Discretization of Temporal Models with Application to Planning with SMT

Jussi Rintanen*
Department of Information and Computer Science
Aalto University, Helsinki, Finland

Abstract

The problem of planning or discrete control for timed system has earlier been solved with various constraint-based solution methods, including Constraint Programming, SAT solvers, SAT modulo Theories solvers, and Mixed Integer-Linear Programming. In this work we investigate the encoding of time in such constraint-based representations. A main issue with existing encodings is the necessity to allow arbitrary interleavings of concurrent actions' starting and ending times. The complex combinatorics of this can lead to poor scalability of leading search methods. We show how real or rational time in temporal models can in many practically important cases be replaced by integer time, and how this leads to far simpler encodings of planning as constraints. We demonstrate that the simplified encodings substantially improve the scalability of constraint-based planning.

Introduction

Temporal planning, similarly to other important problems about timed systems, can be reduced to constraint-based languages such as Constraint Satisfaction Problems (CSP), Mixed Integer Linear Programming (MILP) and Satisfiability modulo Theories (SMT), and solved with general-purpose solvers for these languages. Encodings of planning in SMT are in several respects similar to encodings of classical planning in SAT, but the far more complex model of time means that some of the core issues in SMT encodings do not have a counterpart in SAT encodings. The most central issues we will be addressing in this work are action exclusions and delayed effects.

Extensions of the SAT problem were first applied to problems such as planning with numerical state variables more than 15 years ago (Wolfman and Weld 1999). The modeling languages for temporal planning favored by the planning competition (IPC) community were shown by Shin and Davis (2005) to be effectively representable in the SMT framework (Wolfman and Weld 1999; Audemard et al. 2002). The Shin and Davis framework is very general, addressing planning in complex timed systems with continuous change and far exceeding both the complexity of typical temporal planning problems and the capabilities of existing search methods for timed and continuous systems. This suggests that parts of the generality may impede efficient implementations, which has motivated various hybrid approaches that perform action selection in the first phase and action scheduling in a separate phase. A representative of such approaches is that of Rankooh and Ghassem-Sani (2012; 2013). Other planners replace the first phase with other search paradigms (Vidal and Geffner 2006).

Similarly to classical planning (Kautz and Selman 1992; Rintanen 2009), a problem instance in temporal planning can be reduced to a sequence of constraint-satisfaction problems. Each of these problems represents a sequence of *steps*, which correspond to those states encountered during the execution of a plan in which a (discrete) change takes place (Shin and Davis 2005). In contrast to classical planning, temporal planning involves choosing the *absolute times* for each of the steps as well as the scheduling of the actions and their effects to these steps. Most notably, the *effects* of an action taken at step i are not necessarily at step i or the most closely following steps, but could be arbitrarily far, depending on the duration of the action and the number of other actions taken before or after. Standard encodings have to allow for arbitrary scheduling of actions' start and end points on the sequence of all steps. The difficult combinatorics of this incurs a high computational cost.

In this work we investigate the possibilities of replacing the real or rational timeline with an integer timeline, and devise methods for simplifying the encoding of planning as constraints when the integer timeline can be used. Our experimental results demonstrate the efficiency gains that are possible with simpler and more effective encodings.

The structure of the paper is as follows. We first present a simple yet general model of temporal planning, and sketch constraint-based encodings for it. As the main result of the work we prove that for important classes of temporal planning problems, time can be discretized in the sense that the continuous or dense real or rational timeline can be replaced by an integer timeline. Then we show how for actions with short durations this allows dramatically simpler encodings, with substantial performance improvements. We conclude

*Also affiliated with Griffith University, Brisbane, Australia, and the Helsinki Institute of Information Technology, Finland. This work was funded by the Academy of Finland (Finnish Centre of Excellence in Computational Inference Research COIN, 251170).

the paper by pointing further research topics.

A Model of Temporal Planning

The temporal model used by us differs from that of Shin and Davis (2005), whose use the PDDL, where explicit exclusion holds for action starting points only, and overlap of actions is handled with state variables so that a variable is made *false* in the start of the action, and other exclusive actions cannot be taken because they require the variable to be true.

In this paper we use Boolean *state variables* with values 0 and 1, as well as *unary resources* that can be used by at most one action at a time. If x is a state variable, then x and $\neg x$ are literals. The *complement* $\bar{l}$ of literal l is defined by $\overline{x} = \neg x$ and $\overline{\neg x} = x$.

