Spanning the Middle Ground between Classical and Temporal Planning

Peter Wullinger[1], Ute Schmid[1], and Ulrich Scholz[2]

[1] Otto-Friedrich-University Bamberg
Feldkirchenstr. 21, D-96052 Bamberg
{peter.wullinger,ute.schmid}@uni-bamberg.de
[2] European Media Laboratory GmbH
Schlo-Wolfsbrunnenweg 33, D-69118 Heidelberg
ulrich.scholz@eml-d.villa-bosch.de

Abstract. A recent evaluation of the complexity of temporal planning by Cushing et al. [1] unveiled the interesting result that many current temporal planning problems may be transformed into classical ones in linear time; in other words, they are actually not temporal. Based on this insight, we introduce a new, graded classification of classical planning problems that are temporal to some degree. We present a simple and complete solution method for solving temporally non-simple, expressive problems that makes use of this concept.

1 Introduction

Automated planning as a discipline of artificial intelligence is the process of finding the right actions (from a list of available actions) that transform a source state into a target state. Traditionally, state descriptions are given as conjunctions of first order predicates. State modifications are then specified as parameterised "operators" (action templates) that need to be instantiated to form concrete actions. They specify the conditions under which they are applicable and the modifications to the world state that their application executes.

Planning problems are specified on two abstraction levels: An abstract description of a certain situation is given in the "domain" specification of the planning problem, while the concrete initial and final states are specified in the "problem definition". Separating domain and problem definitions makes it possible to reuse planning domains for different concrete situations.

Planning domains can be classified according to the structure of the operators and how they may be positioned in time. *Classical planning* approaches view actions as atomic. A solution plan to a classical planning problem is a partial or total order of actions. For *temporal planning* domains, on the other hand, actions are no longer atomic, as each action gets assigned a duration and may arbitrarily be positioned on the time axis. For temporal planning, finding a correct solution is more complex: The correct solution may depend on the exact position of an action in time and not only of its place in the sequence of actions. Obviously, temporal planning is more complex than classical planning and requires more

elaborate planning methods, which are typically much slower than their classical counterparts.

2 Simple and Expressive Temporal Problems

Since the first appearance of temporal planning at the International Planning Competition (IPC) in 2004 [2], temporal planners have significantly improved in solving the posed problems. Yet, the actual profit gained by these improvements is disputable: Most current temporal planners are incomplete. Cushing et al. [1] establish this fact and provide also a classification for the affected planners. The authors show that temporal planning problems are encountered in two complexity classes: Simple and expressive. They also show that most current planners can solve only simple problems and give a classification for the planners within this class.

In a subsequent paper, the authors [3] provide a simple classification for temporally simple problems and apply them to the IPC 2002, 2004, and 2006 problem domains, which are all found to be temporally simple. By studying existing sample problems, the authors conclude, that temporally expressive problems seem to be quite rare. This observation is additionally supported by the fact that the IPC 2008 sample problems[3] also do not contain any expressive problems.

Yet, expressive problems are rare, although the language PDDL [2] used to define allows their formulation. This fact is problematic, as temporally expressive problems cause most current planners to simply fail without a suitable solution plan, even given infinite time and space.

2.1 Decision Epoch Planning

According to Cushing et al. [1], most current temporal planners classify as decision epoch planners (DEPs). The term refers to a performance "*trick*" that is implemented by the particular planners: Instead of considering to schedule an action at any possible point in time, only certain points in time —namely *decision epochs*— are considered by a DEP, which significantly reduces the search space. This approach allows a planner to adapt the concepts of classical planners for the new challenge of temporal planning.

But what exactly is a decision epoch? To give a proper definition, we first need to bring to mind a few well-known concepts: First, remember, that PDDL —in both its classical and temporal versions— allows effects to only take place at discrete points in time, now called *effection points*. Note, that an effection point t_e differs from the concept of a *happening* [4] in that the happening refers to the set of concurrently happening effects, while the effection point refers to a point on the time axis.

Definition 1. *Any point in time, where the world state is modified by an action is called an* **effection point (EP)**.

