Resource-based Planning with Timelines

Debdeep Banerjee[1,2] and **Jason Jingshi Li**[3]
Quintiq[1], The Australian National University[2], EPFL[3]
debdeep.banerjee@quintiq.com, jason.li@epfl.ch

Abstract

Real world planning applications typically involve making decisions that consumes limited resources, which requires both planning and scheduling. In this paper we propose a new approach that bridges the gap between planning and scheduling by explicitly modeling the problem in terms of resources, state variables and actions. We show that it is an intuitive way to formulate real world problems with complex constraints, and that solutions can be found by compiling the problem into a constraint satisfaction problem.

Introduction

Real world planning problems typically involve actions with complex temporal constraint, where different consequences of the action come to effect at different phases of the same actions. Consider the example of the turning of a spacecraft in order to point at a target as described in (Smith, 2003). The reaction control system (RCS), must fire the thrusters to provide angular velocity, then the spacecraft coasts until it points to the destination target, then the RCS thrusters are fired again to stop the angular motion of the spacecraft. It means the firing of the thrusters happens in the beginning and the end, and is controlled by the controller. Each time the thrusters are fired, propellants are consumed and it creates vibrations which may prevent some other operation on the spacecraft. It is a complex action that has influences on various domain objects at different times. A challenge for the automated planning research is to create formalisms that efficiently model and solve problems involving such actions.

The main representation language for the planning community are PDDL and its variants (AIPS-98 Planning Competition Committee, 1998; Fox and Long, 2003). In general, the planning models are based on a description of the world in terms of propositional and numeric variables, a set of functions that defined over them, and a set of actions that changes the state of the world. Although PDDL is widely used in the planning research community and is Turing-Complete, it is difficult to use it to model many practical problems due to its lack of support for modeling different kind of resources and temporal constraints that occur in many real world settings. In particular, as argued in (Smith, 2003), complex actions that have intermediate effects are particularly difficult.

Alternatively, one can represent the problem as a scheduling problem in terms of the available resources and the durative activities that have different requirements over these resources. Alternative options to achieve the goals are represented by different modes of action execution. However, the scheduling approach lacks a high-level representation language, and in-depth domain knowledge is needed to model the problems.

In this paper, we aim to bridge the gap between planning and scheduling. Our approach is to frame a planning-scheduling problem in terms of resources, state-variables and actions over a timeline. The actions describe how the resources and state variables evolve over time. Throughout the paper we use a manufacturing setting to illustrate our approach. We show that our approach is an easier way of modeling a planning problems with complex constraints that appear in typical industry-related problems; provides a simple and intuitive semantics for actions and state transitions; and one can encode the problem as a constraint satisfaction problem (CSP) in a straight forward manner.

The work in this paper can be considered as extension of of (Banerjee, 2009), where the former work describes modeling temporal planning problems using actions and transitions. The approach in this paper explicitly models resources, and creates an action representation to allow for delayed effects. The constraint model for resource transition in our paper is related to the support-link scheduling described in (Banerjee and Haslum, 2011). The main contribution of this paper is to show that everything can fit together under a unifying framework that allow us to model resource-based planning problems, and we empirically evaluate our approach with a new solver for such problems.

Our approach is related to the ANML language, where both approaches describe a planning problem in terms of actions and multi-valued state variables. The key difference being that ANML provides temporal qualifiers to represent expressive actions, whereas we represent actions by a set of transitions. Although this restricts our representation from describing more expressive actions effects in ANML, it helps us to develop an efficient and straight forward way to encode and solve real world, resource-based planning problems as constraint satisfaction problems.

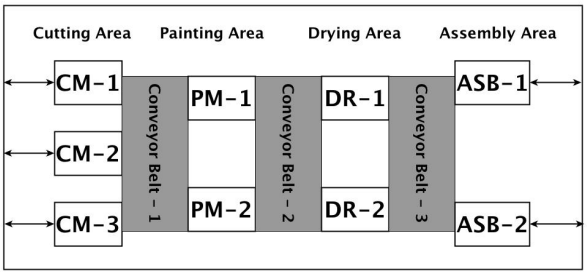

Figure 1: Illustration of a factory with 3 cutting machines (CM), 2 painting machines (PM), 2 dryers (DR) and 2 assembly areas(ABS)

The Setting

We consider the class of planning problem that require the manipulation of some scarce resource. A problem is a tuple $< R, S, A, H, I, G >$, where R denotes the resources involved in the problem, S the set of state variables, A the set of actions, H the planning horizon, I the initial state, and G the set of goal states. We will illustrate our modeling approach through the following example in a practical setting.

