Authoring of and Interacting with Scheduling Constraints in PDDL and in a Rich Web Interface

Jan Dolejsi, Derek Long and Michal Stolba

SLB, UK

Introduction

Planning (especially temporal) and Scheduling are closely related problems and from our experience, most industrial problems contain both elements. In planning, the number of actions required to solve a problem is not specified in the problem itself. Instead, the problem is specified as a condition to be achieved. In (job-) scheduling, the typical problem is a set of jobs that must be done and solving the problem involves deciding start times and assigning resources to the jobs. In both cases, key choices to be resolved are the timing of actions and the allocation of resources, while observing a variety of constraints and optimizing the choices using a cost function. Both planning and scheduling work with ordering constraints between actions/jobs, although these are typically determined in planning problems by the relationship between the preconditions of actions and their achievers, while in scheduling problems these constraints are more often specified as explicit ordering constraints on pairs of job start/end-points.

Modeling Planning-scheduling domains

Attempts to encode *job shop scheduling* problems in PDDL (Fox and Long 2003) lead to verbose models prone to coding errors. A job, if modeled as standard `:durative-action`, needs a parameter for the `location`, where the job takes place and besides other auxiliary parameters, it needs one or more parameters to denote type(s) of `resource`(s) the job requires. Objects of the `location` and `resource` types need a way of specifying their (un-)availability over time. Consider a simplified version of the wall-decorator planning domain introduced by Long and Fox (2001), reduced to a purer scheduling problem, where painters and cleaners perform painting and clean-up jobs in residential two story houses. Both the `paint` and `clean-up` jobs have the `house` parameter and `painter` or `cleaner` resource parameter. In plain PDDL (Fox and Long 2003), the durative action needs to declare conditions and effects asserting the availability of the location and the resource(s), collocation, or correctly toggling the `busy ?r - resource` predicate, etc...

In this system demo, we show how can this repetitive boiler plate PDDL code be alleviated by the introduction of the `:job-scheduling` *requirement*, which brings the job scheduling ontology to PDDL and triggers code injection/-compilation in parsers, planners and editors that decide to support it.

For example, the Figure 1 shows the automatically generated PDDL (highlighted) within the original PDDL code, which could concentrate on the logic, that is specific to each *job*. The demo will show that the auto-generated compilations are done on the fly, so the VAL[1] domain/problem/plan validation and evaluation toolbox works as-is, while planners that decide to embrace the `:job-scheduling` requirement can take advantage of the syntax to directly translate the `:jobs` to their internal scheduling data structures without having to infer it from the combination of conditions and effects and/or expecting the modeler to follow a naming convention for types and predicates/functions if plain PDDL 2.1 was used as the input.

Blending AI-driven Decisions with Human Preferences in a Rich User Experience

Both planning and scheduling problems are encoded with hard constraints (e.g. action ordering, candidate resource applicability). Within those hard constraints, many valid solutions may be found. To nudge the solver to a solution that is not just *valid*, but also *preferred*, PDDL language offers two syntax elements: `:metric` and `:preferences`. However, we observed that the exact balance of the coefficients in the metric expression, or the exact preferences are not known at time of PDDL modeling. They emerge during the user interaction with the schedule. As scheduling problems typically apply to slower-moving processes, where the *human scheduler* participates in the process to validate the schedule and possibly adjust it to their liking.

Analogously to deploying AI planning to autonomous systems, where the plan is not dispatched blindly, step by step at predefined time points, but rather carefully validating the pre-conditions and continuously monitoring the over-all conditions, the demo will show how we take the output of the composite planning-scheduling solver and visualize it in an interactive web-based interface, while preserving the predecessor-successor and other temporal constraints that were part of the input, or inferred by the solver. The user may modify the shape of the schedule by intuitive mouse/-touch gestures dragging jobs along the timeline, or chang-

