Scheduling Problems in PDDL

Derek Long, Jan Dolejsi and Michal Stolba

SLB, UK

Introduction

Planning and Scheduling have been seen as variants of closely related problems for a long time and, indeed, this conference series has acknowledged the link in its title from its earliest foundation. Nevertheless, there are important differences: the canonical propositional planning problem lies in a higher complexity class than scheduling and this is chiefly due to the fundamental issue that, 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 and that condition might be achieved by different plans, possibly achieving the goal in different final states. In scheduling, in contrast, the canonical problem involves executing a specified set of actions (or, equivalently, completing a collection of tasks). Despite these differences, there are significant overlaps: in both cases, key choices to be resolved are the timing of actions and the allocation of resources to the execution of tasks or actions, with the constraint that certain resources will not be free to execute multiple tasks concurrently. There are often ordering constraints between actions, although these are frequently 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 tasks.

Scheduling via Planning

Given the close relationship between the problems, one might expect that scheduling problems should be straightforward to model for a planner and that a planner would then be a good solver for such problems. Unfortunately, this is not usually the case. As a simple example, consider a problem, introduced by Long and Fox (2001), in which a collection of decorators must paint a collection of walls, with each wall requiring a sequence of paints to be applied in order, and decorators moving between walls in order to paint them. This problem is a version of a multi-processor scheduling problem, in which the decorators are processors and the painting tasks are the jobs, with the ordering of paints being the equivalent of an ordering on jobs. The time for a decorator to move between walls is equivalent to a set up time to swap a processor from one series of jobs to another. FF (Hoffmann and Nebel 2001) and LPG-td (Gerevini et al. 2004) can both solve large instances of this. We explored behaviour on instances with tens of walls, 5-10 decorators and a few (up to 4) jobs at each wall. All the planners could generate plans in very short times (eg a few seconds for a plan of 140 steps with FF). However, the plans are all terrible examples of schedules: in every case, the planners allocate the bulk of the work to a single decorator, while the other decorators remain idle for the majority of the plan. POPF (Coles et al. 2010) achieves a far better parallelisation of the work, but is slower. This is a symptom of the near symmetry between resource choices (near symmetry because the decorators do not all start at the same locations), which leads to a wide branching factor as the planner searches alternative choices of resource assignment in its attempt to keep the makespan short.

Modelling Issues

In addition to the poor performance of these planners (which is shared by other planners, too), the encoding of the problems is not made straightforward in PDDL (Fox and Long 2003). Ordering constraints on jobs depend on the appropriate construction of pre- and post-condition dependencies, with static conditions then being used to capture which orderings are applicable; the movement costs of resources (decorators) between jobs has to be encoded with explicit function assignments to every possible path in the initial state; the jobs that must be carried out cannot be easily referred to by name, but, instead, rely on an explicit construction of an effect of those actions that abstracts the resource, so that the goal requires the execution of the actions, without specifying the resource that will perform them. This is all cumbersome and not intuitive.

A Way Forward

Given that there are schedulers designed for the purpose, one might accept (albeit with some disappointment) that planners are simply the wrong tool for the job and that the right way to proceed is to deploy a scheduler. However, it has been our experience that there are examples of problems in practical applications that lie at the boundaries between planning and scheduling: a large part of the problem involves scheduling a collection of specific tasks, but alongside that, there are parts of the problem that involve selecting between alternative ways to achieve the goal and choosing how those

actions should be interleaved with one another and, possibly, with the scheduling problem itself. An example of this is problems in which the primary problem is to schedule a large number of tasks on a collection of resources, but those resources require occasional planned maintenance, with several ways to achieve the maintenance and different requirements for the maintenance of different resources. In examples like this, schedulers are ill-equipped to tackle the general case, while planners remain constrained by the awkwardness of the model, the symmetry in the scheduling problem and, often, the scale of the scheduling problem. For example, it is not at all uncommon to face scheduling hundreds of jobs over tens of resources; the common approach of modern planners, to proceed by grounding, leads to the construction of many thousands of grounded actions, with high degrees of symmetry, and offering little guidance in the heuristics they generate.

We have started to explore hybrid solutions that combine the power of planning and scheduling in solving problems with this character. In this paper, we discuss the work we have carried out in improving the way that PDDL can support modelling the scheduling elements of these problems. The solutions we are exploring reduce the burden on the modeller and provide a more explict representation of the scheduling problem for the solver, enabling a clearer separation of the scheduling component from any planning component, while also exposing any interactions between them. Further important motivation for this is that, as we have experienced the challenges in maintaining PDDL models of domains as those domains extend and the applications that depend on them evolve. A more explicit scaffolding for scheduling problems avoids ad hoc models from being developed in different applications and smooths the problems of maintenance and development of modelling skills: aspects of the Knowledge Engineering challenges that are associated with capturing and using planning and scheduling domains.

Related Work

The relationship between planning and scheduling is, as we note above, one that has been widely observed and explored. There is a large body of work that examines the problems that lie, in particular, at the boundaries, but also that focus on scheduling as a solution to a range of planning-like problems. A review of some of the related material, but approaching it from outside the planning research perspective, is work by Tan and Khoshnevis (Tan and Khoshnevis 2000).