Example

Consider a simple manufacturing plant with 4 areas: cutting, painting, drying and assembly. In the cutting area there are a number of cutting machines that are used to produce a fixed type of parts from raw materials; the painting area has some painting machines to paint the parts with a specific color; the drying area has some drying units to dry the recently painted parts; and finally the assembly area where the parts are being assembled together on a number of assembly desks. Cutting area, painting area and assembly areas of factory floors are connected via conveyor belts, each part after being cut in the cutting machine, travels via these conveyor belts from one location to other location. Each order (o) consists of a number of different order-parts (op), each of which that needs to be cut, painted, dried and assembled with other parts. Automated conveyor-belts move the order parts from one area to another. An order is completed if all its parts are assembled.

The cutting machines are used to cut the raw material into parts. The machine has its own configuration time and offload time, where it needs a worker to configure it before it can cut a part, and a worker to offload the part and send it to the painting machines once it is done. The cutting operation also produces some waste byproducts, which must be cleaned after some iterations.

Each painting machine in the painting area is capable of painting a part with a color. If a machine has to change the color of its paint, it would require some set up time to wash the nozzles depending on what the next color would be. Setup times for changing a color from other color is given.

After the part is painted, it must be sent to the drying area to be dried. A drying unit, while it is running, can dry an unlimited number of parts. However, it may only be in continuous operation for a certain length of time, after which they would need to be switched off to be cooled for a fixed period before it can be switched back on again.

After all the parts of an order are cut, painted and dried, they are sent to an assembly desk where they are put together by a number of workers. Once it is done, the order is completed. Workers are needed in cutting area to configure machines and offloading parts when they are processed. They are also needed in the assembly areas to complete orders. For a worker going from one location to other takes time.

Resource

In our approach, at the center of the problem representation are resources. In essence, resources are domain objects in the planning world that has a finite capacity, and that they are required in order for an action to be executed. The availability of a resource is also reflected on the timeline, where at any instant in time we have both the amount of the resource available and its maximum capacity.

We divide resources in two broad categories: *Reservoir Resources (RSR)*, which are either consumed or produced by an action; and *Reusable Resources (RUR)*, which are borrowed by an action at the beginning of its execution, and returned when the action is completed. So in our example, RSRs are raw materials and waste by products in the plant, and RURs are factory machines and worker pool.

A given resource r has the following attributes:

- $capacity(r)$: an integer that denotes the maximum amount of resource units;
- $type(r)$: the type of the resource, being either Reservoir or Reusable;
- $level(r, t)$: the amount of resource used at time t, with the constraint $0 \leq level(r, t) \leq capacity(r)$;
- $freeSpace(r, t)$: the amount of resource remain available at time t.

In our example, we have the following resources with their associated capacities:

- Reservoir Resources:
  - CuttingMachinWaste: $capacity = X$ (cleaning interval)
- Reusable Resources:
  - CuttingMachine: $capacity = 1$
  - WorkersPool $capacity = Y$ (number of the workers)
  - PaintMachine $capacity = 1$

State Variables

State variables are the domain objects in the planning world that can be in one of many finitely possible states at any given point in time. In contrast to resources, they do not have a capacity. However, they may still be conditions for which actions may be executed.

A state variable sv has the following attributes:

- $dom(sv)$: the set of possible domain values of sv;
- $state(sv, t)$: the domain value of sv that holds at time t.

In our example, there are a number of orders to complete, each of which consists of multiple order parts. For each order (o), order part (op), drying unit (du), we have the following state variables with the associated domain values to denote their status:

- OrderStatus(o): {*incomplete, completed*}
- OrderPart(op): {*uncut, cut,painted,dried,assembled*}
- DryingUnit(du): {*on, off*}

We view state variables and resources as *timelines* as described in control-based modeling of planning and scheduling problems. That means by timelines of state variable and resource we will mean their evolution over time in terms of states and resource availability.

