The Test Laboratory Scheduling Problem

Florian Mischek, Nysret Musliu

Christian Doppler Laboratory for Artificial Intelligence and Optimization for Planning and Scheduling
TU Wien
Favoritenstraße 9-11, 1040 Vienna, Austria

November 29, 2018

This report introduces a real-world scheduling problem arising in an industrial test laboratory. It is related to the well-known Resource Constraint Project Scheduling Problem (RCPSP), which it extends by several additional constraints and objectives, as well as a novel form of grouping smaller activities to larger units to reduce operational overheads. In addition to a formal definition of the problem, the report includes a description of an instance generator which can be used to create random instances based on real-world data for this problem. Two such generated instance sets with different characteristics are provided.

1 Introduction

Project scheduling problems arise naturally in many different contexts, including industrial production and manufacturing processes, construction work, software development and many more. All those areas have in common that activities have to be scheduled and have resources assigned to them subject to various different constraints. Since such problems can become quite complex and practical instances are often very large, automated solution approaches are essential for providing high quality solutions.

This report describes a real-world scheduling problem that arises in a test laboratory of a large industrial company. In the Test Laboratory Scheduling Problem (TLSP), a large number of tasks (tests) have to be scheduled over a long time period, subject to resource availability constraints, time windows, precedence relations between tasks

and several other side constraints. However, instead of directly scheduling those tasks, they are first grouped into larger units, called jobs. Only the jobs themselves have times and resources assigned to them and derive their properties from the contained tasks.

There are several advantages to be gained from such a grouping: It reduces overhead due to multiple context switches, instead focusing on long, uninterrupted work on similar activities. In the same vein, TLSP features setup times required for building up test assemblies for each test. These can be reused between similar tests if they are performed using the same resources. This can easily be modeled by adding the required time at the beginning of each job. The duration of tasks can vary between several minutes and multiple weeks. Grouping shorter tasks into longer units allows for coarser time units in scheduling, without incurring prohibitive overheads due to rounding durations. Further, it improves flexibility because tests within a job can easily be switched around without generating any scheduling conflicts. Finally, the reduced number of jobs compared to the total number of tasks makes the schedule more comprehensible for humans, both those scheduled to perform the tests and any human planners in cases where manual corrections or semi-automatic, interactive scheduling is desired.

TLSP is related to the well-known Resource Constrained Project Scheduling Problem (RCPSP), and several of its extensions, including multiple modes (MRCPSP), release dates and deadlines, heterogeneous resources and others. For an overview of related literature on RCPSP and its extensions, we refer to surveys by Brucker et al. [3], Hartmann and Briskorn [4], or Mika et al. [7].

Of particular relevance for the scheduling part of TLSP is the Multi-Skill Project Scheduling Problem (MSPSP), introduced by Bellenguez and Néron [2]. MSPSP also features resource availability constraints, which are modeled as skills possessed by resources and required for activities. Young et al. [10] have achieved very good results for MSPSP using a constraint programming approach.

Bartels and Zimmermann [1] describe a scheduling problem in a related context, where tests for experimental vehicles are scheduled. It includes several aspects that are also present in TLSP, although it has a different resource model and a completely unrelated objective (i.e. minimizing the number of test vehicles built).

Concepts similar to the grouping aspect of TLSP are found in other works in the form of batching (e.g. [9, 8]) or schedule-dependent setup times (e.g. [5, 6]), although it is usually handled implicitly, i.e. the batches arise from the finished schedule, instead of the other way round.

The rest of the report is structured as follows: A summary of the problem is presented in Section 2, followed by a description of the input parameters of a problem instance in Section 3. Section 4 describes the concept of jobs and introduces notation for the properties of jobs derived from the tasks contained in them. The constraints and objective function of TLSP are formally defined in Section 5. Section 6 lists different usage scenarios relevant in practice, together with their non-functional requirements. In the last part of the report, an instance generator to randomly generate instances based on real-world data is described in Section 7, with two such datasets given in Section 8.