An action consists of a *precondition* which determines whether an action can be taken at a given point of time, as far as the values of the state variables are concerned, *resource requirements* which determine whether the action can be taken in temporal relation to other actions, and *effects* which determine how and when state variables change after the action has been taken.

Definition 1 *Let X be a finite set of state variables and R a finite set of resources. An action is a triple $\langle p, r, e\rangle$ where*

- *the precondition p is a propositional formula over X,*
- *the resource requirement r consists of triples (t_0, t_1, v) where t_0 and t_1 are rational numbers such that $0 \leq t_0 \leq t_1$ and $v \in R$ is a resource, and*
- *the effect e is a set of pairs (t, l) where $t \geq 0$ is a rational number and l is a literal over X.*

We define plans as finite sets $\pi \subseteq \mathbb{Q} \times A$ of pairs (t, a) that assign a starting point to a set of actions.

A necessary condition for taking an action at time point t is that its precondition is true at t. Notice that we allow an action to have effects at the starting time point 0 where also the action's precondition is evaluated. We assume that the time 0 effects and the precondition don't share state variables. Additionally we rule out the possibility of two actions being taken simultaneously with each satisfying (part of) the preconditions of the other. We call this our *acyclicity assumption* for time 0 effects. This assumption is used later in the proof of Theorem 3. A slightly weaker theorem would clearly be possible without this assumption, for example one that assumes that simultaneous preconditions and effects do not share state variables.

If an action is taken at t_0, an effect (t, l) changes the literal l true at $t_0 + t$.

If an action taken at t has resource requirement (t_0, t_1, v), and another action, taken at t', has the resource requirement (t_2, t_3, v), then the actions respectively require the same resource at time intervals $]t + t_0, t + t_1[$ and $]t' + t_2, t' + t_3[$. Since a unary resource may be used by at most one action, taking the second action may not violate the following condition, expressed in terms of a *clock* c that measures the time that has passed since taking the first action.

$$c + t_1 \leq t_2 \text{ or } t_3 \leq c + t_0 \quad (1)$$

variable	description
$x@i, x \in X, 0 \leq i \leq T$	state variables
$a@i, a \in id(A), 0 \leq i < T$	actions
$\tau@i, 0 \leq i \leq T$	absolute time at step i
$\Delta@i, 0 < i \leq T$	time difference of steps $i-1, i$
$c_a@i, a \in id(A), 0 \leq i < T$	clock for action a

Table 1: Variables used in the SMT encodings

Resources (t_0, t_1, v) with $t_0 < t_1$ are interpreted as open intervals $]t_0, t_1[$, and with $t_0 = t_1$ they are interpreted as closed intervals $[t_0, t_1]$. If we limited to durations > 0 only, it would suffice – from the point of view of allowing consecutive actions with a zero-duration gap between them – to have the intervals half-open. However, intervals $[t_0, t_0]$ have important uses, and it sometimes is desirable to allow them *between* two open intervals.

Our modeling language is *temporally expressive* in the sense of (Cushing et al. 2007) in that it can express problems that require actions to be taken concurrently. Many temporal planning problems – both real-world and standard benchmark problems – have purely sequential solutions which do not involve concurrency at all, and these solutions can often be found with classical planners. In this work we are not interested in this possibility, as the main reason for making time explicit is that finding plans with *a short makespan* (the time from the first action until the end of the last) is the main objective, and plans with the shortest makespan typically are not sequential. Although some of the most scalable search methods are not guaranteed to find the plans with the shortest possible makespan, all our results preserve the makespan and allow finding the plans with the shortest possible makespan. In the following, we call a plan *optimal* if it has the shortest possible makespan.

Encoding in SMT

The variables used in an encoding for T steps are listed in Table 1. Name of an action $a \in A$ is denoted by $id(a)$.

The precondition p of an action taken at step i has to be true at step i.

$$id(a)@i \rightarrow p@i. \quad (2)$$

Here we denote by $p@i$ the formula p with every state variable x has been replaced by $x@i$ for a step index i.

