Local Search Neighborhoods for Industrial Test Laboratory Scheduling with Flexible Grouping

Florian Mischek[1] · **Nysret Musliu**[1] · **Andrea Schaerf**[2]

Keywords Project Scheduling · RCPSP · Industrial Test Laboratory · Local Search

1 Introduction

The Test Laboratory Scheduling Problem (TLSP) arises in a real-world industrial test laboratory, where a large number of activities in multiple projects has to be scheduled, subject to several legal and operational constraints. It is an extension of the well-known (Multi-Mode) Resource-Constrained Project Scheduling Problem ((M)RCPSP) (see e.g. [2,5]) which, in addition to other extensions, includes several unique features.

Most importantly, the activities to be scheduled (*jobs*) are not monolithic, but composed of multiple smaller units called *tasks* and derive all their properties from the tasks they contain. The grouping of tasks into jobs must be determined by the solver as part of the solution process. A similar concept exists in the form of batch scheduling (e.g. [14,12]) or schedule-dependent setup times (e.g. [8,9]). The difference is that in these settings, tasks are scheduled directly and batches arise implicitly from the final schedule.

TLSP also uses heterogeneous resources, with general restrictions on which units of a resource can be used for each task. While usually, variants of RCPSPS assume a homogeneous resource model, similar restrictions can be found in the Multi-Skill RCPSP (MSPSP) [1], where each resource unit possesses a set of skills and requirements are also formulated in terms of skills.

[1] Christian Doppler Laboratory for Artificial Intelligence and Optimization for Planning and Scheduling
DBAI, TU Wien
E-mail: {fmischek,musliu}@dbai.tuwien.ac.at
[2] University of Udine
E-mail: schaerf@uniud.it

Finally, TLSP introduces the notion of *linked tasks*, which have to be performed by the same employees. To the best of our knowledge, no other published variant of TLSP contains a similar concept. The only reasonable close approximation can be found in [13], where (some) resource assignments are modeled as different modes and some activities are constrained to be performed in the same mode.

Previous solution approaches have focused on a subproblem of TLSP, where a suitable grouping of tasks into jobs is given as input and cannot be modified by the solver (TLSP-S) [11,4]. However, such a "known-good" grouping cannot always be provided, and simple greedy grouping approaches often result in inferior or even infeasible groupings.

In this extended abstract, we extend the metaheuristic algorithms from [11] by new neighborhoods that deal with the grouping of tasks into jobs. This way, we obtain a solution approach for TLSP that does not require a provided initial grouping. We show that using Simulated Annealing (SA) with these new neighborhoods, we can quickly create high-quality schedules even for large instances. Preliminary results indicate that this approach produces better results than both SA for TLSP-S (even considering that the latter has the advantage of knowing a good grouping from the start). It also outperforms other solution approaches for TLSP, including an exact Constraint Programming (CP) model and, under tight time limits, a Very Large Neighborhood Search (VLNS)[3].

2 Problem definition

TLSP was first defined in a technical report [10], which also contains the full problem description. We provide here a summary of the main properties and constraints.

In TLSP, the solver has to find a schedule which consists of a partitioning of the tasks into jobs, and an assignment of a mode, timeslot and resources for each job.

A job derives its properties from the tasks it contains, which must all come from the same project and family. Within a job, tasks are executed sequentially, but without any defined order. This implies that it must fulfill all requirements of each task for its whole duration, which is the sum of the durations of its tasks plus an additional setup time. For example, the set of available units of each resource is the intersection of the available units of each contained task.

Feasible schedules must satisfy a number of constraints. This includes release dates and deadlines, available resources, linked tasks (and jobs), precedence constraints, fixed assignments, and others.

The quality of a schedule is determined via an objective function that is the weighted sum of several criteria, such as the number of jobs, the total completion time from the start to the end of each project, the number of employees assigned to each project, preferred employee assignments and internal target dates, which are usually slightly before the actual deadline.

3 Local Search framework

In this section, we give an overview of the local search framework and the Simulated Annealing (SA) metaheuristic described in [11], where it was used to solve TLSP-S.

The main concept of the framework is that of *neighborhoods*, which provide various methods to access the *moves* they contain. Solver implementations, such as metaheuristics, can be easily implemented and adapted for different problem variants simply by setting the neighborhoods they operate on.