2 Problem description

In TLSP, a list of projects is given, which each contain several tasks. For each project, the tasks must be partitioned into a set of jobs, with some restrictions on the feasible partitions. Then, those jobs must each be assigned a mode, time slots and resources. The properties and feasible assignments for each job are calculated from the tasks contained within.

A solution of TLSP is a schedule consisting of the following parts:

- A list of jobs, composed of one or multiple similar tasks within the same project.
- For each job, an assigned mode, start and end time slots, the employees scheduled to work on the job, and an assignment to a workbbench and equipment.

The quality of a schedule is judged according to an objective function that is the weighted sum of several soft constraints and should be minimized. Among others, these include the number of jobs and the total completion time (start of the first job until end of the last) of each project.

3 Input parameters

A TLSP instance can be split into three parts: The laboratory *environment*, including a list of resources, a list of *projects* containing the tasks that should be scheduled together with their properties and the current state of the *existing schedule*, which might be partially or completely empty.

3.1 Environment

In the laboratory, resources of different kinds are available that are required to perform tasks:

- *Employees* $e \in E = \{1, \ldots, |E|\}$ who are qualified for different types of tasks.
- A number of *workbenches* $b \in B = \{1, \ldots, |B|\}$ with different facilities. (These are comparable to machines in shop scheduling problems.)
- 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 called G^*.

The scheduling period is composed of *time slots* $t \in T = \{0, \ldots, |T| - 1\}$. Each time slot represents half a day of work.

Tasks are performed in one of several *modes* labeled $m \in M = \{1, \ldots, |M|\}$. The chosen mode influences the following properties of tasks performed under it:

- The *speed factor* v_m, which will be applied to the task's original duration.
- The number of *required employees* e_m.

3.2 Projects and Tasks

Given is a set P of *projects* labeled $p \in \{1, \ldots, |P|\}$. Each project contains *tasks* $pa \in A_p$, with $a \in \{1, \ldots, |A_p|\}$. The set of all tasks (over all projects) is $A^* = \bigcup_{p \in P} A_p$.

Each task pa has several properties:

- It has a *release date* α_{pa} and both a *due date* $\bar{\omega}_{pa}$ and a *deadline* ω_{pa}. The difference between the latter is that a due date violation only results in a penalty to the solution quality, while deadlines must be observed.
- $M_{pa} \subseteq M$ is the set of *available modes* for the task.
- The task's *duration* d_pa (in time slots, real-valued). Under any given mode $m \in M_{pa}$, this duration becomes $d_{pam} := d_{pa} * v_m$.
- Most tasks must be performed on a workbench. This is indicated by the boolean parameter $b_pa \in \{0, 1\}$. If required, this workbench must be chosen from the set of *available workbenches* $B_{pa} \subseteq B$.
- Similarly, it requires qualified *employees* chosen from $E_{pa} \subseteq E$. The required number depends on the mode. A further subset $E^{Pr}_{pa} \subseteq E_{pa}$ is the set of preferred employees.
- Of each equipment group $g \in G^*$, the task requires r_{pag} devices, which must be taken from the set of *available devices* $G_{pag} \subseteq G_g$.
- A list of direct *predecessors* $\mathcal{P}_{pa} \subseteq A_p$, which must be completed before the task can start. Note that precedence constraints can only exist between tasks in the same project.

Each project's tasks are partitioned into *families* $F_{pf} \subseteq A_p$, where f is the family's index. For a given task pa, f_{pa} gives the task's family. Only tasks from the same family can be grouped into a single job.

Additionally, each family f is associated with a certain *setup time* s_{pf}, which is added to the duration of each job containing tasks of that family.

Finally, it may be required that certain tasks are performed by the same employee(s)[1]. For this reason, each project p may define *linked tasks*, which must be assigned the same employee(s). Linked tasks are given by the equivalence relation $L_p \subseteq A_p \times A_p$, where two tasks pa and pb are linked if and only if $(pa, pb) \in L_p$.