Actions and Transitions

Actions are the components that manipulate both resources and state variables. In our approach, we break down an action into the individual effects of an action on either a resource or a state variable, each with its start time and duration on the timeline. We call such individual effect a *Transition*, which may be interpreted as a temporal constraint on a specific domain object. Hence, we represent an action as a set of transitions, which is a set of synchronized durative effects with different durations. This is in contrast to the PDDL-based representation where each action has its durations and all effects takes place either at the beginning or the end of the action. Our approach allows s to intuitively model actions with multiple delayed effect, which is ubiquitous in real world applications.

A transition T has the following attributes:

- $act(T)$: the action that T is a part of;
- $req(T)$: the requirement for T to commence execution;
- $dur(T)$: the duration of T;
- $start(T)$: the start time of T;
- $end(T)$: the end time of T;
- $offset(T)$: the time delay between the start of action and the start of transition.

A transition involves only a single domain object, being either a state variable or a resource. It is typed according to its effect on the domain object. If the transition involves a state variable, it can be of either an *EFFECT* transition, one that changes the assignment of the variable from one value to another; or a *PREVAIL* transition, one that preserves the assigned value of a state variable for its duration. If the transition involves a resource, then it can either *BORROW* a certain amount of resource at the start and return it at the end; *CONSUME* the resource or *PRODUCE* the resource. In the following section we will describe in detail each type of transitions.

EFFECT Transitions For a given EFFECT transition on state variable sv, written as T^E_{sv}, $req(T^E_{sv})$ is a tuple $< s_{from}, s_{to} >, s_{from}, s_{to} \in dom(sv)$. The requirement denotes the value of sv before and after the transition. More specifically, for the transition T^E_{sv} be valid, the following constraints must be satisfied:

1. sv must be assigned to s_{start} at the start of the transition;
$$state(sv, start(T^E_{sv})) = s_{start} \quad (1)$$
2. sv must be assigned to s_{end} at the end of the transition;
$$state(sv, end(T^E_{sv})) = s_{end} \quad (2)$$
3. sv must be undefined between the start and the end of the transition.
$$\forall t \mid start(T^E_{sv}) < t < end(T^E_{sv}) : state(sv, t) = \emptyset \quad (3)$$

Given an EFFECT transition T^E_{sv} on a state variable, $\mathrm{pre}(T^E_{sv})$ denotes the pre-condition,i.e $\mathrm{pre}(T^E_{sv}) = s_{from}$ and $\mathrm{post}(T^E_{sv})$ denotes the post-condition of T^E_{sv}, i.e $\mathrm{post}(T^E_{sv}) = s_{to}$. We say the T^E_{sv} achieves the state $\mathrm{post}(T^E_{sv})$ from the state $\mathrm{pre}(T^E_{sv})$.

PREVAIL Transitions For a given PREVAIL transition on state variable sv, written as T^P_{sv}, $req(T^P_{sv})$ is a tuple $< s_p >, s_p \in dom(sv)$. The requirement denotes that sv must remain s_p for the entire duration of the transition. More specifically, for T^P_{sv} be valid, the following must constraints be satisfied:

$$\forall t \mid start(T^P_{sv}) \leq t \leq end(T^P_{sv}) : state(sv, t) = s_p \quad (4)$$

Note that for a PREVAIL transition pre- and post-conditions are the same, i.e. $\mathrm{pre}(T^E_{sv}) = \mathrm{post}(T^P_{sv}) = s_p$.

PRODUCE Transition A PRODUCE transition on resource r, written as T^R_r, reserves the amount of free-space as described by $req(T^R_r)$ at the beginning of its execution $start(T^R_r)$ for its entire duration, and produces $req(T^R_r)$ amount of resource when it is completed at time point $end(T^R_r)$. The free-space is consumed at the end of the transition.

CONSUME Transition A CONSUME transition on resource r, written as T^C_r, is the complement of a PRODUCE transition. It consumes $req(T^C_r)$ amount of resource levels at at the beginning of its execution $start(T^C_r)$, while reserves the same amount of free-space. It releases the free-space at the end of its execution.

