Position Paper: Designing a Scheduling Competition

Mark S. Boddy
Adventium Labs
mark.boddy@adventiumlabs.org

Introduction

Judging by the effects that similar competitions have had in other research areas, a successful scheduling competition has much to offer the field. However, the difficulties to be overcome are quite significant; to my knowledge, there have been at least two previous attempts to organize such a competition, both ultimately unsuccessful.

In this position paper, I explore some of the issues, looking in particular to the International Planning Competition (IPC) for lessons in what might go wrong, and how to avoid those problems. Following that, I present a brief, informal proposal for a competition language.

Lessons From the Planning Competition

Last year saw the fifth IPC. Running every two years, these competitions have grown considerably since the first one. Grown in several ways: there are more competitors, there is a broader range of problems addressed (including the introduction of the Probabilistic Track), and the language used for defining problem instances has steadily expanded in expressive power.

The genesis of the IPC was the agreement among a sizable committee of planning researchers to define the *Planning Domain Description Language* (PDDL) in a minimalist fashion. PDDL 1.0 was an almost purely classical planning language, with some simple Hierarchical Task Network (HTN) features added on. Reductionist though it was, the initial dialect of PDDL was further simplified for the initial planning competition by removal of the HTN features of the language.

This narrow focus was essential to getting the IPC off the ground. A small language minimized the amount of work competitors needed to do to parse the new language and to deal with the planning constructs thus specified. A small set of similar problem instances constrained the limited field of competitors (there were five) to a small number of different classes of problems, so that meaningful comparisons were possible. The initial problem domains were suitable (or at least feasible) for solution using propositional STRIPS-rule planners. This was a common technical approach, arguably the one supported by the largest group within the ICAPS community, and so a good choice for initial focus. Of the 5 competitors in the first IPC, only two ventured beyond STRIPS rules into ADL.[1]

Lesson 1: Start with a small, easy-to-parse language, and a set of problem domains that are similarly simple, and structurally similar.

As the IPC has matured, however, this approach has become a hindrance. Successive competition organizers have introduced more complex problem domains, more directly motivated by specific real-world applications. This process culminated with the fourth IPC, which included 7 domains, of which 6 were specifically designed to reflect real-world applications. [2] However, the organizers' desire to continue to support the most basic planning representation (propositional STRIPS rules) led them to structure some of the domains in unintuitive and unrealistic ways, specifically so that they could be compiled into STRIPS representations of a reasonable size.

Lesson 2a: As the competition matures, focus more on defining relevant problems, than on supporting a favored technology.

Similar issues have arisen in the probabilistic planning competition. The winner for the probabilistic track of IPC-4, which also "won" in IPC-5 (it was not an official entry), was FF-rePlan, a system that includes no explicit reasoning about probabilities at all. Instead, it replans whenever the state in which it finds itself differs from that predicted by a deterministic plan.

There are a number of ways in which FF-rePlan could be defeated, or at least hampered. For example, the domain can be partially observable, such that reasoning explicitly about what *may* have happened is required to do well. Or the domain can include stochastic failure outcomes, requiring explicit reasoning about the likelihood of falling into a sink state. Of course, both of these features can also present substantial difficulties for probabilistic planners as well, the former by significantly increasing the computational cost of planning, and the latter by requiring either planning all the way to the goal state, or very good heuristics, as a way of avoiding those same sink states.

Lesson 2b: If a different technology, especially a simpler one, does a better job on your problem domains, don't immediately assume that you've done a bad job of problem definition. Maybe the favored technology isn't really appropriate.

There is a third lesson in this saga, as well: it is not obvious that an appeal to Moore's law will "right" things. Even if we assume that the rigorous stochastic planners are given infinite computing resources, either to construct an initial plan or at run-time, nobody that I know of has quantified the loss accepted in using FF-rePlan. How far off optimal, by any measure, will the results be?

There are real-world examples where the answer to this question is that the loss will be minimal. For example, in the mid 1990's Barry Fox and Mark Ringer set up and maintained a benchmark site for Resource-Constrained Project Scheduling Problems. This site presented a series of bench-

[1] ftp://ftp.cs.yale.edu/pub/mcdermott/aipscomp-results.html

[2] http://ls5-www.cs.uni-dortmund.de/ edelkamp/ipc-4/

mark problems of increasing complexity (in terms of the constraints involved), and people were invited to submit the solutions they found using their favorite solvers.[3] The problem instances were drawn from aircraft assembly scheduling problems within McDonnell Douglas. For these problems, a very simple heuristic approach involving no search at all provides results that are within about 5% of optimal makespan, with solution times measured in milliseconds, on desktop systems then available. This approach was independently developed by two different organizations, and is called either *Schedule Packing* or *Doubleback Optimization*[4][5]

Lesson 3a: Try the simple approach *first*.

Lesson 3b: Eventually, it's not that the technology is immature, it's that it isn't appropriate. The more the problem domains are supposed to reflect real applications, the more force this point has. Artificial complexity is rarely if ever tolerated in fielded solutions, and any complexity that can be engineered away at minimal cost will be.

