Generalised Merge and Shrink Abstractions for Temporal Planning

MARTIM BRANDÃO, King's College London, United Kingdom
AMANDA COLES, King's College London, United Kingdom
ANDREW COLES, King's College London, United Kingdom
REBECCA EIFLER, LAAS-CNRS, France

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 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 general temporal planning problems, in a way that is applicable to both partial- and total-order temporal planners. We extend a previous publication to allow the formalism to apply to temporal problems with non-compression safe actions, in particular through the use of a classical planning surrogate of a temporal planning task. The method relies on pre-computing heuristics as formulas of temporal variables that are evaluated at search time, and it allows to use standard merging, shrinking and pruning strategies. 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 expanded nodes to solve optimal problems, as well as leading to improvements in unsolvability-proving of problems with deadlines, and the time to compute Minimally Unsolvable Goal Subsets (MUGS). We exhaustively test the method over these problems and various usage settings, showing improvements in coverage of up to 53%, computation time up to 60%, and expanded nodes up to 75%.

JAIR Associate Editor: Patrik Haslum

JAIR Reference Format:
Martim Brandão, Amanda Coles, Andrew Coles, and Rebecca Eifler. 2026. Generalised Merge and Shrink Abstractions for Temporal Planning . *Journal of Artificial Intelligence Research* 85, Article 30 (March 2026), 47 pages. DOI: 10.1613/jair.1.18212

1 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 (Fan et al. 2018; Helmert, Haslum, Hoffmann, et al. 2007; Nissim et al. 2011; Sievers and Helmert 2021) into PDDL temporal planning, and use these to improve the efficiency of two families of temporal planners: partial-order and total-order planners. We provide a new formalism with respect to our previous work (Brandao et al. 2022), by using a classical planning surrogate of a temporal planning task in order to build

Authors' Contact Information: Martim Brandão, ORCID: 0000-0002-2003-0675, martim.brandao@kcl.ac.uk, King's College London, London, United Kingdom; Amanda Coles, ORCID: 0000-0002-1838-8301, amanda.coles@kcl.ac.uk, King's College London, London, United Kingdom; Andrew Coles, ORCID: 0000-0002-4954-9235, andrew.coles@kcl.ac.uk, King's College London, London, United Kingdom; Rebecca Eifler, ORCID: 0000-0001-8275-7813, rebecca.eifler@laas.fr, LAAS-CNRS, Toulouse, France.

DOI: 10.1613/jair.1.18212

merge-and-shrink abstractions (and admissible heuristics) for temporal planning problems with or without non-compression-safe actions.

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) (A. Coles et al. 2010; Do and Kambhampati 2003; Haslum 2009), though the context-enhanced additive heuristic (Eyerich et al. 2009) and landmark-based pruning strategies (Marzal et al. 2014) have also been applied to temporal planning.

In this paper we build on the work of temporal mutex analysis (Bernardini et al. 2018) and (classical) merge-and-shrink abstractions (Fan et al. 2018; Helmert, Haslum, Hoffmann, et al. 2007; Nissim et al. 2011) 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, temporal problems with deadlines, and temporal Minimally Unsolvable Goal Subsets (MUGS) computation problems. Additionally, we apply it both to heuristics used for state pruning and for open-list sorting; and both solving problems and proving unsolvability—showing consistent improvement across the board compared to the TRPG heuristic which is typical in temporal planners. Finally, and 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 though slightly more in the total-order setting.

The contributions of the paper are: 1) the proposal of a merge-and-shrink method for temporal planners, based on a new formalism compared to our previous publication (Brandao et al. 2022) which is now applicable to problems with non-compression-safe actions; 2) the demonstration of the effectiveness of merge-and-shrink in both partial- and total-order temporal planners, in a variety of problems (makespan-optimization problems, temporal problems with deadlines, and MUGS problems) and in a variety of settings (solving and unsolvability proving, heuristic-based state pruning and open-list sorting).

2 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. While TRPG-based planners are typically focused on satisfycing (non-optimal) planning, the time at which the goals are reached in the TRPG is an admissible makespan estimate, so can be used in the case of optimal planning. Another approach to obtaining admissible makespan 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 et al. 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 et al. 2009). Beyond heuristic-search, other work approaches optimal temporal planning via Optimization Modulo Theory (Panjkovic and Micheli 2024) or via Constraint Programming (Vidal 2011).

The method proposed in this paper is based on the merge-and-shrink literature (Fan et al. 2018; Helmert 2006; Helmert, Haslum, Hoffmann, et al. 2007; Nissim et al. 2011; Sievers and Helmert 2021), which has focused on classical planning problems so far. We retain the restriction to the propositional case (without numeric state variables), but broaden our focus to planning for temporal problems, in particular following the semantics of PDDL2.2 (Hoffmann and Edelkamp 2005). The underlying search mechanics on which we build follows that of prior partial-order forward-search-based planners (Benton et al. 2012; A. Coles et al. 2010; A. J. Coles and A. I. Coles 2016), but is broadly applicable to search-based approaches where a time-constrained plan is being built in a forward direction, and a makespan estimate of reaching the goals is required. We do not consider richer temporal and numeric planning models, such as those modeled in PDDL+ (Fox and Long 2006) which can be solved by a range of approaches (Cashmore et al. 2020; Piotrowski et al. 2016; Scala et al. 2020); but note that our heuristic could be adopted to provide guidance for at least the propositional causal structure of such models which, as a relaxation, would retain admissibility.

3 Background

We begin by discussing the relevant background materials, in particular the temporal planning formalism we are concerned with, and key concepts in merge-and-shrink.

3.1 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 $dom(v)$. The total variable state space is therefore $\mathcal{S}_\mathcal{V} = dom(v_1) \times ... \times dom(v_n)$.*
- *$\mathcal{O}$ is a finite set of durative actions[1], 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 $\delta^+ \geq \delta^-$. For notational convenience, we use $pre_\vdash(a)$, $pre_\leftrightarrow(a)$, $pre_\dashv(a)$, $eff_\vdash(a)$, $eff_\dashv(a)$, $\delta^-(a)$, $\delta^+(a)$ to refer to the respective tuple members of a.*
- *Each action a can be split into two "snap" actions $a_\vdash = \langle pre_\vdash, eff_\vdash \rangle$ and $a_\dashv = \langle pre_\dashv, eff_\dashv \rangle$ (Long and Fox 2003a), which need to be sequenced and scheduled such as to respect $pre_\leftrightarrow(a), \delta^-(a), \delta^+(a)$.*
- *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 that 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 that 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 (partial variable assignments) called the "goal" of the task.*

Within a temporal planning task, a useful distinction can be made between actions that are 'compression safe', and those that are not:

[1]We assume without loss of generality, that any actions that are instantaneous are represented as zero-duration durative actions, with all preconditions and effects at the start.

Definition 2. *"Compression-safe" actions $O_c \subseteq O$ are durative actions where, for each $a \in O_c$, there is no logical necessity to sequence other actions between the start and end of a (A. Coles et al. 2009).*

In temporal SAS+ parlance, an action a is compression safe iff *both of the following conditions hold:*

- *$pre_{\dashv}(a) \subseteq pre_{\leftrightarrow}(a)$. Once a has started, its invariants must hold; hence, in this case, its end preconditions are satisfied – there is no need to apply intermediate actions between the start and end of a, in order to satisfy these.*
- *For each $\langle v, w \rangle \in eff_{\dashv}(a)$, any action a' with a precondition or effect on v, at any point within a', is mutually exclusive with the action a (according to the analysis of (Bernardini et al. 2018)). Hence, there is no logical need to apply intermediate actions between the start and end of a, prior to its end effect on v, as no action referring to v can occur during the execution of a.*

The set of non-compression-safe actions is denoted by $O_n = O \backslash O_c$.

Even though splitting actions into "snap" actions in principle leads to an explosion of the search space, we note that for the special case of compression-safe actions, with the appropriate management of temporal constraints, the end of the action can be applied immediately after its start (rather than considering this a search decision), without compromising completeness, soundness or optimality (A. Coles et al. 2010). In any case, adopting a forward-chaining search approach, we define a state as:

Definition 3. *A temporal planning state $\mathtt{s} = \langle U, P, T \rangle$ is a tuple of:*

- *U: assignments of values to all variables in $\mathcal{V}$*
- *P: a vector of m snap actions which represent the plan to reach $\mathtt{s}$ from s_0. Each one of these m actions is called a "step" of P.*
- *T: a set of temporal constraints over the actions in P, each of the form $t^- \leq t(j) - t(i) \leq t^+$. 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, i and j are indices of actions in P, while $t^-, t^+ \in \mathbb{Q}_0^+$ and $t^- \leq t^+$.*

We define a (partial) plan as follows:

Definition 4. *A (partial) plan for the planning task Π, associated with a temporal planning state $\mathtt{s}$ is defined by snap-actions $P = [p(1), ..., p(m)]$ and an assignment of timestamps $[t(1), ..., t(m)]$ to each of the m actions in P, such that the logical and temporal constraints of the problem are respected.*

For our purposes, the timestamps required for a (partial) plan Π can be derived from a state by taking its vector of snap actions P, and associated temporal constraints T, and solving a Simple Temporal Problem (STP) (Dechter et al. 1989) built from T to find the earliest timestamp $[t(1), ..., t(m)]$ for each snap action. By finding the shortest paths from each step to time zero within a Simple Temporal Network (STN) representation of the STP (Dechter et al. 1989), we find the earliest timestamp for each step; and hence the start time and (STP-chosen) duration for each action. Moreover, if the STP cannot be solved, the temporal constraints are inconsistent. We assume all such inconsistent states are discarded upon creation: a snap-action may be first thought of being applicable due to its logical constraints, but then the resulting state discarded due to an inconsistent STP.

The constraints in T, and hence the STP, are a partial order comprising sequence constraints due to the logical interactions between preconditions and effects (each $d \leq t(j) - t(i)$, where $d \in \{0, \epsilon\}$) or duration constraints on actions: $\delta^-(a) \leq t(j) - t(i) \leq \delta^+(a)$ if an action a started at step i and finished at step j. The convenience function *exec*, defined in Algorithm 1, represents the step indices where actions started in some partial plan P, but have not finished by its end. Briefly, iterating through it step-by-step, when an action a_i starts the respective step index is added to $exec[a_i]$ (line 4); and when it ends, the corresponding start step index is determined by inspecting T (line 6), and this is removed from $exec[a_i]$ (line 7).

Algorithm 1: Finding the actions executing after some plan

Data: A plan P, temporal constraints T
Result: Step indices of the starts of actions still executing after P

```
exec ← {⟨a_i, ∅⟩ | a_i ∈ O};
foreach plan step p_l ∈ {1, ..., m} where m = |P| do
    if P[p_l] = a_i⊢ then
        exec[a_i] ← exec[a_i] ∪ {p_l};
    else if P[p_l] = a_i⊣ then
        p_m ← the step index such that (δ⁻(a_i) ≤ t(p_l) − t(p_m) ≤ δ⁺(a_i)) ∈ T;
        exec[a_i] ← exec[a_i] \ {p_m};
return exec
```

Definition 5. *A state $\mathsf{s} = \langle U, P, T\rangle$ provides a solution plan for the task Π if $G \subseteq U$, all conditions of actions in P are satisfied, all actions have finished executing ($\forall_{a_i \in O}\ exec(P,T)[a_i] = \emptyset$), and the temporal constraints T are consistent. A solution is optimal if its "makespan" $max_{i \in 1,\ldots,m} t(i)$ is minimal.*

Definition 6. *Variable annotations (A. Coles et al. 2010) are stored for each temporal planning state $\mathsf{s} = \langle U, P, T\rangle$ during planning, and consist of:*

- *$V^{\mathit{eff}}(\mathsf{s}, v)$: the index of the step (action) in P that most recently had an effect upon variable v;*
- *$VP(\mathsf{s}, v)$, where each $\langle i, d\rangle \in VP(\mathsf{s}, v)$ comprises the index i of a step in P that had a condition on 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 (Fox and Long 2003).*

We note that while the state annotations introduced in (A. Coles et al. 2010) were written in a propositional PDDL formalism, here they refer to SAS$^+$ variables. As the SAS$^+$ variables correspond to sets of mutually exclusive propositions, the semantics of the annotations are unchanged.

The role of variable annotations is as a basis for defining temporal constraints that are sufficient to ensure a plan is logically sound. They are updated hand-in-hand with the temporal constraints as snap-actions are applied:

Definition 7. *If a start snap action $a_\vdash$ were applied as step j in some state $\mathsf{s} = \langle U, P, T\rangle$, a state $\mathsf{s}' = \langle U', P', T'\rangle$ would be reached, first defined as:*

$$\begin{aligned} U' &= \bigcup_{v \in \mathcal{V}} \begin{cases} \{\langle v, w\rangle\} & \text{if } \langle v, w\rangle \in \mathit{eff}(a_\vdash) \\ \{U[v]\} & \text{otherwise} \end{cases} \\ P' &= P + [a_\vdash] \\ T' &= T \qquad \text{(then updated as below)} \end{aligned}$$

The snap action is logically applicable iff*:*

$$\mathit{pre}_\vdash(a) \subseteq U \wedge \bigwedge_{a_i \in O\,:\, exec(P',T')[a_i] \neq \emptyset} \mathit{pre}_\leftrightarrow(a_i) \subseteq U'.$$

The snap action is temporally applicable iff *after the following updates to T', it remains consistent:*

- *For each condition $\langle v, w\rangle \in \mathit{pre}_\vdash(a)$, the constraint $\left(\epsilon \leq t(j) - t(V^{\mathit{eff}}(\mathsf{s}, v))\right)$ is added to T'. This ensures step j comes after the achievers of each of its preconditions.*
- *For each effect $\langle v, w\rangle \in \mathit{eff}_\vdash(a)$:*

  - *For each* $\langle i, d\rangle \in VP(\mathsf{s}, v)$, $(d \leq t(j) - t(i))$ *is added to* T'. *This ensures step* j *comes after any conditions it might threaten.*
  - $\big(\epsilon \leq t(j) - t(V^{\mathit{eff}}(\mathsf{s}, v))\big)$ *is added to* T'. *This places a total order on the effects of* v, *so its value is known in* s'.
- *For each condition* $\langle v, w\rangle \in pre_{\leftrightarrow}(a)$, *the constraint* $\big(0 \leq t(j) - t(V^{\mathit{eff}}(\mathsf{s}, v))\big)$ *is added to* T'. *This ensures step* j *comes after the achievers of each of its invariants.*

Having applied $a_{\vdash}$, *the variable annotations in* s' *are determined first by copying those from* s, *then updating them according to the conditions and effects of* $a_{\vdash}$:

- *For each condition* $\langle v, w\rangle \in pre_{\vdash}(a)$, $\langle j, \epsilon\rangle$ *is added to* $VP(\mathsf{s}', v)$. *This ensures that subsequent steps with an effect on* v *are ordered after step* j.
- *For each effect* $\langle v, w\rangle \in \mathit{eff}_{\vdash}(a)$, $V^{\mathit{eff}}(\mathsf{s}', v)$ *is changed to* j, *as step* j *has now the most recent effect on* v.
- *For each* $\langle v, w\rangle \in pre_{\leftrightarrow}(a)$, $\langle j, 0\rangle$ *is added to* $VP(\mathsf{s}', v)$.

If an end snap action $a_{\dashv}$ is to be applied in a state reached by plan P, as a new step k, its start step index may be any $j \in \mathit{exec}(P, T)[a]$. Having chosen a start step index j[2], we define ending an action as follows:

Definition 8. *If an end snap action* $a_{\dashv}$ *were applied as step* k *after the plan in some state* $\mathsf{s} = \langle U, P, T\rangle$, *to end an instance of* a *that started at step* j, *a state* $\mathsf{s}' = \langle U', P', T'\rangle$ *would be reached, first defined as:*

$$
\begin{aligned}
U' = \quad & \bigcup_{v \in \mathcal{V}} \begin{cases} \{\langle v, w\rangle\} & \text{if } \langle v, w\rangle \in \mathit{eff}(a_{\dashv}) \\ \{U[v]\} & \text{otherwise} \end{cases} \\
P' = \quad & P + [a_{\dashv}] \\
T' = \quad & T \cup \{(\delta^{-}(a) \leq t(k) - t(j) \leq \delta^{+}(a))\} \qquad \textit{(then updated as below)}
\end{aligned}
$$

$a_{\dashv}$ *is logically applicable* iff:

$$
pre_{\dashv}(a) \subseteq U \wedge \bigwedge_{a_i \in O \,:\, \mathit{exec}(P', T')[a_i] \neq \emptyset} pre_{\leftrightarrow}(a_i) \subseteq U'
$$

...and it is temporally applicable iff *after the following updates to* T', *it remains consistent:*

- *For each* $\langle v, w\rangle \in pre_{\leftrightarrow}(a)$, *the constraint* $\big(0 \leq t(k) - t(V^{\mathit{eff}}(\mathsf{s}, v))\big)$ *is added to* T'.
- *For each* $\langle v, w\rangle \in pre_{\dashv}(a)$, *the constraint* $\big(\epsilon \leq t(k) - t(V^{\mathit{eff}}(\mathsf{s}, v))\big)$ *is added to* T'.
- *For each effect* $\langle v, w\rangle \in \mathit{eff}_{\dashv}(a)$:
  - *For each* $\langle i, d\rangle \in VP(\mathsf{s}, v)$, $(d \leq t(k) - t(i))$ *is added to* T'.
  - $\big(\epsilon \leq t(k) - t(V^{\mathit{eff}}(\mathsf{s}, v))\big)$ *is added to* T'.