An early example of integration of the two can be seen in HSTS (Muscettola 1993), the foundation of several planning systems used at NASA AMES and developed in a system used in the construction of plans for Mars planetary rovers, Spirit and Opportunity. Cesta and his team of researchers have investigated the ground in this integration for some time, leading to examples such as workflow planning and scheduling (R-Moreno et al. 2007). Garrido et al (Garrido, Onaindia, and Sapena 2008) have also invested research in this integration challenge.

Space applications have proved a rich source for problems in this area. While HSTS was originally designed to work with Hubble, and later inspired the development of MAPGEN (Ai-Chang et al. 2004) for the Mars rovers, the James Webb Space Telescope has also inspired a collection of work exploring similar problems (Johnston and Lad 2018). These problems share a need to follow a significant collection of procedural operations, but also to combine them with goal-directed planned activity in which science data is collected using activities organised through causal linkages.

Approaching the problem from a more scheduling focused end, Laborie (Laborie 2003) has also considered the question of hybrid planning-scheduling problems, leading to work on scheduling with a wide range of interesting resource and capacity constraints and, significantly, the ILOG Scheduling tool from IBM (Laborie et al. 2018).

This work is generally focused on solving the integrated problems and, where it considers modelling, does so with languages unique to those tools. We are interested in making the modelling of hybrid problems accessible within a more broadly used language and providing tool support for both modelling and, subsequently, solving these problems.

PDDL for Scheduling

It is well understood in the Knowledge Engineering community that encoded ontologies can provide powerful and flexible frameworks for capturing models of domains and problems. An ontology can supply the core structure for representation of a domain, with the modeller then taking responsibility for the elements that are specific and particular to a given model, while relying on the ontological structure as scaffolding for the model. This approach allows the model to be expressed more concisely, since it need only focus on particular details, while supporting a more easily maintainable representation, since the ontology can be maintained, corrected and extended as an independent process, benefiting all the users simultaneously. Thus, we adopt an approach in which PDDL is extended syntactically with elements to support the scheduling ontology.

We use the requirements field in PDDL to explicitly note the use of the extension. We have found that planners are often very permissive with requirements: they typically ignore the field, but even if they do not, they do not reliably check that the requirements are respected in the domain model, or reliably reject models that demand more than the planner can support. This is largely a consequence of planners being primarily a research tool. In deployed use. respecting the requirements has more significant value, helping users to understand the limits of the planner and, potentially, supporting tailored deployment of solutions within the planner. We add the `:job-scheduling` requirement that determines that a domain uses the (experimental) scheduling extension of PDDL. The use of this requirement automatically causes the inclusion of ontological elements of scheduling problems: the types and predicates that are relevant. These types are `available`, `resource` and `location`. The first of these is the type of objects that can have windows of availability specified for them (which include resources and locations). Locations are the places at which jobs are performed, with resources being required to move between locations in order to

perform the jobs. This same concept can also be seen as set-up time, as the resources move into a state from which the jobs become accessible. The automatically generated predicates are `(is_available ?a - available)`, `(located_at ?r - resource ?l - location)` and `(busy ?r - resource)`. In addition, we generate the function `(travel_time ?f ?t - location ?r - resource)`, used to capture the time it takes for a given resource to move between the locations of different jobs.

We extend the durative action concept to include a `:job`. The use of this concept automatically implies the structure of a durative action with a particular pattern for its duration. The construction of a job automatically generates structure for the implied durative action, some of which can be modified by explicit declaration. This includes the duration, which is set, by default, to take the value of an automatically generated function `xxx_job_duration`, where `xxx` is the name of the job structure, and with parameters taken from the job, but, by default, excluding the resource parameter(s). Appropriate `over-all` conditions are generated to ensure that the resource and location are available throughout and that the resource is at the location throughout. Effects are created to record that the job has started, at the beginning, and completed at the end: `xxx_job_started` and `xxx_job_done`. Additional actions are automatically generated to model the movement of resources between jobs (with appropriate default preconditions, effects and duration).

It is important to emphasise that the automatic generation of content, which reflects the ontology of the underlying scheduling problem, is not rigid: the user is free to add parameters to jobs, preconditions and effects, and to modify the durations or other elements of the jobs as they are generated. Where necessary, these modifications propagate into the automatically generated content. For example, in the fragment of automatically generated PDDL shown in Figure 1, the user has chosen to add arguments to the job to capture that the decorating job is defined by both a house and floor – this is propagated into the duration constraint automatically. Where a model explicitly captures particular constraints, automatically generated elements are either extended appropriately, or disabled to allow the modeller's choices to dominate.

