# Lessons Learned in Applying Domain-Independent Planning to High-Speed Manufacturing

**Minh Do and Wheeler Ruml**
Embedded Reasoning Area
Palo Alto Research Center
{minhdo,ruml}@parc.com

### Abstract

Much has been made of the need for academic planning research to orient towards real-world applications. In this paper, we relate our experience in adapting domain-independent planning techniques to a real industrial problem. We present a simpler formulation of a temporal planning graph-style heuristic, show how to extend it to take resources into account, and evaluate its importance in practice. We also derive several general lessons from our experience which might guide researchers looking to increase the relevance of their work or industrial practitioners seeking to apply planning research to real problems.

## Introduction

In our previous work (Ruml, Do, and Fromherz, 2005), we described a manufacturing problem domain that emphasizes on-line continual problem solving. This paper makes two main contributions: (i) heuristic extension techniques utilizing logical and resource mutexes to meet the productivity of the more complicated and faster manufacturing plants built recently, and (ii) the lessons learned in adapting domain-independent planning techniques in a real-world application.

**Domain background:** The domain is based on a manufacturing process control problem encountered by one of our industrial clients. It involves planning and scheduling a series of job requests which arrive asynchronously over time. The plant runs at high speed, with several job requests arriving per second, possibly for many hours. Therefore, the first requirement for the planner is to be able to produce plans consistently within a fraction of a second while being able to run for a long period of time. It needs to take into account all the real-time constraints such as various delays in network communication or in getting machine controller's responses. The typical plants can be schematically represented as a network of transports linking multiple machines from a few to a few hundred machines and transports (Figure 1). Most manufacturing actions require the use of physical plant resources, so planning for later jobs must take into account the resource commitments in plans for previous jobs, some are already released for production.

To summarize, our domain is finite-state, fully-observable, with classical goals of achievement. However, planning is on-line with additional goals arriving asynchronously. Actions have real-valued variable durations and use resources. Plans for new goals must respect resource allocations of previous plans. Execution failures can occur, but are rare enough that we don't explicitly plan ahead for them. A more detailed discussion on this manufacturing domain can be found in (Ruml et al., 2005).



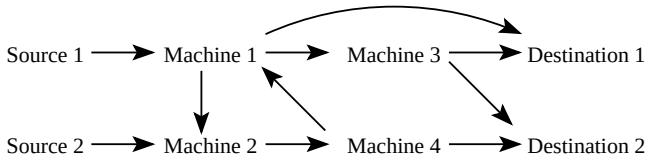


Figure 1: A schematic view of a manufacturing plant.

Our current main objective function is to optimize the productivity of the manufacturing plant. The combination of this objective function, the highly-reconfigurable nature of our domain, the wide variety of plant designs, and the real-time aspect in which plans of different jobs interact in many different ways, make this a suitable planning domain for domain-independent planning techniques. Any pre-compiled domain control rules will not likely to work for different plants with very different types of design and machines, and may also prune optimal solutions when the machine is reconfigured due to user input or runtime exceptions.

**Our Approach:** We model this manufacturing domain as a temporal extension of STRIPS. Before planning begins, a domain description and part of the initial and goal states are provided. Then the problem descriptions arrive on-line, containing logical goal state and part of initial state. The initial state is then built by combining the logical initial state provided with the resource allocations for plans of previous jobs. Our action representation is similar to the durative actions in PDDL2.1 with extensions are: (1) variable action duration with upper and lower bounds; (2) explicit representation of different types of resources; (3) actions have "setup-time". One action may require multiple resource allocations that can overlap with action duration arbitrarily.

We have implemented our own temporal planner using an architecture that is adapted to this on-line domain. The over-

all objective is to minimize the end time of the known jobs. The planner uses state-space regression to plan each job, but maintains as much temporal flexibility as possible in the plans by using a simple temporal constraint network (STN) (Dechter, Meiri, and Pearl, 1991). As we and our customer start exploring more complicated plants, it became obvious that our early implementation based on the admissible temporal $h^1$ heuristic (Haslum and Geffner, 2001) and a "lifted" hybrid regression planner (Ruml et al., 2005) is not powerful enough. After experimenting with grounded progression planner, our current algorithm of choice is a fully grounded regression planner with more informed heuristics based on mutual-exclusion reasoning and runs an order-of-magnitude faster than the previous planner and match the productivity of bigger manufacturing plants. The planner has also been used for several purposes such as: system analysis to help find good plant configurations and to simulate higher level job selection.

## Heuristic Extensions

Our overall objective function is to minimize the earliest possible end time of the plan for the current job. We want effective planning heuristics that are:

- *Admissible:* maintaining high productivity of the plant is an important criterion for our customer.
- *Informed* and *easy to compute*: in most cases, we are only allowed a fraction of a second to find a feasible plan.