Variable annotation updates are similar to the case for $a_{\vdash}$:

- *For each* $\langle v, w\rangle \in pre_{\leftrightarrow}(a)$, $\langle k, 0\rangle$ *is added to* $VP(\mathsf{s}', v)$.
- *For each condition* $\langle v, w\rangle \in pre_{\dashv}(a)$, $\langle k, \epsilon\rangle$ *is added to* $VP(\mathsf{s}', v)$.
- *For each effect* $\langle v, w\rangle \in \mathit{eff}_{\dashv}(a)$, $V^{\mathit{eff}}(\mathsf{s}', v)$ *is changed to* k.

Note the temporal constraints, and *VP* annotations, judiciously use either ϵ or 0 as temporal separation. This is due to the Pddl 2.1 semantics (Fox and Long 2003). For instantaneous conditions—those in $pre_{\vdash}$ or $pre_{\dashv}$—the value of a variable cannot be simultaneously updated by an effect, and inspected to determine whether a precondition is true; so a gap of ϵ is used to ensure the former occurs before the latter. For invariant conditions—those in $pre_{\leftrightarrow}$—these need to hold only in the open interval between the start and end of the action, in which case a gap of 0 is sufficient.

[2]This is a branching point in search—if the end of an action a is applicable in s, it may yield one successor state per currently executing instance of a.

Definition 9. *Variable "use" and "chg" timestamps (A. J. Coles and A. I. Coles 2016) are the earliest timestamps from which a variable can be used or changed, in an extension of a plan forwards from a state* s*, given the preconditions and effects on that variable in the plan to reach* s*, and the lower bound* $t(i)$ *on the timestamp of each step* i *in this plan:*

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

In total-order temporal planners[3], *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 (A. 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 partial- and total-ordered temporal planners – it is not specific to either in particular.

3.2 Merge-and-Shrink

Having introduced matters related to temporal planning, we move on to introducing merge-and-shrink abstractions for classical (non-temporal) planning tasks. As a foundation, a transition graph (Helmert, Haslum, Hoffmann, et al. 2007) is defined as follows:

Definition 10. *A transition graph is a tuple* $\Theta = \langle V, S, L, A, s_0, S_G\rangle$*, where* V *is a set of state variables;* S *is a state space of variable assignments;* $L \subseteq O$ *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 transition graph for a classical planning task Π is denoted $\Theta(\Pi)$. Its variables, states (and goal states) comprise those in the planning task; and the graph has an edge, labeled $l \in O$, from $s \in S$ to $d \in S$, if l is applicable in s and its application results in state d.

An abstraction of a transition graph is defined as follows:

Definition 11. *An abstraction* $\Theta' = \langle V', S', L', A', s'_0, S'_G\rangle$ *of a transition graph* $\Theta = \langle V, S, L, A, s_0, S_G\rangle$*, is defined with respect to an abstraction mapping function* $\alpha : S \to S'$*, such that:*

- $L' = L$
- $\langle \alpha(s), l, \alpha(d)\rangle \in A'$ *for all* $\langle s, l, d\rangle \in A$
- $s'_0 = \alpha(s_0)$
- $\alpha(s_g) \in S'_G$ *for all* $s_g \in S_G$
- $V' \subseteq V$*, with* $v \in V'$ iff *(i)* Θ' *depends on* v*, and (ii)* $v \in V' \Leftrightarrow (\exists s, s' \in S' \mid s \neq s' \wedge \forall_{v' \in (V \setminus \{v\})}\, s(v') = s'(v'))$.

Importantly, the abstraction of a transition graph preserves paths in the transition graph. Hence, abstractions of the transition graph for a planning task are relaxations, and can be used as the basis of heuristics. Merge-and-shrink (Helmert, Haslum, Hoffmann, et al. 2007) 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 $\langle s', l, d'\rangle \in A'$.

[3]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, ordered after all previous steps.

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 2.

Algorithm 2: Generic merge-and-shrink algorithm

Data: $X \leftarrow$ { all single-variable transition graphs }
Result: final transition graph in X

```
while |X| > 1 do
    Θ1, Θ2 ← PickTwoGraphsToMerge(X);
    LabelReduction(Θ1, Θ2) ;
    Θ1 ← α(Θ1); Θ2 ← α(Θ2);          #Shrink
    Θmerged ← Θ1 ⊗ Θ2;                #Merge
    Pruning(Θmerged);
    X ← X ∪ Θmerged \ {Θ1, Θ2};
```

4 Temporal Merge-and-Shrink

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). These properties make it harder to pre-compute of a heuristic function compared to classical planning, and our approach to tackle these challenges is to pre-compute a mapping not from states to numeric values, but from states to *functions* of *use* and *chg* variables.

With this in mind, we will now set out our approach to devising a temporal merge-and-shrink heuristic.

- First, we describe how a classical surrogate planning task can be derived from a temporal planning task, sufficient to capture (most of) its logical constraints. As the surrogate task is classical, it is amenable to existing (classical) merge-and-shrink techniques to produce abstract transition graphs.
- Second, we consider how to re-introduce temporal information into these abstract transition graphs, by overlaying action scheduling information as functions of *use* and *chg* variables for the (abstract) states in the transition graphs.

We then describe how the resulting overlaid transition graphs can be used as the basis for admissible makespan and heuristic functions: in short, heuristics will be pre-computed before starting search, by propagating *use* and *chg* variables from goal states to all states in an abstraction of the classical surrogate, via the actions available. This propagation will yield a goal-makespan formula (i.e. an expression of *use* and *chg* variables) per abstract state, which can be evaluated at search time once we have access to actual values of *use* and *chg* of a specific state $\mathsf{s} = \langle U, P, T \rangle$.

4.1 A Classical Surrogate of a Temporal Planning Task

To construct our abstractions, we need to be able to define the space that we are seeking to abstract. For this, we define a classical surrogate of the temporal planning task. Previous work (such as (Jiménez et al. 2015)) has sought to compile a useful *subset* of temporal planning tasks into classical planning tasks, in a way that ensures solution plans found for these classical tasks can be translated back into sound solution plans for the temporal tasks. Our focus here, however, is different: in the interests of ultimately building abstractions to underpin heuristic estimates for temporal planning tasks *in general*, we define a classical surrogate whose logical structure is a (modest) relaxation of the logical structure of the temporal planning task, such that all paths through the state space of the temporal planning task are preserved as paths through the state space of the classical surrogate.

For compression-safe actions, the mapping from a temporal task to our classical surrogate is relatively straightforward. For non-compression-safe actions, the mapping is more complex. We will discuss the latter of these first.

Definition 12. *The 'execution counter' variables $\mathcal{E}$ for the classical surrogate planning task are additional state variables beyond those in the temporal task. There is one variable $e_i \in \mathcal{E}$ for each non-compression-safe action $a_i \in O_n$, and each $e_i \in \mathcal{E}$ has the domain $\{0, 1, 2\}$.*

If $e_i = 0$, we are in a state where a_i is not currently executing; if $e_i = 1$, we are in a state where a_i is currently executing once; and if $e_i = 2$, we are in a state where a_i is currently executing 2 or more times in parallel. These counter variables are used for three purposes. First, to logically link the starts and ends of actions: the end of an action cannot be applied if its execution counter is 0. Second, to capture that in the initial state and any goal state, no actions are executing. Third, we use them to capture mutual exclusion relationships between actions due to the effects of one interfering with the invariant conditions of the other: at any point where there is such a potential conflict, we have an additional precondition that the relevant action's execution counter is 0.

Where an action cannot overlap with itself, a domain $\{0, 1\}$ for its execution counter variable would be sufficient. However, we also allow for the execution counter variable to take a value of 2 so as to allow the possibility of self-overlapping actions. This is a relaxation, as there may be more than 2 instances of a self-overlapping action running in parallel with each other. Larger values could be added to the domain of such execution counter variables, in order to better capture this; but the value of 2 we choose is a pragmatic balance of not over-burdening the size of the abstraction in the typical case (actions that don't self-overlap) while retaining completeness (and ultimately admissibility) in the case where they can.

Definition 13. *The mutual exclusion precondition set for effects eff, with respect to non-compression-safe actions O_n is defined as $pmutex(eff)$, where:*

$$pmutex(eff) = \{\langle e_i, 0\rangle \mid a_i \in O_n \wedge \exists \quad (\langle v_i, w_i\rangle, \langle v_j, w_j\rangle) \in (pre_{\leftrightarrow}(a_i) \times eff) \text{ such that } v_i = v_j \wedge w_i \neq w_j) \quad \}$$

Each a_i yields six classical actions—three from its start snap-action $a_{i\vdash}$, three from its end $a_{i\dashv}$. For the former we get:

Definition 14. *The three 'start' actions in the classical surrogate planning task, due to a non-compression-safe action $a_i \in O_n$, are:*

$$
\begin{aligned}
a^0_{i\vdash}: \quad & pre(a^0_{i\vdash}) = \{\langle e_i, 0\rangle\} \cup pre_{\vdash}(a_i) \cup (pre_{\leftrightarrow}(a_i) \setminus eff_{\vdash}(a_i)) \cup pmutex(eff_{\vdash}(a_i)) \\
& eff(a^0_{i\vdash}) = \{\langle e_i, 1\rangle\} \cup eff_{\vdash}(a_i) \\
a^1_{i\vdash}: \quad & pre(a^1_{i\vdash}) = \{\langle e_i, 1\rangle\} \cup pre_{\vdash}(a_i) \cup (pre_{\leftrightarrow}(a_i) \setminus eff_{\vdash}(a_i)) \cup pmutex(eff_{\vdash}(a_i)) \\
& eff(a^1_{i\vdash}) = \{\langle e_i, 2\rangle\} \cup eff_{\vdash}(a_i) \\
a^2_{i\vdash}: \quad & pre(a^2_{i\vdash}) = \{\langle e_i, 2\rangle\} \cup pre_{\vdash}(a_i) \cup (pre_{\leftrightarrow}(a_i) \setminus eff_{\vdash}(a_i)) \cup pmutex(eff_{\vdash}(a_i)) \\
& eff(a^2_{i\vdash}) = eff_{\vdash}(a_i)
\end{aligned}
$$

$a^0_{i\vdash}$ and $a^1_{i\vdash}$ correspond to starting the action when 0 (respectively 1) instances of it are currently executing; while ensuring it does not violate the invariants of any executing action, and incrementing the execution counter. For $a^2_{i\vdash}$, as the execution counter is already saturated, we do not increment it further.

Definition 15. *The three 'end' actions in the classical surrogate planning task, due to a non-compression-safe action $a_i \in O_n$, are:*

$$
\begin{aligned}
a^0_{i\dashv}: \quad & pre(a^0_{i\dashv}) = \{\langle e_i, 1\rangle\} \cup pre_{\dashv}(a_i) \cup (pmutex(eff_{\dashv}(a_i)) \setminus \{\langle e_i, 0\rangle\}) \\
& eff(a^0_{i\dashv}) = \{\langle e_i, 0\rangle\} \cup eff_{\dashv}(a_i) \\
a^1_{i\dashv}: \quad & pre(a^1_{i\dashv}) = \{\langle e_i, 2\rangle\} \cup pre_{\dashv}(a_i) \cup pmutex(eff_{\dashv}(a_i)) \\
& eff(a^1_{i\dashv}) = \{\langle e_i, 1\rangle\} \cup eff_{\dashv}(a_i) \\
a^2_{i\dashv}: \quad & pre(a^2_{i\dashv}) = \{\langle e_i, 2\rangle\} \cup pre_{\dashv}(a_i) \cup pmutex(eff_{\dashv}(a_i)) \\
& eff(a^2_{i\dashv}) = eff_{\dashv}(a_i)
\end{aligned}
$$

$a^0_{i\dashv}$ corresponds to ending the action when 1 instance is executing, so that 0 instances of it are then executing, while ensuring it does not violate the invariants of any executing action; noting that the end effects of a_i are allowed to violate its own invariants (as immediately, no other instance of a_i will be executing). $a^1_{i\vdash}$ and $a^2_{i\vdash}$ correspond to ending the action when 2 or more instances are executing, but as we do not know how many there may be, we need two options: one that decrements the execution counter from 2 to 1, to cover the case where there were actually 2 instances executing; and one that does not decrement the execution counter, to cover the case where there were more than 2 instances executing.

For the compression-safe actions, O_c, the mapping is dramatically simpler. As the end snap-action of these can be sequenced immediately after the start, with no adverse logical effects, we can map them to a single classical action with the weakest preconditions, the strongest effects, and that does not violate the invariants of any executing non-compression-safe action. Thus, for each temporal action $a \in O_c$, we make a single classical action a_-:

Definition 16. *The single action a_- in the classical surrogate planning task representing a compression-safe action $a \in O_c$ is:*

$$
\begin{aligned}
a_-: \quad & pre(a_-) = \quad pre_{\vdash}(a) \cup ((pre_{\leftrightarrow}(a) \cup pre_{\dashv}(a)) \setminus eff_{\vdash}(a)) \\
& \qquad\qquad \cup\ pmutex(eff_{\vdash}(a)) \cup pmutex(eff_{\dashv}(a)) \\
& eff(a_-) = \quad \{\langle v, w\rangle \in eff_{\vdash}(a) \mid \nexists \langle v', w'\rangle \in eff_{\dashv}(a)\ s.t.\ v = v'\} \cup eff_{\dashv}(a)
\end{aligned}
$$

Finally, with all these action definitions, we can formally define the classical surrogate of a temporal SAS$^+$ planning task:

Definition 17. *A classical surrogate of a temporal SAS$^+$ planning task* $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, G \rangle$ *is a tuple* $\hat{\Pi} = \langle \mathcal{V} \cup \mathcal{E}, \hat{\mathcal{O}}, \hat{s_0}, \hat{G} \rangle$*, where:*

- $\mathcal{V}$ *are the state variables, unchanged from the temporal planning task.*
- $\mathcal{E}$ *are the execution counter variables, one for each* $a_i \in \mathcal{O}_n$*, defined according to Definition 12.*
- $\hat{\mathcal{O}}$ *is a finite set of classical actions, where each* $a \in \hat{\mathcal{O}}$ *has preconditions* $pre(a)$ *and effects* $eff(a)$*. These follow Definitions 14, 15 and 16.*
- $\hat{s_0}$ *is the initial state, where* $\hat{s_0} = s_0 \cup \bigcup_{e_i \in \mathcal{E}} \{\langle e_i, 0 \rangle\}$.
- $\hat{G}$ *is a set of goals, where* $\hat{G} = G \cup \bigcup_{e_i \in \mathcal{E}} \{\langle e_i, 0 \rangle\}$.

The merge-and-shrink abstraction techniques described in Section 3.2 can now be applied to this classical surrogate of the temporal planning task.

4.1.1 Worked example SAS+ planning task. To demonstrate the concepts introduced so far, we now describe an example temporal *SAS*$^+$ planning task, along with its classical surrogate, and its transition graph. We will later on use the same example to demonstrate the computation of *use/chg* values and heuristics from the classical surrogate.

Consider as a worked example a small logistics-style planning task $\Pi = \langle \mathcal{V}, \mathcal{O}, s_0, G \rangle$ where:

- $\mathcal{V} = \{v_T, v_P, v_D\}$ for a truck, package, and driver respectively;
- $dom(v_T) = \{\text{A,B,C,}\perp\}$ – the truck is at location A, B, or C; or neither, while transitioning between locations
- $dom(v_P) = \{\text{A,B,C,T,}\perp\}$ – the package is at location A, B, or C; or on the truck; or neither, while transitioning from one value to another.
- $dom(v_D) = \{\text{D,L}\}$ – the driver is either ready to drive (D) the truck, or is ready to load/unload packages (L);
- $s_0 = \{v_T \mapsto A, v_P \mapsto B, v_D \mapsto D\}$;
- $G = \{v_P \mapsto C\}$
- $\mathcal{O}$ contains 'drive' actions for the truck, between any pair of locations $x \in \{A, B, C\}$, $y \in \{A, B, C\}$ defined as follows[4]:

$$\begin{array}{lll} \text{(drive x y):} & pre_{\vdash}(\text{drive x y}) = \{v_T \mapsto x\} & eff_{\vdash}(\text{drive x y}) = \{v_T \mapsto \perp\} \\ & pre_{\leftrightarrow}(\text{drive x y}) = \{v_D \mapsto D\} & \\ & pre_{\dashv}(\text{drive x y}) = \{\} & eff_{\dashv}(\text{drive x y}) = \{v_T \mapsto y\} \\ & \delta^-(\text{drive x y}) = 10, \delta^+(\text{drive x y}) = 10 & \end{array}$$

...along with 'load' and 'unload' actions for the package, defined for any location $x \in \{A, B, C\}$ as follows[5]:

[4]A fully enumerated set of drive actions is provided in Figure 7 in the Appendix
[5]A fully enumerated set of load/unload actions is provided in Figure 8 in the Appendix

$$
\begin{array}{lll}
\text{(load x):} & pre_{\vdash}(\text{load x}) = \{v_P \mapsto x\} & eff_{\vdash}(\text{load x}) = \{v_P \mapsto \bot\} \\
& pre_{\leftrightarrow}(\text{load x}) = \{v_T \mapsto x, v_D \mapsto L\} & \\
& pre_{\dashv}(\text{load x}) = \{\} & eff_{\dashv}(\text{load x}) = \{v_P \mapsto T\} \\
& \delta^-(\text{load x}) = 1, \delta^+(\text{load x}) = 1 & \\
\text{(unload x):} & pre_{\vdash}(\text{unload x}) = \{v_P \mapsto T\} & eff_{\vdash}(\text{unload x}) = \{v_P \mapsto \bot\} \\
& pre_{\leftrightarrow}(\text{unload x}) = \{v_T \mapsto x, v_D \mapsto L\} & \\
& pre_{\dashv}(\text{unload x}) = \{\} & eff_{\dashv}(\text{unload x}) = \{v_P \mapsto x\} \\
& \delta^-(\text{unload x}) = 1, \delta^+(\text{unload x}) = 1 &
\end{array}
$$