Since effect axioms depend on the representation of time delays, we will be describing them in detail in the next two sections which present two alternative encodings. We will denote the formula that represents the disjunction of all possible causes (different actions at different times) of a literal l becoming true at step i by $causes_i(l)$. At this state we simply express changes in terms of $causes_i(l)$ as follows.

$$causes_i(l) \rightarrow l@i \quad (3)$$

Frame axioms describe the conditions under which a state variable remains unchanged, or alternatively, lists possible reasons for change from true to false or false to true. They can be similarly expressed in terms of $causes_i(l)$.

$$(\bar{l}@(i-1) \wedge l@i) \rightarrow causes_i(l) \quad (4)$$

Exclusion of actions is represented by two categories of formulas. The first category prevents the simultaneous execution of two actions. This can be easily statically determined by looking at the resource requirements of the actions: if the resource requirements conflict, the actions cannot be executed at the same time.

$$\neg id(a_1)@i \vee \neg id(a_2)@i \tag{5}$$

These formulas can be derived as follows. Let the actions allocate the same resource respectively over the intervals $[t_0, t'_0[$ and $[t_1, t'_1[$ relative to the time when the action is taken. If a_0 and a_1 were to be taken simultaneously at step i, the constraint that would have to be satisfied is

$$(id(a_0)@i \wedge id(a_1)@i) \rightarrow (t'_0 \leq t_1) \vee (t'_1 \leq t_0).$$

As all the constant on the right hand side of the implication are known at the time the encoding is formed, it can always be simplified to either

$$(id(a_0)@i \wedge id(a_1)@i) \rightarrow \bot$$

or

$$(id(a_0)@i \wedge id(a_1)@i) \rightarrow \top$$

where $\bot$ and $\top$ are respectively *false* and *true*. The latter is redundant and can be ignored. The former is equivalent to (5). For all actions, this encoding has a quadratic size in the number of actions. Generalizations of techniques from encoding classical planning in SAT can (often) be applied to obtain linear size encodings but it is not clear whether linear encodings are always possible.

The second category of formulas prevents taking an action if its resource requirements conflict with those of an action taken at an *earlier step*. As these formulas depend on the encoding of time delays, we will be presenting them in the next two sections for the two alternative encodings.

Direct Reference to Steps with Absolute Time

The first encoding of temporal planning (with various extensions, including continuous change) in SMT was by Shin and Davis (2005). The interesting part is how time delays for effects and for resources are handled. Absolute times for all steps i are represented in terms of real variables $\tau@i$. The variables have to satisfy

$$\tau@(i-1) < \tau@i. \tag{6}$$

Delays in action effects are represented with direct references to absolute time: effect (t, l) (with $t > 0$) of action a will take place at step i if there is an earlier step j where a is executed and the time difference between i and j is t. This can be encoded by the formula $\phi^a_l@i$.

$$\phi^a_l@i = \bigvee_{j=0}^{i-1} (id(a)@j \wedge (\tau@i - \tau@j = t))$$

The formula $causes_i(l)$ is simply the disjunction of all formulas $\phi^a_l@i$ for different actions a.[1]

[1]Immediate effects $(0, l)$ are handled in the trivial way, and we will not discuss this special case separately here or later.

Additionally, it is required that if an action is taken, then for every effect (t, l) one later step is t later in absolute time.

$$id(a)@i \rightarrow \bigvee_{j=i}^{T} (\tau@j - \tau@i = t) \tag{7}$$

Let two actions respectively need the same resource for intervals $]t_0, t_1[$ and $]t_2, t_3[$. We derive a constraint for the second action when the first action has already executed earlier. This is directly from Equation 1 and with a disjunction that iterates over all past steps and tests whether the time c in (1) has passed since taking action a_1.

$$\begin{aligned} id(a_2)@i \rightarrow \textstyle\bigvee_{j=1}^{i-1} (id(a_1)@j \rightarrow (t_2 \geq t_1 + (\tau@i - \tau@j) \\ \vee t_0 + (\tau@i - \tau@j) \geq t_3)) \end{aligned} \tag{8}$$

Indirect Reference to Steps through Clocks

We have experimented with a second encoding of time that uses *clocks* which represent the time that has passed since an action was taken. This encoding is an improvement over the previous one in terms of its size, $\mathcal{O}(T)$ instead of $\mathcal{O}(T^2)$. However, the magnitude of T is typically relatively small, and the encoding uses a far higher number of real variables which may negatively impact scalability of SMT solvers.