BORROW Transition A BORROW transition T^B_r on a reusable resource r uses the resource for its duration, and at the end gives back the resource. It may be interpreted as a CONSUME transition, but returns the resource at the end of its execution.

Actions in the Example Our manufacturing example contain many actions. They include cutting, painting and drying the order parts, assembling the parts together for an order, clean a cutting machine, and switch on and off a dryer. Here we will describe the specific actions to illustrate our approach.

- CutOrderPart(order, part, cutting_machine): Cutting a part for an order on the cutting machine CM. It has 5 main transitions (Fig. 2)
  - Transition-1: A BORROW transition on the resource "Worker" for the time it takes to configure part on the cutting machine

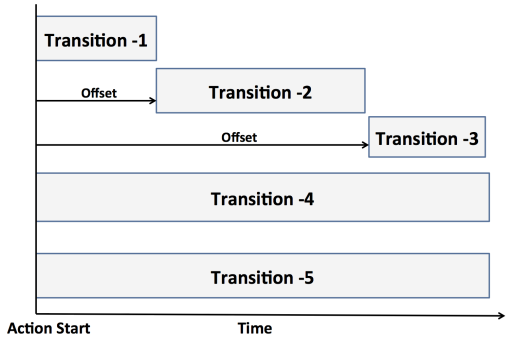

Figure 2: Action CutOrderPart(o, op, cm) and its transitions.

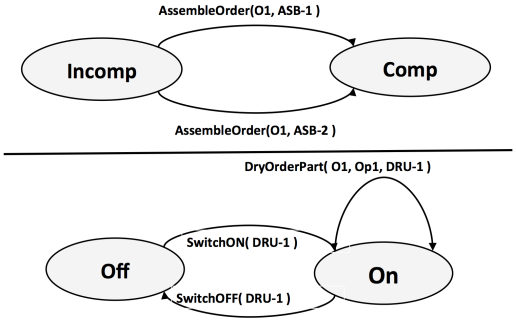

Figure 3: Transitions on order and dryer

  - Transition-2: After configuration time, it has an EFFECT transition on the state variable "Order-Part" where it changes its state from *uncut* to *cut* for the duration when cutting machine cut the part.
  - Transition-3: After cutting machine finishes the processing, it has a BORROW transition on the resource "Worker" for them it takes to offload the part from the machine.
  - Transition-4: During the whole time, it has a BORROW transition on the resource cutting_machine
  - Transition-5: During the time of processing it has a PRODUCE transition on the resource CuttingMachineWaste, where it produced 1 unit of waste.
- ColorOrderPart(order, part): The color part action 2 transitions: one BORROW transition on the resource painting_machine, and an EFFECT transition on the state variable "Order-Part" where it changes state the state *cut* to *painted*.
- DryOrderPart(order, part): This action has 2 state variable transitions: one PREVAIL transition on the state variable "DryingUnit", that needs the state *on*, and an EFFECT transition on the state variable "Order-Part" where it changes the state *painted* to *dried*.

Other actions for which we would not elaborate here include: AssembleOrderPart(order, asb_desk), CleanCuttingMachine(machine), SwitchOnDryer(dryer) and SwitchOffDryer(dryer). Figure. 3 describes how actions changes the assigned values of state variables.

Modeling Setup Constraints

Setup constraints that defines how much time must elapsed between two consecutive tasks. For a transition T, $\mathrm{Setup}(T)$ denotes the setup state of the transition. In our example, there are three scenarios where setup constraints apply; first between any two painting task of different color in a painting machine, second for each worker moving from one location to other, and thirdly between cutting and painting, painting and drying, and drying to assembly tasks of a part as it has to travel via conveyer belts from one area to other. To model setup constraints, we add a setup matrix to each state variable and resource, and assign a setup state to each transition. Setup matrices describes the time delay between pair of setup states. For example, we add a color setup matrix to each painting machine resource. If there are 3 different colors, C1 to C3, then the color setup matrix defines the time needed to change color from a pair of colors. Each resource transition on the painting machine must have a setup state among C1 to C3. For example, each ColorOrderPart(order, part, PM) action has a transition on the resource PM. If the color needed for the part is C1, then the setup state of the transition must be C1.In addition to the color matrix added to the painting machines, in the running example we add a distance location matrix to the resource "WorkerPool" that defines the time needed for each worker to travel from one location to other. All transitions on the resource "WorkerPool" must have one of the locations (locations of individual machines) as the setup state. A distance matrix is added to the state variable "OrderPart" that defines the time to travel from one area to other area via conveyor belt for a part, all transitions on the state variable must one of the areas $:cutting, painting, drying, assembly$ as setup state.