3.3 Initial schedule

All problem instances include an initial (or base) schedule, which may be completely or partially empty. This schedule can act both as an initial solution and as a baseline, placing limits on the schedules of employees and tasks, in particular by defining fixed assignments that must not be changed.

[1] This is used most notably to ensure that documentation is prepared by those employees who also did the tests.

Provided is a set of jobs J^0, where each job $j \in J^0$ contains the following assignments:

- The tasks in the job: $\dot{A}_j$
  - A *fixed* subset of these tasks $\dot{A}_j^F \subseteq \dot{A}_j$. All fixed tasks of a job in the base schedule must also appear together in a single job in the solution.
- The mode assigned to the job: $\dot{m}_j$
- The start and completion times of the job: $\dot{t}_j^s$ resp. $\dot{t}_j^c$
- The resources assigned to the job:
  - Workbench: $\dot{b}_j$
  - Employees: $\dot{E}_j$
  - Equipment: $\dot{G}_{gj}$ for equipment group g

Except for the tasks, each individual assignment may or may not be present in any given job. Fixed tasks are assumed to be empty, if not given. In all other cases, missing assignments will be referred to using the value ϵ. Time slots and employees can only be assigned if also a mode assignment is given.

A subset of these jobs are the *started jobs* J^{0S}. A started job $j^s \in J^{0S}$ must fulfill the following conditions:

- It must contain at least one fixed task. It is assumed that the fixed tasks of a started job are currently being worked on.
- Its start time must be 0.
- It must contain resource assignments fulfilling all requirements.

A started job's duration does not include a setup time. In the solution, the job containing the fixed tasks of a started job must also start at time 0. Usually, the resources available to the fixed tasks of a started job are additionally restricted to those assigned to the job, to avoid interruptions of ongoing work in case of a rescheduling.

4 Jobs and Grouping

For various operational reasons, tasks are not scheduled directly. Instead, they are first grouped into larger units called *jobs*.

A single job can only contain tasks from the same project and family.

Jobs have many of the same properties as tasks, which are computed from the tasks that make up a job. The general principle is that within a job, tasks are not explicitly ordered or scheduled; therefore the job must fulfill all requirements of each associated task during its whole duration[2].

[2] while this might seem overly restrictive, tasks of the same family usually have equivalent or very similar requirements in practice

Let $J = \{1, \ldots, |J|\}$ be the set of all jobs in a solution and $J_p \subseteq J$ be the set of jobs of a given project p. Then for a job $j \in J$, the set of tasks contained in j is $\dot{A}_j$. j has the following properties:

$$\tilde{p}_j \quad \text{and} \quad \tilde{f}_j$$

are the project and family of j.

$$\tilde{\alpha}_j \coloneqq \max_{pa \in \dot{A}_j} \alpha_{pa}, \quad \tilde{\bar{\omega}}_j \coloneqq \min_{pa \in \dot{A}_j} \bar{\omega}_{pa}, \quad \tilde{\omega}_j \coloneqq \min_{pa \in \dot{A}_j} \omega_{pa}$$

are the release date, due date and deadline of j, respectively.

$$\tilde{M}_j \coloneqq \bigcap_{pa \in \dot{A}_j} M_{pa}$$

is the set of available modes.

$$\tilde{d}_{jm} \coloneqq \left\lceil (s_{p_j f_j} + \sum_{pa \in \dot{A}_j} d_{pa}) \ * \ v_m \right\rceil$$

is the (integer) duration of the job under mode m. The additional setup time is added to the total duration of the contained tasks.

$$\tilde{b_j} \coloneqq \max_{pa \in \dot{A}_j} b_{pa}$$

is the required number of workbenches ($\tilde{b}_j \in \{0, 1\}$).

$$\tilde{B}_j \coloneqq \bigcap_{pa \in \dot{A}_j} B_{pa}$$

are the available workbenches for j.

$$\tilde{E}_j \coloneqq \bigcap_{pa \in \dot{A}_j} E_{pa}$$

are the employees qualified for j.