In [11], we implemented different neighborhoods for TLSP-S. One of the best performing configurations was a combination of two neighborhoods, called *JobOpt* and *EquipmentChange*. JobOpt contains moves that modify the mode, timeslot, workbench and employee assignments of a single job, while EquipmentChange contains moves that replace a single assigned equipment unit by a different one. This special handling for equipment was required due to the sometimes huge number of potential equipment assignments, which made a neighborhood that simultaneously swapped all resources unwieldy in practice.

From among several different well known metaheuristics we implemented for our framework, we achieved the best results for TLSP-S with Simulated Annealing (SA) [7].

4 New neighborhoods

The adaptation required to make the solver for TLSP-S described in the previous section also suitable for TLSP is the introduction of new neighborhoods that contain regrouping moves, together with a careful reconfiguration of the search algorithm. If these neighborhoods are combined with those for TLSP-S [11], the same search algorithm can be used to solve also instances for TLSP.

The main challenge presenting itself is the number of potential partitions for the tasks in a family, which grows exponentially with the size of the family (already 115975 combinations for 10 tasks, which is met or exceeded by 6% of all families in our data sets). Moreover, many of these groupings (in particular if the tasks are short) will result in identical or nearly identical jobs.

We developed three new neighborhoods that modify the task grouping of a project in different ways. In combination with the existing neighborhoods for TLSP-S (JobOpt and EquipmentChange), which deal with mode, timeslot and resource assignments, these form a complete and efficient set of neighborhoods for TLSP.

All three new neighborhoods are implemented in such a way as to guarantee that their moves do not result in any new hard constraint violations, except for constraint H8 (Single Assignment) and H11 (Linked Jobs), where conflicts are allowed. This entails the following general restrictions on moves:

- Whenever two existing jobs are involved, they must belong to the same project and family.
- Fixed tasks cannot be moved to a different job.

- Whenever one or multiple tasks are added to an existing job, its mode and resource assignments must be available for all added tasks.
- In addition to the above point, there must also be a valid timeslot assignment for the job, respecting both time windows (which may changed due to the added tasks) and precedence constraints. In particular, moving tasks must not result in cycles in the precedence graph.

The new neighborhoods are:

Single Task Transfer: A single task from a source job is moved to a target job. Resource assignments are kept as-is, except where requirements change. In this case, resource assignments are adjusted as necessary. Mode and timeslot assignments are not modified (only exception: the target job may by moved to an earlier timeslot if necessary to fulfill due date or precedence constraints).

Split Job: A job is split in two by moving a randomly selected subset of tasks to a new job. It is guaranteed that doing so does not introduce cycles in the dependency graph and each job receives at least one task. For the mode, timeslot and resource assignments of the new job, several strategies are possible (see below). Resource assignments of the source job may be adjusted as described above.

Merge Job: All tasks of a job are moved to another job and the source is deleted. As for the transfer, resource assignments and timeslot of the target may have to be adjusted, but are otherwise kept as-is.

We have implemented different strategies to adjust resource assignments for affected jobs, if the requirements change due to a move (if tasks are added or removed, requirements can only increase or decrease, respectively). The resource units to add/remove can be chosen either randomly from all available units or the best units can be chosen - i.e. those that minimize the conflicts involving the job.

For the newly created job in the Split Job neighborhood, which does not have any initial assignments, different options are available. The timeslot can be either directly after the end of the source, chosen randomly (within time window and respecting precedence constraints) or the position that minimizes conflicts. The resource assignments can be copied from the source (and adjusted using either of the above strategies), assigned randomly or chosen such that they minimize conflicts in the selected position.

5 Preliminary Results

For our experiments, we used the same solver implementation as in [11], with the addition of the three new neighborhoods. We used SMAC3 [6], version 0.11.0, to tune the following configuration parameters (six independent parameters in total): The selection probabilities for the five neighborhoods during each step and the strategies to adjust resource assignments or handle assignments for the newly created job in the Split Job neighborhood.

The benchmark data set is the same as the one used in [11], which can be downloaded at `https://www.dbai.tuwien.ac.at/staff/fmischek/TLSP/`. It contains 30 randomly generated instances, ranging in size from 5 projects and around 30 tasks up to 90 projects and over 1500 tasks, as well as 3 anonymized real-world instances from our industrial partner.

Our results show that with a timeout of 10 minutes, feasible solutions could be found for all instances in nearly all runs, starting out from a simple greedily constructed initial grouping. All neighborhoods contributed to finding feasible solutions. Omitting any single neighborhoods led to a strong decrease in the number of feasible solutions, despite low weights for the Split and Merge neighborhoods in the best configuration found by SMAC.