Initial State, Goal State and Planning Horizon

Similar to PDDL, a planning problem contain the description the initial state, which denote the state of the world we start in; and the goal state, the things that we want to be true. It may also contain the description of a planning horizon, which is the maximum time we allow the plan to reach the goal state. Abusing the notation, we would also say that the value a particular domain object (state variable or resource) is in the initial/goal state, if the value assignment is part of the description of the initial/goal state.

The Solution to a Planning Problem

In our setting, the solution to a planning problem is a *valid flexible plan*, which denotes the set of actions to be executed from an initial state to reach a goal state, and maintaining that no constraint is violated at any time during its execution.

Flexible Plan A flexible plan is simply a set of tuples $(a, [X, Y])$, where a is an action, and $[X, Y]$ is a time interval where X and Y are specific time points, which that denotes the range of the possible starting time of action a.

In executing a flexible plan, the agent chooses a starting time from the specified time interval of each action. When

all the starting time of the actions are specified, the resulting plan is a realization of the flexible plan.

Schedule A realization of a flexible plan creates a schedule for every state variable and resource, where a **schedule** is a sequence of transitions with fixed start time. A schedule of a state variable is 'em valid, if and only if the preconditions of the first EFFECT transition is in the initial state, the postconditions of the last EFFECT transition is in the goal state, and the following holds at any given time:

- If an EFFECT transition is in execution, then no other transitions are in execution on this state variable;
- If a PREVAIL transition is in execution, then other PREVAIL transitions in execution on this variable must also require the same state.

A schedule of a resource is valid if and only if the following holds:

- At any time, there exists no set of transitions in execution such that the total resource requirement of the set is greater than the capacity of the resource;
- Immediately after the execution of all the transitions that starts at the initial time point, the level of resource must be less than or equal to the initial level of the resource;
- at the end of the planning horizon the amount of resource in r is within the range defined in the goal state.

Hence, we say that a realization is *valid*, if and only if it creates a *valid schedule* for every state variable and resource, and a flexible plan is *valid* if and only if every possible realization of the plan is a valid realization.

Encoding the Problem as a CSP

In our approach, the search for the solution of a planning problem is to find a *valid flexible plan*, which represents a set of valid schedules for every state variable and resource. This corresponds to a set of temporal constraints on every domain object. Therefore, it is natural to model the planning problem as a Constraint Satisfaction Problem (CSP) on the domain objects.

The constraint model for each state variable can be thought of as the constraints for **causal-links** between pairs of state variable transitions. First introduced in Partial-Order-Planning (McAllester and Rosenblitt, 1991), a *causal-link* $a[p]a'$, represents the fact that action a achieves the pre-condition p for action a'. In our approach, a causal link on a state variable sv, written as $T_{sv}[s]T'_{sv}$, denotes the precedence relation between two transitions. T_{sv} is an EFFECT transition that make the precondition of the latter transition T'_{sv} true. Solving the constraint problem on every state variable is analogous to deciding which causal links hold in the final plan, where all the precedence constraints between those pair of transitions are satisfied.

The constraint model for each resource is based on deciding the **support-links** between pairs of transitions on the resource. On a resource r, a *support-link*, $T_r[\delta]T'_r$, denotes that transition T_r provides δ amount of resource towards the requirement of T'_r. If $\delta = 0$, it means T_r does not provide any support to T'_r. If $\delta > 0$, then the support link implies a precedence relation between T_r and T'_r. By deciding how transitions provide support to other transitions, i.e. creating support links, we build a schedule on each resource.