Lesson 3c: More directly relevant to the competition: make sure the simple approach is one of the competitors.

The evolution of PDDL 3.0 and IPC-5 took a turn away from the further development of real-world domains. Instead, the organizers chose to focus on adding expressive power to the language. Specifically, they added *soft constraints*, so as to define optimization problems in which some goals might not be achieved, or some constraints violated. This is an interesting extension, and may well eventually lead to systems that are more capable in terms of their real-world application, but the immediate effect is to provide a richer set of theoretical, rather than practical, issues to wrestle with.

Lesson 4: Decide explicitly the extent to which the competition focus is on mathematics, versus engineering.

Planning Versus Scheduling

As discussed rather extensively in the course of two previous ICAPS workshops on the topic of "Integrating Planning into Scheduling," the difference between planning and scheduling has more to do with the solution techniques employed than with fundamental differences in the problems being addressed.

With the advent of more recent dialects of PDDL (PDDL 2.1, PDDL+, PDDL3.0), most or all of the features one would expect to find in a scheduling problem can be expressed in the IPC's domain description language. The exclusive use of equipment can be represented using preconditions and postconditions, or by representing a resource as a first-class object. The choice of whether or not to perform a given task can be represented in a planning model as partial goal satisfaction. Classical planning models can be used to encode continuous processes, metric time, and overlapping activities. Classical planning models can, with some pain and inconvenience, be used to represent the creation and destruction of objects. The cost of violating constraints (tardiness, for example), can be represented in PDDL 3.0.

Motivating the addition of a scheduling competition to the already well-established International Planning Competition is hard to do on the grounds of expressive power. The focus needs to be on the relevant features of the domain examples, not on the language itself.

Problem Characteristics

So, what kinds of problems should the first "Interational Scheduling Competition" (ISC-1?) start with? Scheduling algorithms and data structures are broadly adapted for problems including:

1. Large numbers of activities, loosely coupled through simple effects on common propositions (typically represented as resources of one sort or another)
2. Little or no representation of state, other than the simple form in the previous bullet
3. Metric time, including activity duration, and temporal constraints between activities (e.g. ordering constraints), or between activities and a global clock
4. Parallel execution of activities
5. For some problems: strong emphasis on making a choice of which activities to include
6. For some problems: small, interdependent subsets of activities (e.g., the steps in manufacturing a given product) such that one should either include all of them, or none of them
7. Constraints on feasible schedules (e.g., resource bounds)
8. Feasibility or optimality computed according to makespan, deadlines, or throughput, or according to such measures as how many activities were successfully executed, or even more complex measures (the value of the goods produced).

Based on the characteristics above, I propose that the initial competition language should include:

- Real-valued time points and temporal constraints.
- Durative activities
- Instantaneous events
- Real-valued releasable resources, with upper and lower bounds.
- Real-valued consumable resources, with upper and lower bounds.
- Simple, fixed decomposition of tasks into partially-ordered sets of activities.
- Objective functions defined over schedules, which can refer to the values of time points, the inclusion or otherwise of activities, and the values at specified points in time associated with resources.

[3]While the site is not active any longer, it is still accessible, including the benchmark problem data, through the Wayback Machine at web.archive.org: http://tinyurl.com/2pmj6w.

[4]http://cirl.uoregon.edu/research/schedulePacking.html

[5]Numerous, similar "surprises" have occurred in other research areas, for example Charniak's work in the early 1990's on very simple methods for probabilistic parsing. You'd think we'd learn. . .

This language is not sufficiently restrictive to ensure that we construct a set of qualitatively similar problem domains So, I also propose the following restrictions for an initial set of competition problems:

1. Activities are not preemptible.
2. Consumable resource changes are instantaneous.
3. Task decompositions (e.g., steps in a manufacturing process) are specified as simple rules.
4. Context-dependent setup times are excluded.[6]

This suffices to cover job-shop, flow-shop, and resource-constrained project scheduling problems, as well as a broad variety of scheduling problems that include some planning, for example from manufacturing, or the control of autonomous robots.

We should exclude the most reductive problems: job-shop, tours and travelling salesman problems with no significant complicating features. To focus as the CFP for this workshop suggests on problems closer to real applications means that more structure is needed. Specifically *which* structure, within the language features suggested above, is where the meat of the discussion for this workshop will lie. My suggestions would be:

- The more complex versions of RCPS, drawn from the benchmarks site cited above, or from the benchmarks at PSPLIB.[7]
- A simple batch scheduling application, with fixed recipes and fixed paths between equipment.
- A telescope scheduling problem (but this would require context-dependent setup times).
- A satellite downlink scheduling problem (AFSCN, or EOSDIS).

Obtaining or generating data for any of these should be fairly straightforward.

One might also think of looking for other benchmarks that we could adopt here. The OR Library[8] does have some scheduling problems. But I would much rather start with a description of what we want in a set of scheduling domains, then go see what's available.

[6]This will serve to exclude domains involving transport between locations, so might be open for debate.

[7]http://129.187.106.231/psplib/

[8]http://people.brunel.ac.uk/ mastjjb/jeb/info.html