When is Temporal Planning *Really* Temporal?

Paper id: 1540

Abstract

While even STRIPS planners must search for plans of unbounded length, temporal planners must also cope with the fact that each action could start at any point in time. The majority of temporal planners cope with this challenge by restricting action starting times to a small set of *decision epochs*. Indeed, decision epoch planners won the International Planning Competition's Temporal Planning Track in 2002, 2004 and 2006. However, appearances can be misleading; in this case, the benchmark problems fail to expose the serious limitations of the decision epoch architecture. While investigating these weaknesses, we identify the notion of *required concurrency*, which separates expressive temporal action languages from simple ones. We show that decision-epoch planners are only complete for languages in the simpler class – one which is 'equivalent' to STRIPS! While there are complete temporal planners, such as VHPOP, there are none which support the powerful state-based reachability heuristics developed for classical planning. We address this lack, by developing a novel temporal planning algorithm, $TEMPO$, which supports state-based heuristics while retaining completeness.

1 Introduction

Although researchers have investigated a variety of architectures for temporal planning (e.g., plan-space search [Penberthy & Weld, 1994; Younes & Simmons, 2003], extended planning graph [Smith & Weld, 1999; Gerevini & Serina, 2002], reduction to linear programming [Long & Fox, 2003], and others), the most popular current approach is progression (or regression) search through an extended state space [Do & Kambhampati, 2001; Haslum & Geffner, 2001; Kvarnström *et al.*, 2000; Bacchus & Ady, 2001; Chen *et al.*, 2006], in which a search node is represented by a world-state augmented with the set of currently executing actions and their starting times. This architecture is appealing, because it is both conceptually simple and facilitates usage of powerful reachability heursitics, first developed for classical planning [Bonet *et al.*, 1997; Hoffmann & Nebel, 2001; Nguyen *et al.*, 2001; Helmert, 2004]. Indeed, SGPlan [Chen *et al.*, 2006], which won the International Planning Competition's Temporal Planning Track in both 2004 and 2006, is a progression planner.

There is an important technical hurdle that these temporal state-space planners need to overcome: each action could start at any of an infinite number of time points. Most of these planners avoid this infinite branching factor by a (seemingly) clever idea: selecting a very small number of special time points, called *decision epochs*, and only considering starting actions at these epochs. Unfortunately, the popularity of this approach belies an important weakness — we found that decision epoch planners are *incomplete* for many planning problems requiring concurrency.

Seen in juxtaposition with their phenomenal success in the planning competitions, this incompleteness of decision epoch planners raises two troubling issues:

1. Are the benchmarks in the planning competition capturing the essential aspects of temporal planning?
2. Is it possible to make decision epoch planners complete while retaining their efficiency advantages?

In pursuit of the first question, we focused on characterizing what makes temporal planning really temporal — i.e. different from classical planning in a fundamental way. This leads us to formulate the notion of *required concurrency*: the ability of a language to encode problems for which *all* solutions are concurrent. This notion naturally divides the space of temporal languages into those that can require concurrency (*temporally expressive*) and those that cannot (*temporally simple*). What is more, we show that the temporally simple languages are only barely different from classical, non-temporal, languages. This simple class, unfortunately, is the only class for which decision epoch planners are complete.

In pursuit of the second question, we show that the incompleteness of decision epoch planners is fundamental: anchoring actions to absolute times appears doomed. This leaves the temporal planning enterprise in the unenviable position of having one class of planners (decision epoch) that are fast but incomplete in fundamental ways, and another class of planners (partial order ones such as Zeno and VHPOP) that are complete but often unacceptably slow. Fortunately, we find a way to leverage the advantages of both approaches: a temporally lifted state-space planning algorithm called $TEMPO$. $TEMPO$ uses the advantage of lifting (representing action start times with real-valued variables and reasoning about constraints) from partial order planners, while still maintaining the advantage of logical state information (which allows the exploitation of powerful reachability heuristics). The rest of the paper elaborates our findings.

