Timeline-based Planning System for Manufacturing Applications

Minh Do and **Serdar Uckun**
Embedded Reasoning Area, Palo Alto Research Center.
Email: {minh.do,uckun}@parc.com

Abstract

In recent years, the Embedded Reasoning Area (ERA) has been developing a planning system targeting fast online planning problems in manufacturing. The planner, which is based on general-purpose AI planning techniques, has evolved through several iterations and successfully solved applications such as hyper-modular printers, modular packaging machines, material control for LCD manufacturing, and warehouse management. In this paper, we will describe the core techniques underlying the current version of the planner: a combination of timeline-based state representation and action-based planning algorithm. This combination is proven to be flexible and can quickly adapt to new applications and at the same time can scale to complex problems.

1 Introduction

Our research on model-based online planning starts with the Tightly Integrated Parallel Printer (TIPP) project [Ruml *et al.*, 2005; Do *et al.*, 2008; Ruml *et al.*, 2011] where we need to effectively control reconfigurable printing systems. After the success of this project, there have been efforts in adopting the software, in particular the planner to new applications. The first application was controlling modular packaging machine, which shown that the adaptation of the TIPP planner can effectively control (in simulation) a variety of high-speed infeed systems of food flow-wrapper machines. However, this is just the first step in generalizing it to solve a more general class of problems in manufacturing.

After the initial investigation in the packaging domain, we have been further extending our model-based planner so that it can easily be adapted to a wide variety of application domains. Recently, our planner has been used in several funded projects by the IHI Corporation in 2010 and 2011 for different applications: Material Control System (MCS) and Automated Warehouse. In this paper, we outline the architecture of the new planner and the application domains that it was tested on.

The rest of this paper is organized as follows: we start with the timeline-based online planning architecture in the next section. We then follow with two implemented planning algorithms (1) forward state-space; and (2) partial-order in Section 2.3 and Section 2.4. We outline the results of using our planner in the manufacturing applications outlined above and we finish the paper with some future work.

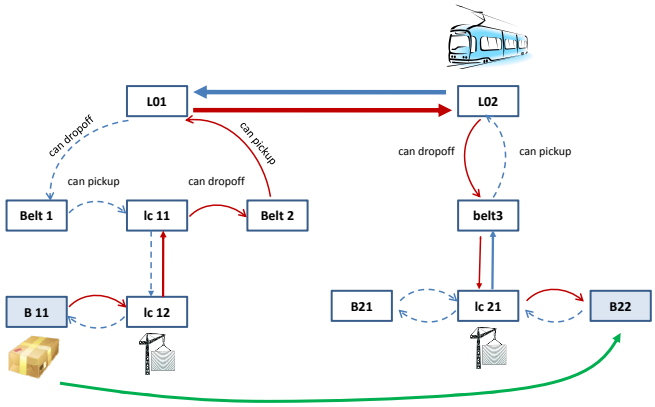


Figure 1: A logistics example

2 Overall Architecture

Planning is the problem of finding a (sequential or parallel) sequence of actions that when executed from a known initial state will achieve all pre-defined goals. Our group has been working on "fast continual on-line planning" problems where user's goals and system updates continuously arrive in concurrent with plan executions of previously found plans. In *fast* we mean the software generally needs to find a complete solution within a few seconds (sub-second in several cases).

Our current Plantrol planner uses a timeline-based planning approach that operates by continually maintaining the *timelines* that capture how different system state variables change their values over time. The planner builds and maintains consistent plans by adding *tokens* to the affected timelines; with each token represents a different operation/change affecting the state variable represented by that timeline. The overall framework allows selection among multiple planning algorithms, all share the same timeline-based state representation, for a given task. In turn, different planning algorithms can call different search algorithms and constraint solvers (e.g., temporal reasoning, uncertainty reasoning) to solve either planning or replanning tasks effectively.