...and the action 'driver-loading', which toggles v_D from D to L at the start, and back to D at the end, creating an opportunity to apply load/unload operators during its execution:

$$
\begin{array}{ll}
\text{(driver-loading):} & pre_{\vdash}(\text{driver-loading}) = \{v_D \mapsto D\} \\
& eff_{\vdash}(\text{driver-loading}) = \{v_D \mapsto L\} \\
& pre_{\leftrightarrow}(\text{driver-loading}) = \{\} \\
& pre_{\dashv}(\text{driver-loading}) = \{\} \\
& eff_{\dashv}(\text{driver-loading}) = \{v_D \mapsto D\} \\
& \delta^-(\text{driver-loading}) = 2 \\
& \delta^+(\text{driver-loading}) = 2
\end{array}
$$

Straightforwardly, all 'drive', 'load' and 'unload' actions are compression safe (so are in O_c). With reference to Definition 2:

- For each such action a, $pre_{\dashv}(a) = \emptyset$; hence, trivially, $pre_{\dashv}(a) \subseteq pre_{\leftrightarrow}(a)$.
- Each such action a has a single end effect $v \mapsto w$; and any action with a precondition or effect on v is mutually exclusive with a. Following the analysis of (Bernardini et al. 2018):
  - All 'drive' actions are pairwise mutex: they are the only actions that change v_T; each requires $v_T \in (dom(v_T) \setminus \bot)$ at the start, before immediately setting $v_T \mapsto \bot$, thereby blocking any other 'drive' actions from starting; before setting v_T to some value in $(dom(v_T) \setminus \bot)$ at the end.
  - Similarly, all 'load' and 'unload' actions are pairwise mutex: they are the only actions that change v_P; each requires $v_P \in (dom(v_P) \setminus \bot)$ at the start, before immediately setting $v_P \mapsto \bot$, thereby blocking any other 'load' or 'unload' actions from starting; before setting v_P to some value in $(dom(v_P) \setminus \bot)$ at the end.
  - 'drive' actions cannot start within the execution of 'load' or 'unload' actions, as each drive action has a start effect $v_T \mapsto \bot$ which conflicts with a precondition $(v_T \in (dom(v_T) \setminus \bot)) \in pre_{\leftrightarrow}(a)$ for each 'load' and 'unload' action a.
  - 'load' and 'unload' actions cannot start within the execution of 'drive' actions: each such 'load' or 'unload' action a has $(v_T \in (dom(v_T) \setminus \bot)) \in pre_{\leftrightarrow}(a)$; but during a 'drive' action, $v_T = \bot$.

Conversely, 'driver-loading' is *not* compression safe, so is in O_n: it has an end effect on v_D, which is a precondition of the 'load' and 'unload' actions; and these are not mutually exclusive with 'driver-loading' – indeed, 'load' and 'unload' actions *must* occur during the execution of 'driver-loading'.

With this in mind, we can construct the classical surrogate for this temporal planning task, $\hat{\Pi} = \langle \mathcal{V} \cup \mathcal{E}, \hat{O}, \hat{s_0}, \hat{G} \rangle$ where:

- $\mathcal{V} = \{v_T, v_P, v_D\}$, unchanged from the temporal planning task.
- $\mathcal{E} = \{e_{DL}\}$ – a single execution counter variable for the single action in O_n, '(driver-loading)'; with $dom(e_{DL}) = \{0, 1, 2\}$[6]

[6]While the reader may have noticed that 'driver-loading' is mutex with itself, hence there is no reachable state where $e_{DL} = 2$, the domain of e_{DL} is intentionally stated as being $\{0, 1, 2\}$, in the interests of consistency with the formalism of the paper.

- $\hat{s_0} = s_0 \cup \{e_{DL} \mapsto 0\}$;
- $\hat{G} = G \cup \{e_{DL} \mapsto 0\}$;
- $\hat{O}$ contains classical surrogates for 'drive' actions for the truck, between any pair of locations $x \in \{A, B, C\}$, $y \in \{A, B, C\}$ defined as follows:

$$\begin{aligned}\text{(drive x y\_):}\quad & \mathit{pre}(\text{drive x y\_}) = \{v_T \mapsto x, v_D \mapsto D\}\\ & \mathit{eff}(\text{drive x y\_}) = \{v_T \mapsto y\}\end{aligned}$$

...along with surrogates for the 'load' and 'unload' actions for the package, defined for any location $x \in \{A, B, C\}$ as follows:

$$\begin{aligned}\text{(load x\_):}\quad & \mathit{pre}(\text{load x\_}) = \{v_P \mapsto x, v_T \mapsto x, v_D \mapsto L\}\\ & \mathit{eff}(\text{load x\_}) = \{v_P \mapsto T\}\end{aligned}$$

$$\begin{aligned}\text{(unload x\_):}\quad & \mathit{pre}(\text{unload x\_}) = \{v_P \mapsto T, v_T \mapsto x, v_D \mapsto L\}\\ & \mathit{eff}(\text{load x\_}) = \{v_P \mapsto x\}\end{aligned}$$

...and start and end surrogate actions for 'driver-loading':

$$\begin{aligned}(\text{driver-loading}^0_{\vdash}):\quad & \mathit{pre}(\text{driver-loading}^0_{\vdash}) = \{e_{DL} \mapsto 0, v_D \mapsto D\}\\ & \mathit{eff}(\text{driver-loading}^0_{\vdash}) = \{e_{DL} \mapsto 1, v_D \mapsto L\}\\ (\text{driver-loading}^1_{\vdash}):\quad & \mathit{pre}(\text{driver-loading}^1_{\vdash}) = \{e_{DL} \mapsto 1, v_D \mapsto D\}\\ & \mathit{eff}(\text{driver-loading}^1_{\vdash}) = \{e_{DL} \mapsto 2, v_D \mapsto L\}\\ (\text{driver-loading}^2_{\vdash}):\quad & \mathit{pre}(\text{driver-loading}^2_{\vdash}) = \{e_{DL} \mapsto 2, v_D \mapsto D\}\\ & \mathit{eff}(\text{driver-loading}^2_{\vdash}) = \{v_D \mapsto L\}\end{aligned}$$

$$\begin{aligned}(\text{driver-loading}^0_{\dashv}):\quad & \mathit{pre}(\text{driver-loading}^0_{\dashv}) = \{e_{DL} \mapsto 1\}\\ & \mathit{eff}(\text{driver-loading}^0_{\dashv}) = \{e_{DL} \mapsto 0, v_D \mapsto D\}\\ (\text{driver-loading}^1_{\dashv}):\quad & \mathit{pre}(\text{driver-loading}^1_{\dashv}) = \{e_{DL} \mapsto 2\}\\ & \mathit{eff}(\text{driver-loading}^1_{\dashv}) = \{e_{DL} \mapsto 1, v_D \mapsto D\}\\ (\text{driver-loading}^2_{\dashv}):\quad & \mathit{pre}(\text{driver-loading}^2_{\dashv}) = \{e_{DL} \mapsto 2\}\\ & \mathit{eff}(\text{driver-loading}^2_{\dashv}) = \{v_D \mapsto D\}\end{aligned}$$

The state transition system for this classical surrogate task is shown in Figure 1. To aid visual clarity, four groupings of nodes are shown, each corresponding to the value of v_P. The topmost grouping corresponds to $v_P \mapsto C$ – its goal value – hence the three goal states are those where $v_P \mapsto C$ and $e_{VD} = 0$, i.e. no actions are executing.

Each of these groupings has a similar structure. The three inner nodes corresponding to states where 'driver-loading' is not executing ($v_D \mapsto D, e_{DL} \mapsto 0$); hence the classical surrogates of 'drive' actions can be applied. The three outer nodes corresponding to states where 'driver-loading' is executing ($v_D \mapsto L, e_{DL} \mapsto 1$), from which the available transitions are either (i) to end 'driver-loading', i.e. apply (driver-loading$^0_{\dashv}$)', returning to an inner node; (ii) to load the package onto the truck (*iff* the package and truck are in the same place); or (iii) to unload the package from the truck.

While the state transition of the classical surrogate task captures the logical semantics of the original temporal planning task, there is no temporal information as of yet. Hence, we now return to temporal matters in the general case, and will later re-visit how this applies to our worked example.

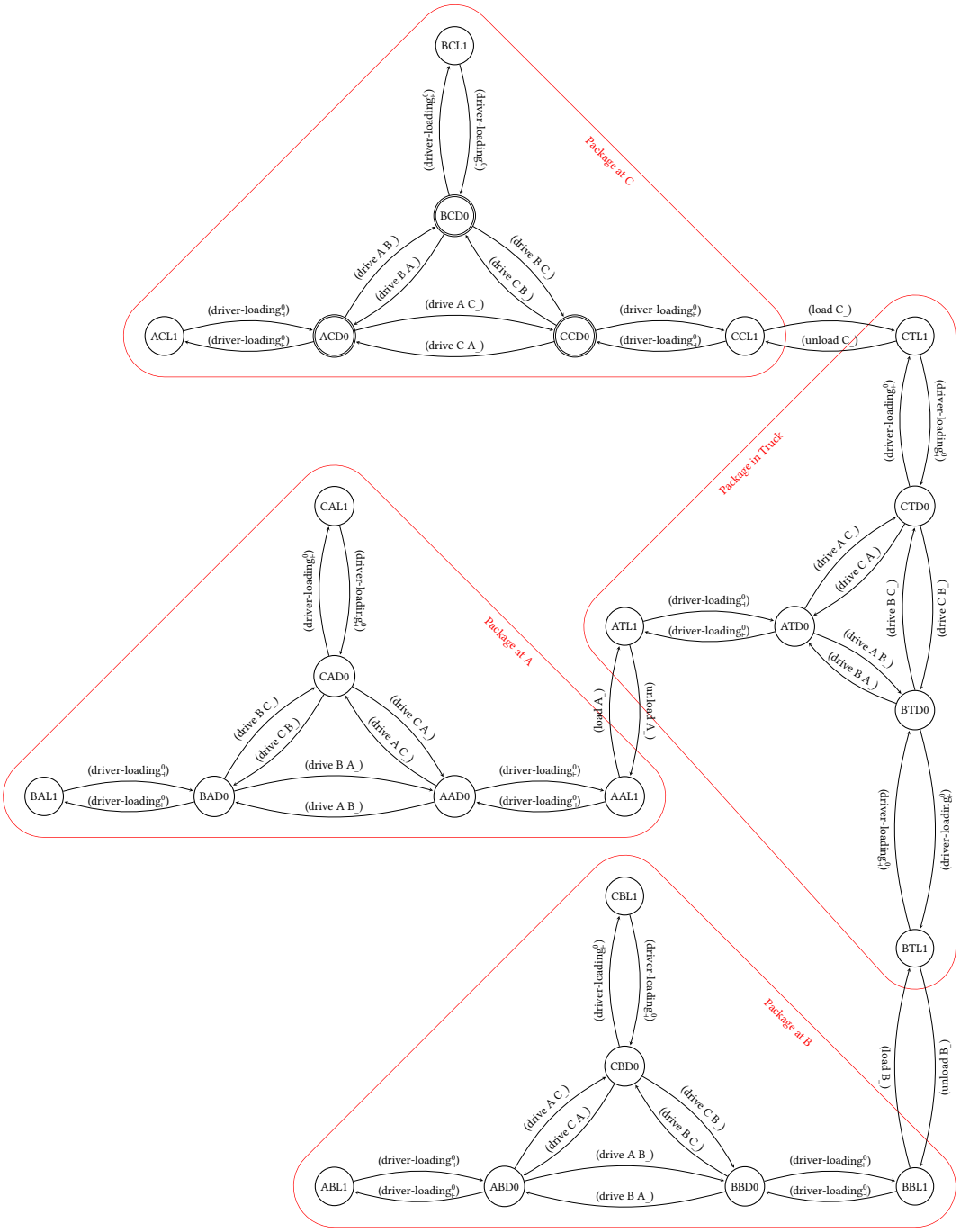


Fig. 1. State Transition System for the Worked Example. Nodes are labeled, in order, with the values of the variables $[v_T, v_P, v_D, e_{DL}]$.

.

4.2 Action Scheduling and *use/chg* Propagation

In the previous section we showed how to obtain a classical surrogate of a temporal planning task. Merge-and-shrink abstraction techniques can then be applied to such classical surrogates. However, building heuristics based on these abstractions requires new methods for action scheduling and the propagation of temporal variables in the abstract space, which we will describe next.

To recap, at some point during search we arrive at a search state $\mathtt{s}$, and we want to compute a lower bound on the time a goal state can be achieved from $\mathtt{s}$ using abstraction Θ. This goal-makespan estimate is a function of the temporal constraints T of the plan P to reach $\mathtt{s}$, which take the form of an STP. Rather than attempt to build an abstraction heuristic that has an embedded STP, we instead build a heuristic as a function of the *use/chg* variables introduced in Definition 9. We will call these functions *goal-makespan formulas*. In order to build them we need to be able to propagate *use/chg* variables from states their neighbors (and eventually from goal states to all other states).

We assume an abstraction is defined over a subset of variables $V \subseteq \mathcal{V}$ and a subset of the execution-counting variables $E \subseteq \mathcal{E}$. We then define variables $use(s, v)$ and $chg(s, v)$ for each $v \in \{V \cup E\}$. We will first set out how these are used, and changes to them are propagated, before returning later to how they relate to concrete lower bound estimates.

If in an abstract state $s \in S$, a classical surrogate action a_-, $a^0_\vdash$, $a^1_\vdash$ or $a^2_\vdash \in \hat{O}$ is applicable, then this corresponds to being able to start a temporal action a. The time at which one of these surrogate actions can be applied is a function of the *use/chg* times of the variables which the action affects and depends on. Conceptually, in line with the search space representation of the non-abstract task, we can use the *use/chg* times to construct a small Simple Temporal Problem (STP) reflecting the constraints that would be applied to $a_\vdash$ (and its corresponding end, $a_\dashv$) were it applied. This yields the following STP, whose variables correspond to the timestamps of $a_\vdash$, $a_\dashv$, the *use* and *chg* variables, and the special variable Z representing time zero; and whose constraints reflect how $a_\vdash$ and $a_\dashv$ must be ordered with respect to these:

Variables: $Z, t(a_\vdash), t(a_\dashv), \forall_{v_i \in V} use(s, v_i), \forall_{v_i \in V} chg(s, v_i)$

Constraints:

$$\epsilon \leq t(a_\vdash) - Z$$

$$\forall_{v_i \in V} \quad 0 \leq use(s, v_i) - Z \qquad \text{(i)}$$

$$\forall_{v_i \in V} \quad 0 \leq chg(s, v_i) - Z \qquad \text{(ii)}$$

$$\delta^-(a) \leq t(a_\dashv) - t(a_\vdash) \leq \delta^+(a) \qquad \text{(iii)}$$

$$\forall_{\langle v_i, w_i \rangle \in pre_\vdash(a) : v_i \in V} \quad \epsilon \leq t(a_\vdash) - use(s, v_i)$$

$$\forall_{\langle v_i, w_i \rangle \in pre_\leftrightarrow(a) : v_i \in V} \quad 0 \leq t(a_\vdash) - use(s, v_i)$$

$$\forall_{\langle v_i, w_i \rangle \in eff_\vdash(a) : v_i \in V} \quad \epsilon \leq t(a_\vdash) - chg(s, v_i)$$

$$\forall_{\langle v_i, w_i \rangle \in pre_\dashv(a) : v_i \in V} \quad \epsilon \leq t(a_\dashv) - use(s, v_i) \qquad \text{(iv)}$$

$$\forall_{\langle v_i, w_i \rangle \in eff_\dashv(a) : v_i \in V} \quad \epsilon \leq t(a_\dashv) - chg(s, v_i) \qquad \text{(v)}$$

Rearranging (iii) we get $t(a_\dashv) \leq t(a_\vdash) + \delta^+(a)$. Substituting this into (iv) and (v), thereby removing the variable $t(a_\dashv)$; and also removing constraint (iii); yields:

Variables: $Z, t(a_\vdash), \bigwedge_{v_i \in V} use(s, v_i), \bigwedge_{v_i \in V} chg(s, v_i)$

Constraints:

$$\begin{aligned}
& & \epsilon \leq\ & t(a_\vdash) - Z & \\
& \bigwedge_{v_i \in V} & 0 \leq\ & use(s, v_i) - Z & \text{(i)} \\
& \bigwedge_{v_i \in V} & 0 \leq\ & chg(s, v_i) - Z & \text{(ii)} \\
& \bigwedge_{\langle v_i, w_i \rangle \in pre_\vdash(a) : v_i \in V} & \epsilon \leq\ & t(a_\vdash) - use(s, v_i) & \\
& \bigwedge_{\langle v_i, w_i \rangle \in pre_\leftrightarrow(a) : v_i \in V} & 0 \leq\ & t(a_\vdash) - use(s, v_i) & \\
& \bigwedge_{\langle v_i, w_i \rangle \in eff_\vdash(a) : v_i \in V} & \epsilon \leq\ & t(a_\vdash) - chg(s, v_i) & \\
& \bigwedge_{\langle v_i, w_i \rangle \in pre_\dashv(a) : v_i \in V} & \epsilon - \delta^+(a) \leq\ & t(a_\vdash) - use(s, v_i) & \\
& \bigwedge_{\langle v_i, w_i \rangle \in eff_\dashv(a) : v_i \in V} & \epsilon - \delta^+(a) \leq\ & t(a_\vdash) - chg(s, v_i) &
\end{aligned}$$

The remaining highlighted constraints, (i) and (ii), are necessary but relatively uninteresting: all they stipulate is that *use/chg* must occur after step zero, but without saying by how much. The actual STP to be solved is hence a function in terms of the values of the *use/chg* variables in $\mathtt{s}$, and solving the STP while accounting for these, to find the minimum amount of time between Z and $a_\vdash$, will return the earliest $a_\vdash$ can occur. We can represent this function as follows:

Definition 18. *$start\_min\_V_\Theta(a, s)$ is a function defining the earliest time a classical surrogate action $a_\dashv$, $a^0_\vdash$, $a^1_\vdash$ or $a^2_\vdash \in \hat{O}$ (derived from temporal action a) can start in state s in an abstraction Θ, due to its preconditions and effects on state variables V:*

$$\begin{aligned}
& start_min_V_\Theta(a, s) \\
& = \max\{\ \epsilon, && \text{(after the initial state)} \\
& \quad \max_{\langle v_i, w_i \rangle \in pre_\vdash(a) : v_i \in V} use(s, v_i) + \epsilon, && \text{(due to its start conditions)} \\
& \quad \max_{\langle v_i, w_i \rangle \in pre_\leftrightarrow(a) : v_i \in V} use(s, v_i), && \text{(due to its invariant conditions)} \\
& \quad \max_{\langle v_i, w_i \rangle \in eff_\vdash(a) : v_i \in V} chg(s, v_i) + \epsilon, && \text{(due to its start effects)} \\
& \quad \max_{\langle v_i, w_i \rangle \in pre_\dashv(a) : v_i \in V} use(s, v_i) - \delta^+(a) + \epsilon, && \text{(due to its end conditions: the start can come no earlier than } \delta^+(a) \text{ before the end)} \\
& \quad \max_{\langle v_i, w_i \rangle \in eff_\dashv(a) : v_i \in V} chg(s, v_i) - \delta^+(a) + \epsilon\ \} && \text{(due to its end effects)}
\end{aligned}$$

As this is analogous to an STP that finds the minimum amount of time a must come after time zero, and no constraints are being added beyond those which would be added when searching for a solution to the original temporal planning task, then $start\_min\_V_\Theta(a, s)$ is admissible.