[3] Available from http://ipc.informatik.uni-freiburg.de/

We now consider planners that start at time t_0 and iteratively schedule operator instances. This mode of operation is called *progressive planning*, as the planner searches through a state space and progressively makes commitments about the current state. Examples include Fast Forward (FF, [5]) or Fast Downward [6] and derived planners.

A progressive temporal planner starts with an empty list of scheduled operator instances and subsequently positions new operator instances on the time axis, increasing the set of already positioned actions. This process also generates an increasing set of *decision epochs*:

Definition 2. *A* **decision epoch** *is an effection point of an already scheduled action.*

The basic idea of DEP is now, that every important planning problem at hand happens at such decision epochs. This argument seems plausible at first glance, as —by definition— every "committed to" effection point coincides with a decision epoch. Now, since conditions are limited to hold at either the start, end, or over the whole duration of an action, the following conclusion seems natural: Future actions (that are yet to be scheduled) only need to start or end at exactly such decision epochs; starting them elsewhere yields no benefit (for further discussion and why this argument is incorrect, see [1]).

2.2 Required Overlap

Scheduling actions only at decision epochs significantly reduces the search space, but is known to yield incomplete planners. Yet, DEP can solve most planning problems, while other approaches based on interval constraint solving (e.g. [7], [8]) are complete, but considerably slower.

In the mentioned work, Cushing et al. [1] relate this difficulty to the concept of *parallel actions*. Consider an optimal solution to a temporal planning problem. It consists of a positioning of operator instances in time. Now, it is often possible to move some of these actions in time without giving up the scheduling's property of being a solution to the planning problem. For example, when two actions are placed in direct succession and no other action is scheduled, the total length of the plan may be expanded indefinitely by simply shifting the second action forward in time. The resulting, transformed plan is not optimal any more, but it is still a valid solution plan. While the shifting often may not be as trivial as in this example, it is still possible in most cases.

Now, in partial order and in temporal planning, the durations of actions may overlap in time. In partial order classical planning, every solution plan may be stretched so that no actions overlap in time. The overlapping of actions is not required, it just serves to reduce planning time when no conflicts are detected. The same need not be the case for temporal planning: In certain circumstances, the effect/condition combination of the participating actions may prevent the actions to be shifted arbitrarily. For example, consider a simple "food slicer" situation, where a special "safety button" needs to be pressed, before the machine

is able to operate. By definition, there is no way to operate the machine, when the button is not pressed and thus any other action needs to overlap with the "push safety button" action.

But as noted, such "safety button" problems seem to be quite rare. Instead, most temporal solution plans contain only actions that may be shifted to yield a new solution with no overlap between the scheduled actions. We call such a transformed solution a *linear rescheduling*.

Definition 3. *A* **linear rescheduling** *of a solution to a temporal planning problem is one, where the actions of the original solution have been shifted, so there is no more overlapping of actions in time.*

2.3 Temporally Simple To Classical Transformation

The most important fact about linear solutions is that problems with linear solutions cannot be really called *temporal* [1]. We show, that a simple algorithm is sufficient to transform a temporally simple planning problem to a classical planning problem and this fact has also been confirmed by our experiments.

- Expand all condition specifications to hold `OVER ALL` except those that are satisfied by the effects of the same action. Remove these self-satisfied conditions.
- Move all effect specifications to come to effect `AT END`. This requires careful consideration of the effect order with regard to PDDL semantics, but has still proved to be possible without major difficulty.
- Drop all actions that are self-contradictory (by self-contradicting conditions or effects that deny a condition).

When these transformations have been performed, it is possible to run a *classical* planner on the new planning problem and obtain a —typically non-optimal— solution to the temporal planning problem. In experiments, we have successfully transformed temporal planning problems to classical planning problems using only the proposed simple transformations. We therefore can confirm Cushing et al.'s statement: A problem with a linear solution is no more complex than a classical planning problem.

Proof Sketch. Assume that the solution to a temporal planning problem is indeed a linear scheduling. As every action follows each other in a strict sequence, every action may only depend on the situation created by all its preceding actions. Since this is the case, it is sufficient to have a single effection point per operator instance.