To illustrate different concepts, we will first present a simple example that will be used throughout the paper:

Example: shown in Figure 1 is an example inspired by IHI's MCS application. In this example, a package located at location *B11* needs to be moved to *B22* using first the crane located at *LC12*, then the overhead vehicle (OHV) that is originally at *L02* and then lastly the second crane originally located at *LC21*. The arrows in solid red color show the path of the package. Note that there are a couple of actions belong to a final plan but are not included in this path such as moving the OHV from *L02* to *L01* and the second crane from *LC21* to

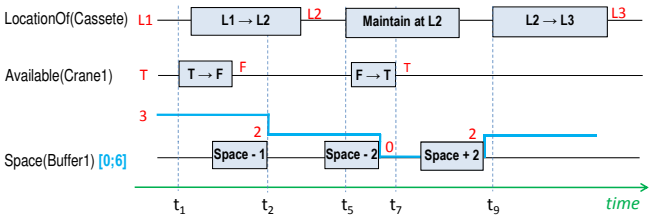


Figure 2: Timeline example

Belt3. They are represented by solid blue arrows. The remaining dotted blue arrows represent the other actions available but are not part of the final plan.

2.1 Timeline-based State Representation

The input to the deterministic online planner consists of:

1. a set of variables V with each $v \in V$ is associated with a given domain of values $D(v)$;
2. a set of actions A, each specified by its *(pre)condition* and *effect* lists. An action condition represents a constraint on the value of a given variable (e.g., $v = x$) and an action effect represents a change to the value of a given variable (e.g., $v \leftarrow y$).
3. a complete variable assignment for all $v \in V$ represents the fully observable initial state I.
4. a partial variable assignment G represents the desired goal condition.

An action a is applicable in state s if all of its conditions are satisfied by s and the resulting state from applying a in s reflects the changes caused by a's effects on s. The planner needs to find a consistent sequence of actions (a plan) P that can connect I to G.

For *online continual* planning scenarios, finding and executing plans and goal-arriving are interleaved. Given that the planner needs to continuously reason about those interleaving processes, we need to effectively maintain the status of different state variables as they change values over time. One good way to do so is through timelines and there are several application-oriented planners, including TIPP, that have used this approach at different levels [J. Frank, 2000; Fratini *et al.*, 2008].

Figure 2 shows an example of the timelines of several variables in our leading example: (1) a *multi-value* (discrete) variable $v_1 = LocationOf(Package)$ that represents the package location; (2) a *binary* variable $v_2 = Available(Crane1)$ represents whether or not *Crane1* is busy carrying some package; (3) a *continuous* variable $v_3 = Space(Buffer1)$ represents the available/empty space in *Buffer1*. While we currently only support three types of variables (which are most common) in our planner, theoretically any variable with a certain value domain can be included in the timeline set managed by the planner.

The timeline for a given variable v consists of a value $c_v \in D(v)$, which is the value of v at the current wall-clock time t_c and a set of *tokens* representing future events affecting the value of v. Those events represent pre-committed assignments of different equipments/resources/objects. Figure 2 shows one example where there are three tokens in the timeline for $v_1 = LocationOf(Package)$ representing the following events (in this order): (1) the value of v_1 changes from the current value $v_1 = L_1$ to a new location $v_1 = L_2$, (2) and $v_1 = L_2$ needs to be maintained for certain duration; then (3) it changes again from L_2 to L_3. Each token tk is represented by:

- Start and end time points $start(tk)$ and $end(tk)$.
- A start value v_s (or bounds on start value $[lb, ub]$ with $lb \leq ub$ for continuous variable).
- Start condition (e.g., $v = v_s$) specifies the condition that needs to be satisfied by the token. Right now, we support: $=, \neq, >, <, \geq, \leq, NONE$.
- Change operation $\langle operator, value \rangle$ (e.g., $v \leftarrow v + 5$ or $v \leftarrow x$) specifies how the variable value is changed within the token duration. Some change operators are: $\leftarrow$, +=, -=, $\times$=, /=, *CHANGE, USE, MAINTAIN*[1].

