Merge and Shrink Abstractions for Temporal Planning

Martim Brandao,[1] Amanda Coles,[1] Andrew Coles,[1] Jörg Hoffmann[2]
[1]King's College London, UK
[2]Saarland University, Germany
{martim.brandao, amanda.coles, andrew.coles}@kcl.ac.uk, hoffmann@cs.uni-saarland.de

Abstract

Temporal planning is a hard problem that requires good heuristic and memoization strategies to solve efficiently. Merge-and-shrink abstractions have been shown to serve as effective heuristics for classical planning, but they have not yet been applied to temporal planning. Currently, it is still unclear how to implement merge-and-shrink in the temporal domain and how effective the method is in this setting. In this paper we propose a method to compute merge-and-shrink abstractions for temporal planning, applicable to both partial- and total-order temporal planners. The method relies on pre-computing heuristics as formulas of temporal variables that are evaluated at search time, and it allows to use standard shrinking strategies and label reduction. Compared to state-of-the-art Relaxed Planning Graph heuristics, we show that the method leads to improvements in coverage, computation time, and number of explored nodes to solve optimal problems, as well as leading to improvements in unsolvability-proving of problems with deadlines.

Introduction

Temporal planning is a hard problem with applications in logistics, manufacturing and other real-world problems. In contrast with classical planning, temporal planning considers durative actions with preconditions and effects at multiple points, and which can be executed concurrently. Due to this added flexibility, temporal planning problems often have larger state spaces. Indeed whilst classical planning is PSPACE hard (Bylander 1994) temporal planning is EXPSPACE hard (Rintanen 2007). Thus the use of plan equivalency and memoization strategies, as well as tight heuristics, is crucial for efficient temporal planning. In this paper we adapt the work on merge-and-shrink abstraction heuristics (Helmert et al. 2007; Nissim, Hoffmann, and Helmert 2011; Fan, Holte, and Müller 2018) into PDDL temporal planning, and use these to improve the efficiency of two families of temporal planners: partial-order and total-order planners.

Heuristics are crucial to improve the efficiency of planners and the quality of plans, both in sorting the open list to guide search towards promising states but also, as we focus on here, in pruning states from which a goal cannot be reached. In optimal temporal problems which optimize the makespan of the plan (i.e. total time from start to goal), heuristics can be used to estimate when the goal will be reached, and thus prune states from which a lower metric value cannot be reached. In problems with deadlines, admissible heuristics can be used to obtain a lower bound of when each fact is achieved, and thus prune states that cannot meet a deadline. Most current state-of-the-art temporal planners use heuristics based on the Temporal Relaxed Planning Graph (TRPG) (Do and Kambhampati 2003; Coles et al. 2010; Haslum 2009), though the context-enhanced additive heuristic (Eyerich, Mattmüller, and Röger 2009) and landmark-based pruning strategies (Marzal, Sebastia, and Onaindia 2014) have also been applied to temporal planning.

In this paper we build on the work of temporal mutex analysis (Bernardini and Smith 2011) and (classical) merge-and-shrink abstractions (Helmert et al. 2007; Nissim, Hoffmann, and Helmert 2011; Fan, Holte, and Müller 2018) to generate merge-and-shrink abstraction heuristics for temporal planning. These abstractions are built by successively taking the product of (single-variable) transition graphs and then shrinking the state-space by "collapsing" abstract states based on their lower-bound estimates of goal-makespan. Due to the temporal nature of the problem, as we will explain later on, such estimates also depend on temporal variables associated with the current state, and thus the computed lower bounds cannot be pre-computed in the form of numeric values—but in the form of algebraic expressions of temporal variables. Similarly, abstract states are collapsed not based on their heuristic values as in classical planning but based on expression equivalency. Even though this complexity results in higher computation times for the heuristic, all expensive computation is done during the pre-computation stage, while during search the heuristic makespan estimates can be quickly obtained by evaluating the pre-computed expressions. We evaluate the approach on makespan-optimization temporal problems and temporal problems with deadlines. Importantly, our approach is generally applicable to both partial-order and total-order temporal planners—and our experiments show that merge-and-shrink heuristics are beneficial in both settings.

The contributions of the paper are: 1) the proposal of a merge-and-shrink method for temporal planners; 2) a new merging strategy that is effective in temporal planning; and 3) the demonstration of the effectiveness of merge-and-shrink in both partial- and total-order temporal planners, in both optimal problems and problems with deadlines.

Related Work

Many temporal planners rely on a variant of a Temporal Relaxed Planning Graph (TRPG) (Do and Kambhampati 2003; Long and Fox 2003a) as a heuristic. The TRPG is a graph that explores the full state space in (timestamped) layers while ignoring delete effects—thus providing admissible estimates of makespan. Another approach to obtaining admissible estimates is Haslum's cost-based heuristic (Haslum 2009), which finds sequential action sets to establish bounds on the amount of concurrency and thus makespan. Other related work on (non-admissible) temporal planning heuristics is that of Marzal et al. (Marzal, Sebastia, and Onaindia 2014), which extracts landmarks from temporal problems and uses them in a non-admissible heuristic during search; as well as Eyerich's work on the context-enhanced additive heuristic (Eyerich, Mattmüller, and Röger 2009).

The method proposed in this paper is based on the merge-and-shrink literature (Helmert et al. 2007; Nissim, Hoffmann, and Helmert 2011; Helmert 2006; Fan, Holte, and Müller 2018), which has focused on classical planning problems so far. Our method also builds on recent advances in temporal planning, particularly the work on partial-order search-based planners (Coles et al. 2010; Coles and Coles 2016; Benton, Coles, and Coles 2012).