Again, since no effect from another action can interfere with the current action, all of its conditions need to be either self-satisfied or provided by preceding actions. It is thus acceptable to require conditions hold over the whole duration of an action.

The resulting operators with only `OVER ALL` conditions and `AT END` effects are simply the temporal formulation of a classical planning problem.

□

Definition 4. *A temporally planning problem with a* linear solution *is called* **temporally simple**. *If all solutions to a problem require overlapping scheduling of actions, the problem is called* **temporally expressive**.

A domain, for which only temporally simple problems may be specified, is temporally simple. A domain, for which problems may be specified, that are not temporally simple, is **temporally expressive**.

(definition rephrased from [1]).

2.4 DEP is not Temporally Simple

In their succession paper, Cushing et al. state *that the planners winning temporal planning competition, which are based on decision epoch planning, are incomplete for any subset of PDDL 2.1.3 that is able to model domains and problems whose solutions require concurrency* [3, page 8]. This argument is somewhat misleading, as the concept of a subset here relates only to the particular set of language restrictions defined in their original paper [1].

In PDDL, conditions and effects are tied to an action's duration. It particular, only the following alternatives are possible:

$C \subseteq \{s, o, e\}$ indicates, that conditions need either be met at the start (s), over (o) the complete duration or at the end (e) of an action.
$A \subseteq \{s, e\}$ indicates, that effects can either happen at the start (s) or at the end (e) of an action.

By categorising conditions and effects, it is possible to divide planning domains by the language subset they use. We define such a subset of the domain definition language by symbols of the form L_A^C, where C and A are defined as above.

We determine this language subset by considering all operators in turn and filling the C and A sets. If an operator contains an effect at one of the above locations s or e, we add the corresponding item to A. If an operator requires a condition to hold at one of the above interval specifications s, o or e, we add the corresponding item to O. In this way, be obtain a specification of used features, i.e. the relative locations of the conditions and effects of all operators.

Categorized as such, Cushing et al. prove, that the languages L_s, L_e, L_s^s, L_s^o, L_e^o, and L_e^e lack the ability to express required concurrency. Yet, using only these language subsets is not sufficiently fine-grained to properly categorize DEP: While any problem formulated in a temporally simple language is really temporally simple and can be solved by DEPs, the reverse direction does not hold: There are problems in temporally expressive language subsets that can be solved by decision epoch planners, which we will show by example.

Our example problem uses a relatively simple problem that is slightly related to the DWR-domain from [9]. Consider three robots (r_1, r_2, r_3) that are situated on three different platforms (l_1, l_2, l_3) as depicted in figure 2. There is only a single operator $move(?robot, ?from, ?to)$ that moves a robot $?robot$ from platform $?from$ to a platform $?to$. The only restriction in the presented world is, that a

robot cannot move onto a platform, if it is still occupied by another robot. A platform may only be occupied by a single robot at any point in time. To keep things simple, though, we will assume, that the robots have a sufficiently advanced path finding and collision avoidance system and thus multiple robots may be on the move at the same time, even between the same platforms. The robot domain may be formulated in PDDL using the language $L^{s,e}_{s,e}$, which is temporally expressive (see figure 1).

```
(:durative-action move
    :parameters (?r - robot
        ?from - location ?to - location
    )
    :duration (= ?duration 1)
    :condition (and
        (at start (at ?r ?from))
        (at end (free ?to))
    )
    :effect (and
        (at start (not (at ?r ?from)))
        (at start (free ?from))
        (at end (at ?r ?to))
        (at end (not (free ?to)))
    )
)
```

Fig. 1. Robot domain in PDDL

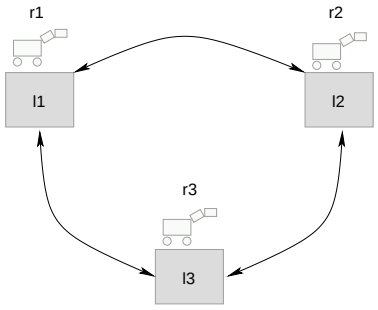

Fig. 2. Sample problem in the robot domain

Assume, that our single goal is to get robot r_1 to occupy platform l_2. For this to work, r_2 — which occupies the target platform initially — must leave

the platform. But since r_2 can only move directly to another platform, r_1 and r_2 need to exchange places simultaneously. Most importantly, r_2 has to leave platform l_2 at least some time *before* r_1 reaches it.