For compression-safe temporal actions, Definition 18 is all that is needed, as it covers the single classical surrogate action $a_\dashv$: there is no associated execution counter variable for $a_\dashv$, so when it can be applied depends only on the variables $v_i \in V$.

For non-compression-safe actions, we also need to consider temporal constraints on their snap actions due to their associated execution counter variables. If we are applying $a^0_{i\vdash}$, this by definition corresponds to starting a_i in a state in which a_i is not currently executing; or in other words, it must be ordered after any previous appearances of a_i in the plan. To capture this, the lower bound on the time at which it can start due to this is $use(e_i)$; unless e_i is not in E, in which case it is zero. On the other hand, if we are applying $a^1_{i\vdash}$ or $a^2_{i\vdash}$ – starting a again, while it is already executing – no such ordering is needed. With this in mind, we can define the time at which any action can start, as follows:

Definition 19. *$start\_min_\Theta(a, s)$ is a function defining the earliest time a classical surrogate action a_-, $a^0_\vdash$, $a^1_\vdash$ or $a^2_\vdash \in \hat{O}$ (derived from temporal action a) can start in state s in an abstraction Θ, due to its preconditions and effects on state variables V, and any necessary orderings due to its associated execution counter variable (if one is to be had):*

$$start\_min_\Theta(a, s) = \max\{start\_min\_V_\Theta(a, s), \begin{cases} use(s, e_i) & \text{if } a = a^0_{i\vdash} \wedge e_i \in E \\ 0 & \text{otherwise} \end{cases} \}$$

We must also consider when one of $a^0_{i\dashv}$, $a^1_{i\dashv}$ or $a^2_{i\dashv} \in \hat{O}$ are applicable in an abstract state s; i.e. when a_i can end. If the abstraction has retained the execution counter variable e_i, we can say the end must come after its start. If not, it must come at least its minimum duration after time zero:

$$after\_start(a_{i\dashv}, s) = \begin{cases} use(s, e_i) & e_i \in E \\ \delta^-(a_i) & \text{otherwise} \end{cases}$$

Additionally, we must also consider relevant preconditions and effects:

Definition 20. *$start\_min_\Theta(a_{i\dashv}, s)$ is a function defining the earliest time a classical surrogate action $a_{i\dashv} \in \{a^0_{i\dashv}, a^1_{i\dashv}, a^2_{i\dashv}\} \subset \hat{O}$ (derived from the end of temporal action a_i) can occur in state s in an abstraction Θ:*

$$\begin{aligned} &start\_min_\Theta(a_{i\dashv}, s) \\ &= \max\{ \quad after\_start(a_{i\dashv}, s), && \text{(as above)} \\ &\qquad \max_{\langle v_j, w_j\rangle \in pre_{\leftrightarrow}(a) : v_j \in V} use(s, v_j), && \text{(due to its invariant conditions)} \\ &\qquad \max_{\langle v_j, w_j\rangle \in pre_{\dashv}(a) : v_j \in V} use(s, v_j) + \epsilon, && \text{(due to its end conditions)} \\ &\qquad \max_{\langle v_j, w_j\rangle \in eff_{\dashv}(a) : v_j \in V} chg(s, v_j) + \epsilon \ \} && \text{(due to its end effects)} \end{aligned}$$

Implicit in the above is that there is a relaxation of the duration constraints linking the start and end of an action: we do not ensure the end of an action is no later than δ^+ after the start. This is deliberate, as in the general case this would require a full STP.

Having now defined *start_min* for starting and ending actions, we can now formalise how we update *use/chg* as a result of action application. These updates are done instead of maintaining and updating VP and V^{eff}, and in turn maintaining an STP, as would be the case in search proper. If we are in abstract state s, and apply a classical surrogate action a at its earliest possible time $start\_min_\Theta(a, s)$, the *use* and *chg* variables in the next state s' are determined by those in state s, and the preconditions and effects of a:

$$\forall v_i \in V \ use(s', v_i) = propagate\_use_\Theta(a, s, v_i)$$
$$\forall v_i \in V \ chg(s', v_i) = propagate\_chg_\Theta(a, s, v_i)$$

$$\forall e_k \in E\ use(s', e_k) = propagate\_use_\Theta(a, s, e_k)$$
$$\forall e_k \in E\ chg(s', e_k) = propagate\_chg_\Theta(a, s, e_k)$$

We first consider the case of applying a surrogate action a_- due to a compression-safe temporal action a. For these, *use* and *chg* are updated with respect to applying the start of a; allowing at least $\delta^-(a)$ amount of time; and then applying its end. The "propagate" functions for variables in V are then:

Definition 21. *If a surrogate action $a_- \in \hat{O}$ (derived from temporal action $a \in O_c$) is applied in a state s in an abstraction Θ, reaching state s', the functions $propagate\_use_\Theta(a_-, s, v_i)$ and $propagate\_chg_\Theta(a_-, s, v_i)$ for variable $v_i \in V$ define $use(s', v_i)$ and $chg(s', v_i)$ conditionally over the following cases:*

- *if a does not condition on or affect v_i:*
$$use(s', v_i) = use(s, v_i)$$
$$chg(s', v_i) = chg(s, v_i)$$
- *if a changes the value of v_i at its end, then regardless of any other effects/conditions on v_i:*
$$use(s', v_i) = start\_min_\Theta(a, s) + \delta^-(a)$$
$$chg(s', v_i) = start\_min_\Theta(a, s) + \delta^-(a)$$
- *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:*
$$use(s', v_i) = start\_min_\Theta(a, s)$$
$$chg(s', v_i) = start\_min_\Theta(a, s) + \delta^-(a)$$
- *if a changes the value of v_i at its start (but* not *the end), and then has an 'invariant' condition on v_i:*
$$use(s', v_i) = start\_min_\Theta(a, s)$$
$$chg(s', v_i) = start\_min_\Theta(a, s) + \delta^-(a) - \epsilon$$
- *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:*
$$use(s', v_i) = start\_min_\Theta(a, s)$$
$$chg(s', v_i) = start\_min_\Theta(a, s)$$
- *if a never changes the value of v_i, but has an 'end' (and optionally a 'start' and/or an 'invariant' condition) on v_i:*
$$use(s', v_i) = use(s, v_i)$$
$$chg(s', v_i) = \max\{chg(s, v_i), start\_min_\Theta(a, s) + \delta^-(a)\}$$
- *if a never changes the value of v_i, but has an 'invariant' condition (and optionally a 'start' condition) on v_i:*
$$use(s', v_i) = use(s, v_i)$$
$$chg(s', v_i) = \max\{chg(s, v_i), start\_min_\Theta(a, s) + \delta^-(a) - \epsilon\}$$
- *if a never changes the value of v_i, but has a 'start' condition on v_i:*
$$use(s', v_i) = use(s, v_i)$$
$$chg(s', v_i) = \max\{chg(s, v_i), start\_min_\Theta(a, s)\}$$

Additionally, the "propagate" functions for variables in E are:

Definition 22. *If a surrogate action $a_{-} \in \hat{O}$ (derived from temporal action $a \in O_c$) is applied in a state s in an abstraction Θ, reaching state s', $propagate\_use_\Theta(a_{\_}, s, e_k)$ and $propagate\_chg_\Theta(a_{\_}, s, e_k)$ for any variable $e_k \in E$ define $use(s', e_k)$ and $chg(s', e_k)$ as being unchanged from s:*

$$use(s', e_k) = use(s, e_k)$$

$$chg(s', e_k) = chg(s, e_k)$$

For surrogate actions due to non-compression-safe temporal actions, we need to take care to update not just *use* and *chg* due to their preconditions and effects on variables in V, but also due to a variable in E. For each $a_{i\vdash} \in \{a^0_{i\vdash}, a^1_{i\vdash}, a^2_{i\vdash}\} \subset \hat{O}$, due to starting some temporal action $a_i \in O_n$, we must meet its start conditions, apply its start effects, and meet its invariants; but we do not consider any end conditions or effects, as they are reserved for $a_{\dashv}$. The propagate functions for each variable $v_j \in V$ are hence defined as follows:

Definition 23. *If a surrogate action $a_{i\vdash} \in \{a^0_{i\vdash}, a^1_{i\vdash}, a^2_{i\vdash}\} \subset \hat{O}$ (derived from temporal action $a_i \in O_n$) is applied in a state s in an abstraction Θ, reaching state s', the functions $propagate\_use_\Theta(a_{i\vdash}, s, v_j)$ and $propagate\_chg_\Theta(a_{i\vdash}, s, v_j)$ for variable $v_j \in V$ define $use(s', v_j)$ and $chg(s', v_j)$ conditionally over the following cases:*

- *if $pre_\vdash(a_i)$, $pre_\leftrightarrow(a_i)$ and $eff_\vdash(a_i)$ do not refer to v_j:*

$$use(s', v_j) = use(s, v_j)$$

$$chg(s', v_j) = chg(s, v_j)$$

- *if $eff_\vdash(a_i)$ changes the value of v_j, and then an 'invariant' condition $pre_\leftrightarrow(a_i)$ refers to v_j:*

$$use(s', v_j) = start\_min_\Theta(a_i, s)$$

$$chg(s', v_j) = start\_min_\Theta(a_i, s) + \delta^-(a_i) - \epsilon$$

- *if $eff_\vdash(a_i)$ changes the value of v_j, and no 'invariant' condition $pre_\leftrightarrow(a_i)$ refers to v_j:*

$$use(s', v_j) = start\_min_\Theta(a_i, s)$$

$$chg(s', v_j) = start\_min_\Theta(a_i, s)$$

- *if $eff_\vdash(a_i)$ does not change the value of v_j, but $pre_\leftrightarrow(a_i)$ (and optionally $pre_\vdash(a_i)$) refer to v_j:*

$$use(s', v_j) = use(s, v_j)$$

$$chg(s', v_j) = \max\{chg(s, v_j), start\_min_\Theta(a_i, s) + \delta^-(a_i) - \epsilon\}$$

- *if $eff_\vdash(a_i)$ does not change the value of v_j, but $pre_\vdash(a_i)$ refers to v_j:*

$$use(s', v_j) = use(s, v_j)$$

$$chg(s', v_j) = \max\{chg(s, v_j), start\_min_\Theta(a_i, s)\}$$

For a surrogate action $a^0_{i\vdash}$, we are starting temporal action a_i in an abstract state where it is not currently executing, so its end cannot occur until at least $\delta^-(a_i)$ time units have passed. Hence, we update *use* and *chg* for its execution-counter variable e_i, and leave the remainder unchanged:

Definition 24. *If a surrogate action $a^0_{i\vdash} \in \hat{O}$ (derived from temporal action $a_i \in O_n$) is applied in a state s in an abstraction Θ, reaching state s', the functions $propagate\_use_\Theta(a^0_{i\vdash}, s, e_k)$ and $propagate\_chg_\Theta(a^0_{i\vdash}, s, e_k)$ for each execution counter variable $e_k \in E$ define $use(s', e_k)$ and $chg(s', e_k)$ as:*

$$use(s', e_k) = \begin{cases} start\_min_\Theta(a_i, s) + \delta^-(a_i) & \text{if } i = k \\ use(s, e_k) & \text{otherwise} \end{cases}$$

$$chg(s', e_k) = \begin{cases} start\_min_\Theta(a_i, s) + \delta^-(a_i) & \text{if } i = k \\ chg(s, e_k) & \text{otherwise} \end{cases}$$

For surrogate actions $a^1_{i\vdash}$ and $a^2_{i\vdash}$, we are starting temporal action a_i in an abstract state where it is already executing. Cautiously, we then update *use* and *chg* for its execution variable to be the earliest of when the incumbent instance of a_i could end, or the newly starting instance:

Definition 25. *If a surrogate action $a_{i\vdash} \in \{a^1_{i\vdash}, a^2_{i\vdash}\} \subset \hat{O}$ (derived from temporal action $a_i \in O_n$) is applied in a state s in an abstraction Θ, reaching state s', the functions $propagate\_use_\Theta(a_{i\vdash}, s, e_i)$ and $propagate\_chg_\Theta(a_{i\vdash}, s, v_i)$ for each execution counter variable $e_k \in E$ define $use(s', e_k)$ and $chg(s', e_k)$ as:*

$$use(s', e_k) = \begin{cases} \min\{use(s, e_k), start\_min_\Theta(a_i, s) + \delta^-(a_i)\} & \text{if } i = k \\ use(s, e_k) & \text{otherwise} \end{cases}$$

$$chg(s', e_k) = \begin{cases} \min\{chg(s, e_k), start\_min_\Theta(a_i, s) + \delta^-(a_i)\} & \text{if } i = k \\ chg(s, e_k) & \text{otherwise} \end{cases}$$

For each surrogate action $a_{i\dashv} \in \{a^0_{i\dashv}, a^1_{i\dashv}, a^2_{i\dashv}\}$, due to ending some temporal action a_i, we must consider that its invariants must hold up until $a_{i\dashv}$ is applied; its end conditions must then be true; and its effects would then occur. Hence:

Definition 26. *If a surrogate action $a_{i\dashv} \in \{a^0_{i\dashv}, a^1_{i\dashv}, a^2_{i\dashv}\} \subset \hat{O}$ (derived from temporal action $a_i \in O_n$) is applied in a state s in an abstraction Θ, reaching state s', the functions $propagate\_use_\Theta(a_{i\dashv}, s, v_j)$ and $propagate\_chg_\Theta(a_{i\dashv}, s, v_j)$ for variable $v_j \in V$ define $use(s', v_j)$ and $chg(s', v_j)$ conditionally over the following cases:*

- *if $eff_\dashv(a_i)$ changes the value of v_j:*

$$use(s', v_j) = start\_min_\Theta(a_{i\dashv}, s)$$

$$chg(s', v_j) = start\_min_\Theta(a_{i\dashv}, s)$$

- *if $eff_\dashv(a_i)$ does not change the value of v_j, but $pre_\dashv(a_i)$ refers to v_j:*

$$use(s', v_j) = use(s, v_j)$$

$$chg(s', v_j) = \max\{chg(s, v_j), start\_min_\Theta(a_{i\dashv}, s)\}$$

- *if $eff_\dashv(a_i)$ does not change the value of v_j, but $pre_\leftrightarrow(a_i)$ refers to v_j:*

$$use(s', v_j) = use(s, v_j)$$

$$chg(s', v_j) = \max\{chg(s, v_j), start\_min_\Theta(a_{i\dashv}, s) - \epsilon\}$$

- *if $eff_\dashv(a_i)$ does not change the value of v_j, and neither $pre_\leftrightarrow(a_i)$ nor $pre_\dashv(a_i)$ refer to v_j:*

$$use(s', v_j) = use(s, v_j)$$

$$chg(s', v_j) = chg(s, v_j).$$

Note that unlike starting actions, where $use(s', e_i)$ and $chg(s', e_i)$ are updated to delay the ends of an action a_i to after its start, we do not need to change their values when ending actions:

Definition 27. *If a surrogate action $a_{i\dashv} \in \{a^0_{i\dashv}, a^1_{i\dashv}, a^2_{i\dashv}\} \subset \hat{O}$ (derived from temporal action $a_i \in O_n$) is applied in a state s in an abstraction Θ, reaching state s', the functions $propagate\_use_\Theta(a_{i\dashv}, s, e_k)$ and $propagate\_chg_\Theta(a_{i\dashv}, s, e_k)$ for each execution counter variable $e_k \in E$ define $use(s', e_k)$ and $chg(s', e_k)$ as being unchanged with respect to s:*

$$use(s', e_k) = use(s, e_k)$$
$$chg(s', e_k) = chg(s, e_k)$$

With these definitions, *use* and *chg* can then be propagated in turn over sequences of actions. If a sequence of actions $a_1, ..., a_k$ is executed from state $s \in S$, leading to the sequence of states $s, s_1, ..., s_k$, then the makespan of this sequence is given by:

$$\max_{v \in \{V \cup E\}} use(s_k, v).$$

Through propagation, this function is ultimately defined in terms of the *use* and *chg* variables in s. If s_k is a goal state, then we now have a function for determining the the makespan of reaching the goal from s via the application of $a_1 ... a_k$.