The default goal definition for the scheduling problem makes use of standard PDDL syntax in the shape of a quantified condition, such as `(forall (?h - house ?f - floor) (paint_job_done ?h ?f))`. However, the requirement that some jobs be ordered is more difficult to capture in straightforward standard PDDL. This is because the constraints are most intuitively captured as ordering on *jobs*, but PDDL currently has no terms to refer to grounded actions. So, for example, we might want to say "paint this wall before that wall", raising two problems: firstly, the constraint refers to actions that, at least from the perspective of standard planning problems, might or might not appear in a solution, and might appear multiple times and, secondly, the reference to the actions implicit in this constraint leaves the resource intentionally unspecified. To support the specification of constraints of this form, we propose an extension of the constraints offered in PDDL3 (Gerevini et al. 2009). We allow `start of` and `end of` as specifiers for the start or end time points of an action instance, referring directly to action instances using grounded parameters. However, to allow the use of such terms without forcing grounding of resource choices, we use an anonymous variable, `?`, to refer to an (implicitly) existentially quantified value. An example of the use of this is as follows:

```
(:constraints
    (ordered
      (end of (remove-old-paint houseA ?))
      (start of (drill-antenna-hole houseA ?))
      (end of (drill-antenna-hole houseA ?))
      (start of (paint houseA ?))))
```

This constraint indicates that the relevant start or end points of these actions will be ordered in the solution. The constraint is, in the context of other assumptions underlying a scheduling problem structure, more specific than if we were to attempt to use it in a general planning problem. In particular, the actions referred to in the constraint are *jobs*, so the automatically generated PDDL ensures that the solution must contain an instance of each of these jobs exactly once (with some resources used in the instances according to the scheduled solution). Thus, the anonymous resource variable indicated by the `?` argument in the two end points of the drilling action term must, in fact, take the same value, since the instance of this action must be unique. If we were to attempt to use a constraint of this form in a general planning context, it would be far less clear how to interpret it: would it imply that these actions *must* appear in the solution and be ordered, or that only if they do appear must they be ordered this way? If multiple copies of the actions appeared in a solution, would they *all* have to satisfy the constraint, or only in each sequence, and how would we interpret the binding of the end points of the drilling action in the context of multiple instances? These ambiguities mean that this constraint syntax can only be reliably exploited in the context of a scheduling problem, where the ambiguities are resolved. The unbound variables in this structure can, of course, be bound, which then enforces both an ordering and a choice of parameter values as constraints on the schedule. As we will discuss later, this mechanism also serves to convey constraints on scheduled parts of a solution into a planner when resolving the planning part of a hybrid problem.

Modelling Support

As part of the VSCode PDDL support project being maintained and developed by the authors (Dolejsi et al. 2019), the extensions described in this paper have been prototyped in the editor and are supported by a variety of quality-of-life elements. In Figure 1 can be seen examples of the presentation of the information, mostly available decoration wedging itself within the white-space and as pop-ups when hovering over relevant parts of the encoding. This keeps the model from being cluttered, but gives the modeller easy access to the content. The content can be expanded explicitly in order to modify or extend it to suit a specific context. Automatically generated predicates and functions join the user-declared components of the model and are accessed through the same menus and auto-completion support in the editor.

The tool is functional and we have confirmed that it allows us to construct and manipulate models of scheduling