There is a simple (the simplest in terms of execution time) solution to this problem. The solution plan schedules two instances of the *move*() operator at time t_0, i.e. directly at the start of the planning run. Scheduling $move(r_1, l_1, l_2)@t_0$[4] and $move(r_2, l_2, l_1)@t_0$ with overlap yields the intended result.

Note, that to find this optimal solution, actions need only to be scheduled at decision epochs and thus the problem is DEP-solvable. Still, the problem is not temporally simple, as the robots' movements are required to overlap to obtain a valid solution plan. The domain by itself is a subset of a temporally expressive

This simple example shows, that the two problem classes *temporally simple* and *DEP-solvable* are not equivalent. The statement that DEP is not complete for any temporally expressive subset of PDDL is correct only if one restricts itself to Cushing et al.'s language categorisation. If one allows more general concept of language categorisation, the assertion does not hold any more.

Proof Sketch. DEP can solve all problems that are temporally simple. A temporally simple problem has at least one — potentially non-optimal — solution that is a linear scheduling. In a linear scheduling every action only has to be placed in direct succession of its predecessor and thus at no other point than a decision epoch.

Yet, by the example above, there exist problems that are not temporally simple, but can be solved by DEP. By definition, a problem instance by itself is a language subset of PDDL. The robot problem is a subset of a temporally expressive language class. Therefore, the assertion that DEP cannot solve problems in any subset of a temporally expressive language class does not hold.

□

DEP is an incomplete approach to solve temporal planning problems that pushes only somewhat beyond the boundary of temporal simplicity. It is not possible to completely capture the possibilities of DEP only using simple language classes based on effect/condition positions.

3 Handling Low Degrees of Temporal Simplicity

By considering *temporally simple* and *temporally expressive* as the extreme points on a scale, we can define a fine-grained partition of the middle ground between them. We then show how DEP can solve problems falling in this middle ground and thus how it can have some limited degree of temporal expressiveness.

[4] We note solution plans for temporal problems with $operator(a, \ldots, z)@t$, when the operator instance $operator(a, \ldots, z)$ needs to start at time t.

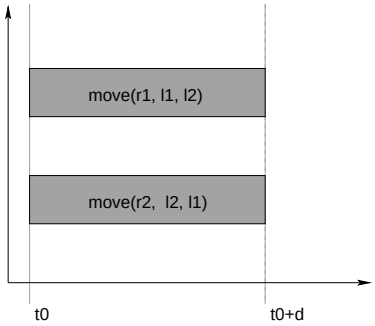

Fig. 3. Presumed solution for the Sample Robot Domain

3.1 Action Combination

The concept of "degrees of temporal simplicity" gives rise to another approach to solve temporal planning problems. Remember, that in PDDL effects of actions still happen at discrete points in time, namely at effection points. Consider the following example.

Figure 4 shows an abstract solution plan for a planning problem. Assume, that the overlapping of op_1 and op_2 is required and thus the planning problem is temporally expressive.

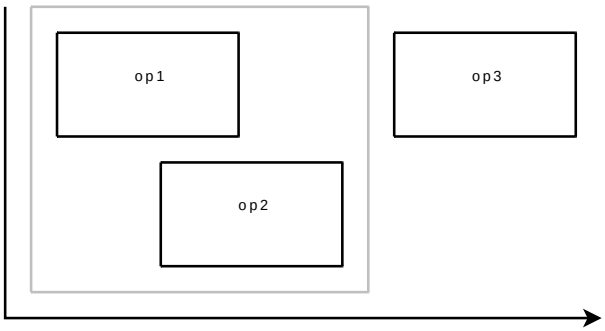

Fig. 4. Action Combination Example

To give an understanding of our approach, assume that the actions op_1 and op_2 are replaced by a single, "larger" operator represented by the grey rectangle. In terms of its conditions and effects, this new "macro operator" shall be indistinguishable from the displayed combination of the two operators op_1 and op_2, i.e. the macro operator is an equivalent replacement. After the replacement, only

the macro operator and op_3 remain in the solution plan. This new arrangement of operators is a linear scheduling and no more overlapping takes place.