In the previous implementation, to estimate the remaining makespan in a regression temporal planner, we built the bi-level temporal planning graph (Smith and Weld, 1999) without mutex that estimates the fastest way to achieve the logical state $L$ as the maximum over the times taken to produce the individual literal in $L$. We have improved on this heuristic by taking different types of mutual exclusion relations into account.

**Logical mutexes:** In our graph expansion algorithm, for each action $a$ and fact $f$, we store the first time points $t_a$ and $t_f$ at which $a$ and $f$ appear in the temporal planning graph. For mutex propagation, we also store the first time point at which each pair of facts $(f_1, f_2)$ of actions $(a_1, a_2)$ become *non-mutex*.

1. $\forall f, a, f_1, f_2, a_1, a_2 : t_a = t_f = t_{(f_1,f_2)} = t_{(a_1,a_2)} = \infty$.
2. $\forall f, f_1, f_2 \in I : t_f = 0, t_{(f_1,f_2)} = 0$.
3. Dynamically update the values of $t_a, t_f, t_{(f_1,f_2)}, t_{(a_1,a_2)}$ starting from the initial state $I$ and time $t = 0$ as follows:
   - $t_a = max(setup\_time(a), max(t_f : f \in Prec(a)), max(t_{(f_1,f_2)} : f_1, f_2 \in Prec(a)))$ (1)
   - $t_f = min(t_a + dur(a) : f \in Eff(a))$ (2)
   - $t_{(f_1,f_2)} = min(t_{(a_1,a_2)} + max(dur(a_1), dur(a_2)) : f_1 \in Eff(a_1), f_2 \in Eff(a_2))$ (3)
   - $t_{(a_1,a_2)} = max(t_{a_1}, t_{a_2}, max(t_{(f_1,f_2)}) : f_1 \in Prec(a_1), f_2 \in Prec(a_2)$ (4)
4. Stop when $\forall g, g_1, g_2 \in G : t_g, t_{(g_1,g_2)} < \infty$ or fix-point.

The time point at which all the goals are achieved pair-wise non-mutex is the heuristic value (under)estimating the remaining makespan. The planning graph expansion process is not done once but may be revisited if goals representing a regressed state do not appear non-mutex in the graph and we haven't reached the fix-point computation in the last round of expansion.

**Function** $CheckEarliest(r, t, d)$
  $R = \{[s_1, e_1], [s_2, e_2], ...[s_m, e_m]\}$: allocs of $r$.
  *min_time := t*
  for *k := 1* to $m$ do
    if $Earliest(s_k) > min\_time + d$ then
      Return *min_time*
    else
      $min\_time := Earliest(e_k)$
  Return($min\_time$)
**end function;**

Figure 2: Check the earliest time for a resource allocation

**Incorporating resource mutexes:** The planning graph discussed until now assumes the interference relations only occur between actions related to a given job that we are planning for. However, most of the time, the plant is not empty and there are plans for jobs that are either (i) executing; or (ii) found by the planner but haven't been sent to the plant. Those plans already made resource reservation, either fixed (for (i)) or flexible (for (ii)), and can interfere with actions related to the current job because they can compete for the same resource.

Thus, to improve the heristic estimate, we take into account resource mutexes, thus incorporating scheduling resource contention constraints into the temporal planning graph. In short, to find the earliest time $t_a$ at which an action $a$ can possibly execute, the condition is not only that all of $a$'s preconditions appear non-mutex in the planning graph but also that there is no resource conflict between any resource $r$ used by $a$ and all current allocations of $r$ (given to previous plans and by external processes.). If the resources used by $a$ is $R_a = \{(r_1, o_1, d_1), (r_2, o_2, d_2), ..., (r_n, o_n, d_n)\}$[1], then the algorithm recursively calls $CheckEarliest(r_i, t_a + o_i, d_i)$ (Figure 2) to find the earliest possible time for $t_a$ at which there will be no resource conflict with previous allocations of $r_i$.

With resource mutexes, the starting times of actions are adjusted to higher than the time points at which their preconditions can be achieved. However, $t_G$ still underestimates the first time that we can achieve the goals and thus is still an admissible heuristic for our main objective function of minimizing the end time of current job.

## Empirical Evaluation

The planner as described in this paper is fully implemented in OCaml and is used both by us and by our industrial research partner to control different manufacturing plants. Figure 3 and 4 show the performances of our planner in two of the biggest built prototype plants by our customer and us.