Given that tokens generally represent conditions and changes caused by actions, there can be temporal relations between tokens that are either: (1) conditions/effects of the same action a; (2) conditions/effects of actions that are related to each other. For example, before we *move* the package from $L1$ to $L2$ using $Crane1$, the crane needs to *pick_up* the package first. Thus, tokens caused by the *pick_up* action need to finish before the tokens added by the *move* action and thus there are temporal orderings between them.

Figure 3 shows an example of tokens on different timelines created by a given action instance. On the left side, we show the action representation in PTDL, a variation of PDDL [Fox and Long, 2003] – a standard planning modeling language, and the right side shows five tokens which would be added to different timelines if action $move$ is added to the plan. The same action starting time point t_s will be the starting time of four tokens and thus those four are constrained to start together.

For a given action a, we will use $T(a)$ to denote the set of tokens caused by a.The set of timelines for all variables is *consistent* if:

- *Value consistent:* Consecutive tokens on the same timeline should make up a consistent sequence of changes. Thus, the end value of a given token should *match* with the start value of the next token[2].
- *Temporal consistent:* All temporal constraints between tokens should not cause any temporal inconsistency. One example of temporal inconsistency is that two temporal orderings: $t_1 < t_2$ and $t_2 < t_1$ are both deductible from the temporal network.

A consistent timeline for v_g achieves a given goal $g = \langle v_g, x \rangle$ (i.e., $v_g = x$) at the end of the timeline for v_g if the end value of the last token matches with x. Alternatively, we say that it achieves g at some point in time if there exist a token T such that the end value of T matches x. For a given goal set G, if for all $g \in G$ the consistent timeline for v_g satisfies g then we say that the set TL of all timelines for all variables satisfy G or $TL \models G$.

2.2 Timeline-based Online Continual Planning

The previous section discusses how the world state is represented and maintained in continual planning by using a set of evolving timelines containing tokens representing actions'

[1]The variable value at the end of the token is calculated based on the start value and the change operation.

[2]In *matching*, we generally mean equal but for continuous variables that are represented by a $[lb, ub]$ interval, matching means that two intervals overlap.

```
(:action move
 :parameters (?v - vehicle ?l1 ?l2 - location)
 :duration (/ (distance ?l1 ?l2) (speed ?v))
 :condition
      ([start,end] (direct-connect ?l1 ?l2))
 :effect
      (over-all
           (change (location-of ?v) ?l1 ?l2)
           (use (path ?l1 ?l2)))
      ([start, start + 2] (change (space-free ?l1) F T))
      ([end - 2, end] (change (space-free ?l2) T F)))
```

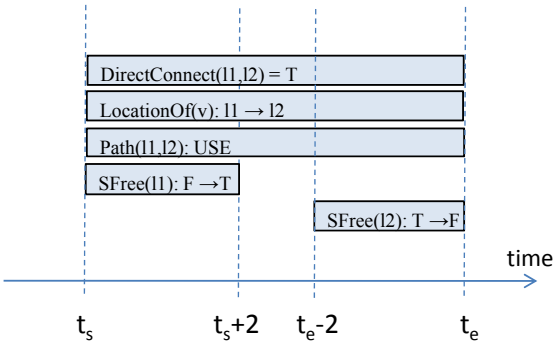


Figure 3: Action and its corresponding tokens

conditions and effects. In this section, we provide a high-level planning algorithm that operates on timelines and finds consistent plans.

Algorithm 1: Timeline-Based Planning Algorithm

input : A consistent timeline set TL, a goal set G
output: Plan P achieves G & an updated timeline set TL

1 Let: $P_0 \leftarrow \emptyset$, $TL_0 \leftarrow TL$, and $s_0 = \langle P_0, TL_0 \rangle$;
2 Initialize the state set: $SQ = \{s_0\}$;
3 **while** $SQ \neq \emptyset$ *and* **done** = false **do**
4 Pick the best state $s = \langle P_s, TL_s \rangle$ from SQ;
5 **if** TL_s *is consistent and* $TL_s \models G$ **then**
6 **done** = *true*
7 **else**
8 Generate zero or more revisions P' of P_s;
9 Generate timeline sets $TL' \leftarrow TL \bigcup T(P')$;
10 Add temporal constraints between temporally related tokens in TL';
11 Add $s' = \langle TL', P' \rangle$ to generated state set SQ;
12 Execute P_s;
13 Revise the master timeline set: $TL \leftarrow TL_s$;