2 Temporal Action Languages

Many different modeling languages have been proposed for planning with durative actions, and we are interested in their relative expressiveness. The TGP language [Smith & Weld, 1999], for example, requires that an action's preconditions hold all during its execution, while PDDL 2.1/3 allows more modeling flexibility. We characterize the space of temporal action languages in terms of the *times* at which preconditions and effects may be 'placed' within an action. Our notation uses superscripts to describe constraints on preconditions, and subscripts to denote constraints on effects: $\text{PDDL}^{\text{preconditions}}_{\text{effects}}$ is the template. The annotations are:

s	"at-start"
e	"at-end"
o:	"over-all" (over the entire duration)

For example, $\text{PDDL}^{o}_{s,e}$ is a language where every action precondition must hold over all of its execution and effects may occur at start or at end. We study different restrictions to PDDL 2.1/3 and in line with its approach allow preconditions to be s, o or e but allow effects to be only s or e.

In this notation, $\text{PDDL}^{s,o,e}_{s,e}$ is, in all important aspects, the same as PDDL 2.1/3. For the bulk of this paper, we exclude consideration of deadlines, exogenous events (timed literals), conditional effects, parameters (non-ground structures), preconditions required at intermediate points/intervals inside an action or effects occurring at arbitrary metric points (as in ZENO [Penberthy & Weld, 1994]). We note that all of these can be, and in practice often are, compiled away. [Fox & Long, 2003; Do & Kambhampati, 2001; Smith, 2003] We exclude metric resources and continuous change for simplicity.

2.1 Basic Definitions

Space precludes a detailed specification of action semantics; thus, we merely paraphrase some of the relevant aspects of the PDDL 2.1/3 semantics for durative actions [Fox & Long, 2003].

Definition 1 (Actions) *A* model, M, *is a total function mapping fluents to values and a* condition *is a partial function mapping fluents to values. An atomic* transition *is given by two conditions: its preconditions, and its effects.*

A durative action, A, *is given by a beginning transition* $begin(A)$, *a final transition* $finish(A)$, *an over-all condition* $o(A)$, *and a positive, rational, duration* $\delta(A)$.

Definition 2 (Plans) *A* plan, $P = \{s_1, s_2, s_3, \ldots, s_n\}$, *is a set of steps, where each* step, s, *is given by a durative action,* $action(s)$, *and a positive, rational, starting time* $t_{begin}(s)$. *The* makespan *of* P *equals*

$$\max_{s \in P}(t_{begin}(s) + \delta(action(s))) - \min_{s \in P}(t_{begin}(s))$$

A rational model of time provides arbitrary precision without Real complications.

Definition 3 (Problems) *A* problem, $\mathcal{P} = (\mathcal{A}, I, G)$, *consists of a set of actions* ($\mathcal{A}$)*, a model representing the initial world state* (I)*, and a condition denoting the goal* (G)*.*

A precise formulation of plan *simulation* is long and unnecessary for this paper; see [Fox & Long, 2003]. Roughly, the set of durative actions is expanded into a sequence of atomic transitions, which may be used to progress the initial world state through a sequence of intermediate states. Each durative action expands into a beginning transition (encoding at-start preconditions and effects), some persistence constraints to check the action's overall condition, and a final transition (encoding the at-end preconditions and effects). The preconditions of each transition (and the persistence constraints) are tested to ensure plan legality. If the goal condition is true in the final state, then the plan is a *solution* to the problem.

Definition 4 (Completeness) *A planner is* complete *with respect to an action language* L*, if for all problems expressible in* L*, the planner is guaranteed to find a solution if one exists. A planner is* optimal, *with respect to language* L *and cost function* c*, if for all solvable problems expressible in* L*, the planner is guaranteed to find a solution minimizing* c*. A planner is* makespan optimal *if it is optimal with* makespan *as the cost function.*

2.2 Required Concurrency

We now come to one of the key insights of this paper. In some cases it is handy to execute actions concurrently; for example, it may lead to shorter makespan. But in other cases, concurrency is essential at a very deep level.