$$\tilde{E}_j^{Pr} \coloneqq \bigcap_{pa \in \dot{A}_j} E_{pa}^{Pr}$$

are the preferred employees of j.

$$\tilde{r}_{jg} \coloneqq \max_{pa \in \dot{A}_j} r_{pag}$$

are the required units of equipment group g.

$$\tilde{G}_{jg} \coloneqq \bigcap_{pa \in \dot{A}_j} G_{pag}$$

are the available devices for equipment group g.

$$\tilde{\mathcal{P}}_j \coloneqq \{k \in J \setminus \{j\} : \exists pa \in \dot{A}_j, pb \in \dot{A}_k \text{ s.t. } pb \in \mathcal{P}_{pa}\}$$

is the set of predecessor jobs of j. Finally,

$$\tilde{L}_p \coloneqq \{(j, k) \in J \times J : j \neq k \wedge \exists pa \in \dot{A}_j, pb \in \dot{A}_k \text{ s.t. } (pa, pb) \in L_p\}$$

defines the linked jobs in project p.

In addition, a solution contains the following assignments for each job:

- $\dot{t}^s_j \in T$ the scheduled start time slot
- $\dot{t}^c_j \in T$ the scheduled completion time
- $\dot{m}_j \in M$ the mode in which the job should be performed
- $\dot{b}_j \in B$ the workbench assigned to the job (ϵ if no workbench is required)
- $\dot{E}_j \subseteq E$ the set of employees assigned to the job
- $\dot{G}_{jg} \subseteq G_g$ the set of assigned devices from equipment group g

5 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.

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

5.1 Hard Constraints

H1: Job assignment Each task must be assigned to exactly one job.

$$\begin{aligned} &\forall p \in P, pa \in A_p : \\ &\quad \exists! j \in J \text{ s.t. } pa \in \dot{A}_j \end{aligned}$$

H2: Job grouping All tasks contained in a job must be from the same project and family.

$$\begin{aligned} &\forall j \in J, pa \in \dot{A}_j : \\ &\quad p = \tilde{p}_j \\ &\quad f_{pa} = \tilde{f}_j \end{aligned}$$

H3: Fixed tasks Each group of tasks assigned to a fixed job in the base schedule must also be assigned to a single job in the solution.

$$\begin{aligned} &\forall j^0 \in J^0 : \\ &\quad \exists j \in J \text{ s.t. } \dot{A}^F_{j^0} \subseteq \dot{A}_j \end{aligned}$$

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

$$\begin{aligned}&\forall j \in J: \\ &\quad \dot{t}^c_j - \dot{t}^s_j = \tilde{d}_{j\dot{m}_j}\end{aligned}$$

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

$$\begin{aligned}&\forall j \in J: \\ &\quad \dot{t}^s_j \geq \tilde{\alpha}_j \\ &\quad \dot{t}^c_j \leq \tilde{\omega}_j\end{aligned}$$

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

$$\begin{aligned}&\forall j \in J, k \in \tilde{\mathcal{P}}_j: \\ &\quad \dot{t}^c_k \leq \dot{t}^s_j\end{aligned}$$

H7: Started jobs A job containing fixed tasks of a started job in the base schedule must start at time 0.

$$\begin{aligned}&\forall j \in J, j^s \in J^{0S}: \\ &\quad t^F_{j^s} = 1 \wedge \dot{A}^F_{j^s} \subseteq \dot{A}_j \implies \dot{t}^s_j = 0\end{aligned}$$

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

$$\begin{aligned}&\forall b \in B, t \in T: \\ &\quad |\{j \in \mathcal{J}_t : \dot{b}_j = b\}| \leq 1 \\ &\forall e \in E, t \in T: \\ &\quad |\{j \in \mathcal{J}_t : e \in \dot{E}_j\}| \leq 1 \\ &\forall g \in G^*, d \in G_g, t \in T: \\ &\quad |\{j \in \mathcal{J}_t : d \in \dot{G}_{jg}\}| \leq 1\end{aligned}$$