Background

Temporal Planning

Definition 1 *A temporal SAS$^+$ planning task is a tuple $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, G \rangle$, where:*

- *$\mathcal{V} = \{v_1, ..., v_n\}$ is a set of state variables. Each variable $v \in \mathcal{V}$ is associated with a domain $\mathcal{D}_v$, which includes an "undefined" value $\perp$. The total variable state space is therefore $\mathcal{S}_{\mathcal{V}} = \mathcal{D}_{v_1} \times ... \times \mathcal{D}_{v_n}$.*
- *$\mathcal{O}$ is a finite set of durative actions, where a durative action $a \in \mathcal{O}$ is a tuple $\langle pre_{\vdash}, pre_{\leftrightarrow}, pre_{\dashv}, eff_{\vdash}, eff_{\dashv}, \delta^-, \delta^+ \rangle$, comprising the sets of pre-conditions and effects at the start ($\vdash$) and end ($\dashv$) of the action and a set of invariant conditions ($\leftrightarrow$) which must hold throughout the execution of the action. The bounds $\delta^-, \delta^+ \in \mathbb{R}_0^+$ constitute a duration constraint on the action, with $\delta^+ \geq \delta^-$.*
- *Each action can be split into two "snap" actions $a_{\vdash} = \langle pre_{\vdash}, eff_{\vdash} \rangle$ and $a_{\dashv} = \langle pre_{\dashv}, eff_{\dashv} \rangle$, which need to be sequenced and scheduled such as to respect $pre_{\leftrightarrow}, \delta^-, \delta^+$.*
- *"Compression-safe" actions $\mathcal{O}_c \subseteq \mathcal{O}$ are durative actions where, without loss of completeness, soundness or optimality, the end snap action can be sequenced immediately after the start snap action (Coles et al. 2009). The set of non-compression-safe actions is denoted by $\mathcal{O}_n = \mathcal{O} \backslash \mathcal{O}_c$.*
- *Conditions $pre_{\vdash}, pre_{\dashv}, pre_{\leftrightarrow}$ are partial variable assignments in the form of pairs $\langle v, w \rangle$ of variables v and the values w they must hold in the start, end, or the whole period between the start and the end of the action, respectively. Therefore, an action has a start, end, or invariant condition on v_i if $\exists \langle v, w \rangle \in pre_{\vdash / \dashv / \leftrightarrow}$ such that $v = v_i$.*
- *Effects $eff_{\vdash}, eff_{\dashv}$ are partial variable assignments in the form of pairs $\langle v, w \rangle$ of variables v and the values w they will hold after the execution of the respective snap action. Therefore, an action has a start or end effect on v_i (or "affects" variable v_i at the start/end) if $\exists \langle v, w \rangle \in eff_{\vdash / \dashv}$ such that $v = v_i$.*
- *$s_0 \in \mathcal{S}_{\mathcal{V}}$ is called the initial state*
- *G is a set of preconditions called the "goal" of the task.*

Even though splitting actions into "snap" actions leads to an explosion of the search space, care can be taken so as to alleviate the issue on compression-safe actions—by identifying compression-safe actions and automatically inserting end actions (rather than considering this a search decision) if doing so does not compromise completeness (Coles et al. 2009).

Definition 2 *A temporal planning state $\mathtt{s} = \langle U, P, T \rangle$ is a tuple of:*

- *U: a set of assignments for all variables in $\mathcal{V}$*
- *P: a vector of snap actions which represent the plan to reach $\mathtt{s}$ from s_0. Each of these actions is called a "step" of P*
- *T: a set of temporal constraints over the actions in P, each of the form $t^- \leq t(b) - t(a) \leq t^+$. Here, $t(i)$ is called a "timestamp" and is a variable representing the time at which the snap action with index i in P is scheduled to be executed. So, a and b are indices of actions in P, while $t^-, t^+ \in \mathbb{R}_0^+$ and $t^- \leq t^+$.*

Definition 3 *A (partial) plan associated with a temporal planning state $\mathtt{s}$ is defined by P and an assignment of timestamps $t(1), ...t(m)$ to each of the m actions in P, obtained by solving a Simple Temporal Problem (STP) (Dechter, Meiri, and Pearl 1989) built from the constraints T.*

The STP finds the earliest possible timestamp for each action. In practice, it can be solved by finding the shortest-path within a Simple Temporal Network (STN) representation of the STP (Dechter, Meiri, and Pearl 1989).

Definition 4 *A solution is a plan for which all goal preconditions G are satisfied after the final step, all conditions of actions in P are satisfied, and all actions have finished executing. A solution is optimal if its "makespan" $max_{i \in 1,...,m} t(i)$ is minimal.*

Definition 5 *Variable annotations (Coles et al. 2010) are stored for each temporal planning state $\mathtt{s} = \langle U, P, T \rangle$ during planning, and consist of:*

- *$V^{eff}(\mathtt{s}, v)$: the index of the step (action) in P that most recently had an effect upon variable v;*
- *$VP(\mathtt{s}, v)$, where each $\langle i, d \rangle \in VP(\mathtt{s}, v)$ comprises the index i of a step in P that referred to the variable v since the last effect on v; and $d \in \{0, \epsilon\}$, encoding the temporal separation needed to respect the PDDL mutual exclusion semantics. A step depends on v if it either has a*

precondition on v; an effect needing an input value of v; or is the start of an action with a duration depending on v.

We note that while the state annotations introduced in (Coles et al. 2010) were written in a propositional PDDL formalism, here they refer to SAS$^+$ variables.

Definition 6 *Variable "use" and "chg" timestamps (Coles and Coles 2016) are timestamps from which a variable can be used or changed, given the preconditions and effects on that variable:*

- $use(\mathtt{s}, v) = t(V^{eff}(\mathtt{s}, v))$,
- $chg(\mathtt{s}, v) = max\{t(i) + d | \langle i, d\rangle \in VP(\mathtt{s}, v)\}$.

On total-order temporal planners[1], *use* and *chg* timestamps are equal to the timestamp of the last action in the plan (or timestamp of the state) in all variables. However, that is not the case in planners that only partially order solution plans—in which case *use* can be derived from the last action to add a fact, and *chg* from the last action to add or condition on the fact (Coles et al. 2010). In this paper we consider the most general case, of separate *use* and *chg* timestamps for each variable, in order to make the methods applicable to both kinds of temporal planners.

Merge-and-Shrink

Definition 7 *A transition graph (Helmert et al. 2007) is a tuple $\Theta = \langle V, S, L, A, s_0, S_G\rangle$, where $V \subseteq \mathcal{V}$ is a subset of the variables in $\mathcal{V}$, S is a state space of variable assignments $S = S_V \subseteq \mathcal{S}_{\mathcal{V}}$, $L \subseteq \mathcal{O}$ is a finite set of transition labels, $A \subseteq S \times L \times S$ is a set of labeled transitions, $s_0 \in S$ is the initial state in Θ, and $S_G \subseteq S$ is the set of goal states in Θ. The graph has a transition labelled $l \in L \subseteq \mathcal{O}$ from $s \in S$ to $d \in S$ if action l is applicable in s and applying the action results in state d.*

We denote the transition graph associated with a planning task Π by $\Theta(\Pi)$. A transition graph is an abstraction of an original planning problem, and thus can be used to obtain a heuristic that helps solve the problem. Merge-and-shrink is an approach to build a small-yet-informative transition graph, which works by incrementally combining ("merging") and simplifying ("shrinking") smaller transition graphs associated with different variables until a single transition graph remains.