The described observation leads to an interesting result: If it is possible to combine overlapping actions into "macro actions" that exhibit the same result as if the individual actions were scheduled, we are able to transform a temporally expressive problem into a temporally simple one. The trade-offs are an increased set of operators and —again— the potential loss of optimality.

The idea is not new: Macro operator generation is a common method to reduce complexity: For planning, an early example is presented by Korf [10]. Recent approaches include the following: Case based techniques [11] analyse prior solutions to find commonly used operator combinations. Botea at al. ([12], [13]) generate abstract components (i.e. macro operators) from analysis of the static problem graph of classical planning problems. To our knowledge, no application of macro operator generation exists for the domain of temporal planning.

3.2 *n*-ary Simple Problems

As temporal planning is truly more complex than classical planning, we cannot simply generate all possible macro operations, run a classical planner on the temporal planning problem and thus completely reduce temporal planning to classical planning. As one expects, the number of possible macro operations grows exponentially with the maximum number of actions in the desired solution plan.

Proof Sketch. To combine a maximum number of m operators, it is sufficient to schedule their start and end times independently into a sequence.

As per PDDL's language definition, there are at most two effection points per operator (`AT START` and `AT END`). Without loss of generality, we assume that there are exactly two effection points per operator, i.e. $2 \times m$.

Since operator durations may be zero, positioning effection points is equivalent to finding a partitioning x_i, such that $\sum_i x_i = 2 \times m$. The number of these partitionings may be —for example— calculated with Rademacher's convergent series [14] and is known to grow exponentially.

The analysis above ignores the constraints between the various effection points. The fact that `AT END` effects cannot happen before the `AT START` effects of a single action and the additional constraints introduced by the differing durations of the various actions are ignored in the analysis above. Considering these constraints would introduce an exponential reduction factor, but that grows much less than the number of partitionings.

Consequently, the number of possible schedulings of m actions also grows exponentially in m.

□

Table 1 shows the number of macro operators generated for the robot domain as mentioned above. Note that, while exponential growth of the number of

generated action tuples cannot be prevented, many generated action tuples cannot be scheduled, because their conditions and effects are contradictory. Pruning these invalid action tuples provides a significant reduction in the total number of macro operators.

tuple size	number of macro operations
2	26
3	818
4	47394

Table 1. Number of generated schedulings for the robot domain

Note also, that the solution to the robot problem requires only the overlapping of two actions and thus the generation of action combinations of more than two actions is not required.

Indeed, one could assume, that — since *all* problems presented in recent international planning competitions (IPC) have been temporally simple — problems that require the parallel scheduling of two actions are already rare. Consequently, domains that require the combination of a tuple of three or even four operators are even less likely to occur in practice. We thus define the concept of *n-ary temporally simple problems*:

Definition 5. *A planning problem is* ***n*-ary (temporally) simple***, if there is a solution plan that requires only the temporal overlapping of at most n actions.*

This definition allows recasting the definition of **temporally simplicity** as used by [1]. A planning problem is temporally simple, if it is *unary* temporally simple. The example robot problem is binary temporally simple, as it requires the overlapping of at most two actions.

4 Evaluation and Future Work

We have built a prototype to test empirically, if macro operator generation is a feasible approach to further push the temporal planning limits.

The prototype generates *operator combinations* from the ungrounded domains rather than *action combinations* using only grounded operators with no variables. This was done to improve the re-usability of the generated target domains. Unfortunately, this design choice also significantly increases the complexity of both the required transformation algorithms and the transformed domains.

The abstract transformation step is very time consuming, owing to the fact that complex verifications and checks has to be made to guarantee a complete and correct result. Given a set of source operators, every possible relative positioning of these operators have to be considered. Additionally, the conditions and effects of the generated macro operator depend on the actual values of the

source operators' parameters. Depending on the structure of the source operators, certain parameter combinations may force us to generate multiple macro operators with differing condition and effect sets. (see [15] for a full discussion).