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

$$\begin{aligned}&\forall j \in J: \\ &\quad \dot{b}_j = \epsilon \iff \tilde{b}_j = 0\end{aligned}$$

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

$$\forall j \in J : \\ |\dot{E}_j| = e_{\dot{m}_j}$$

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

$$\forall j \in J, g \in G^* : \\ |\dot{G}_{jg}| = \tilde{r}_{jg}$$

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

$$\forall j \in J : \\ \dot{b}_j = \epsilon \vee \dot{b}_j \in \tilde{B}_j$$

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

$$\forall j \in J : \\ \dot{E}_j \subseteq \tilde{E}_j$$

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

$$\forall j \in J, g \in G^* : \\ \dot{G}_{jg} \subseteq \tilde{G}_{jg}$$

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

$$\forall p \in P, (j,k) \in \tilde{L}_p : \\ \dot{E}_j = \dot{E}_k$$

5.2 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 The number of jobs should be minimized.

$$C^{S1} \coloneqq |J|$$

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

$$\forall j \in J : \\ C_j^{S2} \coloneqq |\{e \in \dot{E}_j : e \notin \tilde{E}_j^{Pr}\}|$$

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

$$\forall p \in P : \\ C_p^{S3} \coloneqq |\bigcup_{j \in J_p} \dot{E}_j|$$

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

$$\forall j \in J : \\ C_j^{S4} \coloneqq \max(\dot{t}_j^c - \tilde{\omega}_j, 0)$$

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

$$\forall p \in P : \\ C_p^{S5} \coloneqq \max_{j \in J_p} \dot{t}_j^c \; - \; \min_{j \in J_p} \dot{t}_j^s$$

Constraint S1 favors fewer, longer jobs over more fragmented solutions. This helps reducing overhead (fewer setup periods necessary, rounding of fractional durations), but even more important, it reduces the complexity of the final schedule, both for the employees performing the actual tasks and any human planners in those cases where manual corrections or additions become necessary.

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 reduce 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 tasks 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 and may be adjusted according to the current situation. For preliminary evaluations, we have assumed a uniform weight of 1 for all soft constraints.

6 Usage Scenarios

In the daily lab operation, three usage scenarios can be identified, with slightly different assumptions on the input (in particular the initial schedule) and requirements. TLSP is flexible enough to cover all three of those scenarios without additional changes. However, solution approaches might have to be adapted according to the additional restrictions and non-functional requirements of each usage scenario.

6.1 Full Scheduling

Here, the goal is to find a good (/optimal) schedule for all currently known projects, starting either from an empty schedule or a complete base schedule, which is usually, but not necessarily, already feasible. All hard constraints have to be respected, but other than that, the solver is free to regroup tasks and reschedule jobs of all projects.

It is expected that the solution process will be able to run during off hours, e.g. over night or on weekends and thus solution times are not critical (several hours, up to several days).

This scenario is meant to build an initial schedule or reoptimize a schedule after disruptions and subsequent repairs performed in one of the other scenario.

6.2 Project Insertion

For this scenario, the task is to insert a new project into the schedule, minimizing disruptions for existing projects. Thus, the initial schedule will be complete and feasible, except for one single project designated as new, which is not scheduled at all.

In contrast to the Full Scheduling scenario, solutions should be available within a few minutes at the most, as the feasibility of a project must be decided quickly and during the working day. However, suboptimal solutions can be accepted, if this enables additional projects to be fit into the schedule.

6.3 Modification & Repair

Finally, it regularly happens that already scheduled projects are modified before or during their execution. Additional tests might be added or existing ones removed and parameters of tests (in particular time and resource requirements) can change. Therefore, the initial schedule will be complete, though not necessarily feasible for the given environment and project data.

The focus of this mode is to retain or restore feasibility of the schedule under the modified conditions. In particular for the case of disruptions arising during the execution of a project (lateness due to unforeseen complications or errors, employee illness, ...), it is crucial to repair the schedule with minimal effect on the timeliness of the current and other projects.