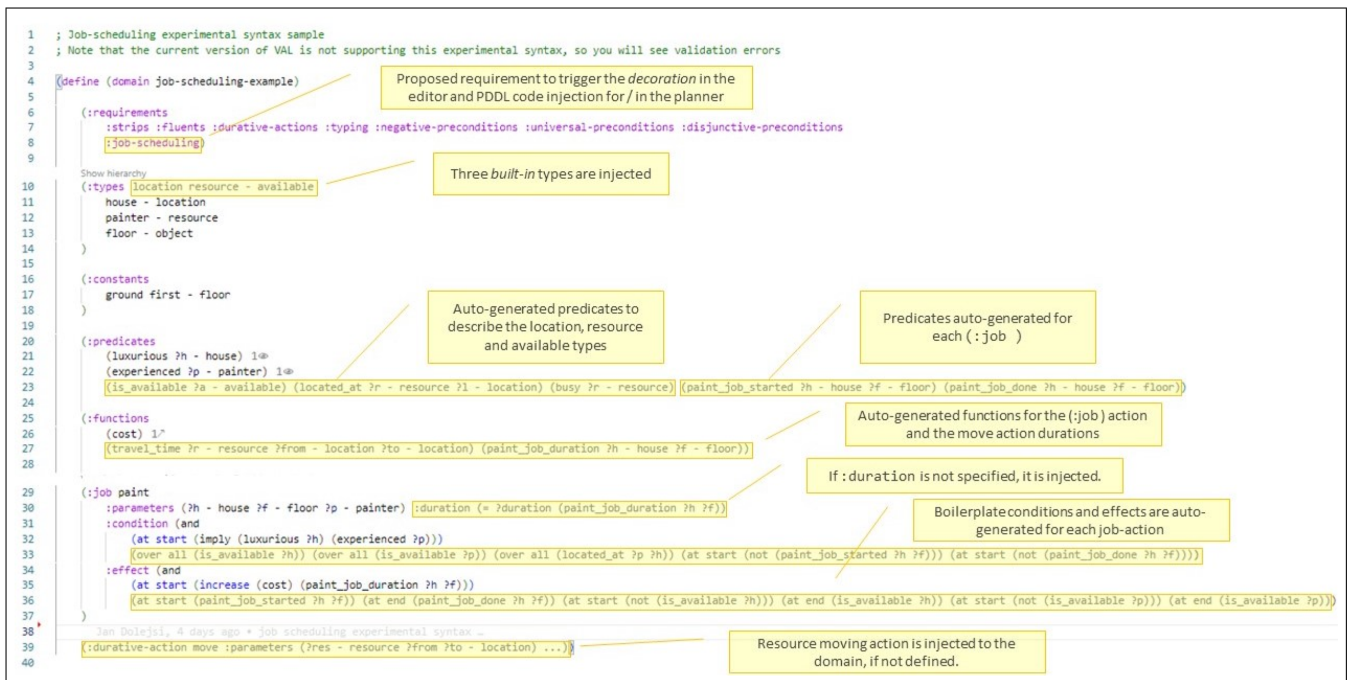

Figure 1: Example of auto-generated PDDL scheduling structure. Highlighted content is automatically inserted as a *decoration* by the editor and compiled-in by/for the solver, while the darker font content is written by the modeller.

and hybrid scheduling-planning problems with relative ease. Of course, work remains to be done to ensure that the models continue to support the user as the complexity of hybrid interactions increases. As an illustration of the easy drift from scheduling problem into planning problem, the original decorators problem is a useful example. The original `paintwall` action is declared as follows (note that this domain was not defined as a temporal variant in its original form):

```
(:action paintwall
 :parameters (?d - decorator ?x - wall ?f ?t - colour)
 :precondition
   (and (painted ?x ?f)
        (have ?t)
        (can_cover ?t ?f)
        (by_wall ?d ?x))
 :effect (and
           (painted ?x ?t)
           (not (painted ?x ?f))))
```

The `can_cover` predicate allows the sequence of paints that must be applied to a wall to be structured according to static relations defined in the initial state of the problem. The goal for this problem requires that the walls each arrive at a final colour, but these requirements imply that the walls must be painted with a sequence of colours to arrive there, so the schedule will include not only the final painting actions, but also the preparatory sequences of actions, all ordered by the `can_cover` relations between colours. As we will briefly discuss below, it is simple enough to identify the implied jobs for scheduling if the paint sequences are unique (which will be the case if the `can_cover` relation is a tree structured partial ordering). On the other hand, if the ordering constraints allow for alternative paths in the ordering between colours, then the collection of implied jobs depends on finding the shortest paths in that graph for each wall to be painted in its sequence of jobs (this was, in fact, the subject of discussion in the original work presenting this domain (Long and Fox 2001)). It requires a very small modification to the domain model to make supply of the relevant paint (encoded by `(have ?p - colour)`) become a dynamic property of the domain (for example, allowing decorators to have access to a supplier from which it can be acquired) to make the decision about the precise colour sequences to adopt become a trade off between the number of paints required and the number of painting jobs that are required, so that the problem switches from being an obvious scheduling problem into the grey area between planning and scheduling. With a simple model in which paint has either been acquired or not, the problem remains NP-hard (the space is finite and no actions need to be repeated), but relatively minor extensions such as making the painting consume the paint and limiting the capacity for carrying paint will already risk pushing the problem further into the planning space and certainly makes its solution inaccessible to a straightforward scheduling approach.

The tool support is focused on extension of PDDL2.1 as the basis for the experimental syntax, because scheduling is inherently a temporal problem (not withstanding the decorators example above). An example of the view provided through the tool in support of scheduling model construction is shown in Figure 1.

Some common elements of scheduling problems (including practical problems we have encountered) include reservoir or pooled constraints: materials that are used in amounts measured continuously rather than discretely and are drawn from an available capacity limit. We have only considered

examples in which the materials are leased by a job and returned on completion, but there are obviously use-cases in which materials are consumed from a capacity constrained supply over time. We have not yet considered extensions to support such cases.

Quality of Schedules

Although the quality of plans is sometimes less significant than simply finding a feasible plan with a reasonable length, it is much more likely that there are many feasible schedules and the challenge is to find a high quality solution. PDDL offers some tools for capturing the quality of plans and supporting comparison of plans based on these values, but these tools do not instrument all of the properties of a plan and some of the values that could be of interest are difficult to measure or access. For example, resource idle time – the time spent by a resource waiting between jobs – is not associated with any value that can be easily accessed in PDDL plans. This is because, other than `total-time`, which measures the makespan of a plan, no instruments are available other than *during* the execution of actions. This has the awkward property of requiring some sort of background acivity to be running in order to measure gaps between other actions, and even this requires machinery to invoke (clips and envelope actions (Fox, Long, and Halsey 2004) or continuous effects). We support secondary metrics through external use of these constructs, rather than wrapping them into the models used for actual planning or scheduling, but our current means to convey metric parameters to a scheduler is programmatic: work remains to be completed on resolving a simpler way to encode and convey this information.

As part of the support in our VSCode module, the end-user interface can show the quality metric or parts of it, using line plots of material consumption and cumulative cost. Where metrics are declared as additional or secondary metrics in the PDDL problem file (currently supported by the VAL toolset), the solver can be used to also export the data for the line plot and the plan/schedule (currently supported by the PDDL extension for VSCode, leveraging VAL)[1].