Definition 5 (Required Concurrency) *Let* $P = \{s_i\}$ *be a plan. Two steps,* $s, s' \in P$*, are* concurrent *when either* $t_{begin}(s) \leq t_{begin}(s') \leq t_{begin}(s) + \delta(action(s))$ *or* $t_{begin}(s') \leq t_{begin}(s) \leq t_{begin}(s') + \delta(action(s'))$*. A plan is* concurrent *when any two of its steps are concurrent. A planning problem has* required concurrency *if it is solvable and all solutions are concurrent.*

To make this concrete, consider the plan in Figure 1(a). The literals above the action denote its preconditions and below denote effects. If A and B are the only actions, then the problem of achieving G has required concurrency. Assuming that R is initially false, no sequential solution can make G true.

Definition 6 (Temporally Simple / Expressive) *An action language,* L*, is* temporally expressive *if* L *can encode a problem with required concurrency; otherwise* L *is* temporally simple.

2.3 Temporally Simple Languages

Theorem 1 PDDL^{o}_{e} *is temporally simple (and so is the TGP representation*[1] *[Smith & Weld, 1999]).*

[1]While TGP is temporally simple, there is no perfect correspondence to any strict subset of PDDL 2.1/3, because they have slightly different semantics and hence different mutex rules. PDDL^{o}_{e} is extremely close, however.

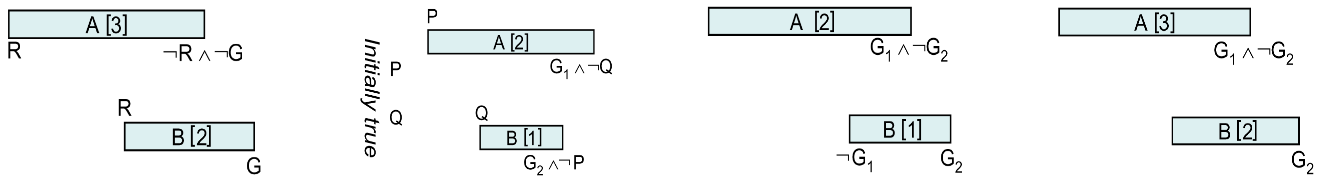

Figure 1: (**a**): Action A provides a resource, R, at the start of execution and later deletes it. Because B requires R as a precondition (only at-start), the two actions *must* execute concurrently, as shown, in order to achieve G. **(b)**: Each action's effects clobbers the other's precondition; $G_1 \wedge G_2$ can be achieved only by concurrently executing A and B. **(c)**: Preconditions are not even needed for a language to be temporally expressive. **(d)**: DEP is not optimal with respect to makespan; it can't generate this plan, because no action ends at the time when B must start to minimize duration.

Space precludes proofs; see the long version of our paper.

Theorem 1 is interesting, because a large number of temporal planners (TGP, TP4 [Haslum & Geffner, 2001], HSP* [Haslum, 2006], TLPlan [Bacchus & Ady, 2001], and CPT) have restricted themselves to the TGP representation, which is now shown to be so simple that essential temporal phenomena can't be modeled! Note, for example, that the common motif of temporary resource utilization (Figure 1(a)) can't be encoded in these representations and hence not generated by these planners.

Yet some of these planners did extremely well in the last three International Planning Competitions. The reality: the majority of the problems in the Temporal Track don't require concurrency! And these problems can even be solved using classical techniques:

Theorem 2 *Let $\mathcal{P}$ be a planning problem in a temporally simple subset of $\text{PDDL}_{s,e}^{s,o,e}$. If P is a solution to $\mathcal{P}$, then there exists a corresponding STRIPS problem $\mathcal{P}'$ where durations are ignored and all preconditions and effects are treated as a single atomic transition. Furthermore, there exists a homomorphic STRIPS plan P' which is a solution to $\mathcal{P}'$, P' can be rescheduled into a solution to $\mathcal{P}$ with makespan less than or equal to that of P, and this scheduling takes time which is linear in the length of the plan.*

Theorem 2 shows that temporal planning is essentially equivalent to STRIPS planning if the action language is temporally simple. Since no concurrency is required in the temporal plan, it can be serialized, and a STRIPS planner will find the corresponding plan.