Any problems should be corrected as fast as possible (few seconds), to minimize disruptions to the operation of the lab.

At least the following modifications should be supported by the solution procedure:

- Adding tasks to a project.
- Changing the properties (e.g. duration, resource requirements, precedence, ...) of a number of tasks
- Adding or removing resources, globally or restricted to a certain time period.
- Fixing the tasks or assignments of a job to its current, or different values.

The type of modification is given as additional input to allow for specialized solution approaches. The weights of the soft constraints are chosen to set a clear focus on ensuring or restoring feasibility of the schedule, especially for the beginning of the scheduling period.

7 Instance generator

In order to be able to generate instances of specific size and complexity on demand, we developed an instance generator for TLSP. It randomly generates instances of various sizes that are based on the real-world data in the laboratory of our industrial partner and can be configured to produce instances of various sizes and properties, including the usage scenarios of Full Scheduling and Project Insertion[3] The generator was written in Java.

To generate a new instance, one has to specify the number of expected projects and the length of the scheduling period, plus optionally various configuration parameters that refine the desired properties of the instance. From this, an instance is generated as follows: First, the laboratory environment, including the available resources, is defined according to the desired number of projects and the length of the scheduling period. Then, the required number of projects is generated, together with a set of jobs for each project. This is used to populate the reference solution, which is guaranteed to be a feasible solution for the final problem instance. In the third step, task properties (resource requirements and availabilities, precedence constraints, time windows, ...) are defined in accordance with the reference schedule. Finally, the reference solution is modified to receive the base schedule, which completes the instance generation.

7.1 Environment generation

From the number of projects given (together with their average total work) and the length of the scheduling period, a measure for the expected workload per time slot can be extracted. The number of employees and workbenches required to achieve a certain mean degree of utilization can be estimated from this measure.

Equipment is generated by a separate component, which can be passed to the instance generator. This component also handles equipment requirements of tasks in Step 7.2. Currently supported are two different implementations:

[3]Support for Rescheduling will be added at a later date

Mode	v_m	e_m
Single	1	1
Shift	0.6	2
External	1	0

Table 1: Task modes used by the instance generator.

Lab equivalent mode generates devices for exactly those equipment groups that are relevant for planning at our industrial partner. The equipment requirements for tasks also closely corresponds to the distribution of requirements in the real-world laboratory.

General mode is more flexible and creates between 3 and 6 equipment groups, together with corresponding devices. Equipment requirements for each groups are selected to be either unitary (i.e. tasks require at most device of this group) or randomly generated for each project.

In both cases, the number of devices in each group depends on the same measure for workload as for the employees and workbenches, modified by the expected number of required devices per task.

In this step, also the task modes shown in Table 1 are generated.

7.2 Reference solution

Each project is assigned a certain total workload, which is taken from a distribution that is as close as possible to the real-world data. It is also assigned to a random interval of the scheduling period, where earlier intervals are slightly favored over later ones. Then a number of tasks are created, still without a duration. In general, the expected number of tasks grows with the project's workload. These tasks are then distributed into families and further into jobs. The families can either be taken from the real-world data or generated randomly, depending on the generator configuration.

Each generated job is then randomly assigned a preliminary duration (taken from the total workload of the project), a mode (with only a small probability for *External* mode), and equipment requirements according to the chosen equipment generation mode (see Step 7.1). Most jobs will also be set to require a workbench.

Once these parameters are determined, the job is placed into the reference schedule. This is done by first randomly choosing a seed point within the project's assigned interval. Starting at this point, the job is grown outward in both directions as long as any feasible resource assignment for it exists or until the desired duration is reached. If no feasible placement at the whole duration can be found, the position is still accepted as long as the duration is not much smaller than expected. Where this is not the case or no resource assignment exists even for the initial seed, the procedure is repeated for another random seed point, up to a certain maximum number of iterations. While

this process cannot guarantee that a feasible schedule can be found, experiments so far have shown that this is sufficient in most cases.