Merging consists of taking the synchronized product of two transition graphs $\Theta \otimes \Theta' = \langle V \cup V', S \times S', L, A \otimes A', \langle s_0, s'_0\rangle, S_G \times S'_G\rangle$, where $A \otimes A'$ is such that a transition exists from $\langle s, s'\rangle$ to $\langle d, d'\rangle$ via label l iff $\langle s, l, d\rangle \in A$ and $\langle s', l, d'\rangle \in A'$.

Shrinking consists of creating an abstract transition graph $\alpha(\Theta) := \langle V, \alpha(S), L, \{\langle \alpha(s), l, \alpha(d)\rangle | \langle s, l, d\rangle \in A\}, \alpha(s_0), \alpha(S_G)\rangle$, where α is a function on S that maps multiple states into a single abstract state.

For merge-and-shrink to be computationally efficient, other operations are also conducted before or after each shrink step: label reduction, and pruning. Label reduction decreases the number of transitions by replacing groups of transitions by a single abstract transition. It consists of creating an abstract transition graph $\tau(\Theta) := \langle V, S, \tau(L), \{\langle s, \tau(l), d\rangle | \langle s, l, d\rangle \in A\}, s_0, S_G\rangle$, where τ is a label mapping function which maps multiple (same-cost/duration) labels into a single one. Finally, "pruning" consists of removing states from a transition graph that are not reachable from the initial state, or that are not connected to the goal.

A generic merge-and-shrink abstraction-building algorithm is shown in Algorithm 1.

Algorithm 1: Generic merge-and-shrink algorithm

```
  Data: $X \leftarrow \{$ all single-variable transition graphs $\}$
  Result: final transition graph in $X$
1 while $|X| > 1$ do
2     $\Theta_1, \Theta_2 \leftarrow$ PickTwoGraphsToMerge($X$);
3     LabelReduction($\Theta_1, \Theta_2$);
4     $\Theta_1 \leftarrow \alpha(\Theta_1)$; $\Theta_2 \leftarrow \alpha(\Theta_2)$;        #Shrink
5     $\Theta_{\text{merged}} \leftarrow \Theta_1 \otimes \Theta_2$;                  #Merge
6     Pruning($\Theta_{\text{merged}}$);
7     $X \leftarrow X \cup \Theta_{\text{merged}} \setminus \{\Theta_1, \Theta_2\}$;
```

Temporal Merge-and-Shrink

We now introduce merge-and-shrink for temporal planning. For simplicity, we focus on the case of non-numeric temporal domains without "required concurrency" (Cushing et al. 2007), i.e. where all actions are compression-safe $\mathcal{O}_n = \emptyset$. In the last section we discuss what an extension to domains with non-compression-safe actions would entail.

Challenges

Concurrency in temporal planning introduces a challenge when building and using merge-and-shrink heuristics and abstractions. While in classical planning a heuristic can be computed taking shortest-paths (e.g. from (Dijkstra 1959)) on the abstract transition graph, in temporal planning the makespan of a sequence of transitions is not necessarily equal to the sum of the transitions' durations. The makespan of a sequence of transitions from s to a goal s_G depends not only on transition durations but also on the times at which each of the transitions from s to s_G can start. Additionally, it depends on the times of previous actions that have already been added before s was reached (i.e. the times of actions in P). This will complicate the pre-computation of a heuristic function—since we will now need to pre-compute a mapping not from states to numeric values, but from states to *functions* of *use* and *chg* variables.

We will now describe how to compute the starting time of actions as functions of *use* and *chg* variables, and use this knowledge to describe *use/chg* propagation and the computation of makespan and heuristic functions.

[1]Planners that enforce a total-ordering on actions, i.e. neighbors of a search state will always add an action to the end of the plan.

Action Scheduling and *use/chg* Propagation

At a given stage during search we arrive at a search state s, and we want to compute a lower bound on the time a goal state can be achieved from s using abstraction Θ. First we project s to $s \in S$, then we compute when each action can be executed, successively through multiple transitions in Θ until a goal state is reached.

The time at which an action a can execute in an abstract state $s \in S$ depends on the *use/chg* times of the variables which the action affects and depends on. If a is applied in s then it can start no earlier than:

$$\begin{aligned} &\mathit{start\_min}_\Theta(a, s) = \max\{ \\ &\max_{\langle v_i,w_i\rangle \in \mathit{pre}_\vdash(a) : v_i \in V} \mathit{use}(s, v_i) + \epsilon, \\ &\max_{\langle v_i,w_i\rangle \in (\mathit{pre}_\leftrightarrow(a) \setminus \mathit{eff}_\vdash(a)) : v_i \in V} \mathit{use}(s, v_i), \\ &\max_{\langle v_i,w_i\rangle \in \mathit{eff}_\vdash(a) : v_i \in V} \mathit{chg}(s, v_i) + \epsilon, \\ &\max_{\langle v_i,w_i\rangle \in \mathit{eff}_\dashv(a) : v_i \in V} \mathit{chg}(s, v_i) - \delta^- + \epsilon\}. \end{aligned}$$

Note that since *use* and *chg* are not values, $\mathit{start\_min}_\Theta(a, s)$ is also not a value: it is a formula. Furthermore, only conditions and effects on variables that are part of the abstraction are considered (due to constraint $v_i \in V$), thus rendering $\mathit{start\_min}_\Theta$ a lower bound on action start time. Applying action a then leads to a change in the times at which the variables that are affected/depended by the action can be changed and used. Therefore, the *use* and *chg* variables in the next state s' are determined by those in state s, and the preconditions and effects of a:

$$\mathit{use}(s', v_i) = \mathit{propagate\_use}_\Theta(a, s),$$

$$\mathit{chg}(s', v_i) = \mathit{propagate\_chg}_\Theta(a, s),$$

where the "propagate" functions are such that:

- if a does not condition on or affect v_i:
$$\mathit{use}(s', v_i) = \mathit{use}(s, v_i)$$
$$\mathit{chg}(s', v_i) = \mathit{chg}(s, v_i)$$
- if a changes the value of v_i at its end, then regardless of any other effects/conditions on v_i:
$$\mathit{use}(s', v_i) = \mathit{start\_min}_\Theta(a, s) + \delta^-$$
$$\mathit{chg}(s', v_i) = \mathit{start\_min}_\Theta(a, s) + \delta^-$$
- if a changes the value of v_i at its start (but *not* the end), and then has an 'end' and an 'invariant' condition on v_i:
$$\mathit{use}(s', v_i) = \mathit{start\_min}_\Theta(a, s)$$
$$\mathit{chg}(s', v_i) = \mathit{start\_min}_\Theta(a, s) + \delta^-$$
- if a changes the value of v_i at its start (but *not* the end), and then has an 'invariant' condition on v_i:
$$\mathit{use}(s', v_i) = \mathit{start\_min}_\Theta(a, s)$$
$$\mathit{chg}(s', v_i) = \mathit{start\_min}_\Theta(a, s) + \delta^- - \epsilon$$
- if a changes the value of v_i at its start (but *not* the end), and has neither an 'invariant' or 'at end' condition on v_i:
$$\mathit{use}(s', v_i) = \mathit{start\_min}_\Theta(a, s)$$
$$\mathit{chg}(s', v_i) = \mathit{start\_min}_\Theta(a, s)$$
- if a never changes the value of v_i, but has an 'end' and an 'invariant' condition on v_i:
$$\mathit{use}(s', v_i) = \mathit{use}(s, v_i)$$
$$\mathit{chg}(s', v_i) = \max\{\mathit{chg}(s, v_i), \mathit{start\_min}_\Theta(a, s) + \delta^-\}$$
- if a never changes the value of v_i, but has an 'invariant' condition on v_i:
$$\mathit{use}(s', v_i) = \mathit{use}(s, v_i)$$
$$\mathit{chg}(s', v_i) = \max\{\mathit{chg}(s, v_i), \mathit{start\_min}_\Theta(a, s) + \delta^- - \epsilon\}$$
- if a never changes the value of v_i, but has an 'at start' condition on v_i:
$$\mathit{use}(s', v_i) = \mathit{use}(s, v_i)$$
$$\mathit{chg}(s', v_i) = \max\{\mathit{chg}(s, v_i), \mathit{start\_min}_\Theta(a, s)\}$$

If a sequence of actions $a_1, ..., a_k$ is executed from state $s \in S$, leading to the sequence of states $s_1, ..., s_k$, then the makespan of this sequence is given by:

$$\max_{i \in 1,...,|V|} \mathit{use}(s_k, v_i).$$

Temporal M&S Heuristic

Definition 8 *A temporal transition graph Θ of a problem satisfying $\mathcal{O}_c = \mathcal{O}$ is a transition graph where: a transition is labeled $l \in L \subseteq \mathcal{O}$ from $s \in S$ to $d \in S$ if (i) $l_\vdash$ is applicable in s, (ii) applying $l_\vdash$ results in state s' that satisfies $l_\leftrightarrow$, and (iii) $l_\dashv$ is applicable in s' and applying it results in state d.*

Definition 9 *A heuristic is a function h^Θ, associated with the transition graph Θ, which assigns to each state $s \in S$ the makespan of the lowest-makespan path in Θ, from s to any goal state $s_G \in S_G$.*

Please note that the merge-and-shrink literature (e.g. (Helmert et al. 2007; Nissim, Hoffmann, and Helmert 2011)) define a heuristic in terms of cost values, while here, for the purposes of temporal planning, we define it as minimizing (timestamp) makespan.

As we have seen in the previous section, in general the makespan estimate provided by our heuristic function is precomputed as an expression of *use* and *chg* variables, and only evaluated into a numeric value during search. In the rest of the paper, we will call this expression a "goal-makespan *formula*".

Goal-makespan Formulas

Definition 10 *A goal-makespan formula $tg^\Theta(s)$ for a state $s \in S$ is a formula that expresses the value of $h^\Theta(s)$ in terms of use and chg timestamps.*

For goal states $s_G \in S_G$, $tg^\Theta(s_G) := \max\{\mathit{use}(s_G, v_i) | v_i$ is a goal variable in $\Theta\}$, i.e. the makespan will be at least equal to the time at which all goal variables in that state can be used. For non-goal states, tg^Θ is a formula of nested min and max constraints over the *use* and *chg* variables, to represent the temporal constraints of the different paths that can be taken from s to the goal states. States for which no path to a goal exist are associated with $tg^\Theta = \infty$.

Computing Goal-makespan Formulas We compute the goal-makespan formulas for all states in an abstract transition graph as outlined in Algorithm 2. Basically, the algorithm starts by computing the goal-makespan formulas of goal states and initializing all other formulas to infinity. When it updates the formula of a state, it will then update its neighbors' formulas accordingly by placing the state in an update queue. "MakespanThroughNeighbor()" computes the new goal-makespan formula for a neighbor s as the minimum between its previous makespan estimate, and the formula given by s' plus the required time to complete the transition. Note that the time to complete a transition is a function of the timestamps of precondition variables. The first formulas to be updated are the goal state formulas, and the next ones will be those associated to states that lead to goal states in a single transition. The formula of a single state may be updated multiple times as new paths from its neighbors' to the goal are discovered. The "Simplify()" function makes sure a formula is not redundant and therefore eliminates loops and definitely-longer paths from the goal-makespan formula. Eventually, the goal-makespan formulas for all states will be computed (i.e. no more states will be added to the queue Q) and the algorithm terminates.

Algorithm 2: Pre-computing tg^Θ formulas

```
Data: Abstraction $\Theta$, state space $S$, transitions $A$
Result: Each abstract state $s \in S$ has its formula
        $tg^\Theta(s)$ updated to reflect paths to goals
1  Q ← [] ;
2  foreach state $s \in S$ do
3      if $s$ is a goal state then
4          $tg^\Theta(s) \leftarrow$ Formula( $\max\{use(s, v_i) \mid$
            $v_i$ is a goal variable$\}$) ;
5          append $s$ to $Q$;
6      else
7          $tg^\Theta(s) \leftarrow \infty$;
8  while $Q \neq \emptyset$ do
9      pop $s'$ off the front of $Q$;
10     foreach edge $\langle s, a, s' \rangle \in A'$ do
11         $new\_tg \leftarrow$
             MakespanThroughNeighbor($s, a, s'$);
12         $new\_tg' \leftarrow$ Formula( $\min\{tg^\Theta(s), new\_tg\}$);
13         $new\_tg' \leftarrow$ Simplify($new\_tg'$);
14         if $new\_tg' \neq tg^\Theta(s)$ then
15             $tg^\Theta(s) \leftarrow new\_tg'$;
16             append $s$ to $Q$;
```

The two key functions in this algorithm are "MakespanThroughNeighbor(s, a, s')" and "Simplify(tg)".