Thus, it is natural to consider using a classical planner to generate temporal plans. The first step, converting the durative $\mathcal{P}$ to STRIPS $\mathcal{P}'$ is a simple linear-time procedure: each action's duration is ignored and all preconditions and effects are treated as a single atomic transition. There is no change to the number of actions, propositions, or plan lengths.

Secondly, one must solve the classical problem. The only subtlety here stems from the fact that there is no guarantee that the shortest serial STRIPS plan will correspond to the concurrent plan with optimal makespan, so one must think carefully about the correct heuristics for such a search.

Finally, it is easy to convert the serial STRIPS plan into a corresponding parallel, durative plan. Although [Bäckström, 1998] showed that finding the optimal parallelized order in which to schedule a STRIPS plan is NP-hard, we don't need to do that much work. A straightforward algorithm suffices: left-shift each action (starting from the head of the serial plan) as far as possible without causing preconditions and effects to clash; by caching the latest modification time for each fluent, only linear time is required[2].

Previous researchers have proposed ways of compiling temporal planning problems to classical ones, but ours is notable because it does not change the number of actions or the number of actions in a plan. In contrast, the mapping proposed in [Fox & Long, 2003] expands each temporal action into three STRIPS actions. Furthermore, the strategy used in MIPS [Edelkamp, 2003] has an expansion factor of two.

2.4 Temporally Expressive Languages

We have already seen (Figure 1(a)) one example of a language which can express problems with required concurrency. We now list several others below.

Theorem 3 $\text{PDDL}_{s,e}^{o}$ *is temporally expressive.*

It's no surprise that by adding at-start effects (in PDDL_{e}^{o}), one can represent required concurrency, but it is interesting to note that merely shrinking the time that preconditions must hold (from overall in temporally simple PDDL_{e}^{o} to at-start) also increases expressiveness. Figure 1(b) proves the following:

Theorem 4 PDDL_{e}^{s} *is temporally expressive.*

Even more surprising is the fact that preconditions aren't necessary for temporal expressiveness, as shown by the example of Figure 1(c).

Theorem 5 $\text{PDDL}_{s,e}$ *is temporally expressive.*

2.5 Summary

Figure 2 places the languages under discussion in the context of a lattice of the PDDL sub-languages, and shows the divide between temporally expressive and simple. An easy way to grasp the demarcation is to note that the temporal expressiveness of a language depends on whether or not the language allows a "*temporal gap*" between the preconditions and (or) effects of an action. Specifically, a durative action is said to allow temporal gap if there is no single time point during the

[2]Bäckström's results says that this scheduling procedure may fail to recognize when the causal structure could be broken to yield a shorter-makespan plan composed from the same steps. In such a case there exists a *different* STRIPS plan (with the same steps) which would get scheduled correctly. So as long as the STRIPS planner uses correct heuristics, we will find the makespan optimal plan.

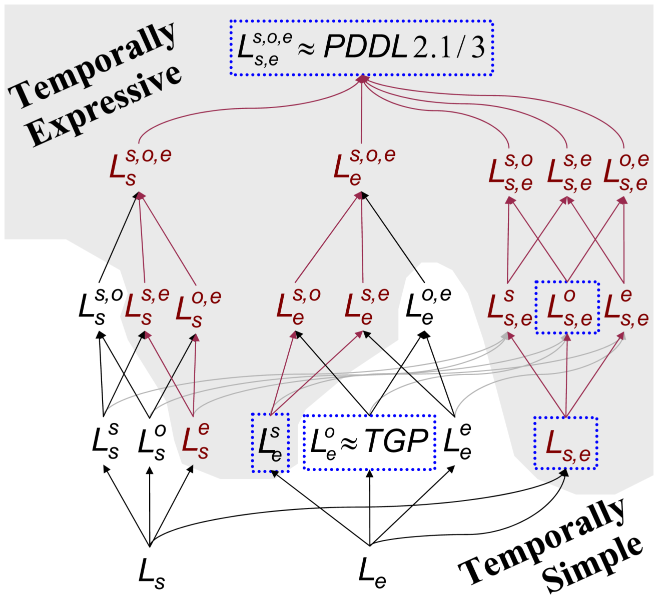

Figure 2: The taxonomy of temporal languages and their expressiveness; those outlined by dotted lines are discussed in the text.