[1] Allocation $(r, o, d)$ of $a$ means $a$ uses $r$ starting from offset $o$ from the starting time of $a$ for a duration of $d$

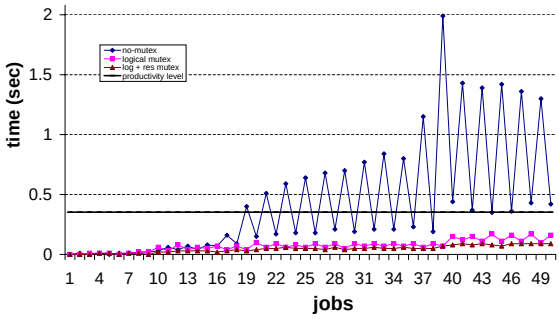


Figure 3: Performance in the plant by our industrial partner.

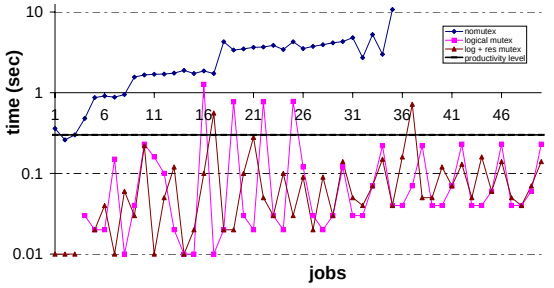


Figure 4: Performance for the current plant built by us.

The plant in Figure 3 is a simpler one with 25 main components and 35 action schemata with the productivity of 170 jobs per minute (i.e. 0.353sec/job). The shortest possible plan contains 8 actions and the planner needs to handle the interaction of around 100 actions. The planner with no mutex, only logical mutexes, and both logical and resource mutexes take respectively an average of 0.4191, 0.0732, and 0.0458 secs to solve one job.

The plant in Figure 4 is more complicated with around 90 machines, 212 action schemata with the shortest possible plan contains 16 actions. The planner needs to regularly reason about the interactions between more than 150-200 actions. The productivity level is 200 jobs-per-minute (i.e. 60/200 = 0.3sec/job). Without mutex, the planner quickly go over the base time of 0.3 second after a few jobs. With mutexes (logical, resource) the planner generally output plans faster than 0.3 secs and occasionally takes more than 0.3 seconds. However, because it plans ahead 10 jobs , those jumps in plan time still doesn't hurt the planner in matching with the full productivity of the plant. The planner averages 0.1336 secs with only logical mutexes and 0.0928 secs (1.44x improvement) with both types of mutexes.

## Lessons Learned

Given that our real-time planning domain is substantial more complicated than the traditional STRIPS domain but we still are able to use $A^*$ search algorithm with admissible heuristics to find reasonable size optimal plans (15-40 actions) while maintaining consistencies with hundreds more actions of other plans previously found in short time (fraction of a second), there are lessons learned in making it possible. In this section, we list the most important ones and hope that they can be useful for both application developers and academic researchers in planning.

*Modeling is important:* We model the domain using the two-layer modeling language. Through discussion with our users and industrial partners, we feel that the machine-centric language used by the end-user involving modules, machine instances, inter-connections is easier for them to understand and the compiled-down representation is much easier for us to adopt STRIPS planning techniques. We also experienced that while modeling, knowing the search algorithm and the heuristics that we will use, careful modeling the actions, goals, and initial states can produce quite different results. We experienced a scenario in which adding two extra predicates representing subgoal completion achieved the speedup of nearly 10x for some product lines. While we solely encode the domain "physics" but not any search guidance, we want to emphasize that the same "physics" can be represented differently, even if limited to STRIPS, and finding the right mix with the search algorithm of choice can dramatically affect the planner's performance. As application developers, not having to work with fixed benchmark domain representation allows us to exploit another dimension in modelling to improve planner's performance.

*The most suitable planning algorithm depends on the application specifications:* We went through several implemetations of different planning algorithms before finally settled for the backward state-space planner. It is much faster than forward-planner, which is dominating the planning competition. Therefore, understanding your domain, the important constraints involved, your objective function, and how different planning algorithms work can help selecting the most suitable planning strategy.

*Having a fast and robust temporal reasoner is very important:* In our planner, even though the source code for the Simple Temporal Network (STN) totals less than 200 lines of code, it's very important in handling all temporal relations between actions and resource allocations in a single plan and between different plans. In a real-world application where there are various temporal constraints and factored in delays such as "communication delays", "setup-time", "coordination delay" and time synchronization problem between the planner and the other components in the overall control architecture, rapidly keeping the overall temporal consistency is one of the most important things to keep the planner running without interruption for a long period of time.

*Many uses of the planner:* Besides its main job of controlling different plants, the planner is also recently used extensively for system analysis purpose. Thus, the planner is tested against (i) different plant designs; (ii) plants with various broken machines for reliability analysis. Our customer just finished running an extensive test consists of 11760 different planner runs for variations of a single plant configuration. Recently the planner is also used to test the performance of the job-sequencer. We managed to finish a

job mix of 50000 jobs without any break, which is more intensive than the regular real-life plant operation. Besides those activities, there are also intentions to use the planner in the near future for other purposes such as sophisticated objective functions, to handle machine dynamics. Through all those experiences, we learned that when a planner is successfully used for one application, it's likely to be useful for many other related applications, some may seem to be very different with the original one.