"MakespanThroughNeighbor(s, a, s')" computes the goal-makespan formula for a state s, given the formula of state s' and the action a. It does so by replacing $use(v)$ and $chg(v)$ variables in $tg^\Theta(s')$ according to the actions' preconditions and effects. In particular, the functions $propagate\_use_\Theta(a, s)$ and $propagate\_chg_\Theta(a, s)$ can be used to derive a substitution rule as follows:

$$sub(s, a, s') = \bigcup_{v_i} \{use(s', v_i) \mapsto propagate\_use_\Theta(a, s),$$

$$chg(s', v_i) \mapsto propagate\_chg_\Theta(a, s)\}.$$

meaning that $tg^\Theta(s)$ will be obtained by taking formula $tg^\Theta(s')$ and replacing variable $use(s', v_i)$ with the expression given by $propagate\_use_\Theta(a, s)$—and similarly for chg. Function "MakespanThroughNeighbor" thus reflects the option of reaching a goal from s by applying a (thus reaching s') and taking a path from there.

Finally, the "Simplify(tg)" function in Algorithm 2 manipulates formulas to remove redundancy. First, it applies a canonical form to the formula where all constant additions are moved to the leaves of the formula (e.g. $1 + \max\{use(v_1), use(v_2)\}$ becomes $\max\{use(v_1) + 1, use(v_2) + 1\}$). Then, it removes terms whose values are dominated by other terms (e.g. $\max\{use(v_1), use(v_1) + 3\}$ becomes $use(v_1) + 3$). Formula domination is established by an inequality definition over formulas, as defined below.

Ordering Goal-makespan Formulas In order to facilitate the ordering of formulas, we represent goal-makespan formulas as expressions of the type $tg := max\{u_1.use(s, v_i) + t^u_1, ..., u_{|V|}.use(s, v_{|V|}) + t^u_{|V|}, c_1.chg(s, v_i) + t^c_1, ..., c_{|V|}.chg(s, v_{|V|}) + t^c_{|V|}\}$, where $u_i \in \{0, 1\}$ and $c_i \in \{0, 1\}$ are indicator variables, and t^u_i, t^c_i are constants[2]. A formula can therefore be conveniently implemented as a tuple of vectors $\langle u, t^u, c, t^c \rangle$.

Definition 11 *One goal-makespan formula tg is dominated by (definitely lower or equal than) another tg', i.e. $tg \leq tg'$, if one of the following conditions holds:*

- $max_i(max(t^u_i, t^c_i)) \leq \infty \wedge max_i(max(t^{u'}_i, t^{c'}_i)) = \infty$
- $\forall_i\{u_i = 0 \wedge c_i = 0 \wedge u'_i = 0 \wedge c'_i = 0\} \wedge max_i(max(t^u_i, t^c_i)) \leq max_i(max(t^{u'}_i, t^{c'}_i))$
- $\forall_i \ \{u_i \leq u'_i \wedge c_i \leq c'_i \wedge t^u_i \leq t^{u'}_i \wedge t^u_i \leq t^{c'}_i \wedge t^c_i \leq t^{c'}_i\}$

Evaluating Goal-makespan Formulas During planning, for each temporal planning state $\mathtt{s} = \langle U, P, T \rangle$ that is evaluated we obtain the corresponding *use/chg* timestamps $use(\mathtt{s}, v_i)$, $chg(\mathtt{s}, v_i)$, and the abstract state $s \in S$. From the abstract state we obtain the pre-computed goal-makespan formula $tg^\Theta(s)$ and evaluate it using the values of $use(\mathtt{s}, v_i)$ and $chg(\mathtt{s}, v_i)$:

$$h^\Theta(s) = tg^\Theta(s)\Big|_{\substack{use(s,v_i)=use(\mathtt{s},v_i)\forall_i \\ chg(s,v_i)=chg(\mathtt{s},v_i)\forall_i}}.$$

Admissibility The temporal merge-and-shrink heuristic, computed by evaluating goal-makespan formulas, is admissible. To see this, note that (as in classical planning) every path in the full transition graph $\Theta(\Pi)$ is also a path in an abstract transition graph Θ, since merging and shrinking operations keep all transitions. In addition to this, we

[2]These constants arise from action durations along paths to a goal state, which have been discovered through *use/chg* propagation as described previously.

need to prove that goal-makespan formulas provide admissible estimates of makespan in time. We do this in steps. First, we note that *use/chg* propagation is admissible because $start\_min_{\Theta}$ is a lower bound (a max() over a subset of all variables is a lower bound of the max over all variables). Propagation assumes that all actions take their minimum duration δ^{-} to execute, which is again a lower bound. Second, Algorithm 2 propagates *use/chg* variables from goal nodes to all other nodes in the transition graph, and stores the makespan of all non-cyclical paths to goal states as a function of *use/chg* variables. Line 12 makes sure that all non-cyclical ("simplified") paths to goal states are considered since it uses a min() of previous and propagated formulas; and path simplification (line 13) is also admissibility-preserving since it only removes paths that are provably worse. Since goal-makespan formulas are a lower bound of makespan over all non-cyclical paths to the goal, they are also a lower bound of makespan over all possible paths to the goal.

Classical/Temporal M&S Differences To summarize, the main difference between M&S abstractions in classical and compression-safe temporal planning is that precomputing the shortest path to a goal involves precomputing all possible non-cyclical paths to the goal as a function of temporal variables which are evaluated at search time. Transition graphs and abstractions are defined in the same way in classical and temporal planing—only the applicability of actions is different—while all temporal aspects of a problem relevant to makespan are captured by goal-makespan formulas. The only temporal aspect not captured by goal-makespan formulas is the possibility for actions to execute for longer than δ^{-}. Formulas assume all actions execute for their minimum duration, which guarantees admissibility but means that goal-makespan formulas of the full transition graph $\Theta(\Pi)$ are a relaxation of Π (where actions can execute for a duration in the interval $[\delta^{-}, \delta^{+}]$).

Shrinking Algorithms

h-preserving Shrink An h-preserving shrink algorithm groups states that have the same heuristic value into a single abstract state. In temporal planning, states that have the same *goal-makespan formula* are grouped into abstract states.