Initially, clocks are initialized to high values to indicate that the corresponding actions are not active. We take this to be $1 + max_a$, where max_a is the maximum t for an effect (t, l) of a or t_1 for a resource requirement (t_0, t_1, v) of a:

$$c_a@0 = 1 + max_a \tag{9}$$

for all $a \in A$. Similarly, at the last step of the plan no action may be active.

$$c_a@T >= max_a \tag{10}$$

Time differences between consecutive steps are positive.

$$\Delta@i > 0 \tag{11}$$

When an action is taken, its clock is reset to zero.

$$id(a)@i \rightarrow (c_a@i = 0) \tag{12}$$

At other steps the clock progresses by Δ.

$$\neg id(a)@i \rightarrow (c_a@i = c_a@(i-1) + \Delta@i) \tag{13}$$

If action a with an effect (t, l) has been taken, there has to be a step where the action's clock has value t. This means that the value of the clock may not jump past t:

$$(c_a@(i-1) < t) \rightarrow (c_a@i \leq t) \tag{14}$$

To form the effect and frame axioms 3 and 4, if action a has effect (t, l), then the action contributes $c_a@i = t$ as one disjunct to $causes_i(l)$.

Finally, constraints for resource conflicts are expressed in terms of clocks as in Equation 1. Let two actions a_1 and a_2 respectively need the same resource for intervals $]t_0, t_1[$ and $]t_2, t_3[$. The constraint is as follows.

$$id(a_2)@i \rightarrow (t_2 \geq c_{a_1}@i + t_1 \vee c_{a_1}@i + t_0 \geq t_3) \tag{15}$$

This completes the encoding with explicit clock variables.

As is obvious from the above, most of the complexity in both encodings stems from the fact that the steps where action's effects will take place is not known at the encoding time, and active actions can be interleaving and nesting active actions in multiple possible ways. Similarly, it is not known how long (in terms of the number of steps) an action allocates resources. In the next section we propose methods for reducing this complexity.

Discretization to Integer Time

A problem with both encodings in the previous section is the need to anticipate arbitrary schedulings of effects to later steps. In this section we show how in many practically interesting cases this scheduling can simplified, and in the following section we show how this can lead to dramatically improved SMT encodings. The basic insight is that in many important cases real or rational time can be discretized to integer time, and the correspondence between time points and steps can be determined at the encoding time without the need of representing all possible such correspondences in the encoding itself. For example, for many problem types it can be determined that the effects of an action taken at step i will be at step $i+n$ for some small integer n which is known at the time of reduction to SMT. This often dramatically simplifies the encodings.

Definition 2 *An action (p, r, e) has* integer time *if*

- *for every $(t, l) \in e$, t is an integer, and*
- *for every $(t_0, t_1, v) \in r$, both t_0 and t_1 are integers.*

For our main result we need the following lemmas which express rather general conditions that guarantees a temporal separation between two effects or effects and a precondition.

Lemma 1 *Let two actions respectively have effects e_1 at t_1 and e_2 at t_2, and let the actions respectively allocate the same resource at the intervals $]t_1^s, t_1^e[$ and $]t_2^s, t_2^e[$.*

Let d be any real number. If $t_1^e - t_2^s \geq t_1 - t_2 + d$ and $t_2^e - t_1^s \geq t_2 - t_1 + d$, then in any valid execution with the actions, there is at least time d between the time points where the effects e_1 and e_2 take place.

Proof: Let the two actions be respectively taken at time points t and t'. Because of the conflicting resource requirements, t and t' have to satisfy one of the following conditions.

$$t + t_1^e \leq t' + t_2^s \tag{16}$$
$$t' + t_2^e \leq t + t_1^s \tag{17}$$

The first condition says that the first action has to release the resource at the latest at the same time when the second action allocates it. The second condition corresponds to the second action using the resource first. We rewrite these conditions as follows for later use.

$$t \leq t' + (t_2^s - t_1^e) \tag{18}$$
$$t' \leq t + (t_1^s - t_2^e) \tag{19}$$

The requirement that the effects e_1 and e_2 are not closer to each other than d induces the following two conditions, one of which has to be satisfied.

$$t + t_1 + d \leq t' + t_2 \tag{20}$$
$$t' + t_2 + d \leq t + t_1 \tag{21}$$

The first condition says that the first action's effect e_1 is at least d earlier than the second action's effect e_2. The second condition is analogous. We rewrite these conditions as follows.

$$t \leq t' + (t_2 - t_1 - d) \tag{22}$$
$$t' \leq t + (t_1 - t_2 - d) \tag{23}$$

Now we can derive a condition on t_1, t_1^s, t_1^e, t_2, t_2^s, t_2^e and d that guarantees that the resource requirements prevent the two effects cannot be closer to each other than d. This condition is obtained by observing the similar form of 22,23 and 18,19, and the fact that if $t \leq t' + c$ and $c \leq c'$ then $t \leq t' + c'$: for one of the latter to be satisfied it is sufficient that both of the following hold.

$$t_2^s - t_1^e \leq t_2 - t_1 - d \tag{24}$$
$$t_1^s - t_2^e \leq t_1 - t_2 - d \tag{25}$$

which is what we set out to prove. □

Lemma 2 *Let two actions respectively allocate the same resource at the intervals $]t_1^s, t_1^e[$ and $]t_2^s, t_2^e[$ and let the first action have an effect e_1 at t_1.*

Let d be any real number. If $t_1^e - t_2^s \geq t_1$ and $t_2^e - t_1^s \geq d - t_1$, then in any valid execution with the actions, The effect e_1 is does not take place during time d after taking the second action.