*Exceptions:* Given that the planner interacts with other parts that are either higher or lower on the control hierarchy, exceptions can come at various forms. The user can reconfigure the machines, other processes may request resource allocation repeatedly (cyclic resource allocations), and the machines in the plant can break down in different ways. We believe that besides our domain, similar exceptions would occur in most applications where the planner interacts with physical world. While robust exception handling (replanning) is important, and in fact we plan to dedicate most of our near-future research on this topic, we found that there are much less research on this topic compared to other branches of domain-independent planning.

Above are some of our observations and lessons learned from working on developing and deploying our planner. It may be from one domain, and it may not endure the level of complexity of other well-known domain but we hope that it can help researchers to develop planning techniques that are more ready to be used in the real-world environment and are also useful for planning application developer to pick out the right approaches.

## Related Work

The dynamic programming rules used in building the temporal planning graph are similar to the the way planner TP4 (Haslum and Geffner, 2001) calculating its $h^2$ heuristic and TPG planning in building the temporal planning graph with some slight differences. While there are some surveyed work (Smith, Frank, and Jonsson, 2000) of integrating planning and scheduling (Ghallab and Laruelle, 1994; Muscettola, 1994; Beck and Fox, 1999), we are not aware of any similar work using scheduling constraints to improve planning heuristic.

There are several successful deployed planning system such as EUROPA/MAPGEN(Bresina, Jonsson, Morris, and Rajan, 2005), HSTS(Muscettola, 1994), or IxTeT(Ghallab and Laruelle, 1994). However all of them use domain-knowledge search guidances for their applications while our planner do not rely on domain knowledges.

Fromherz, Saraswat, and Bobrow (1999) discuss on-line constraint-based scheduling methods for controlling physical machines. They use precomputed plans and their formalization cannot model systems such as ours with possible cycles and an infinite number of potential plans.

## Future Work

There are several challenges with using and extending our planner. First, as our industrial partner aiming for more ambitious projects that involve faster, more complicated, and higher number of components, the planning problems that we need to solve become harder. At the same time, because the plants will have a higher productivity, the planner will have less time to find a plan. The second challenge is related to the highly-reconfigurable nature of our domain. One requirement for the planner is that it should work well even if some error occurs and some components of the plant go online/offline at any given time. Effective handling various types of exception is one of the most critical problem for us at the moment.

We plan to investigate the multi-objective optimization planning algorithm to support different quality criteria. Another direction is to investigate a different objective entirely: minimizing machines "wear and tear". Under this objective, one would like the different machines in the plant to be used the same amount over the long term. However, because machines are often cycled down when idle for a long period and cycling them up introduces wear, one would like recently-used machines to be selected again soon in the short term. Optimize this objective function alone, or in conjunction with our current time-based criteria pose great challenge.

## Acknowledgments

The members of our research group provided helpful comments and suggestions. Our industrial collaborators not only provided domain expertise but were invaluable in helping us to simplify and frame the application in a useful way.

## References

Beck, Chris, and Mark Fox. 1999. Scheduling alternative activities. In *Proc. of AAAI-99)*.

Bresina, John, Ari Jonsson, Paul Morris, and Kanna Rajan. 2005. Activity planning for the mars exploration rovers. In *Proc. of ICAPS05*, 40–49.

Dechter, Rina, Itay Meiri, and Judea Pearl. 1991. Temporal constraint networks. *Artificial Intelligence* 49:61–95.

Fromherz, Markus P.J., Vijay A. Saraswat, and Daniel G. Bobrow. 1999. Model-based computing: Developing flexible machine control software. *Artificial Intelligence* 114(1–2):157–202.

Ghallab, Malik, and Hervé Laruelle. 1994. Representation and control in IxTeT, a temporal planner. In *Proc. of AIPS-94*, 61–67.

Haslum, Patrik, and Héctor Geffner. 2001. Heuristic planning with time and resources. In *Proc. of ECP-01*.

Muscettola, Nicola. 1994. HSTS: Integrating planning and scheduling. In *Intelligent scheduling*, ed. Monte Zweben and Mark S. Fox, chap. 6, 169–212. Morgan Kaufmann.

Ruml, Wheeler, Minh Binh Do, and Markus Fromherz. 2005. On-line planning and scheduling for high-speed manufacturing. In *Proc. of ICAPS05*, 30–39.

Smith, David E., Jeremy Frank, and Ari K. Jonsson. 2000. Bridging the gap between planning and scheduling. *Knowledge Engineering Review* 15.

Smith, David E., and Daniel S. Weld. 1999. Temporal planning with mutual exclusion reasoning. In *Proc. of IJCAI-99*, 326–333.