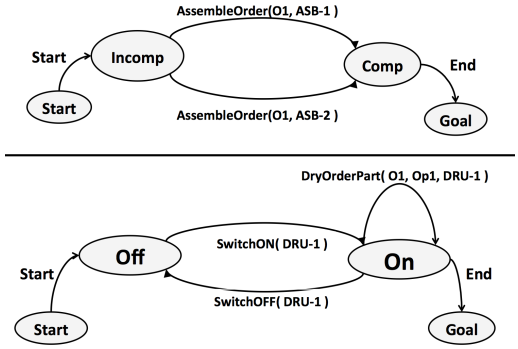

Figure 4: Additional states and actions

Each *causal* and *support link* implies a precedence or ordering relation between a pair of transitions. A precedence relation between two transitions $T \rightarrow T'$ means that T' starts after T finishes its execution. Since each transition is non-preemptive, and the start times of transitions and their corresponding actions are synchronized, each precedence constraint implies a precedence relation between the actions of the transitions. The constraint model for actions maintains the transitive closure of these precedence relations.

Preprocessing

We first introduce a preprocessing step before encoding the problem as a CSP. It introduces new states, actions and transitions such that it allows the resulting CSP to be solved in a more efficient manner.

Additional States for State Variables For each state variable $sv \in SV$ we add two additional states to its domain of possible states: $start_{sv}$ and end_{sv}.

Dummy Start **and** End **actions** We add two dummy actions Start and End into the set of actions A, where Start and End mark the achievement of the initial state and goal respectively. The Start action is constrained to appear at the beginning of the plan, before any other action in the plan, and the End action is constrained to appear at the end of the plan, after every other action. Introducing these dummy Start and End actions is a standard practice in modeling partial order causal link (POCL) planning (McAllester and Rosenblitt, 1991). Note that all transitions of all dummy actions, Start, End and other dummy actions that we introduce below, have duration 0.

On each state variable $sv \in SV$, the Start action has an EFFECT transition T_{sv}^{start} that changes the state of sv from the dummy $start_{sv}$ state to the initial state $\mathrm{init}(sv)$, representing the achievement of the initial state of sv. The End action has an EFFECT transition T_{sv}^{end} on each state variable sv that changes its state form either the goal state to the end_{sv} state (Fig. 4).

Similar to state variables, on each resource $r \in R_{reserve} \cup R_{reuse}$, the Start action has a resource transition T_r^{start}, and

the End action has a resource transition T_r^{end}.

The DUMMY start action's transitions will be referred as initial transitions and the DUMMY end actions' transitions will be referred as goal transitions in the following sections.

CSP Variables and Domains

To formulate the problem as a CSP we create the following CSP variables:

- **next[T]**: For each state variable transition T, except for the goal transitions, the CSP variable next[T] represents which EFFECT transition immediately follows T. Domain of next[T] contains all EFFECT transitions that can immediately *follow* T. That is, domain of next[T] contains all state variable transitions T′ such that $\mathrm{post}(T) = \mathrm{pre}(T')$. Note that for a transition T, all transition in the next[T] domain are the transitions on the same state variable as T.
- **previous[T]**: For each transition T, except for the initial transitions, previous[T] represents which EFFECT transition is immediately before T. Domain of previous[T] is the set of EFFECT transitions that can appear immediately *before* T. That is, domain of previous[T] contains all T′ such that $\mathrm{pre}(T) = \mathrm{post}(T')$ where both T and T' are on the same state variable.
- **inplan[A]**: For each action A, inplan[A] represents if the action A is in the plan or not. There are two possible values for inplan[A], *true* or *false*.
- **support[T_r,T'_r]**: For each pair of resource transitions $< T_r, T'_r >$, where T_r and T'_r are on the same resource and T_r is not a goal transition and T'_r is not an initial transition, the variable $\mathrm{support}(T_r, T'_r)$ represents the amount of resource (a non-negative integer) T_r provides to T'_r. Note, on a reservoir resource, there can be two types of transitions: PRODUCE transitions and CONSUME transitions. A PRODUCE transition T_p produces $\mathrm{req}(T_p)$ amount of resource at the end, and a CONSUME transition T_c consumes $\mathrm{req}(T_c)$ amount of resource at the start. For this reason we say that a PRODUCE transition can only provide support to a CONSUME transition and vice versa. Note that both PRODUCE and CONSUME transition can provide support to goal transitions on resources. So what all this means is the for each $\mathrm{support}(T_r, T'_r)$ variable we define, if T_r is a PRODUCE transition, then T'_r must either a CONSUME transition or goal transition, and if T_r is a CONSUME transition then T'_r must be a PRODUCE transition or goal transition. If δ_{T_r,T'_r} denotes the maximum amount of resource that T_r can provide to T'_r, then

$$\delta_{T_r,T'_r} = \min\left(\mathrm{req}(T_r), \mathrm{req}(T'_r)\right)$$

  The domain of each $support(T_r, T'_r)$ is the interval $[0, \delta_{T_r,T'_r}]$, where 0 indicates that T_r does not support T'_r.