Algorithm 1 shows at a high-level a planning algorithm operating on timelines. Some notations used in this algorithm, and all subsequent algorithms described in the next several sections are:

- For each time point tp (e.g., token's start/end time-point): $est(tp)$ and $lst(tp)$ represent the earliest and latest possible times that tp can happen.
- For an action set A: $T(A)$ is the set of tokens caused by all actions in A. Similarly, $T(P)$ is the set of tokens caused by all actions in the plan P.

The planner starts with a consistent timeline set TL representing all changes and constraints related to all state variables from the current wall-clock time. It needs to find a plan P such that (1) adding $T(P)$ to TL does not cause any inconsistency, (2) achieve all goals, and (3) executable (i.e., all tokens caused by this plan should be able to start after the wall-clock time at which the plan is found). The planner starts with an empty plan and keeps revising it until achieving these objectives (lines 8-11). The planner tries to find the *best* plan by maintaining a set of generated states (which is composed of a plan P and the timelines resulted from adding tokens caused by P to the original timelines) and at each step picks the best from the generated set to check for being a valid plan. When the best plan P is found, we execute P (line 12) and incorporate its effects in the continually maintained timelines (line 13).

This high-level algorithm obviously lacks many details such as: how to revise P_s (line 8)? what is the *best* plan? or what exactly is the representation of the plan during the planning process? On the other hand, it's general enough to capture both systematic and local-search style of planning, and for different planners that can handle different set of variables and constraints. In the next two sections, we describe two implemented algorithms based on this framework.

2.3 Forward State-Space Planner on Timeline

Forward state-space (FSS) planners move forward in time through fixed-time complete state. It starts with an empty plan and gradually add actions at some fixed wall-clock time to the end of the currently expanding partial plan until the final sequence of actions satisfies the goals. In short, a visited "planning state" s of a FSS planner consists of: (1) a time-stamp t_s of s; (2) a set of timelines in which all tokens (i) end after t_s and (ii) have fixed start and end times.

The algorithm starts searching from the current wall-clock time t_c but will execute the plan at the (expected) wall-clock time $t_e > t_c$ when the plan is found. To start the planning process, the planner "freezes" all tokens in all timelines and remove all tokens that end before t_e. This step simplifies the token and timeline representation and also reduces their sizes. The key details here are the successor generating functions to create subsequent search nodes:

- *Applicable*: for each action a, the FSS planner moves forward in time from the current state's time stamp t_s until it finds an earliest time $t_a \geq t_c$ that if a executes at t_a then all new tokens added will not cause any inconsistency. Any action t_a that we can find a *consistent* execution time t_a is added to our candidate set.
- *Apply*: the planner generates successors by creating tokens corresponding to action's conditions and effects and add them to the current timelines.
- *AdvanceTime*: this is a special action that helps move the state time-stamp t_s forward closer to the goal. When moving the time-stamp forward, it basically sets the newer lower-bound on the future action execution time and thus: (1) simplifying the timelines (remove all tokens finish before the new time-stamp); and (2) reducing the interactions between existing tokens and future actions.