execution of the action where all its preconditions and all its effects must hold together. It is easy to see from Figure 2 that the languages that model actions allowing temporal gap are expressive. For example, PDDL^s_e allows temporal gap and is thus expressive. In contrast, by requiring that all preconditions be true over the entire duration of the action PDDL^o_e precludes modeling actions allowing temporal gap, and is thus temporally simple. This makes intuitive sense since without a temporal gap, the duration of an action becomes a secondary attribute such as cost of the action.[3]

Coming back to the space of languages, we have already noted that several popular temporal planners restrict their attention to temporally simple languages, which are essentially equivalent to STRIPS. The next section shows that most of the planners which claim to address temporally expressive languages, are actually incomplete in the face of required concurrency.

3 Decision Epoch Planning

The introduction showed that most temporal planners, notably those dominating the recent IPC Temporal Tracks, use the decision epoch (DE) architecture. In this section, we look in detail at this method, exposing a concerning weakness: incompleteness for temporally expressive action languages.

SAPA [Do & Kambhampati, 2001], TLPlan [Bacchus & Ady, 2001], TP4, and HSP$^*_a$ [Haslum & Geffner, 2001], among others, all have decision-epoch based planners. Rather than consider each in isolation, we abstract the essential elements of their search space by defining DEP. The defining attribute of DEP is search in the space of temporal states:

Definition 7 *A temporal state, N, is given by a model of the world, $state(N)$, a time, $t(N)$, and a plan, $agenda(N)$, which is used for recording which actions have not yet finished.*

The central attribute of a temporal state, N, is the world state, $state(N)$. Indeed, the world state information is responsible for the success and popularity of DEP, because it enables the computation of state-based reachability heuristics developed for classical, non-temporal, planning. We now define DEP's search space by showing how temporal states are refined; there are two ways of generating children:

Fattening: Given a temporal state, N, we generate a child, N_A, for every action, $A \in \mathcal{A}$. Intuitively, N_A represents an attempt to start executing A; thus, N_A differs from N only by adding a new step, s, to $agenda(N_A)$ with $action(s) = A$ and $t_{begin}(s) = t(N)$.

Advancing Time: We generate a single child, N_{epoch}, by simulating forward in time just past the next atomic transition in the agenda: $N_{epoch}{=}simulate(N, d + \epsilon)$, where $d{=}min\{t : s \in agenda(N)$ and $(t{=}t_{begin}(s)$ or $t{=}t_{begin}(s) + \delta(action(s)))$ and $t \geq t(N)\}$.

Our definition emphasizes simplicity over efficiency. We rely on *simulate* (as defined by [Fox & Long, 2003]) to check action executability; inconsistent temporal states are pruned. Obviously, a practical implementation would check these during fattening.

Theorem 6 *DEP is complete for temporally simple languages, but not makespan-optimal.*

Figure 1d. presents an example of sub-optimality; DEP would find a serial solution, but not the plan with concurrency. Completeness follows trivially from Theorem 2 (just advance time whenever possible).

Theorem 7 *DEP is incomplete for temporally expressive languages.*

Since every temporally expressive language is a superset of PDDL^s_e, PDDL^e_s, or $\text{PDDL}_{s,e}$ (see Figure 2), it suffices to prove incompleteness for these three minimal languages. Figure 1c gives a PDDL^s_e example which stumps DEP— in order to achieve $G_1 \wedge G_2$ action B must be dispatched between one and two units after A, but there are no decision epochs in that interval. Similar examples work for PDDL^s_e and PDDL^e_s.

4 Generalized Decision Epoch Planning

As the example of Figure 1c shows, DEP doesn't consider enough decision epochs. Specifically, it makes the mistaken assumption that every action will start immediately after some other action has started or ended. In Figure 1c, however, action B has to *end* (not start) after A ends. Thus, it is natural to wonder if one could develop a complete DE planner by exploiting this intuition. In short, the answer is "No."