4.2.1 Propagate functions in the worked example. We now demonstrate the computation of propagation functions in our worked example. Recall the drive action defined in Section 4.1.1 as follows:

(drive x y):
$pre_\vdash$(drive x y) = $\{v_T \mapsto x\}$ — $eff_\vdash$(drive x y) = $\{v_T \mapsto \bot\}$
$pre_\leftrightarrow$(drive x y) = $\{v_D \mapsto D\}$
$pre_\dashv$(drive x y) = $\{\}$ — $eff_\vdash$(drive x y) = $\{v_T \mapsto y\}$
δ^-(drive x y) = 10, δ^+(drive x y) = 10

$start\_min_\Theta$((drive x y _), s), following Definition 19 and including Definition 18, is then:

$start\_min_\Theta$((drive x y _), s)
= max{
- ϵ, *(after the initial state)*
- $use(s, v_T) + \epsilon$, *(c.f. start condition on v_T)*
- $use(s, v_D)$, *(c.f. invariant condition on v_D)*
- $chg(s, v_T) + \epsilon$, *(c.f. start effect on v_T)*
- $chg(s, v_T) - \delta^+$(drive x y) + ϵ *(c.f. end effect on v_T)*
- 0 } *(no execution counter variable)*

Note that the maximization term '$chg(s, v_T) - \delta^+$(drive x y) + ϵ' is redundant in this particular case, as it is dominated by $chg(s, v_T) + \epsilon$. In any case, with this in hand, and recalling that (drive x y) is a compression-safe action, we follow Definition 21 to determine $propagate\_use_\Theta$((drive x y _), s, v_i) and $propagate\_chg_\Theta$((drive x y _), s, v_i) for variables $v_i \in V$:

i) There are no conditions/effects on v_P, hence:
$use(s', v_P) = use(s, v_P)$
$chg(s', v_P) = chg(s, v_P)$

ii) There is an end effect on v_T, hence:
$use(s', v_T) = start\_min_\Theta$((drive x y _), s) + δ^-(drive x y)
$chg(s', v_T) = start\_min_\Theta$((drive x y _), s) + δ^-(drive x y)

iii) There is an invariant on v_D, hence:
$use(s', v_D) = use(s, v_D)$
$chg(s', v_D) = \max\{chg(s, v_D), start\_min_\Theta$((drive x y _), s) + δ^-(drive x y) − $\epsilon\}$

For variables $e_k \in E$, we refer to Definition 22: $propagate\_use_\Theta((\text{drive x y\_}), s, e_k)$ and $propagate\_chg_\Theta((\text{drive x y\_}), s, e_k)$ are no-ops.

For 'load' and 'unload' actions, we follow a similar process – these actions are also compression safe, and follow the same pattern of changing the value of a variable (v_P) over a durative period, while having an invariant on others (v_T, v_D). Hence, $start\_min_\Theta((\text{load x\_}), s)$ is:

$$
\begin{array}{lll}
start\_min_\Theta((\text{load x\_}), s) & & \\
= \max\{ & \epsilon, & \textit{(after the initial state)} \\
& use(s, v_P) + \epsilon, & \textit{(c.f. start condition on } v_P\textit{)} \\
& \max\{use(s, v_T), use(s, v_D)\}, & \textit{(c.f. invariant conditions)} \\
& chg(s, v_P) + \epsilon, & \textit{(c.f. start effect on } v_P\textit{)} \\
& chg(s, v_P) - \delta^+(\text{load x}) + \epsilon & \textit{(c.f. end effect on } v_P\textit{)} \\
& 0 \quad \} & \textit{(no execution counter variable)}
\end{array}
$$

...and for (unload x) is:

$$
\begin{array}{lll}
start\_min_\Theta((\text{unload x\_}), s) & & \\
= \max\{ & \epsilon, & \textit{(after the initial state)} \\
& use(s, v_P) + \epsilon, & \textit{(c.f. start condition on } v_P\textit{)} \\
& \max\{use(s, v_T), use(s, v_D)\}, & \textit{(c.f. invariant conditions)} \\
& chg(s, v_P) + \epsilon, & \textit{(c.f. start effect on } v_P\textit{)} \\
& chg(s, v_P) - \delta^+(\text{unload x}) + \epsilon & \textit{(c.f. end effect on } v_P\textit{)} \\
& 0 \quad \} & \textit{(no execution counter variable)}
\end{array}
$$

Then, *propagate_use* and *propagate_chg* for $v \in V$ again follow Definition 21. First, for (load x)_:

i) There is an end effect on v_P, hence:
$use(s', v_P) = start\_min_\Theta((\text{load x\_}), s) + \delta^-(\text{load x})$
$chg(s', v_P) = start\_min_\Theta((\text{load x\_}), s) + \delta^-(\text{load x})$
ii) There is an invariant on v_T, hence:
$use(s', v_T) = use(s, v_T)$
$chg(s', v_T) = \max\{chg(s, v_T), start\_min_\Theta((\text{load x\_}), s) + \delta^-(\text{load x}) - \epsilon\}$
iii) There is an invariant on v_D, hence:
$use(s', v_D) = use(s, v_D)$
$chg(s', v_D) = \max\{chg(s, v_D), start\_min_\Theta((\text{load x\_}), s) + \delta^-(\text{load x}) - \epsilon\}$

...and for (unload x)_:

i) There is an end effect on v_P, hence:
$use(s', v_P) = start\_min_\Theta((\text{unload x\_}), s) + \delta^-(\text{unload x})$
$chg(s', v_P) = start\_min_\Theta((\text{unload x\_}), s) + \delta^-(\text{unload x})$
ii) There is an invariant on v_T, hence:
$use(s', v_T) = use(s, v_T)$
$chg(s', v_T) = \max\{chg(s, v_T), start\_min_\Theta((\text{unload x\_}), s) + \delta^-(\text{unload x}) - \epsilon\}$
iii) There is an invariant on v_D, hence:
$use(s', v_D) = use(s, v_D)$
$chg(s', v_D) = \max\{chg(s, v_D), start\_min_\Theta((\text{unload x\_}), s) + \delta^-(\text{unload x}) - \epsilon\}$

For variables $e_k \in E$, as in the case of 'drive' actions, we refer to Definition 22.

For '(driver-loading)', as it is not compression safe, we proceed a little differently. Repeating its definition for reference:

$$
\begin{aligned}
\text{(driver-loading):}\quad & \mathit{pre}_{\vdash}(\text{driver-loading}) = \{v_D \mapsto D\} \\
& \mathit{eff}_{\vdash}(\text{driver-loading}) = \{v_D \mapsto L\} \\
& \mathit{pre}_{\leftrightarrow}(\text{driver-loading}) = \{\} \\
& \mathit{pre}_{\dashv}(\text{driver-loading}) = \{\} \\
& \mathit{eff}_{\dashv}(\text{driver-loading}) = \{v_D \mapsto D\} \\
& \delta^{-}(\text{driver-loading}) = 2 \\
& \delta^{+}(\text{driver-loading}) = 2
\end{aligned}
$$

We begin by considering surrogate actions corresponding to *starting* (driver-loading): applying $(\text{driver-loading})^k_{\vdash}$ for $k \in \{0, 1, 2\}$. First, consider $k = 0$: starting the action when it is not currently executing. Definition 19 will prescribe that it can occur no earlier than (i) the earliest feasible time due to its preconditions/effects on state variables V, following Definition 18, and (ii) after the last execution of (driver-loading) completed, as reflected in the *use* value of its execution counter variable e_{DL}:

$$
\begin{aligned}
& \mathit{start\_min}_{\Theta}((\text{driver-loading}^0_{\vdash}), s) \\
= \max\{\ & \epsilon, && \textit{(after the initial state)} \\
& \mathit{use}(s, v_D) + \epsilon, && \textit{(c.f. start condition on } v_D\textit{)} \\
& \mathit{chg}(s, v_D) + \epsilon, && \textit{(c.f. start effect on } v_D\textit{)} \\
& \mathit{chg}(s, v_D) - \delta^{+}(\text{driver-loading}) + \epsilon, && \textit{(c.f. end effect on } v_D\textit{)} \\
& \mathit{use}(s, e_{DL}) && \}\quad \textit{(order w.r.t. } e_{DL}\textit{)}
\end{aligned}
$$

Otherwise, for $k \in \{1, 2\}$, it is not constrained with respect to e_{DL} and hence is:

$$
\begin{aligned}
& \mathit{start\_min}_{\Theta}((\text{driver-loading}^k_{\vdash}), s) \\
= \max\{\ & \epsilon, && \textit{(after the initial state)} \\
& \mathit{use}(s, v_D) + \epsilon, && \textit{(c.f. start condition on } v_D\textit{)} \\
& \mathit{chg}(s, v_D) + \epsilon, && \textit{(c.f. start effect on } v_D\textit{)} \\
& \mathit{chg}(s, v_D) - \delta^{+}(\text{driver-loading}) + \epsilon, && \textit{(c.f. end effect on } v_D\textit{)} \\
& 0 && \}
\end{aligned}
$$

Definition 23 uses $\mathit{start\_min}_{\Theta}((\text{driver-loading}^k_{\vdash}), s)$ to prescribe $\mathit{propagate\_use}_{\Theta}((\text{driver-loading}^k_{\vdash}), s, v)$ and $\mathit{propagate\_chg}_{\Theta}((\text{driver-loading}^k_{\vdash}), s, v)$ for $k \in \{0, 1, 2\}$ and $v \in V$ as follows:

i) There are no preconditions or effects on v_P, hence:
$\mathit{use}(s', v_P) = \mathit{use}(s, v_P)$
$\mathit{chg}(s', v_P) = \mathit{chg}(s, v_P)$

ii) There are no preconditions or effects on v_T, hence:
$\mathit{use}(s', v_T) = \mathit{use}(s, v_T)$
$\mathit{chg}(s', v_T) = \mathit{chg}(s, v_T)$

iii) $\mathit{eff}_{\vdash}$(driver-loading) changes v_D, hence:
$\mathit{use}(s', v_D) = \mathit{start\_min}_{\Theta}((\text{driver-loading}^k_{\vdash}), s)$
$\mathit{chg}(s', v_D) = \mathit{start\_min}_{\Theta}((\text{driver-loading}^k_{\vdash}), s)$

We must also consider the execution counter variable e_{DL}. Here, again, the treatment of starting the action when it is not currently executing ($k = 0$) differs from when it is already executing. For $k = 0$, from Definition 24, $\mathit{propagate\_use}_{\Theta}((\text{driver-loading})^0_{\vdash}, s, e_{DL})$ and $\mathit{propagate\_chg}_{\Theta}((\text{driver-loading})^0_{\vdash}, s, e_{DL})$ are defined respectively as:

$$\mathit{use}(s', e_{DL}) = \mathit{start\_min}_{\Theta}((\text{driver-loading}^0_{\vdash}), s) + \delta^{-}(\text{driver-loading})$$

$$chg(s', e_{DL}) = start\_min_\Theta((\text{driver-loading}^0_\vdash), s) + \delta^-(\text{driver-loading})$$

For $k \in \{1, 2\}$, from Definition 25, $propagate\_use_\Theta((\text{driver-loading})^k_\vdash, s, e_{DL})$ and $propagate\_chg_\Theta((\text{driver-loading})^k_\vdash, s, e_{DL})$ are defined respectively as:

$$use(s', e_{DL}) = \min\{use(s, e_{DL}), start\_min_\Theta((\text{driver-loading}^k_\vdash), s) + \delta^-(\text{driver-loading})\}$$

$$chg(s', e_{DL}) = \min\{chg(s, e_{DL}), start\_min_\Theta((\text{driver-loading}^k_\vdash), s) + \delta^-(\text{driver-loading})\}$$

Having now considered starting (driver-loading), we finally consider *ending* it. First, following Definition 20, we obtain $start\_min_\Theta((\text{driver-loading})^k_\dashv, s)$ for $k \in \{0, 1, 2\}$:

$start\_min_\Theta((\text{driver-loading}^k_\dashv), s)$
$= \max\{ \quad use(s, e_{DL}) \quad$ *(after its start)*
$\quad\quad\quad chg(s, v_D) + \epsilon \quad \}$ *(c.f. end effect on v_D)*

Then, this can be used with Definition 26 to obtain $propagate\_use_\Theta((\text{driver-loading}^k_\dashv), s, v)$ and $propagate\_chg_\Theta((\text{driver-loading}^k_\dashv), s, v)$ for $v \in V$ as follows:

i) There are no preconditions or effects on v_P, hence:
$use(s', v_P) = use(s, v_P)$
$chg(s', v_P) = chg(s, v_P)$
ii) There are no preconditions or effects on v_T, hence:
$use(s', v_T) = use(s, v_T)$
$chg(s', v_T) = chg(s, v_T)$
iii) $eff_\dashv(\text{driver-loading})$ changes v_D, hence:
$use(s', v_D) = start\_min_\Theta((\text{driver-loading}^k_\dashv), s)$
$chg(s', v_D) = start\_min_\Theta((\text{driver-loading}^k_\dashv), s)$

Finally, following Definition 27, $propagate\_use_\Theta((\text{driver-loading}^k_\dashv), s, e)$ and $propagate\_chg_\Theta((\text{driver-loading}^k_\dashv), s, e)$ are no-ops for execution counter variables $e \in E$.

4.3 Temporal M&S Heuristic

In the previous section showed how to compute the time at which surrogate actions can be applied, and how to propagate *use* and *chg* variables between abstract states. We now leverage these to obtain a heuristic for an abstraction.

We begin with some preliminaries:

Definition 28. *A temporally extended abstract transition graph for the classical surrogate of a temporal SAS$^+$ planning task is a tuple $\Theta = \langle V \cup E, S, L, A, s_0, S_G\rangle$, where $V \subseteq \mathcal{V}$ is a subset of the variables in $\mathcal{V}$, $E \subseteq \mathcal{E}$ is a subset of the variables in $\mathcal{E}$, S is a state space of variable assignments $S = (S_V \times S_E) \subseteq (\mathcal{S}_\mathcal{V} \times \mathcal{S}_\mathcal{E})$, L is a finite set of transition labels (all those needed for $\hat{O}$), $A \subseteq S \times L \times S$ 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 labeled $l \in L \subseteq \hat{O}$ from $s \in S$ to $d \in S$ if surrogate action l is applicable in s and applying the action results in state d.*

The synchronized product of two temporally-extended transition graphs is $\Theta \otimes \Theta' = \{V \cup E \cup V' \cup E', S \times S', L, A \otimes A', \langle s_0, s'_0\rangle, S_G \times S'_G\}$. Transitions $A \otimes A'$ are 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 $\langle s', l, d'\rangle \in A'$.

Note that in problems where all actions are compression safe, $E = \emptyset$, and A reduces to the same as in the classical planning setting (each temporal action yields a single action in the classical surrogate).

Definition 29. *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, Haslum, Hoffmann, et al. 2007; Nissim et al. 2011)) define a heuristic in terms of cost values, while here, for the purposes of temporal planning, we define it as minimizing (timestamp) makespan.

In general the makespan estimate provided by our heuristic function is pre-computed 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*". Roughly, as we will see, to obtain goal-makespan formulas for all states of an abstraction we use the *use/chg* propagation rules introduced in Section 4.2 to back-propagate *use/chg* variables (and thus goal-makespan expressions) from goal states to all other states.

4.3.1 *Goal-makespan Formulas.*

Definition 30. *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 $s_G \in S_G$, $tg^{\Theta}(s_G) := \max\{use(s_G, v_i) | v_i \text{ is a goal variable in } \Theta\}$, 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 $tg^{\Theta} = \infty$.

4.3.2 *Computing Goal-makespan Formulas.*

We compute the goal-makespan formulas for all states in an abstract transition graph through breadth-first-search, re-opening states when different paths are found to ensure all non-dominated paths from each state to each goal state are kept.

This algorithm is outlined in Algorithm 3. From lines 2 to 7, the goal-makespan formulas of states are initialised. For goal states, these can be safely set to be 'the latest makespan of any goal variable'. For non-goal states, these are initialised to infinity, as no path to a goal state has yet been found.

The main breadth-first-search loop begins at line 8. Repeatedly, for some state s', it considers each of its neighbors s (where there is an edge from s to s'), to see if this gives another non-dominated path from s to a goal state via s'. Briefly:

- "MakespanThroughNeighbor()" computes the new goal-makespan formula for a neighbor s to a goal state via s';
- A formula taking the minimum of this and the prior goal-makespan formula for s is generated;
- The "Simplify()" function makes sure a formula is not redundant and therefore eliminates loops and definitely-longer paths from the goal-makespan formula – it keeps all non-dominated paths, hence all candidates to be the lowest-makespan path;
- If the result is a new goal-makespan formula for s, it contains another non-dominated path. s is then enqueued for expansion (or re-expansion) on a subsequent iteration.