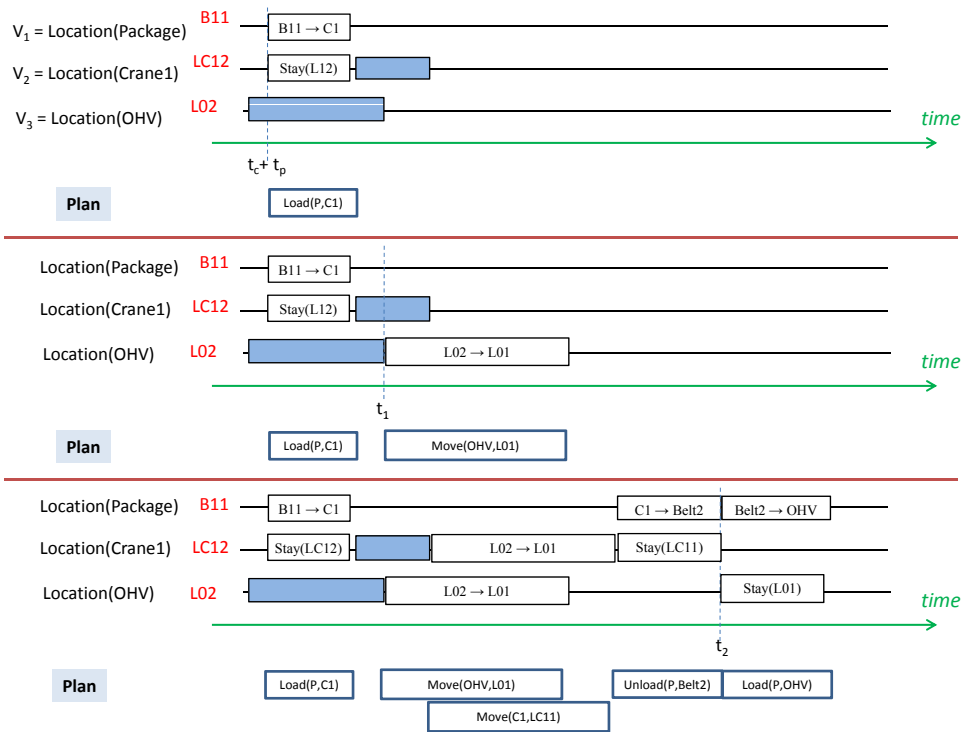


Figure 4: Example illustrating several steps of the FSS on timeline algorithm: adding actions in "forward" direction.

Given that the plan returned by the FSS algorithm has all actions and tokens tied to some fixed wall-clock times, depending on a particular search algorithm and heuristic setting, the FSS planning algorithm may not return the plan with all actions start at the earliest possible time according to their temporal relation. The "fixed-time" plan found by the FSS algorithm can easily be converted to flexible temporal plans using techniques described in [Do and Kambhampati, 2003].

Figure 4 shows several steps with the goal of having a package inside an OHV. In the timelines for the three variables mentioned above, tokens represented by solid rectangles are from previous planning episodes and thus tokens created by the current planning process should not overlap with them. We start by setting up the time stamp t_e, and the planner starts by adding an action of loading the package into *Crane 1* at t_e. This action addition creates two fixed-time tokens on the timelines for v_1 and v_2. We then apply the *AdvanceTime* action (several times) to reach t_1 and apply the second action to move the OHV to $L01$ (this adds one token to the timeline of v_3). After several steps of adding regular actions (e.g., $Move(Crane, LC11)$, $Unload(P, Belt2)$) and several *AdvanceTime* actions, we *load* the package into OHV. At this time, all timelines are consistent and achieving all goals so we terminate the planning process.

2.4 Partial-Order Planner (POP) on Timeline

The FSS algorithm described in the previous section finds plans by moving forward through a sequence of consistent timelines until a given timeline set satisfying all goals. On the other hand, the POP algorithm finds plans by starting with an inconsistent timeline set and systematically refines it until it becomes consistent. The planner searches backward from the goals. For that, it first creates special tokens representing the goals and the planner's objective is to create enough tokens through action addition so that those goal tokens are all eventually supported. Instead of finding *Applicable* actions as in the FSS algorithm, it finds *Relevant* actions, which can contribute new tokens that support some currently un-supported tokens. We have two-level branching: (1) over actions that are deemed *relevant*; and (2) over token ordering where the new tokens introduced by the newly added actions can be added in the respective timelines. Note that there is no fixed starting time for all actions and tokens but their start/end times are represented by floating time points.