Integer time alone is insufficient for discretization. In the following theorem we need further assumptions to be able to prove that (optimal) plans are not excluded by discretization. The first two assumptions respectively guarantee that potentially conflicting effects or conflicting effect and precondition will not be moved to the same time point as a result of discretization. The third assumption guarantees that actions can be moved to the preceding integer time point one by one, starting from the earliest actions, without causing resource conflicts with later actions.

Theorem 3 *Let all actions have integer time. Additionally assume the following.*

1. *If actions a and b respectively have effects (t_a, x) and $(t_b, \neg x)$, then they have resource requirements (t_a^s, t_a^e, r) and (t_b^s, t_b^e, r) for some resource r such that $t_1^e - t_2^s \geq t_1 - t_2 + 1$ and $t_2^e - t_1^s \geq t_2 - t_1 + 1$.*
2. *If action a has effect (t_a, l) and action b has precondition $\bar{l}$, then a and b respectively have resource requirements (t_a^s, t_a^e, r) and $(0, t_b^e, r)$ for some resource r such that either $t_a - t_a^s \geq 1$ or $t_b^e \geq 1$.*
3. *$t_0 = 0$ for all $(t_0, t_1, v) \in r$.*

Then, if there are plans for the problem instance, then there is a plan where all actions are scheduled at integer time points with an at most the same makespan.

Proof: We show that every action can be moved to the preceding integer time point. These moves can be done independently one by one, starting from the earliest actions, and what is obtained after each move is a valid plan.

Consider a plan π. Assume there is $(t, a) \in \pi$ such that t is not an integer. Let $t_0 = \lfloor t \rfloor$. We show that the plan $\pi' = \pi \backslash \{(t, a)\} \cup \{(t_0, a)\}$ obtained by moving a to the immediately preceding integer time point is a valid plan.

Since (t, a) is the earliest non-integer action in the plan, no other action is taken at any t_2 such that $t_0 < t_2 < t$. If another action is taken at t, we can always choose an action whose precondition was not made true by time 0 effects of a simultaneous action, which is possible by the acyclicity assumption discussed right after Definition 1. Hence the truth of the precondition of a in t entails its truth in t_0. Similarly, all resources available at t are available at t_0 as well, and by assumption 3 the action allocates all of its resources starting at t and not later. Hence a continues to be executable when moved from t to t_0.

It remains to show that moving a earlier does not overwrite the effects of other actions, nor falsifies the preconditions of later actions, nor violates resource requirements.

Consider an effect (t_1, l) of a. By assumption 1 and Lemma 1 there is no action with effect $\bar{l}$ taking place between time points $t_0 + t_1$ and $t + t_1$. Hence moving a from t to t_0 cannot overwrite the effects of other actions.

Assume a has an effect (t_1, l) at $t+t_1$, and there is another action with precondition $\bar{l}$ taken at some time point t_2 such that $t_0 + t_1 \leq t_2 \leq t + t_1$. Hence moving a earlier would falsify the other action's precondition. There can be no such action by assumption 2: the second action would require resource r starting at t_2 and a until $t+t_1$, and one of them for at least one unit of time, making the requirements conflict. Hence no such action can exist. Hence moving a from t to t_0 does not falsify the preconditions of any actions in the plan.

By assumption 3 resource conflicts cannot arise when the action is moved from t to t_0, because resources of all earlier actions are released at integer time points, and for a and later actions conflict with a at t_0 implies conflict with a at t. Hence moving a from t to t_0 is always safe.

The same moves can be done for all actions, one by one, and as a result we have a plan with integer time points only. The plan has a makespan at most that of the original plan. □

The next examples show why the assumptions of the theorem cannot be substantially relaxed.

First we explain why actions can only overlap if there is a gap of duration of at least 1 between their conflicting effects. This justifies assumption 1 of Theorem 3.