[1] https://github.com/KCL-Planning/VAL

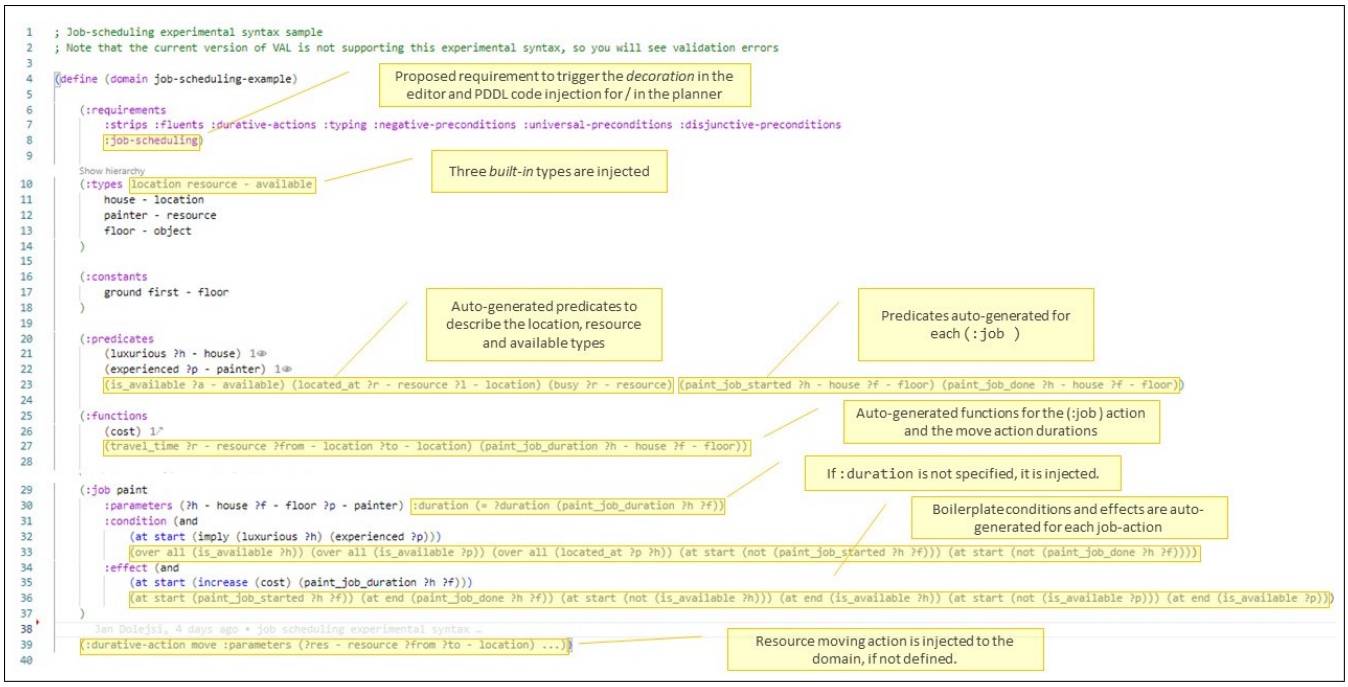

Figure 1: Example of auto-generated PDDL scheduling structure. Highlighted content is automatically inserted as a *decoration* by the editor and compiled-in by/for the solver, while the darker font content is written by the modeller.

ing the resource assignment by dragging the job between resource swim-lanes. All such modifications are sent back to the solver, which validates the new version of the schedule and re-calculate the score. The solver behind the user interface is a descendant of the POPF (Coles et al. 2010) temporal-numeric solver, coupled with a scheduling solver comprising of an initial greedy allocation algorithm followed by an implementation of tabu search enhanced by scheduling-specific large neighborhood operators.

The Figure 2 shows a static snapshot of the otherwise very interactive user interface showing a schedule for the house painting problem.

Conclusion

The demonstrated combination of PDDL notation tailored for job-scheduling and the intuitive web interface for the human to interact with the schedule within the bounds of the constraints, helps pushing the boundary of modeling and then automating a wider range of operational decision-making problems that combine both planning and scheduling paradigms and for which the pure job scheduling approach would lead to over-simplification, while the PDDL-based modeling approach would yield models that would be verbose, repetitive and tedious to maintain.