Figure 5 shows several steps in the POP algorithm finding the plan with the same set of actions as the FSS algorithm shown in Figure 4. The planner starts by creating a special token $v_1 = In(OHV)$ at the end of the timeline for v_1. It then adds an action $Load(P, OHV)$ to the plan because that action can add a token to support $v_1 = In(OHV)$. Appropriate temporal orderings are also added between related time points (we show some of them in the figure). The algorithm keeps picking un-supported tokens and add actions to support them until the timelines are consistent and the final plan is found.

FSS vs. POP: Two algorithms have distinctive advantages. The *fixed-time* and the association of a time-stamp for each search state during the planning process lead to:

- Smaller state representation: (1) any token ends before the current state's time-stamp can be removed from consideration; (2) no order between different tokens need to be stored. They are implicitly implied by the fixed start/end time of all tokens.
- Lower branching factor: each applicable action generates exactly one successor.

Therefore, the FSS planner likely find some valid plan faster. On the other hand, the POP algorithm employs a more

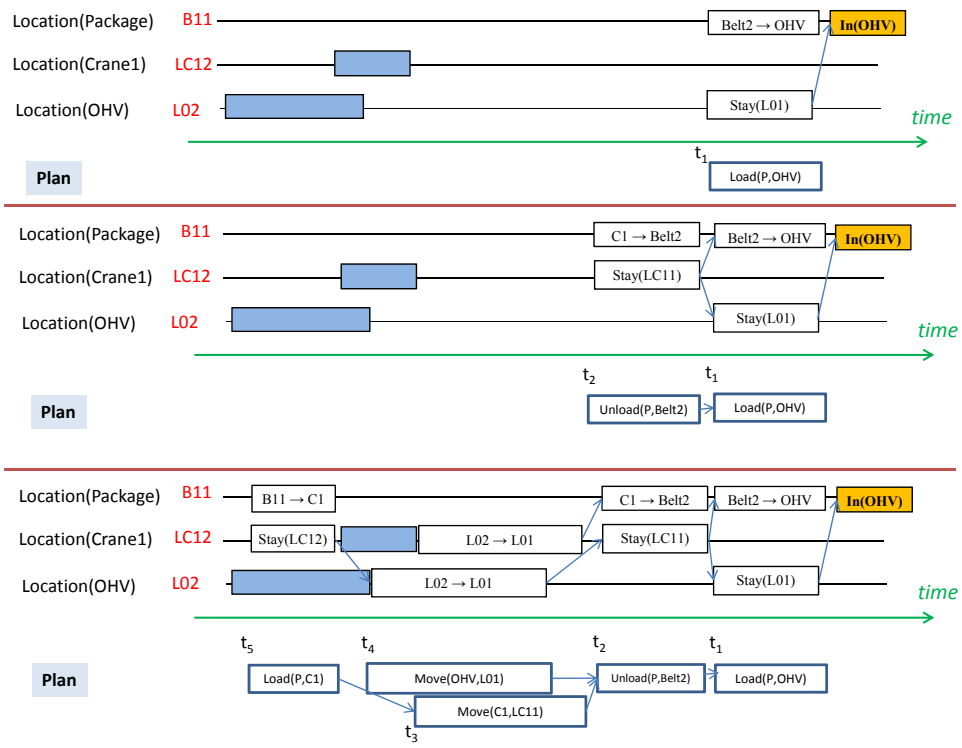


Figure 5: Example illustrating several steps of the POP on timeline algorithm: adding flexible actions in "backward" direction.

complete branching rule and thus does not rule out any valid solution. FSS planner, due to the fact that it doesn't consider all possible action starting times (by only moves the time-stamp forward to the next significant time point) may miss some solutions.

Depending on the actual application, one algorithm may be more appropriate: FSS is likely more suitable for applications where finding a plan quickly is of critical; and POP may be more appropriate where planning time is not critical but plan quality is more important.

2.5 Makespan-estimation Heuristic

Planner's performance, especially search-based, highly depends on the quality of the heuristic guiding its exploration of its solution space. In our targeted problems of Plantrol, we concentrate on finding plans that optimize for goal achievement time, which is highly related to *makespan* (i.e., plan execution time). Our heuristic is based on building the relaxed temporal planning graph (RTPG) and adjusting its estimation with the potential conflicts with tokens of the previous plan.