Example 1 *Consider actions* $\langle a, \emptyset, \{(1, \neg b), (1, c)\}\rangle$ *and* $\langle a, \emptyset, \{(1, b)\}\rangle$. *With an initial state that satisfies* $a \wedge \neg b \wedge \neg c$ *there is no unique optimal plan for reaching* $b \wedge c$. *The second action has to be started a non-zero amount of time later than the first to avoid the conflict and having its effect* b *overridden. The shortest plan with integral starting times has the second action started at time 1, and there are significantly shorter plans than this one.*

The next example shows that an action cannot falsify another action's precondition if there is only a small gap between them. This justifies assumption 2 of Theorem 3.

Example 2 *Consider actions* $a_1 = \langle a, \emptyset, \{(1, \neg a), (1, c)\}\rangle$, $a_2 = \langle a, \emptyset, \{(1, \neg b), (1, d)\}\rangle$, $a_3 = \langle b \wedge c, \emptyset, \{(1, e)\}\rangle$. *With initial state that satisfies* $a \wedge b \wedge \neg c \wedge \neg d \wedge \neg e$ *the plan* $(0, a_1), (0.5, a_2), (1, a_3)$ *reaches the goal* $d \wedge e$. *This is an optimal plan. Consider a discretized plan. Action* a_3 *cannot be earlier than 1 because its precondition is produced by* a_1. *Action* a_2 *cannot be at 0 because it would falsify the precondition of* a_3, *and it cannot be at 1 because its precondition would be falsified by* a_1. *There are no discretized plans for this problem.*

The theorem shows when action schedules can be limited to integer time points. Hence plans (with optimal makespan) can have a particularly simple structure.

- Progression of time can be limited to integers only.
- No other actions or effects take place between an action with duration 1 and its effects.
- For pairs of actions with unit durations, resource requirements can conflict only if the actions start simultaneously.

These properties can be utilized in any search method. Later we will see how this can substantially simplify SMT encodings of temporal planning.

Extensions for State Resources

Some of the benchmark problems used by the planning community use *over all* conditions of PDDL. The general mapping of these conditions to our resource-based representations involves state resources (Baptiste and Le Pape 1996) which in some cases are allocated for a single time point (interval $[t, t]$ with duration 0.)

Theorem 3 is stated for unary resources, which induce pairwise action exclusions. The proof trivially applies to resource conflicts caused by two actions allocating the same state resource in different states. For handling state resources allocated at single time points $[t, t]$ the proof needs to be extended.

Theorem 4 *In addition to the assumptions of Theorem 3, additionally assume that state resources are allowed and they fulfill the following properties.*

1. *All duration 0 allocations of a state resource are for the same state (that is, mutually non-conflicting).*

Proof: Sketch follows. The proof first proceeds exactly as in Theorem 3, with the duration 0 state resource allocations completely ignored. Then we argue that in the resulting discretized plan and execution, there are no new resource conflicts with the duration 0 state resources.

So consider any allocation of a duration 0 state resource at some time point t. Due to the process of moving all actions to an integer time point, this allocation will be moved to some integer time $t_0 \leq t$. Consider any conflicting allocation of the same resource. By assumption 1, no other allocation with duration 0 can conflict with the allocation. Hence any potentially conflicting allocation is for a non-zero interval $]t_1, t_2[$ such that $t_0 \leq t_2 \leq t$. Due to the discretization process, this allocation now is over some absolute interval $]t'_1, t'_2[$ such that $t'_2 = t_0$. Since this interval is open, it does not conflict with the duration 0 allocation at t_0. □

This theorem allows discretizing most of the standard benchmark problems with *over all* conditions.

Utilizing Discretization in SMT Encodings

Theorem 3 shows – for many practically important temporal planning problems – that actions with unit duration never need to be interrupted by intermediate events. Hence for any such action taken at a given step of a plan all of its effects can take place at the next step. Analogously, actions with small integer duration n can – under the same conditions – directly entail their effects n steps later. And, finally, constraints on conflicting resource usage can similarly directly refer to action variables at steps that are known at encoding time. This observation often allows eliminating all or much of the combinatorially hard parts of existing encodings.

We utilize the theorem as follows. First, as the theorem is only applicable to a subset of temporal planning problems, with integer delays and resource allocations, as well as the additional conditions in Theorem 3, our planner tests all those conditions, and proceeds with a standard encoding if they are not satisfied. Second, even when the theorem's conditions are satisfied, reduction of all actions with a "long" integer duration proceeds with standard encodings because representing all intermediate steps explicitly would explode the size of the encoding.

Only actions with integer durations of at most 5 are handled specially. For these actions (which we call *short actions*) our encoding is the following.