Bisimulation A bisimulation-based shrinking algorithm uses *bisimulation* (Nissim, Hoffmann, and Helmert 2011) to identify states to aggregate. Two states s, s' are bisimilar if "every transition label leads into equivalent abstract states" from s and s' (Nissim, Hoffmann, and Helmert 2011). The computation of the coarsest bisimulation can be made efficiently (Nissim, Hoffmann, and Helmert 2011) but involves grouping states by heuristic value. In the temporal domain (since values are only accessible at search time) we instead group states by goal-makespan formula equality.

Label Reduction

Label reduction decreases the number of transitions by replacing groups of (same-source, same-destination) transitions by a single abstract transition. In temporal planning, care needs to be taken when comparing the durations of transitions. We implement label reduction as follows. If two transitions associated with actions a_1, a_2, both connecting s to s', are such that:

$$start\_min_{\Theta}(a_1, s) + \delta_1^{-} \leq start\_min_{\Theta}(a_2, s) + \delta_2^{-},$$

and all the variables that the actions affect and depend on belong to V, then we replace the two transitions by a single a_1-labeled transition (and similarly for a_2). The variable condition makes sure the synchronized product works properly, i.e. it avoids transitions being lost when merging new variables that the respective actions depend on or affect, since $start\_min_{\Theta}$ works with projected conditions (i.e. only conditions on variables $v_i \in V$). In the expression above, the inequality is as in Definition 11.

Results

Planner Setup

We implemented the methods proposed in the previous sections in the temporal planner OPTIC (Benton, Coles, and Coles 2012). OPTIC performs search starting from the initial state, applying start/end snap actions whose preconditions are satisfied at each state (and which do not delete the invariants of actions that have started executing but have not yet finished in the state)[3]. When each new state is generated, the planner builds an STN to check the temporal consistency of the plan. If an inconsistency is found then the state is pruned. Otherwise, the search heuristic value of the state is evaluated and it is added to the open list. OPTIC uses as its search heuristic the number of actions in the temporal relaxed plan to reach the goal from the state. In addition to that it uses admissible makespan estimates to prune states where deadlines can no longer be reached, and in the optimal planning case it prunes states whose admissible makespan estimates are higher than the best makespan found so far. In the benchmarks that follow, we compare the results of the planner when the admissible makespan estimates are taken from the merge-and-shrink abstractions, versus when they are taken from the TRPG.

We express deadlines as Timed Initial Literals (TILs) (Hoffmann and Edelkamp 2005). While there is no explicit way to represent deadlines in PDDL, they can be modelled using PDDL2.2 TILs which allow deletion or addition of a fact at a fixed time (e.g. (at 10 (not (can-deliver package1)))). If such a fact is a precondition of any action that adds a given goal fact, then this effectively places a deadline on reaching that goal. This can easily be detected in preprocessing, allowing the planner to identify deadlines on goal facts.

Merge-and-Shrink Strategy

In our experiments we implement merge-and-shrink as in Algorithm 1, using FastDownward's (Helmert 2006) merge-

[3]In this work, as in prior work, OPTIC's search uses the PDDL fact representations of state and actions and the SAS+ representation is used only for our new heuristic, with the annotations "translated" to SAS+ for that purpose. Since the focus of this work is on a SAS+ based heuristic, we focused on that formalism here and wrote in those terms—so as to not duplicate the search written in terms of the traditional PDDL semantics (which can be found in (Coles et al. 2010)).

	Domain	L-hshrink	L-bisim	R500-hshrink	R500-bisim	R5k-hshrink	R5k-bisim
(optimal)	depots	404.52	368.0	1.01	0.85	57.14	70.36
	driverlog	316.19	224.24	0.79	0.8	117.68	289.44
	pipesworld	-	-	0.75	0.94	228.03	7.72
	rovers	158.9	244.14	0.19	0.27	3.19	6.04
	satellite	11.06	18.47	0.16	0.28	0.95	1.36
	zenotravel	237.03	202.38	0.26	0.37	1.87	1.68
(deadlines)	depots	884.99	893.92	3.66	4.46	210.62	261.52
	driverlog	27.82	30.34	2.24	2.18	55.52	45.29
	floortile	-	-	7.12	5.3	353.91	371.04
	logistics	0.08	0.16	0.83	1.48	1.57	4.36
	trucks	17.26	18.33	1.46	1.88	53.15	59.23
	zeno	170.05	170.18	169.39	169.72	167.89	168.09

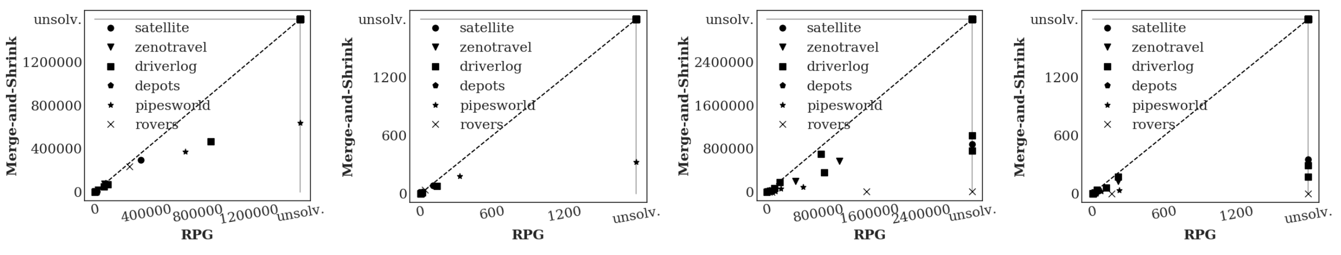
Table 1: Merge-and-shrink pre-computation time (s)

Domain	RPG	L-hshrink	L-bisim	R-hshrink500	R-bisim500	R-hshrink5k	R-bisim5k
depots	2 (11)	0 (6)	0 (7)	2 (11)	2 (11)	2 (11)	2 (11)
driverlog	5 (15)	1 (2)	1 (3)	5 (15)	5 (15)	5 (13)	5 (13)
pipesworld	3 (9)	0 (0)	0 (0)	4 (9)	4 (9)	3 (8)	1 (7)
rovers	1 (19)	1 (19)	1 (19)	1 (19)	1 (19)	1 (19)	1 (19)
satellite	3 (9)	3 (9)	3 (9)	3 (9)	3 (9)	3 (9)	3 (9)
zenotravel	6 (13)	6 (14)	6 (14)	6 (14)	6 (14)	7 (14)	7 (13)
Sum	20 (76)	11 (50)	11 (52)	**21 (77)**	**21 (77)**	21 (74)	19 (72)