[3]This understanding of temporal expressiveness in terms of temporal gap is reminiscent of the "unique main sub-action restriction" [Yang, 1990] used in HTN schemas to make handling of task interactions easy. The resemblance is more than coincidental, given that temporal actions decompose into atomic transitions in much the same way as HTNs specify expansions (see Section 2.1).

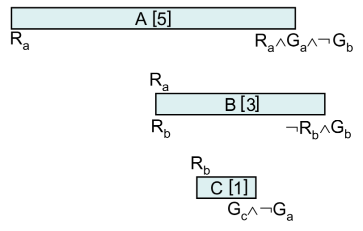

Figure 3: DEP+ can't find a plan to achieve $G_a \wedge G_b \wedge G_c$.

but the reason why the effort fails is instructive, so we present the DEP+ algorithm below.

Let Δ be the maximum duration of an action ($\Delta = \max \delta(\mathcal{A})$). In contrast to DEP, which adds new actions at time $t(N)$, DEP+ will add them at time $t(N) + \Delta$. Specifically, children of a temporal state, N, are added as follows:

Fattening: For every action A, we create two children of N. N^s_A is analogous to N_A in DEP— we commit to starting action A by adding a step s to $agenda(N^s_A)$ with $action(s){=}A$ and $t_{begin}(s){=}t(N^s_A) + \Delta$.

The latter, N^e_A, the essential difference from DEP, differs from N only in that $agenda(N^e_A)$ contains a new step, s, with $action(s){=}A$ and $t_{begin}(s){=}t(N^e_A) + \Delta - \delta(A)$.

Advancing Time: N_{epoch} is obtained from N by simulating to just after the first time where $t(N_{epoch}) + \Delta$ is the start or end of a step in $agenda(N_{epoch})$. Specifically, $N_{epoch}{=}simulate(N, d + \epsilon)$ where $d{=}\min\{t|s{\in}agenda(N)$ and $(t{=}t_{begin}(s)$ or $t{=}t_{begin}(s) + \delta(action(s)))$ and $t \geq t(N) + \Delta\}$.

DEP+ is more powerful than DEP, but, in only one way:

Theorem 8 *DEP+ is makespan optimal for simple temporal languages, but incomplete for expressive temporal languages.*

Makespan optimal plans in simple temporal languages always start or end actions immediately after some other action ends. By removing the action which ends last (in a makespan optimal solution), one can perform an induction on makespan to show that DEP+ will generate makespan optimal solutions.

Theorem 9 *DEP+ is incomplete for expressive temporal languages.*

DEP+ can't generate the plan in Figure 3, because there is no epoch to use for starting C until B has been added (ending just after A ends), but by this point $t(N)$ has advanced too far to start C. Similar examples exist for the other expressive languages. Furthermore, arbitrarily complex examples of chaining may be constructed; in these the necessary dispatch time for an action depends on actions an unbounded number of Δs in the future.

5 Lifted Progression Planning

The key observation about decision-epoch planning is that decisions about *when* to execute actions are made very eagerly – before all the decisions about *what* to execute are made. DEP attempts to create a "left-shifted" plan, in which each action is started as early as possible. Unfortunately, for temporally expressive languages, this translates into the following two erroneous assumptions:

***1** Every action will start immediately after some other action has started or ended.

***2** The only conflicts preventing an earlier dispatch of an action, however indirect, involve actions which start earlier.

In developing DEP+, we noted the first flaw, and attempted to address it by allowing synchronization on the *finish*()s of actions as well as their *begin*()s. However, there does not appear to be any (practical) way of addressing the second flaw within the decision-epoch approach. One must either define *every time point* to be a decision epoch (branching over dense time!) or pick decision-epochs forwards and backwards, arbitrarily far, through time (as in [Long & Fox, 2003]).

Instead, we develop a complete state-space approach, by exploiting the idea of *lifting* over time: delaying the decisions about *when* to execute until all of the decisions about *what* to execute have been made. Note that [Younes & Simmons, 2003] also lifts over time – we take a different approach that allows us to preserve state information at each search node.