Hybrid Planning and Scheduling

One of the motivations for our work on modelling scheduling problems in a PDDL framework is that we have encountered real problems that lie on the boundaries between scheduling and planning. We have been frustrated to see that planners, despite solving problems that include scheduling problems as a subset, are typically not very good at finding useful schedules. However, schedulers are not equipped with the necessary capacity to solve planning problems, so the hybrid problems lie outside their scope. Although the primary focus of this paper has been on presenting our proposals and prototype for aiding modellers in the construction of scheduling problems within planning problems, we here offer a brief account of the work we have conducted in exploring the solution of such problems.

We begin with a few observations: the grounding strategy used by planners is challenged by scheduling problems at scale – a few tens of resources and hundreds of jobs, each paremeterised by, say, a job type and a location, already creates a setting that is difficult for planners, even if only because of the symmetries it creates. The choices a planner is typically best equipped to make are the choices between causal paths to a goal that lies multiple actions from the current state; scheduling has a rather different character, since the common challenge is not to thread together an intricate web of causality, but to ensure efficient division of labour between the appropriate resources, squeezing the necessary actions into as compact a block of time as possible. Goals in planning are often compact, requiring a conjunction of just a few key conditions to be achieved, while goals in scheduling are both sprawling (a large enumerated set of jobs must all be completed) and structurally simple, but focused on what must be done, rather than on why. At the boundary between these problems lie many realistic use-cases in which many actions are mandated by operating procedures, or prescribed by the lack of choices about how to achieve key conditions, while others provide logistic support that offers opportunities for clever planned efficiency gains and interleavings of activity to achieve goals by carefully coordinated action.

It is useful to note that, if a scheduling problem is embedded within a planning problem, then identifying it and solving it as a stand-alone problem offers a relaxation of the hybrid problem. In the same way that other relaxations have been demonstrated to have high value in planning, this relaxation can offer valuable insights into the structure of the solutions to the hybrid problem. A second way in which relaxations can play an important role is that, if we know which actions within a domain form the jobs to be scheduled, and we also know which parameters of those actions are the resources, we can relax the problem to remove the resource argument (and all the pre- and post-conditions that affect it), yielding a problem whose solution tells us which jobs must be scheduled and what ordering constraints apply. In fact, the first relaxed plan generated by a planner using delete-relaxations to guide search will usually contain all this information, making it easy to extract and already offering a powerful way to reduce the number of actions that must be grounded in the domain.

Once the jobs relevant to the scheduler have been identified (and the entanglement between these actions and anything relevant to the planning problem has been relaxed) a scheduler can be applied to generate a reasonable allocation of jobs to resources and time slots. These decisions can then be used to pass constraints to a planner, restricting the choices of resources to those identified by the scheduler, constraining the order of application of actions to that identified by the scheduler and including, once again, all the previously relaxed preconditions on the jobs that link the problem to the planning component. It is useful that the same ordering constraint identified as the means to communicate constraints to the scheduler can also be used to communicate the scheduler decisions to the planner (the resource parameters now instantiated). This workflow is illustrated in the sequence diagram in Figure 2.

[1] https://github.com/jan-dolejsi/vscode-pddl#line-plots-for-multiple-metric-expressions