Once a job is scheduled, it is assigned a feasible set of resources. Its final duration is split between its tasks, minus the setup time required for its family.

To model started jobs, the scheduling period is extended to the past by one month (about 40 time slots) in the beginning of this step. After all jobs have been scheduled, the scheduling period is reduced to its original duration. Jobs ending before the new first time slot are removed completely, including their contained tasks – it is assumed that those have been completed before the start of the scheduling period. Those overlapping the first time slot have their duration reduced accordingly (potentially including removing some of the contained tasks) and are defined as started jobs.

7.3 Task properties

After the reference schedule has been completed, the remaining properties of the tasks are finalized.

Release dates, deadlines and due dates for tasks of a project are set to the first start, respectively last end, of any job in the project. They are then extended outwards by an additional number of time slots (minimum of 0), which is usually smaller for the due date than for the deadline.

The available resources are set for each job (and all contained tasks) such that they include at least the assigned resources. The number of available resources is taken from the real-world data for employees (including preferred employees) and workbenches, and handled by the chosen equipment generation mode for equipment. A small subset of tasks may also have additional available resources beyond those of the other tasks in the job.

The available task modes are chosen such that tasks of a job with the *External* mode in the reference schedule must be performed in this mode, while all other tasks can be performed in *Single* mode, plus optionally in *Shift* mode. The latter is always the case if the task's job has *Shift* mode in the reference schedule.

Started jobs will always have their time window, available resources and modes set to exactly those values that they are assigned to, to ensure that they cannot be altered by the solver.

The generation of precedence constraints is again delegated to a separate component, which can be set in the configuration. Two implementations are currently supported, both start by building a maximum graph of possible dependencies according to the reference schedule for each project:

Ranked precedence constraints assign ranks to a subset of tasks such that tasks of higher rank have all tasks of lower rank as prerequisites.

General precedence constraints randomly choose arcs in the maximum dependency graph that will result in actual dependencies between tasks.

In both cases, the number of precedence constraints between tasks is rather low and most tasks don't have any prerequisite tasks at all. This circumstance is directly

taken from the real-world data, where only few dependencies between tasks appear and contrasts with other project scheduling problems that include tighter constraints on the order of activities.

Finally, possible candidates for linked tasks are identified, both within and between jobs, and a small subset of those is chosen randomly.

7.4 Base schedule

The last step in the instance generation process is the derivation of a base schedule from the reference solution.

Again, there are several supported options for this, which the generator can be configured to use. Currently, there are two main implementations supported:

Delete mode Removes some assignments of the jobs in the reference schedule, but leaves the remaining assignments intact.

Random mode Replaces assignments of the jobs in the reference schedule by random values. Resource availability constraints and time windows are respected, but other constraints may be violated by these changes.

Either mode has a parameter that defines the strength of the perturbation (given as the percentage of jobs affected). Further, various flags can modify the behavior, including keeping certain types of assignments intact (e.g. the grouping of tasks into jobs).

Any perturbations do not affect started jobs as well as a number of jobs selected as *fixed*, which have their tasks fixed and either all or some assignments fixed to the current value by restricting the time window, available resources and/or modes to their assigned values.

This step also includes an option to specify the new project in the Project Insertion usage scenario. This project will have all its jobs (though not the contained tasks) removed from the initial schedule.

8 Data sets

Using the instance generator described above, we provide randomized datasets of various sizes that can be used to evaluate solution approaches for different usage scenarios.

Currently, there are two different datasets, which are intended for the Full Scheduling scenario. In both cases, the base schedule of all instances contains jobs providing a grouping for all tasks in the instance, but no other assignments except for started and fixed jobs. Feasible solutions using the provided job grouping are guaranteed to exist, although they might not be optimal.

The first dataset (LabStructure) is modeled as closely as possible to the real-world data. The second (General) uses the same distribution of work among the projects and tasks, but is more flexible regarding several other problem features. The instance generator configurations used to create these two datasets are listed in Table 2.