When running this algorithm, 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. 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.

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

"MakespanThroughNeighbor(s, a, s')" computes a candidate 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

Algorithm 3: Pre-computing tg^{Θ} formulas

Data: Abstraction Θ, state space S, transitions A
Result: Each abstract state $s \in S$ has its formula $tg^{\Theta}(s)$ updated to reflect paths to goals

```
Q ← [] ;
foreach state s ∈ S do
    if s is a goal state then
        tg^Θ(s) ← Formula( max{use(s, v_i) | v_i is a goal variable}) ;
        append s to Q;
    else
        tg^Θ(s) ← ∞;
while Q ≠ ∅ do
    pop s' off the front of Q;
    foreach edge ⟨s, a, s'⟩ ∈ A' do
        new_tg ← MakespanThroughNeighbor(s, a, s');
        new_tg' ← Formula( min{tg^Θ(s), new_tg});
        new_tg' ← Simplify(new_tg');
        if new_tg' ≠ tg^Θ(s) then
            tg^Θ(s) ← new_tg';
            append s to Q;
```

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:

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

Hence, a candidate goal-makespan formula for s will be obtained by taking formula $tg^{\Theta}(s')$ and replacing variable $use(s', v_i)$ with the expression given by $propagate\_use_{\Theta}(a, s, v_i)$—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, "Simplify(new_tg')" manipulates formulas to remove redundancy, i.e. to remove either $tg^{\Theta}(s)$ or new_tg from $new\_tg' = \min\{tg^{\Theta}(s), new\_tg\}$, in case one dominates the other.

To check for domination between two formulas in an expression of the type $\min\{tg, tg'\}$, let each of these be first converted into the following canonical form:

$$tg := \max\{ \quad u_1.use(s, v_1) + t^u_1, ..., u_{|V|+|E|}.use(s, v_{|V|+|E|}) + t^u_{|V|+|E|}, \\ c_1.chg(s, v_1) + t^c_1, ..., c_{|V|+|E|}.chg(s, v_{|V|+|E|}) + t^c_{|V|+|E|}\}$$

where $u_i \in \{0, 1\}$ and $c_i \in \{0, 1\}$ are indicator variables, and t^u_i, t^c_i are constants[7]. Then, domination between such formulas can be computed using the following definition.

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

Definition 31. *One goal-makespan formula tg of the above form dominates (is definitely lower or equal than) another tg′, i.e. tg ≤ tg′, if one of the following conditions holds:*

- $max_{i=1,\ldots,|V|+|E|}(max(t_i^{u'}, t_i^{c'})) = \infty$
  (i.e. tg′ has an infinite constant, and therefore the max of all its terms is infinite)
- $(\forall_i\{u_i = 0 \wedge c_i = 0 \wedge u'_i = 0 \wedge c'_i = 0\}) \wedge$
  $max_{i=1,\ldots,|V|+|E|}(max(t_i^{u}, t_i^{c})) \leq max_{i=1,\ldots,|V|+|E|}(max(t_i^{u'}, t_i^{c'}))$
  (i.e. tg and tg′ are equal to a max of constants, and the highest constant of tg is lower or equal than that of tg′)
- $\forall_i\ \{u_i \leq u'_i \wedge c_i \leq c'_i \wedge t_i^u \leq t_i^{u'} \wedge t_i^u \leq t_i^{c'} \wedge t_i^c \leq t_i^{c'}\}$
  (i.e. all constants and u_i, c_i indicator variables of tg are pairwise lower or equal than those of tg′)

4.3.3 Computing (some) goal makespan formulas for the worked example. We will now demonstrate how (some) of the goal makespan formulas are computed for the state transition system of the worked example, as shown in Figure 1.

First, in the initialization phase of Algorithm 3 (lines 2– 7):

- For each goal state $s \in \{\text{ACD0}, \text{BCD0}, \text{CCD0}\}$, recalling v_P and e_{DL} are the variables on which goals are specified (resp. the package is at C, and no actions are executing), $tg^{\Theta}(s) \leftarrow \text{Formula}(\max\{use(s, v_P), use(s, e_{DL})\})$. Q then comprises these three goal states; the order is arbitrary, but for the sake of example, we assume $Q = [\text{ACD0}, \text{BCD0}, \text{CCD0}]$.
- For any non-goal state s, $tg^{\Theta}(s) \leftarrow \infty$.

The main loop at line 8 then begins, with each iteration proceeding as follows.

0) ACD0 is popped off Q *Remaining:* $Q = [\text{BCD0}, \text{CCD0}]$

There are three edges into ACD0. Two are inconsequential – $\langle \text{BCD0}, (\text{drive B A})\_, \text{ACD0}\rangle$ and $\langle \text{CCD0}, (\text{drive C A})\_, \text{ACD0}\rangle$ – as these correspond to transitions to ACD0 from the other two goal states. These transitions could never contribute to their respective goal-makespan formulas – no path out of a goal state can do better than its initial formula, so would be removed by the call to 'Simplify'.

For the edge $\langle \text{ACL1}, (\text{driver-loading})^0_{\dashv}, \text{ACD0}\rangle$, 'MakespanThroughNeighbor()' computes a goal-makespan formula for ACL1 based on that of ACD0, using the substitution rule. As $tg^{\Theta}(\text{ACD0}) = \text{Formula}(\max\{use(s, v_P), use(s, e_{DL})\})$, and $propagate\_use_{\Theta}((\text{driver-loading}^0_{\dashv}), s, v)$ and $propagate\_chg_{\Theta}((\text{driver-loading}^0_{\dashv}), s, v)$ are no-ops for all $v \in V$, substitution yields:

$$new\_tg = \text{Formula}(\max\{use(s, v_P), use(s, e_{DL})\})$$

As $tg^{\Theta}(\text{ACL1})$ was initialized to ∞:

$$new\_tg' = \text{Formula}(\min\{\infty, \max\{use(s, v_P), use(s, e_{DL})\}\})$$

Trivially, $\text{Simplify}(new\_tg')$ then eliminates the ∞, giving a new goal-makespan formula for ACL1:

$$tg^{\Theta}(\text{ACL1}) = \text{Formula}(\max\{use(s, v_P), use(s, e_{DL})\})$$

...and ACL1 is appended to Q. This goal-makespan formula makes intuitive sense: in ACL1, the goal $v_P = C$ has already been achieved, so no extra time is needed beyond $use(s, v_P)$; and while e_{DL} does not yet have its goal value of 0, no extra time is needed for this either, as this change in value happens instantaneously when $(\text{driver-loading}^0_{\dashv})$ is applied.

1) BCD0 is popped off Q *Remaining:* $Q = [\text{CCD0}, \text{ACL1}]$

This proceeds similarly to popping ACD0 off Q, resulting in:

$$tg^{\Theta}(\text{BCL1}) = \text{Formula}(\max\{use(s, v_P), use(s, e_{DL})\})$$

...and BCL1 being appended to Q.

2) CCD0 is popped off Q *Remaining:* $Q = [\text{ACL1}, \text{BCL1}]$
This proceeds similarly to popping ACD0 or BCD0 off Q, resulting in:

$$tg^{\Theta}(\text{CCL1}) = \text{Formula}(\max\{use(s, v_P), use(s, e_{DL})\})$$

...and CCL1 being appended to Q.

3) ACL1 is popped off Q *Remaining:* $Q = [\text{BCL1}, \text{CCL1}]$
There is a single edge into ACL1: $\langle \text{ACD0}, (\text{driver-loading}^0_{\vdash}), \text{ACL1}\rangle$. As this edge is from a goal state ACD0 it is not especially interesting, as it will not improve its goal-makespan formula. But, as *propagate_use* and *propagate_chg* are not no-ops, it does serve to better illustrate the substitution in 'MakespanThroughNeighbor' – starting with the goal-makespan formula for $s' = ACL1$:

$$\text{Formula}(\max\{use(s', v_P), use(s', e_{DL})\})$$

..through substitution this becomes, in turn:

$$\begin{aligned} new\_tg &= \text{Formula}(\max\{use(s, v_P), start\_min_{\Theta}((\text{driver-loading}^0_{\vdash}), s) + \delta^{-}(\text{driver-loading})\}) \\ &= \text{Formula}(\max\{use(s, v_P), \max\{use(s, v_D) + \epsilon, chg(s, v_D) + \epsilon, use(s, e_{DL})\} + 2\}) \\ &= \text{Formula}(\max\{use(s, v_P), use(s, v_D) + 2 + \epsilon, chg(s, v_D) + 2 + \epsilon, use(s, e_{DL}) + 2\}) \end{aligned}$$

This can be read as follows – if one is in the state ACD0 and wishes to reach the goals via a path that goes through ACL1, the earliest time at which the goal can be reached, and the execution of all actions completed, is the maximum of either:

- the time $use(s, v_P)$ in ACD0 at which the goal $v_P \mapsto C$ was achieved – since the path via ACL1 doesn't affect this in any way; or,
- the time at which (driver-loading) would be able to start (ϵ after $use(s, v_D)$, due to its precondition and effect on v_D; and after $use(s, e_{DL})$ due to its execution counter); plus its minimum duration of 2, to give the time at which it could end.

Quite sensibly, however, this path would be disregarded as ACD0 is already a goal state. More formally, $tg^{\Theta}(\text{ACD0})$ was $\text{Formula}(\max\{use(s, v_P), use(s, e_{DL})\})$, hence:

$$\begin{aligned} new\_tg' = \text{Formula}(\min\{ \quad & \max\{ \quad use(s, v_P), use(s, e_{DL})\}, \\ & \max\{ \quad use(s, v_P), use(s, v_D) + 2 + \epsilon, \\ & \qquad chg(s, v_D) + 2 + \epsilon, use(s, e_{DL})\}\}) \end{aligned}$$

Then, trivially, as the terms of the first 'max' are a subset of those in the second, only the first would be kept – it is never any larger than the second. As a result, $new\_tg' = tg^{\Theta}(\text{ACD0})$, so the condition at line 14 of Algorithm 3 does not hold – there is no need to update $tg^{\Theta}(\text{ACD0})$ and place ACD0 on Q.

4) BCL1 is popped off Q *Remaining:* $Q = [\text{CCL1}]$
Much as when ACL1 was popped off Q, there is a single edge into BCL1 from a goal state (in this case, BCD0) so no goal-makespan formulas will be updated in this iteration of the loop, as going via BCL1 cannot shorten the makespan from a goal state back to itself.

5) CCL1 is popped off Q *Remaining:* $Q = []$
There are two edges into CCL1. The first is uninteresting – from goal state CCD0. Much like the edges into ACL1 and BCL1, this edge will not lead to any goal-makespan formulas being updated.

The other edge into CCL1 is ⟨CTL1, (unload C)_, CCL1⟩. This leads to the computation of a goal-makespan formula for CTL1, based on that of CCL1, using the substitution rule. Following iteration 2, we determined:

$$tg^{\Theta}(\text{CCL1}) = \text{Formula}(\max\{use(s, v_P), use(s, e_{DL})\})$$

Then, from $propagate\_use_{\Theta}((\text{unload\_}), s, v_P)$ and $propagate\_use_{\Theta}((\text{unload\_}), s, e_{DL})$ (which is a no-op):

$$new\_tg = \text{Formula}(\max\{start\_min_{\Theta}((\text{unload x\_}), s) + \delta^{-}(\text{unload x}), use(s, e_{DL})\})$$

Then, embedding $start\_min_{\Theta}((\text{unload x\_}), s)$ and $\delta^{-}(\text{unload x}) = 1$ and $\delta^{+}(\text{unload x}) = 1$ gives:

$$\begin{aligned} new\_tg = \text{Formula}(\max\{ \quad \max\{ \quad & \epsilon + 1, \\ & use(s, v_P) + \epsilon + 1, \\ & use(s, v_T) + 1, \\ & use(s, v_D) + 1, \\ & chg(s, v_P) + \epsilon + 1, \\ & chg(s, v_P) - 1 + \epsilon + 1, \\ & 0 + 1\}, \\ use(s, e_{DL})\}) & \end{aligned}$$

Tidying up redundant terms and flattening the nested 'max'es then gives:

$$\begin{aligned} new\_tg = \text{Formula}(\max\{ \quad & \epsilon + 1, \\ & use(s, v_P) + \epsilon + 1, \\ & use(s, v_T) + 1, \\ & use(s, v_D) + 1, \\ & chg(s, v_P) + \epsilon + 1, \\ & use(s, e_{DL})\}) \end{aligned}$$

As $tg^{\Theta}(\text{CTL1})$ was initialized to ∞, after Simplify, $new\_tg' = new\_tg$, hence this becomes the new goal-makespan formula $tg^{\Theta}(\text{CTL1})$, and CTL1 is pushed onto Q.

Again, this goal-makespan formula makes intuitive sense. From top to bottom, $\epsilon + 1$ corresponds to the theoretical lower bound of assuming (unload x_) is applied as soon as theoretically possible, recalling no action can occur before time ϵ, and the action has a minimum duration of 1. Then, the terms on state variables reflect the need to order (unload x_) after actions that support its preconditions/conflict with its effects; and the term for e_{DL} reflects the fact that (driver-loading) is currently executing, so a goal state cannot be reached any sooner than the time at which this would end.

6) CTL1 is popped off Q *Remaining:* $Q = []$

There are two edges into CTL1. First, we consider ⟨CCL1, (load C)_, CTL1⟩. In the previous iteration ⟨CTL1, (unload C)_, CCL1⟩ gave us the goal-makespan formula for CTL1, so it makes no intuitive sense for this edge into CTL1 to usefully contribute to the goal-makespan formula for CCL1 – it would correspond to pointlessly loading the package onto the truck from its goal location, only to need to unload it again. Thankfully, 'Simplify' picks this up. In short, by using to $propagate\_use_{\Theta}((\text{load\_}), s, v_P)$, $propagate\_use_{\Theta}((\text{load\_}), s, e_{DL})$ (which is a no-op), and $start\_min_{\Theta}((\text{load x\_}), s)$:

$$\begin{aligned} new\_tg = \text{Formula}(\max\{ \quad & 2\epsilon + 2, \\ & use(s, v_P) + 2\epsilon + 2, \\ & use(s, v_T) + \epsilon + 2, \\ & use(s, v_D) + \epsilon + 2, \\ & chg(s, v_P) + 2\epsilon + 2, \\ & use(s, e_{DL})\}) \end{aligned}$$

As the incumbent $tg^{\Theta}(\text{CCL1}) = \text{Formula}(\max\{use(s, v_P), use(s, e_{DL})\})$ is strictly better than this, Simplify($\min\{tg^{\Theta}(\text{CCL1}), new\_tg\}$) would return $tg^{\Theta}(\text{CCL1})$. Hence, the condition at line 14 would be unsatisfied, and CCL1 would not be pushed onto Q.

The second edge into CTL1 is $\langle \text{CTD0}, (\text{driver-loading}^0_{\vdash}), \text{CTL1} \rangle$. This leads to the computation of a goal-makespan formula for CTD0, based on that of CTL1, using the substitution rule. Following iteration 5, we determined:

$$\begin{aligned} tg^{\Theta}(\text{CTL1}) = \text{Formula}(\max\{ \quad & \epsilon + 1, \\ & use(s, v_P) + \epsilon + 1, \\ & use(s, v_T) + 1, \\ & use(s, v_D) + 1, \\ & chg(s, v_P) + \epsilon + 1, \\ & use(s, e_{DL})\}) \end{aligned}$$

From $propagate\_use_{\Theta}((\text{driver-loading}^0_{\vdash}), s, v)$ and $propagate\_chg_{\Theta}((\text{driver-loading}^0_{\vdash}), s, v)$ for $v \in \{v_P, v_T, v_D, e_{DL}\}$:

$$\begin{aligned} new\_tg = \text{Formula}(\max\{ \quad & \epsilon + 1, \\ & use(s, v_P) + \epsilon + 1, \\ & use(s, v_T) + 1, \\ & start\_min_{\Theta}((\text{driver-loading}^0_{\vdash}), s) + 1, \\ & chg(s, v_P) + \epsilon + 1, \\ & start\_min_{\Theta}((\text{driver-loading}^0_{\vdash}), s) + \delta^{-}(\text{driver-loading})\}) \end{aligned}$$

Then, embedding $start\_min_{\Theta}((\text{driver-loading}^0_{\vdash}), s)$ and $\delta^{-}(\text{driver-loading}) = 2$ and $\delta^{+}(\text{driver-loading}) = 2$ gives:

$$\begin{aligned} new\_tg = \text{Formula}(\max\{ \quad & \epsilon + 1, \\ & use(s, v_P) + \epsilon + 1, \\ & use(s, v_T) + 1, \\ & \max\{ \quad \epsilon + 1, \\ & \qquad use(s, v_D) + \epsilon + 1, \\ & \qquad chg(s, v_D) + \epsilon + 1, \\ & \qquad chg(s, v_D) - 2 + \epsilon + 1, \\ & \qquad use(s, e_{DL}) + 1\}, \\ & chg(s, v_P) + \epsilon + 1, \\ & \max\{ \quad \epsilon + 2, \\ & \qquad use(s, v_D) + \epsilon + 2, \\ & \qquad chg(s, v_D) + \epsilon + 2, \\ & \qquad chg(s, v_D) - 2 + \epsilon + 2, \\ & \qquad use(s, e_{DL}) + 2\}\}) \end{aligned}$$

Tidying up redundant terms and flattening the nested 'max'es then gives:

$$\begin{aligned} new\_tg = \text{Formula}(\max\{ \quad & \epsilon + 2, \\ & use(s, v_P) + \epsilon + 1, \\ & use(s, v_T) + 1, \\ & chg(s, v_P) + \epsilon + 1, \\ & use(s, v_D) + \epsilon + 2, \\ & chg(s, v_D) + \epsilon + 2, \\ & use(s, e_{DL}) + 2\}) \end{aligned}$$

As $tg^{\Theta}(\text{CTD0})$ was initialized to ∞, after Simplify, $new\_tg' = new\_tg$, hence this becomes the new goal-makespan formula $tg^{\Theta}(\text{CTD0})$, and CTD0 is pushed onto Q.