Definition 8 *A lifted temporal state, $\mathcal{N}$, is given by the current temporal variable, $\tau(\mathcal{N})$, a model, $state(\mathcal{N})$, a lifted plan, $agenda(\mathcal{N})$, and a set of temporal constraints, $constraints(\mathcal{N})$.*

We retain the terminology used in DEP, and DEP+, to highlight the similarity of the approaches, despite the differences in details which arise from lifting time. For example, the agenda in a lifted temporal state is different from that in a (ground) temporal state — we replace exact dispatch times ($t_{begin}()$) with temporal variables ($\tau_{begin}()$), and impose constraints through $constraints(\mathcal{N})$. In fact, we associate every step, s, with two temporal variables: $\tau_{begin}(s)$ and $\tau_{finish}(s)$. All the duration constraints $\tau_{finish}(s) - \tau_{begin}(s) = \delta(action(s))$ and mutual exclusion constraints $\tau_x(s) \neq \tau_y(t)$, for mutually exclusive atomic transitions $x(action(s))$ and $y(action(t))$ (x and y are each one of *begin* or *finish*), are always, implicitly, part of $constraints()$. With respect to this notion of "state", we define our complete approach, $TEMPO$, to state-space temporal planning by describing the child-generator function.

Fattening: Given a lifted state, $\mathcal{N}$, we generate a child, $\mathcal{N}_A$, for every action, $A \in \mathcal{A}$. As before $\mathcal{N}_A$ represents starting A; we add a step s to $agenda(\mathcal{N}')$ with $action(s) = A$. In addition, we add "$\tau_{begin}(s) \geq \tau(\mathcal{N})$" to $constraints(\mathcal{N}')$. Unlike before, we immediately simulate: $\mathcal{N}_A{=}simulate(\mathcal{N}', \tau_{begin}(s))$. In particular, everything in $agenda(\mathcal{N}_A)$ has already started.

Advancing Time: For every $s{\in}agenda(\mathcal{N})$, we generate a child, $\mathcal{N}^A_{epoch}$, where $A{=}action(s)$. Note that A has already started; this is a decision to end A. Specifically, we add "$\tau_{finish}(s) \geq \tau(\mathcal{N})$" to $constraints(\mathcal{N}')$ and then simulate:

$\mathcal{N}^{A}_{epoch}{=}simulate(\mathcal{N}', \tau_{finish}(s))$

In essence, $TEMPO$ is searching the entire space of sequences of atomic transitions ($begin$()s and $finish$()s of all the actions), in prefix order. That is, every search state corresponds to the unique sequence of atomic transitions that, if (assigned dispatch times and) executed, result in the given (lifted) temporal state. Of course, just before terminating, $TEMPO$ must actually pick some particular assignment of times satisfying $constraints(\mathcal{N})$ (for a solution $\mathcal{N}$) in order to return a ground plan. Since $constraints(\mathcal{N})$ will, among other things, induce a total ordering, this will not be very difficult. So it should not be very surprising that $TEMPO$ is guaranteed to find solutions — if there is a solution, it has a sequence of atomic transitions, and $TEMPO$ will eventually visit that sequence, and find an assignment of times.

Theorem 10 *$TEMPO$ is complete (for both temporally simple and expressive languages), moreover, makespan optimal.*

6 Conclusion

This paper presents an in-depth study of temporal planning with focus on expressiveness, completeness and optimality. We introduce the notion of *required concurrency* which divides temporal languages into *temporally simple* (where concurrency is not required for correctness) and *temporally expressive*. We present a lattice of sub-languages of PDDL 2.1/3and relate the demarcation between its expressive and simple sub-languages to the syntactic notion of *temporal gap*. We show that minor modifications of classical planning techniques are sufficient to handle temporally simple languages. In short, we have tightly characterized when temporal planning is *really* temporal.