1. If not all other actions are short, then a short action of duration n implies duration and time constraints: the next n time differences between consecutive steps are all 1. This allows mixing the new compact encoding with the earlier general encodings for non-short actions.
2. The contributions of effects $(t, l) \in e$ of short actions $a = (p, r, e)$ to $causes_i(l)$ are particularly simple, because the step indices are known at encoding time. We simply have $id(a)@j$ for some $j < i$ as a disjunct of $causes_i(l)$.
3. Formulas for resource constraints directly refer to action variables in the relevant steps. Essentially, we reformulate (15) by replacing the real interval for the clock variable by a disjunction of action variables for different steps.

$$id(a_2)@i \rightarrow \bigwedge_{j=\max(0,\min(\lceil t_2-t_1 \rceil, i-1))}^{\max(0,\min(\lfloor t_3-t_0 \rfloor, i-1))} \neg id(a_1)@j \quad (26)$$

property	crewplanning	elevators	elevators-numeric	peg solitaire	floortile	matchcellar	openstacks	parking	sokoban	storage	tms	transport	turnandopen
discretizable	✓	✓	✓	✓	✓	✓		✓	✓		✓	✓	✓
some short	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓	✓
all short				✓	✓	✓		✓	✓	✓			✓
all unit				✓					✓				

Table 2: Properties of problems

		C	Cd	ITSAT
2008-pegsol	30	30	30	30
2008-sokoban	30	5	13	16
2011-floortile	20	5	18	20
2011-matchcellar	10	5	8	10
2011-parking	20	7	8	10
2011-turnandopen	20	10	16	20
2008-crewplanning	30	10	9	30
2008-elevators	30	4	7	15
2008-transport	30	0	4	E
2011-tms	20	8	8	20
2008-openstacks	30	1	1	24
2008-openstacks-adl	30	2	3	E
2011-storage	20	0	0	E
total	320	87	125	195

Table 3: Numbers of problems solved in 30 minutes

Experiments

We have translated IPC benchmark sets (2008 and 2011) for temporal planning into our modeling language. This is mostly straightforward. Parcprinter was excluded, due to its complexity. Table 2 classifies these problems according to whether the problem is discretizable (Theorems 3 and 4), whether some or all of discretizable actions are short (duration at most 5), and whether all actions have unit durations. The last two cases are particularly interesting: encodings in those cases do not contain real variables, and the result is a pure SAT problem. Further, in these two cases plans with the smallest possible number of steps have optimal makespan.

Then the translations from earlier in this work were applied to produce input for SMT solvers. Temporal invariants (Rintanen 2014) were included in the SMT encodings. Table 3 gives numbers of problem instances solved for a number of benchmark problems by our planners Cd and C in 1800 seconds of SMT solving time (the Z3 SMT solver on a Intel Xeon E3-1230 at 3.2 GHz). C is our baseline planner, with a Shin&Davis style encoding of steps for all actions.[2] Cd is the same planner with the encoding using discretization as presented in the previous section. Both planners increase the number of steps by 3 after the SMT solver determines the

[2]The clock-based encodings – though much more compact – perform clearly worse, and are not included in the experiments.

current formula to be unsatisfiable. For instances with short durations Cd therefore is guaranteed to always find plans with makespan at most 2 from optimal.

The column ITSAT presents runtimes for the ITSAT planner (Rankooh and Ghassem-Sani 2013). Some domains were not solved because of internal errors, marked with E. ITSAT is one of the strongest temporal planners overall. It is based on a reduction to SAT with encodings for classical planning (Rintanen, Heljanko, and Niemelä 2004; Wehrle and Rintanen 2007). The SAT encodings used in ITSAT allow far more parallelism than SMT encodings for temporal planing, which is likely to be the main reason for the excellent performance of ITSAT in comparison to our planners. A smaller difference may be that ITSAT uses the Precosat SAT solver, which performs extraordinarily well for planning problems, whereas the Z3 SMT solver used by us is based on the MiniSAT solver, which performs worse than Precosat with planning problems. Unlike our SMT encodings for discretizable problems with short action durations, the methods applied in ITSAT cannot be easily adapted to find plans with optimal makespan, due to the SAT encodings being unaware of durations.