As a brief sense-check, the terms with '+2' in this goal-makespan formula correspond to those of (driver-loading): as its execution must have completed in a goal state, the earliest this can be achieved is the earliest its execution could have started plus its duration of 2. Likewise the '+1' terms correspond to (unload C), which can occur in parallel to this: the goal $v_P = C$ can be achieved no earlier than the time at which the truck and package variables can be used (resp. used and changed) plus its duration of 1.

7 onwards) Subsequent iterations

As we have shown the use of Algorithm 3 to generate sensible goal-makespan formulas for the states encountered so far, and an exhaustive demonstration would be prohibitively lengthy, we will for the sake of this worked example derive goal-makespan formulas for states along a trajectory back from CTD0 – the state we have reached so far – and the initial state ABD0.

- For CTD0 we have *inter alia* an incoming edge $\langle \text{BTD0}, (\text{drive B C})\_, \text{CTD0} \rangle$. This produces a goal-makespan formula for BTD0:

$$\begin{aligned} tg^{\Theta}(\text{BTD0}) = \text{Formula}(\max\{ \quad & \epsilon + 12, \\ & use(s, v_P) + \epsilon + 1, \\ & use(s, v_D) + 12, \\ & use(s, v_T) + \epsilon + 12, \\ & chg(s, v_P) + \epsilon + 1, \\ & chg(s, v_D) + \epsilon + 2, \\ & chg(s, v_T) + \epsilon + 12, \\ & use(s, e_{DL}) + 2\}) \end{aligned}$$

- For BTD0 we have *inter alia* an incoming edge $\langle \text{BTL1}, (\text{driver-loading})^0_{+1}, \text{BTD0} \rangle$. This produces a goal-makespan formula for BTL1:

$$\begin{aligned} tg^{\Theta}(\text{BTL1}) = \text{Formula}(\max\{ \quad & \epsilon + 12, \\ & use(s, v_P) + \epsilon + 1, \\ & use(s, v_T) + \epsilon + 12, \\ & chg(s, v_P) + \epsilon + 1, \\ & chg(s, v_D) + \epsilon + 12, \\ & chg(s, v_T) + \epsilon + 12, \\ & use(s, e_{DL}) + 12\}) \end{aligned}$$

- For BTL1 we have *inter alia* an incoming edge $\langle \text{BBL1}, (\text{load B})\_, \text{BTL1}\rangle$. This produces a goal-makespan formula for BBL1:

$$\begin{aligned} tg^{\Theta}(\text{BBL1}) = \text{Formula}(\max\{ \quad & \epsilon + 13, \\ & use(s, v_P) + \epsilon + 13, \\ & use(s, v_D) + 13, \\ & use(s, v_T) + 13, \\ & chg(s, v_P) + \epsilon + 13, \\ & chg(s, v_D) + \epsilon + 12, \\ & chg(s, v_T) + \epsilon + 12, \\ & use(s, e_{DL}) + 12\}) \end{aligned}$$

- For BBL1 we have *inter alia* an incoming edge $\langle \text{BBD0}, (\text{driver-loading})^0_{\vdash}, \text{BBL1}\rangle$. This produces a goal-makespan formula for BBD0:

$$\begin{aligned} tg^{\Theta}(\text{BBD0}) = \text{Formula}(\max\{ \quad & \epsilon + 14, \\ & use(s, v_P) + \epsilon + 13, \\ & use(s, v_D) + \epsilon + 14, \\ & use(s, v_T) + 13, \\ & chg(s, v_P) + \epsilon + 13, \\ & chg(s, v_D) + \epsilon + 14, \\ & chg(s, v_T) + \epsilon + 12, \\ & use(s, e_{DL}) + 14\}) \end{aligned}$$

- For BBD0 we have we have *inter alia* an incoming edge $\langle \text{ABD0}, (\text{drive A B})\_, \text{BBD0}\rangle$. This produces a goal-makespan formula for ABD0:

$$\begin{aligned} tg^{\Theta}(\text{ABD0}) = \text{Formula}(\max\{ \quad & \epsilon + 24, \\ & use(s, v_P) + \epsilon + 13, \\ & use(s, v_D) + 24, \\ & use(s, v_T) + \epsilon + 24, \\ & chg(s, v_P) + \epsilon + 13, \\ & chg(s, v_D) + \epsilon + 14, \\ & chg(s, v_T) + \epsilon + 24, \\ & use(s, e_{DL}) + 14\}) \end{aligned}$$

Sense-checking $tg^{\Theta}(\text{ABD0})$, each option in the max corresponds to a critical path through the temporal plan from ABD0 to CCD0. $\epsilon + 24$ corresponding to the critical path from time zero (the time taken to start and complete the actions [(drive A B), (driver-loading), (drive B C), (driver-loading)] leading to goal state CCD0; with durations adding up to 24, and the first action beginning at time ϵ). The other options are based on the times the variables' values are available for use (or to change), and the duration of the critical path after that point; for instance, $use(s, v_P) + \epsilon + 13$ corresponds to there being a critical path of duration 13 ([(load B), (drive B C), (driver-loading)]), with an ϵ gap, that cannot occur before $v_P \mapsto B$ is available for use.

Running the algorithm in full yields goal-makespan formulas for each state in the abstraction, with the formulas implicitly capturing the actions along the paths from each state to goal states, reducing them to their critical duration paths relative to when state variables can be used/changed.

4.3.4 Evaluating Goal-makespan Formulas. We have just showed how to compute goal-makespan formulas for each abstract state, both in theory and in our worked example. We will now set out how these are used in search, as the basis of a makespan heuristic.

Algorithm 4: Translating a temporal planning state

Data: Temporal planning state $\mathsf{s}_t = \langle U, P, T \rangle$
Result: Classical surrogate planning state s_c; the values $use(\mathsf{s}_c, v_j)$ and $chg(\mathsf{s}_c, v_j)$ for all $v_j \in \mathcal{V}$; the values $use(\mathsf{s}_c, e_k)$ and $chg(\mathsf{s}_c, v_k)$ for all $e_k \in \mathcal{E}$

1 $\forall_{p_l \in P} \; t(p_l) \leftarrow$ minimum timestamp of plan step p_l according to the STN T;
2 $\mathsf{s}_c \leftarrow U$;
3 **foreach** $v_j \in \mathcal{V}$ **do**
4 $use(\mathsf{s}_c, v_j) \leftarrow t(V^{\mathit{eff}}(\mathsf{s}_t, v_j))$;
5 $chg(\mathsf{s}_c, v_j) \leftarrow \max\{t(i) + d \mid \langle i, d \rangle \in VP(\mathsf{s}_t, v_j)\}$;
6 **foreach** $a_i \in O_n$ **do**
7 **if** $exec(P, T)[a_i] \neq \emptyset$ **then**
8 $\mathsf{s}_c \leftarrow \mathsf{s}_c \cup \{\langle e_i, \max\{|exec(P, T)[a_i]|, 2\}\rangle\}$;
9 $use(\mathsf{s}_c, e_i) \leftarrow \min_{p_m \in exec(P,T)[a_i]} (t(p_m) + \delta^-(a_i))$;
10 $chg(\mathsf{s}_c, e_i) \leftarrow \min_{p_m \in exec(P,T)[a_i]} (t(p_m) + \delta^-(a_i))$;
11 **else**
12 $s_c \leftarrow \mathsf{s}_c \cup \{\langle e_i, 0\rangle\}$;
13 $use(\mathsf{s}_c, e_i) \leftarrow 0$;
14 $chg(\mathsf{s}_c, e_i) \leftarrow 0$;
15 **return** s_c; $use(\mathsf{s}_c, v)$ *and* $chg(\mathsf{s}_c, v)$ *for all* $v \in \{\mathcal{V} \cup \mathcal{E}\}$

During planning, for each temporal planning state $\mathsf{s}_t = \langle U, P, T \rangle$ that is to be evaluated (i.e. for which we wish to obtain a lower bound on the time the goal can be reached), we first translate it into its surrogate state s_c in the surrogate classical planning task, and find the *use* and *chg* values for s_t due to the plan P that reached it and the temporal constraints in T. To obtain this information, we use Algorithm 4.

The algorithm begins by obtaining the information for state variables: at line 2, the state variable values in s_c are taken from s_t; and from lines 3 to 5, the *use* and *chg* values for state variables are computed, as per Definition 9.

Next, it uses the convenience function $exec(P, T)$ (defined in Algorithm 1) to provide a basis for obtaining the information for execution counter variables. If according to $exec(P, T)$, one or more instances of some $a_i \in O_n$ are still running by the end of P then:

- the execution-counter variable e_i for a_i is assigned a value equal to how many instances of a_i are running, capped at 2 (line 8);
- $use(\mathsf{s}_c, e_i)$ and $chg(\mathsf{s}_c, e_i)$ are set to the earliest time a_i could end, across all executing instances (lines 9 and lines 10, respectively).

Otherwise, if all executing instances of a_i have finished, e_i is assigned the value of 0, and $use(\mathsf{s}_c, e_i)$ and $chg(\mathsf{s}_c, e_i)$ are given harmless default values of zero.

Having obtained the classical surrogate state s_c, and the relevant *use* and *chg* timestamps, evaluating the heuristic is relatively straightforward: identify the abstract state $s \in S$ that corresponds to s_c; and from this abstract state, obtain the pre-computed goal-makespan formula $tg^{\Theta}(s)$ and evaluate it with the values of $use(s, v_i)$

and $chg(s, v_i)$ equal to those in $use(\mathsf{s}_c, v_i)$ and $chg(\mathsf{s}_c, v_i)$ respectively:

$$h^{\Theta}(s) = tg^{\Theta}(s)\Big|_{\substack{use(s,v)=use(\mathsf{s}_c,v)\forall_{v\in\{V\cup E\}} \\ chg(s,v)=chg(\mathsf{s}_c,v)\forall_{v\in\{V\cup E\}}}}.$$

4.3.5 Admissibility. As we will now show, the temporal merge-and-shrink heuristic, computed by evaluating goal-makespan formulas, is admissible. Considering the underlying abstract transition graphs, the full transition graph $\Theta(\Pi)$ for our classical surrogate keeps all transitions from the underlying temporal planning task Π. As in classical planning, every path in $\Theta(\Pi)$ is also a path in an abstract transition graph Θ, since merging and shrinking operations keep all transitions from the full transition graph. Hence the structure of the abstract transition graphs provide the basis for finding admissible heuristics: all paths corresponding to optimal solutions to the underlying temporal planning task are preserved. In addition to this, we need to prove that goal-makespan formulas derived from the abstract transition graphs provide admissible estimates of makespan. We do this in steps. First, we note that *use/chg* propagation is trivially admissible. $start\_min_{\Theta}$ is a lower bound (a max() over a subset of variable assignments, is a lower bound of the max over all variable assignments). Further, propagation assumes that all actions take their minimum duration δ^- to execute, which is again a lower bound. Second, Algorithm 3 is equivalent to finding, for each state in the abstraction, all non-cyclical paths from it to goal states, keeping only the Pareto front defined according to Definition 31 – a path from a state to a goal state will only be removed through simplification (line 13) if another known path is unequivocally better. The goal-makespan formula for each state in the abstraction is then the minimum across the Pareto front of paths from it, to goal states. Finally, as evaluating the goal-makespan formula for a state in the abstraction is equivalent to taking the minimum makespan evaluated across the Pareto front of paths from it to goal states, the function yields the optimal makespan for reaching a goal state in the abstraction. Hence, as the abstraction is a path-preserving relaxation of the transition graph for the planning task, this optimal makespan is an admissible makespan estimate for the planning task.

4.3.6 Classical/Temporal M&S Differences. We now summarize the main differences between M&S abstractions in classical and temporal planning settings. There are two differences: 1) in the temporal setting actions can be in an executing state, and thus each abstract state has a counter of the number of times each (non-compression-safe) action is executing; 2) pre-computing admissible goal-makespan estimates for each abstract state involves pre-computing all possible non-cyclical paths to the goal, and expressing goal-makespan as a function of temporal variables which are evaluated at search time.

These goal-makespan formulas capture all temporal aspects of a problem relevant to makespan. The only temporal aspect which is relaxed in 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 $[\delta^-, \delta^+]$).

4.4 Shrinking Algorithms

In this paper we consider two shrinking algorithms for temporal merge-and-shrink: h-preserving and bisimulation.

h-preserving Shrink. An h-preserving shrink algorithm groups states that have the same heuristic value into a single abstract state (Fan et al. 2018). 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 et al. 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 et al. 2011). The computation of the coarsest bisimulation can be made efficiently by starting from a single equivalence class and iteratively separating non-bisimilar states (Nissim et al. 2011). For

efficiency, as done in FastDownward, the process can also be initialized by grouping states by heuristic value, since states with different heuristic value cannot be bisimilar. Similarly, in the temporal domain, states with different goal-makespan formulas cannot be bisimilar, and therefore we initialize the process by grouping states by goal-makespan formula equality.

4.5 Merging Strategies

We consider two merging strategies: a popular strategy in classical planning (CGGL), and a new randomized strategy proposed by us.

CGGL. One popular merging strategy is CGGL (Causal Graph, Goal, Level (Helmert, Haslum, Hoffmann, et al. 2007)), which is available in the Fast Downward planner (Helmert 2006). The strategy is linear (i.e. it merges a single variable in each iteration), and orders variables by the following criteria: 1) presence of a causal graph arc to already added variables, 2) a goal being defined for the variable, 3) the variable's canonical order according to Fast Downward's "highest level" first ordering (Helmert 2006).

A New Randomized Strategy: CGGR. We propose a new merging strategy that we found to be effective in practice. This strategy is a randomized version of CGGL to which we call CGGR ("R" for "random"). It makes two changes to CGGL. First, it replaces the canonical level-based variable order of CGGL (criteria 3 above) with a random order of remaining variables. Second, it stops merging new variables once a maximum number of transition-graph edges is reached. Since the abstractions obtained by CGGR are random, we propose running the process R times from different random seeds to obtain R different abstractions, and then for heuristic purposes use the maximum of the makespan estimates given by the abstractions.

4.6 Pruning

Similarly to other merge-and-shrink work (Helmert, Haslum, Hoffmann, and Nissim 2014), in the pruning step (Algorithm 2) we prune abstract states that cannot be reached from the initial state, and abstract states that cannot reach a goal state. These operations preserve (forward) admissibility in forward-search planners (Sievers and Helmert 2021), which is the type of planner we apply.

Furthermore, even though we do not implement classical label reduction (Nissim et al. 2011) in this work, we still reduce the number of transitions in step 3 of Algorithm 2 by pruning dominated transitions from abstractions. We do this 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 transition pruning 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 31.

5 Results

5.1 Planner Setup

We implemented the methods proposed in the previous sections in the temporal planner OPTIC (Benton et al. 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)[8]. 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 focus on these comparisons (rather than to other planning approaches, such as constraint programming and optimization modulo theories) as this allows us to cleanly discern the effects of our contribution, both by comparing to a well-established benchmark; and by using the same PDDL planning tasks for all configurations to ensure results are fairly comparable, with differences not arising due to variations in modelling or implementation concerns.

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.

5.2 Merge-and-Shrink Strategy

In our experiments we implement merge-and-shrink as in Algorithm 2, using Fast Downward's (Helmert 2006) merge-and-shrink code as a basis. All experiments use pruning of unreachable and irrelevant abstract states, as in Fast Downward (i.e. we prune abstract states that cannot be reached from the initial state and abstract states that cannot reach a goal state, which preserves forward admissibility in forward-search planners (Sievers and Helmert 2021)). As shrinking strategies we evaluate bisimulation ("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 et al. 2018). The implementation of bisimulation was ported from Fast Downward into OPTIC. Regarding the merging strategies, we use CGGL (Helmert, Haslum, Hoffmann, et al. 2007) and our proposed CGGR as described in Section 4.5. For CGGR we use $R = 10$ abstractions and show results for 500 and 5000 as maximum number of transition-graph edges. We use bisimulation algorithm parameter $N = 100$ (maximum number of equivalent classes), which led to the best results.

5.3 Benchmarks

The benchmarks we used for evaluation were all the temporal problems from the International Planning Competition (IPC) 2002-2014 (A. J. Coles, A. Coles, et al. 2012; Dimopoulos et al. 2006; Long and Fox 2003b; Vallati et al. 2015). Even though only a subset of these were designed to be used in an optimal-planning setting (IPC2002 and 2006), we included optimal-planning versions of newer problems—in order to assess our method's ability to optimize extremely challenging problems. For problems with deadlines we used driverlog, trucks, and zeno domains with deadlines, as in (Marzal et al. 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.

[8]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 (A. Coles et al. 2010)).

Table 1. Merge-and-shrink pre-computation time (s), for partial-order OPTIC, averaged over a subset of problems per domain. In this subset of problems, all methods can compute abstractions within the time and memory budget.