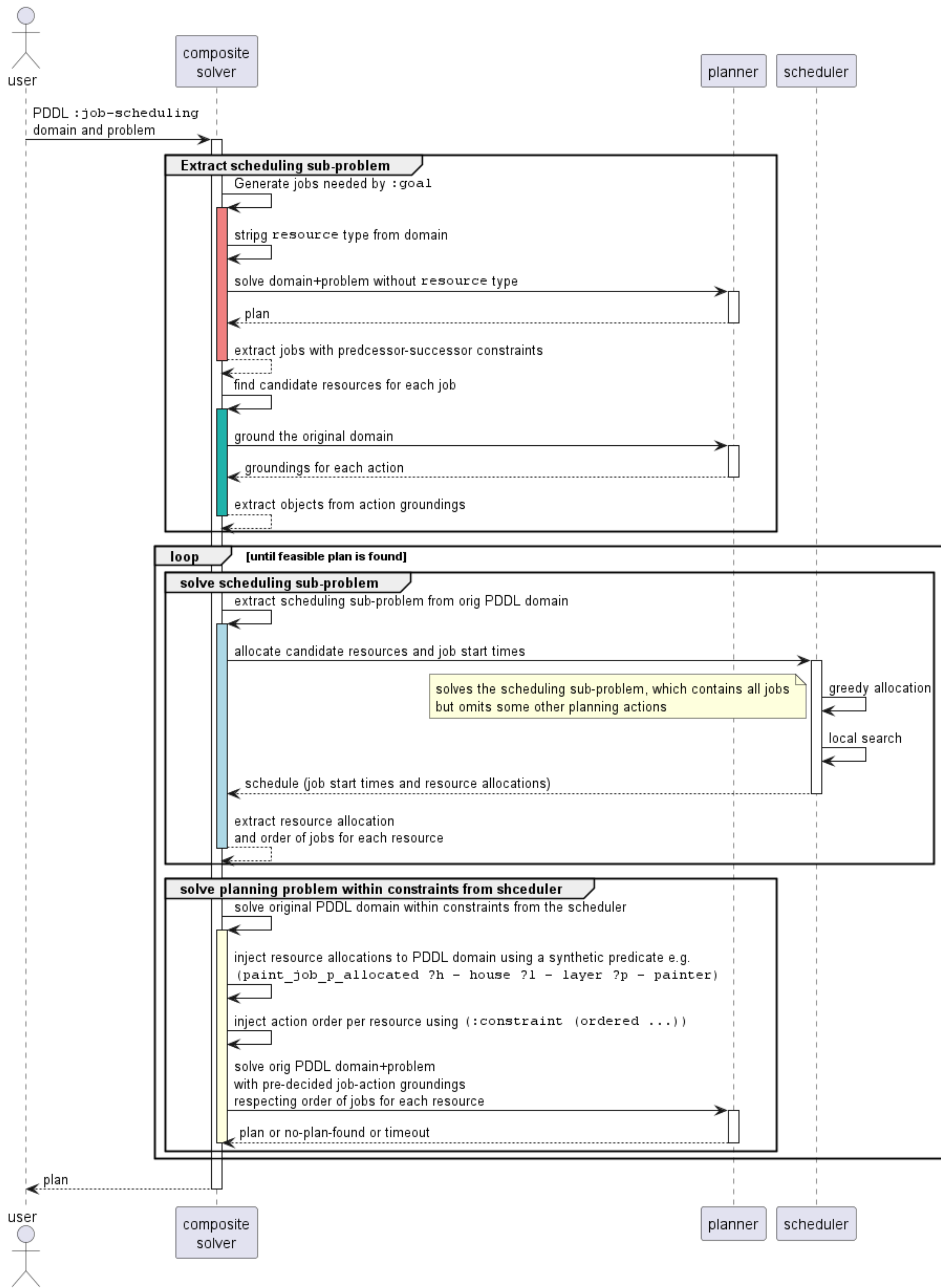

Figure 2: Sequence diagram showing the interactions between a planner and a scheduler, managed via a composite solver.

In the context of the planner, we refer to the ordering constraints generated by the scheduler as an external (partial) plan. The external plan provides guidance to the planner by informing it that the solution to the planning problem is expected to contain the given actions with the given grounding in the given order. This information is used in the following two ways.

First, the information that the provided actions are to be grounded in this particular way, that is, the resources should be assigned to the jobs at the locations, can be used to prune the grounding process. The planner need not ground all the combinations of jobs and resources but instead relies on the information supplied by the scheduler and fills in only the parameters that were not provided by the output of the scheduler because they were not part of the scheduling problem (having been relaxed out of the problem).

Second, the external solution can be used to guide the heuristic search. One possibility is to integrate it into the relaxed plan heuristic (based on the same TRPG used by POPF (Coles et al. 2010)) as follows. The relaxed plan is forced to contain the suffix of the external plan, where the suffix is obtained from the original external plan by successively removing fist appearances of actions encountered during the search (if present in the external plan). This ensures, that the heuristic estimate includes all the remaining actions from the external plan which we know will be present in the final solution, even though they might not be required in the relaxed plan computed in a standard way.

In general, the inclusion of an external solution provided by the scheduler (or directly by the user) can cause the planner to fail to find a solution, even though one exists. This can be caused by two main issues. Firstly, the restricted grounding might exclude an action grounding that is needed to find a solution. This can be mitigated by informing the grounder about the actions explicitly required by the partial plan at that point encoded in the `(:constraints ..)`. Secondly, the external solution might contain an action which is not reachable in the planning problem. Such a situation is easily detected in the first relaxed plan computation. The overarching issue in these cases is that the scheduling problem is too intertwined with the full planning problem and cannot be solved completely independently — for example, resource availability might depend on the actions which are not part of the scheduling problem, resources might be consumed, etc. This class of problems needs a closer integration between the planner and the scheduler where, in the above cases of planning failures, the given schedule is marked as a *nogood* for the scheduler and the scheduler is re-run to find a different solution.

Another completely orthogonal possibility to integrate a scheduling solver into the planner is to use a post-processing phase. Optimal temporal planning is often prohibitively complex and even respecting metrics is often not easy. Moreover the partial-ordering logic in the forward action-chaining algorithm of the temporal planner sometimes is not powerful enough to recognize that two actions should not be ordered, because the nature of their interaction is not tangible in the outcome. By feeding the solution temporal plan into a scheduling solver which understands PDDL, the solution can be optimised within the constraints given by the planner, lifting resources and rescheduling by reallocating them, subject to the structural limitations imposed by the causal links in the plan.

Integrating Planning into Scheduling

A planner benefits from integration with a scheduler, but there are also opportunities vice versa. The relaxation-based reachability analysis often used by planners can be used to prune unreachable job-resource assignments, thus reducing the search space of the scheduler. For example, if painters must be trained before allowed to decorate a luxury house, and some painters cannot meet the preconditions for training, then grounding will reveal which painters can be considered. The expressivity of PDDL combined with the filtering by advanced grounding or relaxed reachability allows the user to specify jobs more concisely without the need to explicitly enumerate candidate resources for each job.

