	855	**yes**
14	175	20	62	16	16	3	132	735	?
15	175	30	113	23	23	3	18	1685	?
16	175	30	105	23	23	3	201	1582	?
17	175	40	126	31	31	3	100	1398	?
18	175	40	138	31	31	3	271	1826	?
19	175	60	208	46	46	6	219	2669	?
20	175	60	212	46	46	3	397	2912	?
21	521	20	76	6	6	5	42	915	?
22	521	20	71	6	6	3	67	955	?
23	521	40	196	12	12	4	18	3688	?
24	521	40	187	12	12	3	146	2516	?
25	521	60	260	18	18	6	148	3368	?
26	521	60	239	18	18	3	218	3968	?
27	783	60	270	13	13	4	16	3147	?
28	783	60	247	13	14	3	196	2686	?
29	783	90	384	19	19	5	139	4672	?
30	783	90	401	19	19	3	284	5983	?

Table 4. Detailed results for Chuffed in the final experiment (all solutions are feasible for their instance)

6 Conclusion

In this paper we have investigated different possibilities to model a complex real-world project scheduling problem. For some of the constraints, we first experimented with approaches which were already used in related project scheduling problems. To deal with this more complex problem and larger instances we introduced several extensions in modeling. We have shown that also for our problem the new MiniZinc search procedure `priority_search` has a lot of potential. We have evaluated our approach on a set of 30 benchmark instances. Using CP techniques we could find feasible solutions for all considered instances. Furthermore, optimal solutions for 9 instances could be provided for the first time. We also compared our CP based approach to a state-of-the-art MIP solver and showed that CP technology performs better for this problem.

For the future work, we plan to investigate exact techniques to solve both stages of the TLSP including grouping and scheduling simultaneously.

Acknowledgments 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

1. Bartels, J.H., Zimmermann, J.: Scheduling tests in automotive r&d projects. European Journal of Operational Research **193**(3), 805 – 819 (2009). https://doi.org/10.1016/j.ejor.2007.11.010
2. Bellenguez, O., Néron, E.: 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. pp. 229–243. Springer Berlin Heidelberg, Berlin, Heidelberg (2005). https://doi.org/10.1007/11593577_14
3. Brucker, P., Drexl, A., Möhring, R., Neumann, K., Pesch, E.: Resource-constrained project scheduling: Notation, classification, models, and methods. European Journal of Operational Research **112**(1), 3 – 41 (1999). https://doi.org/10.1016/S0377-2217(98)00204-5
4. Chu, G.: Improving combinatorial optimization. Ph.D. thesis, University of Melbourne, Australia (2011), http://hdl.handle.net/11343/36679
5. Dauzère-Pérès, S., Roux, W., Lasserre, J.: Multi-resource shop scheduling with resource flexibility. European Journal of Operational Research **107**(2), 289 – 305 (1998). https://doi.org/10.1016/S0377-2217(97)00341-X
6. Drezet, L.E., Billaut, J.C.: A project scheduling problem with labour constraints and time-dependent activities requirements. International Journal of Production Economics **112**(1), 217 – 225 (2008). https://doi.org/10.1016/j.ijpe.2006.08.021, special Section on Recent Developments in the Design, Control, Planning and Scheduling of Productive Systems
7. Elmaghraby, S.E.: Activity networks: Project planning and control by network models. John Wiley & Sons (1977)
8. Feydy, T., Goldwaser, A., Schutt, A., Stuckey, P.J., Young, K.D.: Priority search with minizinc. In: ModRef 2017: The Sixteenth International Workshop on Constraint Modelling and Reformulation at CP2017 (2017)

9. Hartmann, S., Briskorn, D.: A survey of variants and extensions of the resource-constrained project scheduling problem. European Journal of Operational Research **207**(1), 1 – 14 (2010). https://doi.org/10.1016/j.ejor.2009.11.005
10. IBM, CPLEX: 12.8.0 cplex users manual. https://www.ibm.com/analytics/cplex-optimizer (2017)
11. LLC, G.O.: Gurobi optimizer reference manual. http://www.gurobi.com (2018)
12. Mika, M., Waligóra, G., Węglarz, J.: Overview and state of the art. In: Schwindt, C., Zimmermann, J. (eds.) Handbook on Project Management and Scheduling Vol.1, pp. 445–490. Springer International Publishing, Cham (2015)
13. Mischek, F., Musliu, N.: A local search framework for industrial test laboratory scheduling. In: Proceedings of the 12th International Conference on the Practice and Theory of Automated Timetabling (PATAT-2018), Vienna, Austria, August 2831, 2018. pp. 465–467 (2018)
14. Mischek, F., Musliu, N.: 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 (2018)
15. Nethercote, N., Stuckey, P.J., Becket, R., Brand, S., Duck, G.J., Tack, G.: 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 (2007). https://doi.org/10.1007/978-3-540-74970-7_38
16. Nudtasomboon, N., Randhawa, S.U.: Resource-constrained project scheduling with renewable and non-renewable resources and time-resource tradeoffs. Computers & Industrial Engineering **32**(1), 227 – 242 (1997). https://doi.org/10.1016/S0360-8352(96)00212-4
17. Pritsker, A.A.B., Waiters, L.J., Wolfe, P.M.: Multiproject scheduling with limited resources: A zero-one programming approach. Management Science **16**(1), 93–108 (1969). https://doi.org/10.1287/mnsc.16.1.93
18. Salewski, F., Schirmer, A., Drexl, A.: Project scheduling under resource and mode identity constraints: Model, complexity, methods, and application. European Journal of Operational Research **102**(1), 88 – 110 (1997). https://doi.org/10.1016/S0377-2217(96)00219-6
19. Schulte, C., Lagerkvist, M., Tack, G.: Gecode 6.10 reference documentation. https://www.gecode.org (2018)
20. Schwindt, C., Trautmann, N.: Batch scheduling in process industries: an application of resource–constrained project scheduling. OR-Spektrum **22**(4), 501–524 (2000)
21. Szeredi, R., Schutt, A.: 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 (2016). https://doi.org/10.1007/978-3-319-44953-1_31
22. Węglarz, J., Józefowska, J., Mika, M., Waligóra, G.: Project scheduling with finite or infinite number of activity processing modes a survey. European Journal of Operational Research **208**(3), 177 – 205 (2011). https://doi.org/10.1016/j.ejor.2010.03.037
23. Young, K.D., Feydy, T., Schutt, A.: Constraint programming applied to the multi-skill project scheduling problem. In: Principles and Practice of Constraint Programming - 23rd International Conference, CP 2017, Melbourne, VIC, Australia, August 28 - September 1, 2017, Proceedings. pp. 308–317 (2017). https://doi.org/10.1007/978-3-319-66158-2_20