Conclusions

We have developed and proved correct a general condition that allows limiting to integer-valued schedules instead of arbitrary real or rational valued schedules for plans in temporal planning. Although our experimental evaluation limits to constraint-based methods, our main result is equally applicable to other search methods including partial-order planning (Vidal and Geffner 2006) and the temporal variant of traditional explicit state-space search. Our experiments showed substantial improvement in scalability over state-of-the-art encodings for temporal planning in SMT. Future work includes investigation of additional methods for improving the scalability of temporal planning. Same ideas that have dramatically improved the scalability of classical planning (Rintanen 2012a; 2012b) are clearly applicable to temporal planning as well. A focus of future research is on topics specific to temporal planning.

References

Audemard, G.; Bertoli, P.; Cimatti, A.; Korniłowicz, A.; and Sebastiani, R. 2002. A SAT based approach for solving formulas over Boolean and linear mathematical propositions. In Voronkov, A., ed., *Automated Deduction - CADE-18, 18th International Conference on Automated Deduction, Copenhagen, Denmark, July 27-30, 2002, Proceedings*, number 2392 in Lecture Notes in Computer Science, 195–210. Springer-Verlag.

Baptiste, P., and Le Pape, C. 1996. Disjunctive constraints for manufacturing scheduling: principles and extensions. *International Journal of Computer Integrated Manufacturing* 9(4):306–310.

Cushing, W.; Kambhampati, S.; Mausam; and Weld, D. S. 2007. When is temporal planning really temporal. In Veloso, M., ed., *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, 1852–1859. AAAI Press / International Joint Conference on Artificial Intelligence.

Kautz, H., and Selman, B. 1992. Planning as satisfiability. In Neumann, B., ed., *Proceedings of the 10th European Conference on Artificial Intelligence*, 359–363. John Wiley & Sons.

Rankooh, M. F., and Ghassem-Sani, G. 2013. New encoding methods for SAT-based temporal planning. In *ICAPS 2013. Proceedings of the Twenty-Third International Conference on Automated Planning and Scheduling*, 73–81. AAAI Press.

Rankooh, M. F.; Mahjoob, A.; and Ghassem-Sani, G. 2012. Using satisfiability for non-optimal temporal planning. In *Logics in Artificial Intelligence, 13th European Conference, JELIA 2012, September 2012, Proceedings*, volume 7519 of *Lecture Notes in Computer Science*, 176–188. Springer-Verlag.

Rintanen, J.; Heljanko, K.; and Niemelä, I. 2004. Parallel encodings of classical planning as satisfiability. In Alferes, J. J., and Leite, J., eds., *Logics in Artificial Intelligence: 9th European Conference, JELIA 2004, Lisbon, Portugal, September 27-30, 2004. Proceedings*, number 3229 in Lecture Notes in Computer Science, 307–319. Springer-Verlag.

Rintanen, J. 2009. Planning and SAT. In Biere, A.; Heule, M. J. H.; van Maaren, H.; and Walsh, T., eds., *Handbook of Satisfiability*, number 185 in Frontiers in Artificial Intelligence and Applications. IOS Press. 483–504.

Rintanen, J. 2012a. Engineering efficient planners with SAT. In *ECAI 2014. Proceedings of the 21st European Conference on Artificial Intelligence*, 684–689. IOS Press.

Rintanen, J. 2012b. Planning as satisfiability: heuristics. *Artificial Intelligence* 193:45–86.

Rintanen, J. 2014. Constraint-based algorithm for computing temporal invariants. In Fermé, E., and Leite, J., eds., *Logics in Artificial Intelligence, 14th European Conference, JELIA 2014, September 2014, Proceedings*, volume 8761 of *Lecture Notes in Computer Science*, 665–673. Springer-Verlag.

Shin, J.-A., and Davis, E. 2005. Processes and continuous change in a SAT-based planner. *Artificial Intelligence* 166(1):194–253.

Vidal, V., and Geffner, H. 2006. Branching and pruning: an optimal temporal POCL planner based on constraint programming. *Artificial Intelligence* 170:298–335.

Wehrle, M., and Rintanen, J. 2007. Planning as satisfiability with relaxed $\exists$-step plans. In Orgun, M., and Thornton, J., eds., *AI 2007 : Advances in Artificial Intelligence: 20th Australian Joint Conference on Artificial Intelligence, Surfers Paradise, Gold Coast, Australia, December 2-6, 2007, Proceedings*, number 4830 in Lecture Notes in Computer Science, 244–253. Springer-Verlag.

Wolfman, S. A., and Weld, D. S. 1999. The LPSAT engine & its application to resource planning. In Dean, T., ed., *Proceedings of the 16th International Joint Conference on Artificial Intelligence*, 310–315. Morgan Kaufmann Publishers.