A 10 minute version of the demo may be found here: https://www.youtube.com/watch?v=YvK$_3QGtZJU$

References

Coles, A.; Coles, A.; Fox, M.; and Long, D. 2010. Forward-Chaining Partial-Order Planning. In *Proc. International Conf. on Automated Planning and Scheduling*.

Fox, M.; and Long, D. 2003. PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains. *Journal of Artificial Intelliigence Research*, 20: 61–124.

Long, D.; and Fox, M. 2001. Multi-Processor Scheduling Problems in Planning. In *Proc. Int. Conf. on AI*.

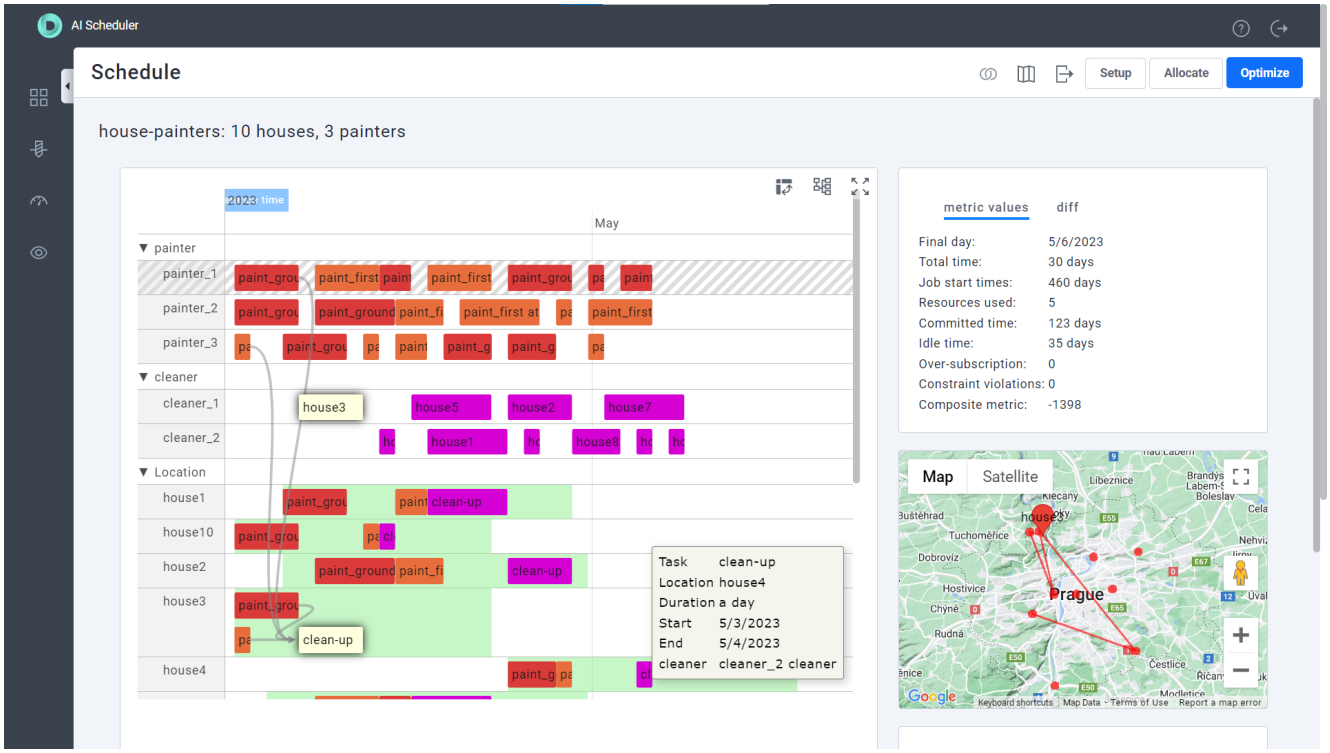

Figure 2: Screenshot of the web-based interactive schedule display showing with pale green background the time periods, where houses are available for the painters and cleaners to perform the `paint` and `clean-up` jobs.