Note, that although either the next or the previous variables alone are sufficient for the encoding, using both provides an opportunity for better propagation. Each assignment of the *next* and *previous* variable creates a *causal-link*, and assignment of *support* variables represents a *support-link*.

There are two additional variables for each transition T: **start[T]**, which represents the start time of T, and **end[T]** representing the end time of T. Similarly, for each action A, a variable **start[A]** represents the start time of A and **end[A]** represents the end time of the action.

We maintain two sets for each transition T, **before(T)** and **after(T)** to represent precedence relations between transitions on the same domain object, where before(T) contains all the transitions T' where $T' \rightarrow T$ holds, and similarly, after(T) contains all the transitions T'' where $T \rightarrow T''$ holds.

For each pair of actions A and B, we maintain a variable dist(A,B) that represents the distance in time from start of A to start of B, i.e. dist(A,B) = Start(B) - Start(A).

The next[T] and previous[T] variables can be assigned to a **not-in-plan** value $\perp$, which will denote that the transition T will not be part of the final plan.

Constraints

1. **EFFECT Position Constraints**: If an EFFECT transition T appears before another EFFECT transition T', then T' must appear after T and vice versa, i.e. $\forall T', T' \in$ EFFECT

$$previous[T'] = T \Leftrightarrow next[T] = T'$$

2. **PREVAIL Position Constraints**: The following constraints holds for all PREVAIL transition T_p that can appear next to T_e and before T_a, where T_e and T_a are EFFECT transitions.

$$previous[T_p] = T_e \wedge next[T_p] = T_a \Rightarrow next[T_e] = T_a$$
$$previous[T_p] = T_e \wedge next[T_e] = T_a \Rightarrow next[T_p] = T_a$$
$$next[T_p] = T_a \wedge next[T_e] = T_a \Rightarrow previous[T_p] = T_e$$

   Note that all next and previous variables' domains are consists of only EFFECT transitions, not PREVAIL transitions. This is case because PREVAIL transitions do not change a state, so they can't appear in the left side of the causal link $T[s]T'$. The next and previous variables model the causal links.

3. **Action Synchronization Constraints**: If an action is in the plan then all the transitions caused by the action must also be in the plan and vice versa, i.e for all action A,

$$inplan[A] = true \Leftrightarrow \forall_{T.act=A} T : \neg(next[T] = \perp).$$

   Note that this constraint is bi-directional, i.e. if $\perp$ is removed from a next variable then it implies that the corresponding action is included in the plan.

4. **Transition Exclusion Constraint**: If a transition T is excluded from the plan, then no transition can appear before or after it, i.e. for all transition T [1],

$$next[T] = \perp \Leftrightarrow \text{previous[T]} = \perp.$$

5. **Action Time Synchronization Constraints:** Start times of transitions must be consistent with the start time of their corresponding actions and vice versa.

$$start[A] = \forall_{\mathrm{act}(T)=A} T : start[T] - offset(T)$$

[1] both EFFECT and PREVAIL transitions

Similarly, each action's end time is must be equal to the maximum of the end times of its transitions.

$$end[A] = \max\{\forall_{\text{act}(T)=A} T : end[T]\}$$

6. **Support Constraints**: Each assignment of a next variable implies a precedence constraint between the transitions.

$$next[T] = T' \Rightarrow T \rightarrow T'$$

Similarly, each assignment of the support variables also implied a precedence constraint between the transiitions.

$$support[T, T'] > 0 \Rightarrow T \rightarrow T'$$

7. **Temporal Position Constraints**: For each precedence constraint $T \rightarrow T'$ we post the following temporal constraints

$$inplan[T] \Rightarrow \text{dist}(\text{act}(T), \text{act}(T')) \geq \text{dur}(T) + \text{offset}(T) - \text{offset}(T') + setuptime(\text{Setup}(T), \text{Setup}(T'))$$

Recall that $\text{dist}(\text{act}(T), \text{act}(T'))$ represents the distance between the starlings of the two actions: $\text{start}(\text{act}(T')) - \text{start}(\text{act}(T)$, and the $setuptime(\text{Setup}(T), \text{Setup}(T'))$ denotes the time delay needed between the given setup states.