Scheduling solvers typically rely on a feasible initial solution and resource allocation. In some use cases (where it is fast enough), planners can be used to come up with the initial solution without having to implement such an algorithm in the scheduling solver. This can be done either on the simpler scheduling problem, or, if the scale allows it, on the original problem including all the planning and scheduling constraints. As mentioned earlier, the FF (Hoffmann and Nebel 2001) and LPG-td (Gerevini et al. 2004) planners are fast enough to generate valid, though sub-optimal, resource assignments for the Decorators (2001) domain.

These ideas can play a role in the interplay between planning and scheduling in both directions. We have already implemented prototypes of these couplings to explore further.

Example

We briefly illustrate the use of the tool to generate PDDL. A simple example is shown, from the perspective of the modeller, in Figure 3.

The associated problem file, including a user-imposed constraint, which adds the goal to have coffee with the owner of the blue house in between of the two `paint` actions is shown below.

```
(define (problem paint-two-houses)
    (:domain job-scheduling-example)
    (:requirements :timed-initial-literals)
    (:objects
        red blue - house
        jay pro - painter
        pub - location)

    (:init
        ; red house (only available in a time window)
        (at 3 (is_available red))
        (at 13 (not (is_available red)))
        (= (paint_job_duration red ground) 4)
        (= (paint_job_duration red first) 4)
        (= (clean-up_job_duration red) 1)

        ; blue house
        (is_available blue)
        (luxurious blue)
        (= (paint_job_duration blue ground) 4)
        (= (paint_job_duration blue first) 4)
        (= (clean-up_job_duration blue) 2)

        ; jay (inexperienced) painter
        (is_available jay)
        (located_at jay pub)
```

```
; Job-scheduling experimental syntax sample
; Note that the current version of VAL is not supporting this experimental syntax, so you will see validation errors

(define (domain job-scheduling-example)

    (:requirements
        :strips :fluents :durative-actions :typing :negative-preconditions :universal-preconditions :disjunctive-preconditions
        :job-scheduling)

    (:types location resource - available
        house - location
        painter - resource
        floor - object
    )

    (:constants
        ground first - floor
    )

    (:predicates
        (luxurious ?h - house)
        (experienced ?p - painter)
        (had-coffee-with-owner ?h - house)
        (is_above ?f1 ?f2 - floor)
        (is_available ?a - available) (located_at ?r - resource ?l - location) (busy ?r - resource) (paint_job_started ?h - house ?f - floor) (paint_job_done ?h - house ?f - floor) (clean-up_job_started ?h - house) (clean-

    (:functions
        (cost)
        (travel_time ?r - resource ?from - location ?to - location) (paint_job_duration ?h - house ?f - floor) (clean-up_job_duration ?h - house))

    (:job paint
        :parameters (?h - house ?f - floor ?p - painter) :duration (= ?duration (paint_job_duration ?h ?f))
        :condition (and
            ; luxurious houses must be decorated by experienced painters
            (at start (imply (luxurious ?h) (experienced ?p)))
            ; example of job predecessor/succcessor encoding: upper floors should be done first
            (at start (forall (?f1 - floor) (imply (is_above ?f1 ?f) (paint_job_done ?h ?f1))))
            (over all (is_available ?h)) (over all (is_available ?p)) (over all (located_at ?p ?h)) (at start (not (paint_job_started ?h ?f))) (at start (not (paint_job_done ?h ?f))))
        :effect (and
            (at start (increase (cost) (paint_job_duration ?h ?f)))
            (at start (paint_job_started ?h ?f)) (at end (paint_job_done ?h ?f)) (at start (not (is_available ?h))) (at end (is_available ?h)) (at start (not (is_available ?p))) (at end (is_available ?p)))
    )
    (:job clean-up
        :parameters (?h - house ?p - painter) :duration (= ?duration (clean-up_job_duration ?h))
        :condition (and
            (at start (and
                (forall (?f - floor)
                    (paint_job_done ?h ?f)
                )
            ))
            (over all (is_available ?h)) (over all (is_available ?p)) (over all (located_at ?p ?h)) (at start (not (clean-up_job_started ?h))) (at start (not (clean-up_job_done ?h))))
        (:effect (at start (clean-up_job_started ?h)) (at end (clean-up_job_done ?h)) (at start (not (is_available ?h))) (at end (is_available ?h)) (at start (not (is_available ?p))) (at end (is_available ?p))))

    (:durative-action coffee
        :parameters (?h - house ?p - painter)
        :duration (= ?duration 1)
        :condition (and (at start (not (had-coffee-with-owner ?h))) (over all (not (busy ?p))) (over all (located_at ?p ?h)))
        :effect (and (at end (had-coffee-with-owner ?h)))
    )

    (:durative-action move :parameters (?res - resource ?from ?to - location) ...))
```

Figure 3: Full listing of a PDDL domain including the editor-rendered auto-generated PDDL syntax. The example can be found in this repository: https://github.com/jan-dolejsi/vscode-pddl-samples/tree/master/JobScheduling

```
        ; pro painter
        (is_available pro)
        (experienced pro)
        (located_at pro pub)

        (= (travel_time pro pub blue) 1)
        (= (travel_time pro pub red) 3)
        (= (travel_time jay pub blue) 1)
        (= (travel_time jay pub red) 3)

        (is_above first ground)

        (= (cost) 0))

    (:goal (and (forall
                (?h - house)
                (clean-up_job_done ?h))))

    (:constraints (and
        (ordered
            (end of (paint_job blue first ?))
            (start of (had-coffee-with-owner blue))
            (end of (had-coffee-with-owner blue))
            (start of (paint_job_started blue ground ?))
        )))

    (:metric minimize (cost)))
```

Finally, a planner generates the solution in Figure 4.