For each planing state s, the RTPG estimates the temporal distance between the current state in the search tree and the final state that the search algorithm tries to reach. In our FSS algorithm (Section 2.3), the heuristic estimates the distance between the current state and the goal state while in the POP algorithm (Section 2.4), the heuristic estimates the distance between the current state and the initial state. Given that the heuristic procedures for both types of planner are very similar, we will just discuss and give an example for the FSS planner.

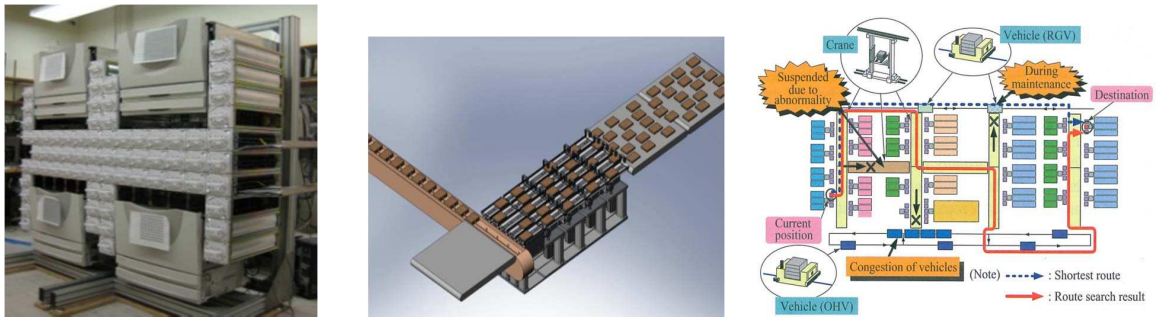
Given a timeline set TL representing a state during the planning process and the goal set G to be achieved, the algorithm will estimate the finishing time of a shortest plan that achieves G and built on top of TL (i.e. extends and includes all tokens in TL). The algorithm starts from the time-stamp $t = t_{TL}$ of TL and moves forward in a similar fashion to the FSS algorithm described in Section 2.3. However, instead of selecting which action to add next, we will optimistically apply all actions that have their conditions satisfied at t and ignore their conflicts. The neglect of conflicts between overlapping actions lead to the name *"relaxed" temporal planning graph*.

When actions with their conditions satisfied at time t are added, we add the tokens caused by their effects (refer to Section 2.1) to the collective pool of tokens that can lead to new values. At any given moment, we maintain the set D of (optimistically) achievable values, starting with the current values at t_{TL} in all timelines. After activating all actions having all of their conditions satisfied at time t and add tokens representing their effects into TL, we move forward (increase t) to the earliest end time t_e of any token in TL and add the new value achieved by all tokens ending in t_e to D. We repeat the process until either: (1) D contains all values of G; (2) there is no additional token in S to advance to its end time (and thus there is no new value to add to D).

3 Applications

Our online temporal planner has been successfully tested on several applications.

Tightly Integrated Parallel Printer (TIPP): The TIPP reconfigurable printer design allows building custom reconfigurable printer configurations from shared components. This project requires a software controller that can work with any design and it starts our work in the integrated planning and control framework. Our first planner built for this application uses the timeline representation for shared resources and combines it with the non-timeline state representation for logical variables. The planner works very well and can control two physical prototypes and hundreds of conceptual designs with



Parallel Printer **Modular Packaging** **Material Control System**

Figure 6: Example instances of the applications addressed by PARC's planner

the productivity of up to 220 page-per-minute (which requires planning/solving time to be less than 0.27 seconds). We have published extensively for this application [Ruml *et al.*, 2005; Do *et al.*, 2008; Ruml *et al.*, 2011].