We also compared our results to those for TLSP-S, using the algorithm and neighborhoods reported in [11]: On average, results for TLSP are better on 19 of the 33 instances, despite the fact that we did not start out from a known good and guaranteed to be feasible grouping. For some instances, we could even generate schedules that are better than the proven optima for TLSP-S under the previously known fixed grouping.

Our results also improve upon those achieved using an exact CP model for TLSP written in MiniZinc. In particular for instances with 20 projects or more, SA always found better solutions, with up to half the penalty of the results using CP.

In comparison with VLNS [3] using the CP model internally, SA finds competitive results overall. As for CP, the relative performance of SA is better on larger instances, where it slightly outperforms VLNS.

Results for longer timeouts of two hours indicate similar findings.

6 Conclusions

We have developed three new local search neighborhoods that contain task (re)grouping moves for TLSP. They can be used in combination with existing neighborhoods for mode, timeslot and resource assignments to solve the full TLSP, without any known initial grouping.

With our local search solver using simulated annealing, we could create high-quality schedules, that show improvements compared to TLSP-S, which starts out from a known good grouping. This approach also outperforms a CP model for TLSP on most instances, and is significantly better on all instances with more than 20 projects. On these large instances, it is also slightly better than a VLNS approach, while remaining competitive on the smaller instances.

For the future, we aim to perform extensive evaluations of our algorithms and their performance, and further improve the task grouping operators. We also plan to investigate other solution approaches for TLSP, such as hyper-heuristics.

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

1. Bellenguez, O., Néron, E.: Lower bounds for the multi-skill project scheduling problem with hierarchical levels of skills. In: E. Burke, M. Trick (eds.) Practice and Theory of Automated Timetabling V, pp. 229–243. Springer Berlin Heidelberg, Berlin, Heidelberg (2005). DOI 10.1007/11593577_14
2. 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). DOI https://doi.org/10.1016/S0377-2217(98)00204-5. URL http://www.sciencedirect.com/science/article/pii/S0377221798002045
3. Danzinger, P., Geibinger, T., Mischek, F., Musliu, N.: Solving the test laboratory scheduling problem with variable task grouping. In: submitted to Internation Conference on Planning and Scheduling (ICAPS) 2020
4. Geibinger, T., Mischek, F., Musliu, N.: Investigating constraint programming for real world industrial test laboratory scheduling. In: Proceedings of the Sixteenth International Conference on the Integration of Constraint Programming, Artificial Intelligence, and Operations Research (CPAIOR 2019) (2019)
5. 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). DOI https://doi.org/10.1016/j.ejor.2009.11.005. URL http://www.sciencedirect.com/science/article/pii/S0377221709008558
6. Hutter, F., Hoos, H.H., Leyton-Brown, K.: Sequential model-based optimization for general algorithm configuration. In: International Conference on Learning and Intelligent Optimization, pp. 507–523. Springer (2011)
7. Kirkpatrick, S., Gelatt, C.D., Vecchi, M.P.: Optimization by simulated annealing. Science **220**(4598), 671–680 (1983). DOI 10.1126/science.220.4598.671. URL https://science.sciencemag.org/content/220/4598/671
8. Mika, M., Waligóra, G., Węglarz, J.: Modelling setup times in project scheduling. Perspectives in modern project scheduling pp. 131–163 (2006)
9. Mika, M., Waligóra, G., Węglarz, J.: Tabu search for multi-mode resource-constrained project scheduling with schedule-dependent setup times. European Journal of Operational Research **187**(3), 1238 – 1250 (2008). DOI https://doi.org/10.1016/j.ejor.2006.06.069. URL http://www.sciencedirect.com/science/article/pii/S0377221706008344
10. 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)
11. Mischek, F., Musliu, N.: A local search framework for industrial test laboratory scheduling. Annals of Operations Research **302**, 533–562 (2021). DOI 10.1007/s10479-021-04007-1
12. Potts, C.N., Kovalyov, M.Y.: Scheduling with batching: A review. European Journal of Operational Research **120**(2), 228 – 249 (2000). DOI https://doi.org/10.1016/S0377-2217(99)00153-8. URL http://www.sciencedirect.com/science/article/pii/S0377221799001538
13. 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). DOI https://doi.org/10.1016/S0377-2217(96)00219-6. URL http://www.sciencedirect.com/science/article/pii/S0377221796002196
14. Schwindt, C., Trautmann, N.: Batch scheduling in process industries: an application of resource–constrained project scheduling. OR-Spektrum **22**(4), 501–524 (2000)