Conclusion

Modelling is a challenging activity. The construction and maintenance of domain models relies on the same discipline and support as writing code. We have been motivated to explore the construction and maintenance of scheduling problems and hybrid planning-scheduling problems, using a familiar modelling language, and providing consistent modelling choices and scaffolding for the construction of these problems. Much work remains to be completed, but we have a prototype illustrating both the modelling process and one strategy for solving such hybrid problems.

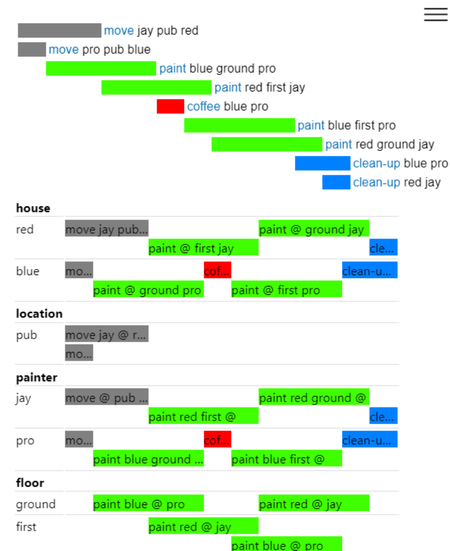

Figure 4: Resulting plan showing the two painters busy decorating two houses, while one of them has the prescribed coffee break.

References

Ai-Chang, M.; Bresina, J.; Charest, L.; Chase, A.; Hsu, J.-J.; Jonsson, A.; Kanefsky, B.; Morris, P.; Rajan, K.; Yglesias, J.; Chafin, B.; Dias, W.; and Maldague, P. 2004. MAPGEN: mixed-initiative planning and scheduling for the Mars Exploration Rover mission. *IEEE Intelligent Systems*, 19(1): 8–12.

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

Dolejsi, J.; Long, D.; Fox, M.; and Muise, C. 2019. From a Classroom to an Industry From PDDL "Hello World" to Debugging a Planning Problem. In *System Demonstrations at the Twenty-Ninth International Conference on Automated Planning and Scheduling (ICAPS)*.

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.

Fox, M.; Long, D.; and Halsey, K. 2004. An Investigation into the Expressive Power of PDDL2.1. In *Proceedings of the 16th European Conference on Artificial Intelligence*, ECAI'04, 328–332. NLD: IOS Press. ISBN 9781586034528.

Garrido, A.; Onaindia, E.; and Sapena, O. 2008. Planning and scheduling in an e-learning environment. A constraint-programming-based approach. *Engineering Applications of Artificial Intelligence*, 21(5): 733–743. Constraint Satisfaction Techniques for Planning and Scheduling Problems.

Gerevini, A.; Saetti, A.; Serina, I.; and Toninelli, P. 2004. LPG-td: a Fully Automated Planner for PDDL 2.2 Domains. In *Proc. Int. Conf. on AI Planning and Scheduling (ICAPS)*.

Gerevini, A. E.; Haslum, P.; Long, D.; Saetti, A.; and Dimopoulos, Y. 2009. Deterministic planning in the fifth international planning competition: PDDL3 and experimental evaluation of the planners. *Artificial Intelligence*, 173(5): 619 – 668. Advances in Automated Plan Generation.

Hoffmann, J.; and Nebel, B. 2001. The FF planning system: Fast plan generation through heuristic search. *Journal of Artificial Intelligence Research*, 14: 253–302.

Johnston, M.; and Lad, J. 2018. Integrated Planning and Scheduling for NASA's Deep Space Network–from Forecasting to Real-time. In *2018 SpaceOps Conference*, 2728.

Laborie, P. 2003. Algorithms for propagating resource constraints in AI planning and scheduling: Existing approaches and new results. *Artificial Intelligence*, 143(2): 151–188.

Laborie, P.; Rogerie, J.; Shaw, P.; and Volim, P. 2018. IBM ILOG CP optimizer for scheduling. *Constraints*, 23: 210––250.

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

Muscettola, N. 1993. HSTS: Integrating planning and scheduling. Technical report, CARNEGIE-MELLON UNIV PITTSBURGH PA ROBOTICS INST.

R-Moreno, M. D.; Borrajo, D.; Cesta, A.; and Oddi, A. 2007. Integrating planning and scheduling in workflow domains. *Expert Systems with Applications*, 33(2): 389–406.

Tan, W.; and Khoshnevis, B. 2000. Integration of process planning and scheduling -— a review. *Journal of Intelligent Manufacturing*, 11: 51—63.