Packaging: After the conclusion of the TIPP project, we investigated the application of our model-based planning+control technology to controlling an automated infeeder for a packaging line of food and consumer packaged goods. In this system, products arrive continuously at high-speed from the end of the production line and need to be arranged into a specific configuration for downstream primary and secondary packaging machines. In collaboration with a domain expert from the packaging industry, we developed an innovative design for a reconfigurable parallel infeed system using a matrix of interchangeable smart belts. We also adapted our online model-based Plantrol planner to this domain. Our planner can control various configurations of the new infeed system through simulation both in nominal planning and when runtime failures occur. We are also building a physical prototype to validate the new design and our software framework. More details are described in [Do *et al.*, 2011a].

Material Control System: Recently, in early 2010, we have successfully applied our planning framework to another application: planning for the Material Control System (MCS) of Liquid Crystal Display (LCD) manufacturing plant in a joint project between the Embedded Reasoning Area at PARC and the Products Development Center at the IHI Corporation. The model-based planner created at PARC was able to successfully solve a diverse set of test scenarios provided by IHI, including those that were deemed very difficult by the IHI experts. The short project time (2 months) proved that model-based planning is a flexible framework that can adapt quickly to novel applications. This the the first project where the the full timeline based representation, as described in this paper, was used for the planner. More details on the domain and the adaptation effort are described in [Do *et al.*, 2011b].

Automated Warehouse: Earlier this year, we collaborated with IHI again on another project on Automated Warehouse control[3]. The adaptation of our planner was able to successfully control a very large (few thousand objects) warehouse system with complex constraints. It consistently found plans up to hundreds of actions in less than one second. We hope to be able to describe the details in the future publication.

[3]Due to the proprietary IHI's warehouse design, we are not able to reveal the details or show any configuration example.

4 Conclusion & Future Work

In this paper, we introduce an automated planning framework for fast online continuous planning applications. The planner combines timeline-based state representation and action-based planning algorithms. The result planner has been used successfully in several manufacturing applications. We are currently working on extending both the expressiveness of our modeling language, adding supports for handling constraints such as uncertainties, and looking to apply our framework for even more applications.

References

[Do and Kambhampati, 2003] Minh Do and Subbarao Kambhampati. Improving the temporal flexibility of position constrained metric temporal plans. In *Proc. of ICAPS-03*, 2003.

[Do *et al.*, 2008] Minh Do, Wheeler Ruml, and Rong Zhou. On-line planning and scheduling: An application to controlling modular printers. In *Proc. of AAAI08*, 2008.

[Do *et al.*, 2011a] Minh Do, Lawrence Lee, Rong Zhou, and Lara Crawford. Online planning to control a packaging infeed system. In *Proc. of the Twenty-Third Annual Conference on Innovative Applications of Artificial Intelligence (IAAI-11)*, 2011.

[Do *et al.*, 2011b] Minh Do, Kazumichi Okajima, Serdar Uckun, Fumio Hasegawa, Yukihiro Kawano, Koji Tanaka, Lara Crawford, Ying Zhang, and Aki Ohashi. Online planning for a material control system for liquid crystal display manufacturing. In *Proc. of the 21st International Conference on Automated Planning and Scheduling (ICAPS-11)*, 2011.

[Fox and Long, 2003] Maria Fox and Derek Long. PDDL2.1: An extension to PDDL for expressing temporal planning domains. *Journal of Artificial Intelligence Research*, 20:61–124, 2003.

[Fratini *et al.*, 2008] S. Fratini, F. Pecora, and A Cesta. Unifying planning and scheduling as timelines in a component-based perspective. *Archives of Control Sciences*, 18(2):231–271, 2008.

[J. Frank, 2000] P. Morris J. Frank, A Jonsson. On reformulating planning as dynamic constraint satisfaction. In *Symposium on Abstraction, Reformulation and Approximation*, 2000.

[Ruml *et al.*, 2005] Wheeler Ruml, Minh B. Do, and Markus Fromherz. On-line planning and scheduling for high-speed manufacturing. In *Proc. of ICAPS-05*, pages 30–39, 2005.

[Ruml *et al.*, 2011] Wheeler Ruml, Minh Do, Rong Zhou, and Markus Fromherz. On-line planning and scheduling: An application to controlling modular printers. *Journal of Artificial Intelligence Research*, 40:415–468, 2011.