Parameter	LabStructure	General
Scenario	Full Scheduling	Full Scheduling
Equipment	Lab equivalent	General
Task families	Lab equivalent	General
Precedence	Ranked	General
Base schedule mode	Delete	Delete
Base schedule perturbation	1.0	1.0
Base schedule options	Grouping constant	Grouping constant

Table 2: Instance generator configuration parameters for the two provided datasets.

Instances	$\|T\|$	$\|P\|$
000 - 004	88	5
005 - 009	88	10
010 - 014	174	20
015 - 019	174	40
020 - 024	174	15
025 - 029	174	30
030 - 034	174	60
035 - 039	520	20
040 - 044	520	40
045 - 049	520	60
050 - 054	782	60
055 - 059	782	90

Table 3: Instances and their properties (identical for both data sets)

Each dataset contains 60 instances of varying sizes. Each instance consists of the following four files, where `<i>`, `<h>` and `<p>` stand for the instance id, number of time slots in the scheduling period and number of projects, respectively:

- `<i>_<h>_<p>_environment.xml` contains environment data (see Section 3.1).
- `<i>_<h>_<p>_tasks.xml` contains the list of projects and tasks (see Section 3.2).
- `<i>_<h>_<p>_schedule.xml` contains the initial schedule (see Section 3.3).
- `<i>_<h>_<p>_reference.xml` contains the reference solution produced by the instance generator (see Section 7.2).

Table 3 shows the instances, together with their sizes. Both instance sets are provided for download at `https://www.dbai.tuwien.ac.at/staff/fmischek/TLS`.

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] J.-H. Bartels and J. Zimmermann. Scheduling tests in automotive r&d projects. *European Journal of Operational Research*, 193(3):805 – 819, 2009.

[2] Odile Bellenguez and Emmanuel Néron. Lower bounds for the multi-skill project scheduling problem with hierarchical levels of skills. In Edmund Burke and Michael Trick, editors, *Practice and Theory of Automated Timetabling V*, pages 229–243, Berlin, Heidelberg, 2005. Springer Berlin Heidelberg.

[3] Peter Brucker, Andreas Drexl, Rolf Möhring, Klaus Neumann, and Erwin Pesch. Resource-constrained project scheduling: Notation, classification, models, and methods. *European Journal of Operational Research*, 112(1):3 – 41, 1999.

[4] Sönke Hartmann and Dirk Briskorn. A survey of variants and extensions of the resource-constrained project scheduling problem. *European Journal of Operational Research*, 207(1):1 – 14, 2010.

[5] Marek Mika, Grzegorz Waligóra, and Jan Węglarz. Modelling setup times in project scheduling. *Perspectives in modern project scheduling*, pages 131–163, 2006.

[6] Marek Mika, Grzegorz Waligóra, and Jan Węglarz. Tabu search for multi-mode resource-constrained project scheduling with schedule-dependent setup times. *European Journal of Operational Research*, 187(3):1238 – 1250, 2008.

[7] Marek Mika, Grzegorz Waligóra, and Jan Węglarz. Overview and state of the art. In Christoph Schwindt and Jürgen Zimmermann, editors, *Handbook on Project Management and Scheduling Vol.1*, pages 445–490. Springer International Publishing, Cham, 2015.

[8] Chris N. Potts and Mikhail Y. Kovalyov. Scheduling with batching: A review. *European Journal of Operational Research*, 120(2):228 – 249, 2000.

[9] Christoph Schwindt and Norbert Trautmann. Batch scheduling in process industries: an application of resource–constrained project scheduling. *OR-Spektrum*, 22(4):501–524, 2000.

[10] Kenneth D. Young, Thibaut Feydy, and Andreas Schutt. Constraint programming applied to the multi-skill project scheduling problem. In J. Christopher Beck, editor, *Principles and Practice of Constraint Programming*, pages 308–317, Cham, 2017. Springer International Publishing.