8. **Non-preemptive Transition Constraints:** Since transitions are non-preemptive, the following condition must hold for all transition T.

$$end[T] - start[T] = T.duration$$

In additions to these constraints we maintain the transitive closure of the precedence relations conditioned on the inclusion of transitions. That means, for transitions T, T' and T'', if $T \rightarrow T'$ and $T' \rightarrow T''$, we only post the precedence relation $T \rightarrow T''$ if and only if inplan[T']= true.

Preliminary Evaluations

We implemented a simple constraint solver to solve such planning problems in C++. As we are not aware of any public available planning benchmarks with complex temporal constraints on resources, nor solvers readily available to easily model and solve such problems, we evaluated our own solver with a set of randomly generated benchmark representing completing orders in the factory depicted in the example in Fig. 1. Our solver ran on servers with AMD Opteron(TM) Processor 6272 at 2.4GHz. We enforce a time limit of 30 minutes and memory limit of 2GB per instance.

We tested our factory setting with 5, 10, 15 and 20 orders at 50 instances each. The table in Fig. 5 reports the number of transitions, the average cpu time for the solved instances, and the number of unsolved instances. Fig.6 shows the performance of our solver against cpu time. The problem is solved almost instantaneously for small instances, and the difficulty of the problem increases with its size. We note that our solver is still under development, and its performance should improve once more powerful propagators are used.

orders	transitions	avg cpu time	failure
5	188.44	2.37	0/50
10	347	32.69	0/50
15	649.95	133.54	8/50
20	855.87	299.87	4/50

Figure 5: Solver statistics for factory with 5 to 20 orders.

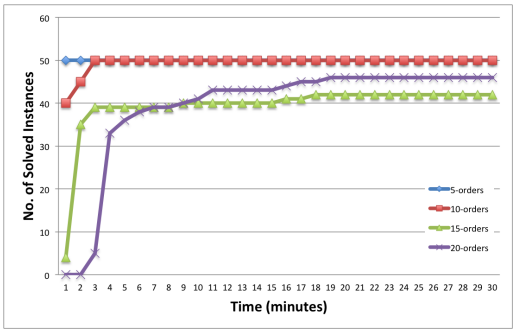

Figure 6: Number of solved instances against CPU time.

Summary

We proposed a new approach to model and solve planning problems with resources. Our approach extends an action-based planning planning domain description that provides an easy and direct way to model practical problems with complex temporal constraints. It concisely compiles the planning problem as a constraint satisfaction problem, and provides an interesting alternative to the current state of the art in modeling a wide ranges of possible planning applications. Possible future work includes improving the solver performance by improving the efficiency of constraint encoding and propagation, and evaluate our solver against other existing solvers on a set of expressive benchmark problems.

References

AIPS-98 Planning Competition Committee. PDDL - The Planning Domain Definition Language. Technical Report, Yale Center for Computational Vision and Control, 1998.

D. Banerjee. Integrating Planning and Scheduling In a CP Framework : A Transition-based Approach In *Proceedings of the 19th International Conference on Automated Planning and Scheduling (ICAPS)*, 2009.

D. Banerjee and P. Haslum. Partial-Order Support-Link Scheduling. In *Proceedings of the 21st International Conference on Automated Planning and Scheduling (ICAPS)*, 2011.

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

David McAllester and David Rosenblitt. Systematic nonlinear planning. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, pages 634639, 1991.

D. Smith The case for Durative Actions: A Commentary on PDDL2.1 *Joural of Artificial Intelligence Research*, 20:149-154, 2003.

D. Smith, J. Frank and W. Cushing. The ANML Language In *ICAPS Workshop on Knowledge Engineering for Planning and Scheduling*, 2008.