Table 2: Optimality coverage (solve coverage), partial-order planner on optimal problems

Domain	RPG	L-hshrink	L-bisim	R-hshrink500	R-bisim500	R-hshrink5k	R-bisim5k
depots	2 (15)	2 (15)	2 (15)	2 (15)	2 (15)	2 (15)	2 (15)
driverlog	6 (16)	8 (16)	8 (16)	8 (16)	8 (16)	9 (16)	9 (16)
pipesworld	6 (10)	7 (10)	7 (10)	6 (10)	6 (10)	6 (10)	6 (10)
rovers	3 (18)	4 (18)	4 (18)	4 (18)	4 (18)	4 (18)	4 (18)
satellite	3 (19)	4 (19)	4 (19)	4 (19)	4 (19)	4 (19)	4 (19)
zenotravel	7 (19)	8 (19)	8 (19)	7 (19)	7 (19)	8 (19)	7 (19)
Sum	27 (97)	**33** (97)	**33** (97)	31 (97)	31 (97)	**33** (97)	32 (97)

Table 3: Optimality coverage (solve coverage), total-order planner on optimal problems



(a) Partial-order #nodes (b) Partial-order comp. time (c) Total-order #nodes (d) Total-order comp. time

Figure 1: Makespan-optimal temporal planning problems, using minimal h-preserving shrink and CGGR-500.

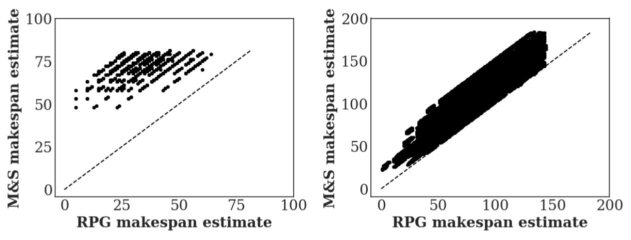


Figure 2: Comparison of the makespan estimates obtained by TRPG vs M&S (total-order planner with CGGR500 and hshrink) on a rovers (left) and driverlog problem (right).

and-shrink code as a basis. All experiments use both label reduction and pruning. As shrinking strategies we evaluate bimisulation ("bisim") and minimal h-preserving shrinking ("hshrink"). hshrink is an h-preserving shrink where all same-formula states are aggregated in a single state (Fan, Holte, and Müller 2018). The implementation of bisimulation was ported from FastDownward into OPTIC. Regarding the merging strategies, we use CGGL (Causal Graph, Goal, Level (Helmert et al. 2007)), as well as a new strategy that we found to be effective in practice. This second strategy is a randomized version of CGGL—to which we call CGGR. It replaces the canonical variable order of CGGL with a random order, and it stops merging new variables once a maximum number of transition-graph edges is reached (we show results for 500 and 5000). Since the abstractions obtained by CGGR are random, we compute 10 different merge-and-shrink abstractions, each starting from a different random seed, and then for heuristic purposes use the maximum of the makespan estimates given by the abstractions. We use bisimulation parameter $N = 100$, which led to the best results.

Benchmarks

The benchmarks we used for evaluation were the temporal optimal-makespan problems of IPC2002 (Long and Fox 2003b) "time-simple-automatic" domains (depots, driverlog, rovers, satellite, zenotravel) and IPC2006 (Dimopoulos et al. 2006) "metric-time" pipesworld. For problems with deadlines we used driverlog, trucks, and zeno domains with deadlines, as in (Marzal, Sebastia, and Onaindia 2014); as well as a selection of automatically generated problems with tight deadlines on depots, driverlog, floortile, logistics, and zeno domains. All experiments use 30min computation time and 4GB memory budget—after which we considered the planner to have failed to solve the problem.

M&S Computation Times Table 1 shows the average time needed to generate merge-and-shrink (M&S) abstractions in both the optimal and deadline problems. This time is spent in a pre-computation phase before the planner starts, and includes both the generation of abstractions (one abstraction in CGGL, 10 in CGGR) and the computation of the goal-makespan formulas. Entries marked "-" indicate that it was not possible to compute the merge-and-shrink abstraction and goal-makespan formulas within the time budget (30min) in all problems of that domain. The table shows that CGGR reduces the computation time overall compared to CGGL. The only exception is the logistics domain, because here abstractions are small and thus CGGR builds 10 full abstractions (and spends approximately 10 times longer on pre-compute). However, total compute time is still between 1 and 4 seconds and thus considerably low.

Optimal Problems To show the generality of our approach, we obtained results for optimal problems with both a partial-order and a total-order planner (i.e. two versions of OPTIC). Tables 2 and 3 show the optimality coverage (i.e. number of problems solved to optimality) and total coverage (i.e. number of problems solved) obtained when using:

- traditional makespan estimates based on TRPG
- M&S with minimal h-preserving shrink (hshrink)
- M&S with bisimulation shrinking (bisim)

The tables show that merge-and-shrink improves coverage on both the partial-order and total-order planner. Particularly, the use of many small abstractions in CGGR500 leads to the partial-order planner solving 1 more problem to feasibility and 1 more problem to optimality, and the total-order planner optimally solving 4 more problems. CGGL improved performance only in the total-order planner, while in partial-order planner it was counterproductive because of the large overhead of computing full abstractions.

Figure 1 shows a comparison of the number of nodes generated and total computation time (including pre-computation) required to solve each problem to optimality—comparing TRPG and M&S with hshrink and CGGR500. The figure shows significant reduction in both the number of states explored and total computation time in partial- and total-order planning.

The reduction in the number of generated nodes is obtained due to better makespan estimates. Figure 2 shows a comparison between TRPG- and M&S-based makespan estimates on all the explored states in two example problems (using CGGR500 and hshrink on a total-order planner). The figure shows that M&S indeed provides better estimates, and is strictly higher than TRPG in some problems.

Problems With Deadlines Table 4 shows results on temporal problems with deadlines, using partial-order search. The benchmark includes both solvable and unsolvable problems, and hence we show both solve coverage (i.e. number of solvable problems solved) and unsolvability-proof coverage (i.e. number of unsolvable problems proved to be unsolvable). The table shows once more that merge-and-shrink with CGGR leads to higher coverage in both kinds of problems—from 6 to 8 more unsolvability proofs, and 4 to 8 more solvable problems solved.