Having shown the expressiveness divide, we go on to prove that a surprisingly large class of popular temporal planners, which branch on a restricted set of decision epochs (*e.g.*, all state-space planners like SAPA, SGPlan), are complete *only* for the temporally simple languages. In fact, there exist problems even in simple languages for which these planners *cannot* be optimal. Evidently the more expressive temporal languages are not being handled by these beyond the parsing phase! Given the "winning" performances of decision epoch planners in the last three planning competitions (2002, 2004, 2006), this leads to the inescapable conclusion that the competition benchmarks are not exercising the essential aspects of temporal planning!

The appeal of the decision epoch planning has been the ability to leverage powerful state-based reachability heuristics. We end on a constructive note by outlining a new temporal planning algorithm $TEMPO$ that uses a lifted notion of state space to retain this advantage while simultaneously avoiding incompleteness.

References

[Bacchus & Ady, 2001] F. Bacchus & M. Ady. Planning with resources and concurrency: A forward chaining approach. In *IJCAI-01*, 2001.

[Bäckström, 1998] Christer Bäckström. Computational aspects of reordering plans. *JAIR*, 9:99–137, 1998.

[Bonet *et al.*, 1997] B. Bonet, G. Loerincs, & H. Geffner. A robust and fast action selection mechanism for planning. In *AAAI-97*, 1997.

[Chen *et al.*, 2006] Y. Chen, C. Hsu, & B. Wah. Temporal planning using subgoal partitioning and resolution in SGPlan. *JAIR*, to appear, 2006.

[Do & Kambhampati, 2001] M. Do & S. Kambhampati. Sapa: A domain independent heuristic metric temporal planner. In *ECP-01*, 2001.

[Edelkamp, 2003] S. Edelkamp. Taming numbers and duration in the model checking integrated planning system. *JAIR*, 20:195–238, 2003.

[Fox & Long, 2003] M. Fox & D. Long. Pddl2.1: An extension to PDDL for expressing temporal planning domains. *JAIR Spl. Issue on Planning Competition*, 20:61–124, 2003.

[Gerevini & Serina, 2002] A. Gerevini & I. Serina. Lpg: A planner based on local search for planning graphs. In *AIPS-02*, 2002.

[Haslum & Geffner, 2001] P. Haslum & H. Geffner. Heuristic planning with time and resources. In *ECP-01*, 2001.

[Haslum, 2006] P. Haslum. Improving heuristics through relaxed search - an analysis of TP4 and HSP*a in the 2004 planning competition. *JAIR*, 25:233–267, 2006.

[Helmert, 2004] Malte Helmert. A planning heuristic based on causal graph analysis. In *ICAPS'04*, pp. 161–170, 2004.

[Hoffmann & Nebel, 2001] J. Hoffmann & B. Nebel. The FF planning system: Fast plan generation through heuristic search. *JAIR*, 14:253–302, 2001.

[Kvarnström *et al.*, 2000] J. Kvarnström, P. Doherty, & P. Hasslum. Extending TALplanner with concurrency and resources. In *ECAI-00*, 2000.

[Long & Fox, 2003] D. Long & M. Fox. Exploiting a graphplan framework in temporal planning. In *ICAPS'03*, pp. 51–62, 2003.

[Nguyen *et al.*, 2001] X. Nguyen, S. Kambhampati, & R. Nigenda. Planning graph as the basis for deriving heuristics for plan synthesis by state space and csp search. *AIJ*, 135:73–123, 2001.

[Penberthy & Weld, 1994] S. Penberthy & D. Weld. Temporal planning with continuous change. In *AAAI-94*, 1994.

[Smith & Weld, 1999] D. Smith & D. Weld. Temporal planning with mutual exclusion reasoning. In *IJCAI-99*, 1999.

[Smith, 2003] D. E. Smith. The case for durative actions: A commentary on PDDL2.1. *JAIR Spl. Issue on Planning Competition*, 20:149–154, 2003.

[Yang, 1990] Q. Yang. Formalizing planning knowledge for hierarchical planning. *Computational Intelligence*, 6:12–24, 1990.

[Younes & Simmons, 2003] H.L.S Younes & R.G. Simmons. Vhpop: Versatile heuristic partial order planner. *JAIR Spl. issue on Planning Competition*, 20:405–430, 2003.