Due to the complexity of the abstract transformation, the generated macro operators may become quite complex. In some cases, the generated domains even caused serious trouble for the planners used for verification, as problems with the PDDL parsers of particular planners are not uncommon. For example, LPG [16] often rejected syntactically valid domains or the program simply crashed when trying to parse some of the transformed domains. These experiences lead to the conclusion, that a possible re-evaluation should be tried using only grounded operator tuples.

Still, despite the technical problems, a set of benchmark runs was completed using artificially generated sample problems and domains. After the operator combination step, 8% of the transformed domains that previously could not be solved by DEP, could then be solved by a classical planner.

Results have shown, that pre-generating overlapping schedulings from temporally expressive domains can be used to transform n-ary temporally simple problems into classical planning problems. We have empirically evaluated the process for $n = 2$ and have seen promising results.

Yet, future work is still required. Many of the generated macro operators are already invalid and can be detected by finding condition and effect conflicts. Additionally, many operator combinations violate domain invariants and thus can never be instantiated. Use of an invariant detection mechanism can thus further reduce the amount of generated macro operators. Additionally, it has to be evaluated, if performing the combination step at the level of grounded actions yields a better performing approach.

Acknowledgements We would like to thank Michael Mendler for providing useful insights from a more theoretical perspective. The work is supported by the Klaus Tschira Foundation.

References

1. Cushing, W., Kambhampati, S., Mausam, Weld, D.S.: When is Temporal Planning Really Temporal? International Joint Conferences on Artificial Intelligence (2007) 1852–1859
2. Fox, M., Long, D.: PDDL 2.1: An extension to PDDL for expressing temporal planning domains. Journal of Artificial Intelligence Research **20**(2003) 61–124
3. Cushing, W., Kambhampati, S., Talamadupula, K., Weld, D.S., Mausam: Evaluating Temporal Planning Domains. International Conference on Automated Planning and Scheduling (ICAPS) (2007) 105–112
4. Edelkamp, S., Hoffman, J.: PDDL 2.2: The Language for the Classical Part of the 4th International Planning Competition. Technical Report 195, Albert-Ludwigs-Universität Freiburg, Institut für Informatik (2004)
5. Hoffmann, J., Nebel, B.: The FF planning system: Fast plan generation through heuristic search. Journal of Artificial Intelligence Research **14** (2001) 253–302

6. Helmert, M.: The Fast Downward planning system. Journal of Artificial Intelligence Research **26** (2006) 191–246
7. Penberthy, J.: Planning with continuous change. PhD thesis, University of Washington (1993)
8. Vidal, V., Geffner, H.: CPT: An Optimal Temporal POCL Planner based on Constraint Programming. Artificial Intelligence **170**(3) (2006) 298–335
9. Ghallab, M., Nau, D., Paolo, T.: Automated Planning – Theory and Practice. Morgan Kaufmann Publishers (May 2004)
10. Korf, R.: Planning as search: a quantitative approach. In: Artificial Intelligence. Volume 33. (1987) 65–88
11. Kambhampati, S.: Supporting Flexible Plan Reuse. Machine Learning Methods for Planning (1993) 397–434
12. Botea, A., Müller, M., Schaeffer, J.: Using Component Abstraction for Automatic Generation of Macro-Actions. International Conference on Automated Planning and Scheduling (2004) 181–190
13. Botea, A.: Using abstraction for heuristic search and planning. In: Proceedings of the 5th International Symposium on Abstraction, Reformulation and Approximation, London, UK, Springer-Verlag (2002) 326–327
14. Rademacher, H.: Collected Papers of Hans Rademacher. Volume 2. MIT Press (1997)
15. Wullinger, P.: Transformation of temporally expressive into temporally simple planning problems. Master's thesis, Otto-Friedrich-University Bamberg (September 2007)
16. Gerevini, A., Serina, I.: LPG: a Planner based on Local Search for Planning Graphs. In: Sixth International Conference on Artificial Intelligence Planning and Scheduling (AIPS'02), Toulouse, France, AAAI Press (2002) 12–22