Regarding CGGL, it is interesting to note that while it underperformed in solvable problems, it was able to obtain more proofs in unsolvable problems. Particularly, it was able to prove unsolvability of logistics problems, which were the only ones for which merge-and-shrink computation was small (under one second, see Table 1). CGGR was able to prove unsolvability of depots problems, which matches pre-computation times being considerably smaller than CGGL

Domain	RPG	L-hshrink	L-bisim	R-hshrink500	R-bisim500	R-hshrink5k	R-bisim5k
depots	3 (25)	0 (7)	0 (7)	3 (25)	3 (25)	3 (25)	3 (25)
driverlog	13 (30)	13 (31)	13 (31)	13 (31)	13 (31)	13 (31)	13 (30)
floortile	0 (4)	0 (0)	0 (0)	0 (4)	0 (4)	0 (4)	0 (4)
logistics	35 (12)	42 (18)	42 (18)	41 (16)	41 (15)	42 (18)	43 (18)
trucks	0 (10)	0 (10)	0 (10)	0 (10)	0 (10)	0 (10)	0 (10)
zeno	3 (32)	3 (32)	3 (32)	3 (32)	3 (32)	3 (33)	3 (33)
Sum	54 (113)	58 (98)	58 (98)	60 (118)	60 (117)	**61 (121)**	**62 (120)**

Table 4: Unsolvability proof coverage (solve coverage), partial-order planner on problems with deadlines

in this domain—thus allowing enough time to exhaust the search space.

Conclusions

In this paper we proposed a methodology for generating merge-and-shrink abstractions for temporal planning problems, and to use them as heuristics to speed up temporal planners. The method involves (re)defining action start times and timestamp variables to account for concurrency, and a search method to pre-compute a heuristic as goal-makespan *formulas* instead of numeric values. Our method is applicable both to partial- and total-order temporal planners, and we showed that it leads to increases in coverage and computation speed in both settings. Another contribution of this paper is CGGR, which uses a collection of abstractions built with a random linear merging order and which might include only a subset of the variables. As far as we know this method is new, and was competitive in comparison to CGGL.

Our results indicated the merge-and-shrink approach to computing makespan estimates is helpful in both optimal planning problems, problems with deadlines, and in obtaining unfeasibility proofs. The fact that the approach is particularly helpful in settings where the state-space needs to be exhausted suggests that this work could be useful for computing "minimal unsolvable goal subsets" and planning explanations (Eifler et al. 2020, 2021). In the future we plan to extend the formulation and implementation to domains with non-compression-safe actions (Cushing et al. 2007). Such an extension will have to consider the number of times each non-compression-safe action is executing as part of the abstract state. Other important future work includes the exploration of different merging and shrinking strategies specifically designed for temporal planning, and the integration with landmarks-based approaches (Marzal, Sebastia, and Onaindia 2014).

Acknowledgments

This work was supported by the Air Force Office of Scientific Research under award number FA9550-18-1-0245, and by the EPSRC project THuMP (EP/R033722/1).

References

Benton, J.; Coles, A.; and Coles, A. 2012. Temporal planning with preferences and time-dependent continuous costs. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*.

Bernardini, S.; and Smith, D. E. 2011. Automatic synthesis of temporal invariants. In *Symposium of Abstraction, Reformulation, and Approximation (SARA)*.

Bylander, T. 1994. The Computational Complexity of Propositional STRIPS Planning. *Artificial Intelligence*, 69.

Coles, A.; Coles, A.; Fox, M.; and Long, D. 2009. Extending the use of inference in temporal planning as forwards search. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*.

Coles, A.; Coles, A.; Fox, M.; and Long, D. 2010. Forward-chaining partial-order planning. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*.

Coles, A. J.; and Coles, A. I. 2016. Have I been here before? state memoization in temporal planning. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*.

Cushing, W.; Kambhampati, S.; Mausam, M.; and Weld, D. 2007. When is Temporal Planning Really Temporal? In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 1852–1859.

Dechter, R.; Meiri, I.; and Pearl, J. 1989. Temporal Constraint Networks. In *Proceedings of Principles of Knowledge Representation and Reasoning (KR)*, 83–93.

Dijkstra, E. W. 1959. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1): 269–271.

Dimopoulos, Y.; Gerevini, A.; Haslum, P.; and Saetti, A. 2006. The benchmark domains of the deterministic part of IPC-5. *Abstract Booklet of the competing planners of ICAPS-06*, 14–19.

Do, M.; and Kambhampati, S. 2003. Sapa: A multi-objective metric temporal planner. *Journal of Artificial Intelligence Research*, 20: 155–194.

Eifler, R.; Brandao, M.; Coles, A. J.; Frank, J.; and Hoffmann, J. 2021. Plan-Property Dependencies are Useful: A User Study. In *ICAPS 2021 Workshop on Explainable AI Planning*.

Eifler, R.; Steinmetz, M.; Torralba, A.; and Hoffmann, J. 2020. Plan-Space Explanation via Plan-Property Dependencies: Faster Algorithms & More Powerful Properties. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*, 4091–4097.

Eyerich, P.; Mattmüller, R.; and Röger, G. 2009. Using the context-enhanced additive heuristic for temporal and numeric planning. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*.

Fan, G.; Holte, R.; and Müller, M. 2018. MS-Lite: A lightweight, complementary merge-and-shrink method. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*.

Haslum, P. 2009. Admissible Makespan Estimates for PDDL2.1 Temporal Planning. In *ICAPS 2009 Workshop on Heuristics for Domain-Independent Planning*.

Helmert, M. 2006. The fast downward planning system. *Journal of Artificial Intelligence Research*, 26: 191–246.

Helmert, M.; Haslum, P.; Hoffmann, J.; et al. 2007. Flexible Abstraction Heuristics for Optimal Sequential Planning. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, 176–183.

Hoffmann, J.; and Edelkamp, S. 2005. The deterministic part of IPC-4: An overview. *Journal of Artificial Intelligence Research*, 24: 519–579.

Long, D.; and Fox, M. 2003a. Exploiting a graphplan framework in temporal planning. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, 51–62.

Long, D.; and Fox, M. 2003b. The 3rd International Planning Competition: Results and analysis. *Journal of Artificial Intelligence Research*, 20: 1–59.

Marzal, E.; Sebastia, L.; and Onaindia, E. 2014. On the use of temporal landmarks for planning with deadlines. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*.

Nissim, R.; Hoffmann, J.; and Helmert, M. 2011. Computing perfect heuristics in polynomial time: On bisimulation and merge-and-shrink abstraction in optimal planning. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI)*.

Rintanen, J. 2007. Complexity of concurrent temporal planning. In *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*, 280–287.