	Domain	L-hshrink	L-bisim	R500-hshrink	R500-bisim	R5k-hshrink	R5k-bisim
(optimal)	02depots	402.17	333.65	0.59	0.63	8.49	8.53
	02driverlog	349.33	356.25	0.56	0.62	25.08	47.7
	02rovers	28.37	30.43	0.19	0.28	6.91	2.31
	02satellite	11.42	16.13	0.16	0.26	0.89	1.23
	02zenotravel	271.24	216.6	0.24	0.35	1.74	1.57
	06openstacks	129.15	126.6	0.25	0.32	3.46	3.91
	06storage	474.6	457.93	0.69	0.79	21.78	23.1
	06trucks	344.3	348.83	1.41	1.47	17.97	16.24
	08crew	508.3	475.22	0.39	0.5	5.21	5.35
	08elevator	431.87	324.62	1.13	1.45	3.96	4.38
	08sokoban	626.69	486.74	0.9	1.11	46.83	42.35
	11crew	380.13	357.52	0.48	0.6	6.75	6.62
	11elevator	403.37	479.91	1.74	2.14	7.76	5.17
	11match	32.23	31.16	0.07	0.1	0.7	0.91
	11parking	595.44	617.92	10.3	10.43	10.48	10.38
	11peg	1399.69	1377.08	0.29	0.45	11.15	11.0
	11sokoban	1786.84	1674.14	0.8	1.52	17.48	30.63
	11turn	251.83	247.93	0.42	0.53	28.66	29.95
	14driverlog	494.64	1063.78	0.44	0.56	8.99	10.67
	14match	24.71	24.27	0.07	0.1	0.72	1.12
	14parking	74.06	74.46	8.35	8.34	8.29	8.31
	14turn	252.18	245.9	0.41	0.57	28.78	29.68
(deadlines)	depots	812.88	830.7	4.37	4.63	45.98	44.42
	driverlog	28.39	30.88	2.26	2.17	49.17	45.29
	logistics	0.05	0.11	0.73	1.42	0.85	2.67
	trucks	17.26	18.33	1.46	1.88	53.15	59.23
	zeno	1.3	1.43	0.64	0.97	4.24	4.45

5.3.1 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. Times are averaged over a subset of problems per domain. We include all problems for which all methods can obtain abstractions and formulas within the budget (30min and 4GB). As averages are computed over the same set of problems, computation times are comparable across methods. For some domains, CGGL is not able to generate formulas within the budget in any of the domain's problems, and therefore these domains are omitted from the table. The table shows that CGGR reduces the computation time overall compared to CGGL, most often by 2-4 orders of magnitude. There are four exceptions in the deadlines domains (driverlog, logistics, trucks and zeno), however, since in these problems the transition systems are small and thus CGGR builds 10 abstractions which are of similar size to CGGL. However, total pre-computation time in these domains, especially for CGGR500, is still considerably low (i.e. within 1-4 seconds in all domains). The table also shows that CGGR abstractions limited to 500 edges (R500-*) are about 1-2 orders of magnitude faster than those limited to 5000 edges (R5k-*), which is expectable given the size of these abstractions is also one order of

Table 2. Optimality coverage, partial-order planner on optimal problems

Domain	RPG	L-hshrink	L-bisim	R-hshrink500	R-bisim500	R-hshrink5k	R-bisim5k
02depots	2	0	0	2	2	2	2
02driverlog	5	1	1	5	5	5	5
02rovers	1	1	1	1	1	1	1
02satellite	3	3	3	3	3	3	3
02zenotravel	6	6	6	6	6	7	7
06openstacks	0	0	0	0	0	0	0
06pipesworld	3	0	0	4	4	3	2
06storage	8	3	3	8	8	8	8
06trucks	4	1	1	4	4	4	4
08crew	3	5	5	5	5	5	5
08sokoban	8	1	1	8	8	8	8
11floor	3	0	0	3	3	0	0
11match	3	3	3	3	3	3	3
11peg	16	2	2	17	17	17	17
Sum	65	26	26	**69**	**69**	66	65

magnitude smaller. CGGR abstractions limited to 500 edges are on average computed within one second for most domains.

5.3.2 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 solve 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)

We omit from the tables domains that could not be solved with any method (i.e. which would be rows of zeros). The tables show that merge-and-shrink with CGGR500 improves coverage on both the partial-order and total-order planner: compared to when using TRPG as a source of makespan estimates, the partial-order planner can solve 4 more problems to optimality and the total-order planner can solve 6-8 more problems (depending on the shrinking method). However, the use of CGGR with larger abstractions (5k edges) lead to either the same or worse performance than TRPG, and the use of CGGL leads to drastically worse performance than TRPG. Therefore, the table shows the importance of selecting a merging strategy that leads to a good trade-off between computation time and makespan estimates.

Figure 2 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 expanded and total computation time in partial- and total-order planning.

From this data we also computed the performance improvement of each method vs the TRPG heuristic *on problems for which search with either will find an optimal solution*. In other words, we ask the question: when search using either the M&S heuristic or TRPG can solve a problem to optimality, what is the average number of expanded nodes and computation time? We show these results in Figure 3. The figure shows that planing using M&S abstractions leads to a decrease in number of expanded nodes of 23-47% compared to TRPG in the

Table 3. Optimality coverage, total-order planner on optimal problems

Domain	RPG	L-hshrink	L-bisim	R-hshrink500	R-bisim500	R-hshrink5k	R-bisim5k
02depots	2	1	1	2	2	2	2
02driverlog	6	8	8	8	8	9	9
02rovers	3	4	4	4	4	4	4
02satellite	3	4	4	4	4	4	4
02zenotravel	7	8	8	7	7	8	7
06openstacks	2	2	2	2	2	2	2
06pipesworld	6	1	0	6	6	5	6
06storage	11	5	5	12	12	8	12
06trucks	5	2	1	6	6	2	4
08crew	4	5	5	5	4	5	5
08sokoban	9	0	0	9	9	8	9
11crew	0	1	1	1	0	1	1
11floor	5	0	0	5	5	4	5
11match	5	13	12	5	5	5	5
11peg	18	0	0	18	18	8	3
11sokoban	2	0	0	2	2	1	2
14match	0	12	11	0	0	0	0
Sum	88	66	62	**96**	94	76	80

partial-order planner, and 45-75% in the total-order planner, depending on the M&S strategy. CGGL has similar pruning capability to CGGR5000, and CGGR500 is slightly worse, though still better than TRPG. Even though CGGL and CGGR5000 prune more states, the overhead in their pre-computation leads to worse performance than TRPG, as can be seen by the plots showing total computation time. CGGR500 actually leads to the best performance, in terms of time to solve to optimality—15% faster than TRPG in the partial-order planner and 43% faster in the total-order planner.

The reduction in the number of generated nodes is obtained due to better makespan estimates. Figure 4 shows a comparison between TRPG- and M&S-based makespan estimates on all the expanded 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.

5.3.3 Optimal Problems With Open-List Sorting By Makespan. While the previous optimal planning experiments used OPTIC's default open-list sorting strategy (by heuristic estimate of distance, i.e., number of actions), the planner allows to open-list sorting by makespan estimates, in which case M&S abstractions can be used to estimate these values.

In the next experiment, we computed the optimality coverage, number of expanded nodes, and computation time of the planner when using sorting by makespan—as shown on Figure 5. For comparison, the figure shows results achieved both with distance and makespan sorting. The figure shows that using M&S (CGGR500 hshrink) estimates decreases the number of expanded nodes, compared to when the estimates are obtained from the TRPG (see "Mksp sort" bars). However, this decrease is more pronounced in the total-order variant (-46%) than in the partial-order variant (-7%). This leads to a large improvement in computation time for total-order (-60%), but not for partial-order variants (+12%)—since in the partial-order case the time saved due to pruning is too small compared to the pre-computation time of M&S abstractions. The use of M&S abstractions also allows the total-order planner to increase optimality coverage by 53%, while reducing it for partial-order planner by 2%.

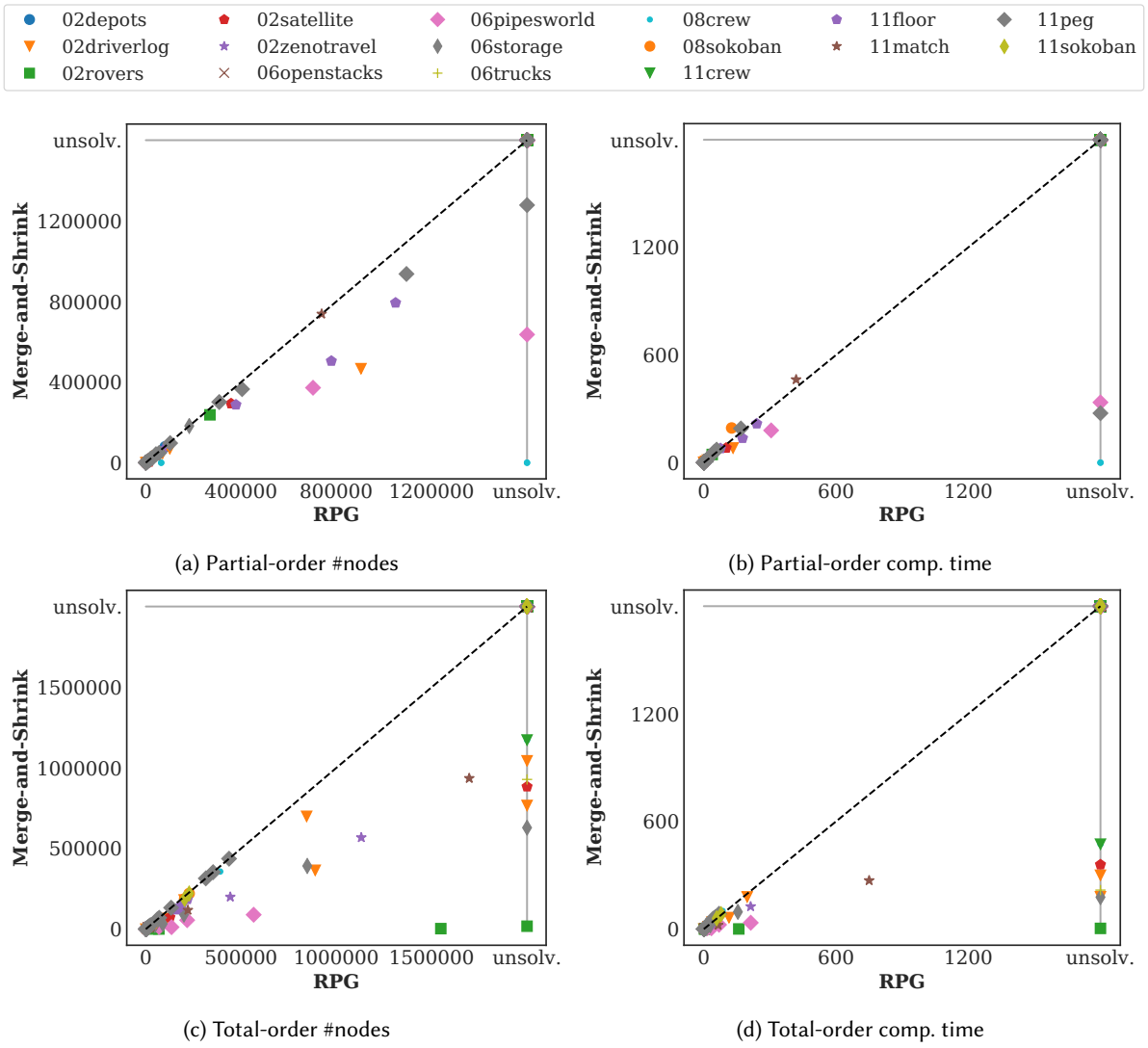
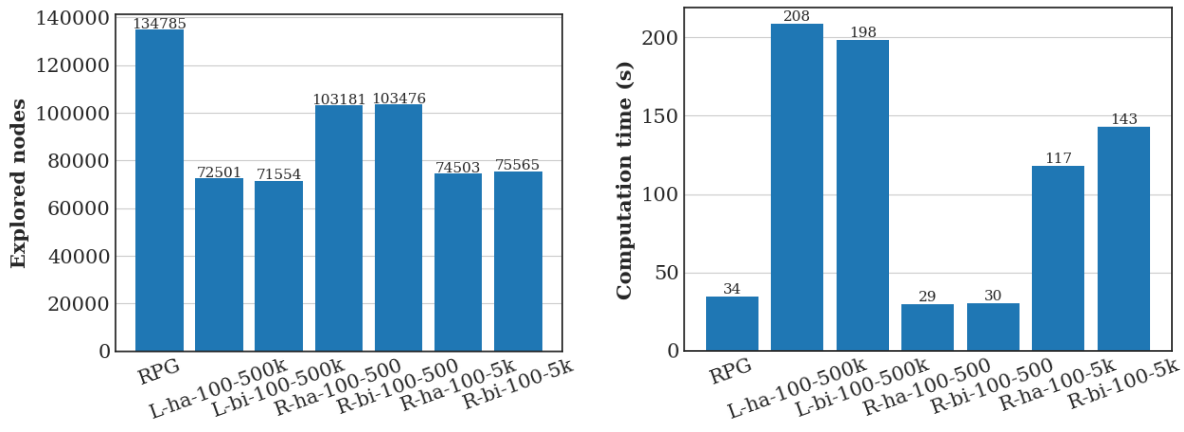


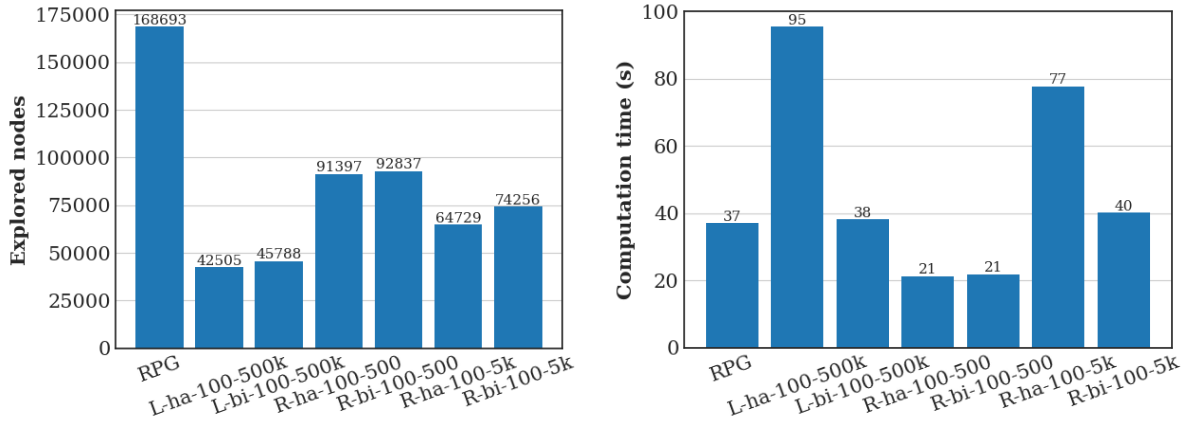
Fig. 2. Makespan-optimal temporal planning problems, using minimal h-preserving shrink and CGGR-500, compared to the TRPG heuristic in terms of nodes expanded (a and c), and computation time (b and d).

Overall, the results show that M&S abstraction can be used both for pruning and open-list sorting in temporal optimality problems, though the benefits are greater for total-order planners than partial-order planners. The best performance on this set of problems, in terms of coverage, was achieved by the total-order variant of OPTIC with distance-based open-list sorting. This combination of methods also led to considerably low computation times—average 21 seconds to solve to optimality.

5.3.4 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



(a) Partial-order planner



(b) Total-order planner

Fig. 3. Comparison of average number of nodes expanded and computation time for solving problems to optimality. Averaged over problems where both RPG and other methods succeed.

(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—8 more unsolvability proofs and 7 more solvable problems solved by CGGR5k and hshrink. CGGR500 led to similar results (6 more proofs and 5 more solves).

5.3.5 MUGS Problems. We now evaluate the use M&S on another type problem that, similarly to optimal planning and unsolvability proofs, requires exhausting the search space—the computation of Minimally Unsolvable Goal Subsets (MUGS). These are sets of facts which cannot be satisfied simultaneously, but which become feasible once one of the facts is removed. These sets have applications in eXplainable AI Planning (XAIP), specifically because they can be used to describe the space-of-plans (Eifler et al. 2020) (or space of achievable goal facts).

We follow the evaluation setup in (Eifler et al. 2020): we compute the optimal makespan of optimal-planning problems, and then obtain unsolvable problems by introducing a deadline of x% of this value. Then, we use

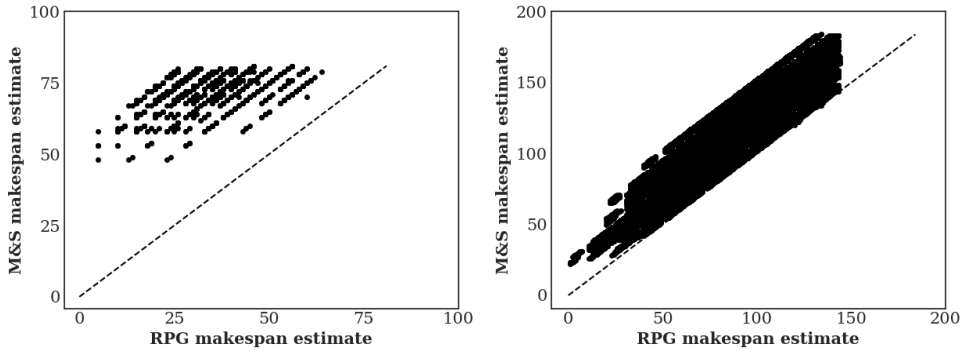
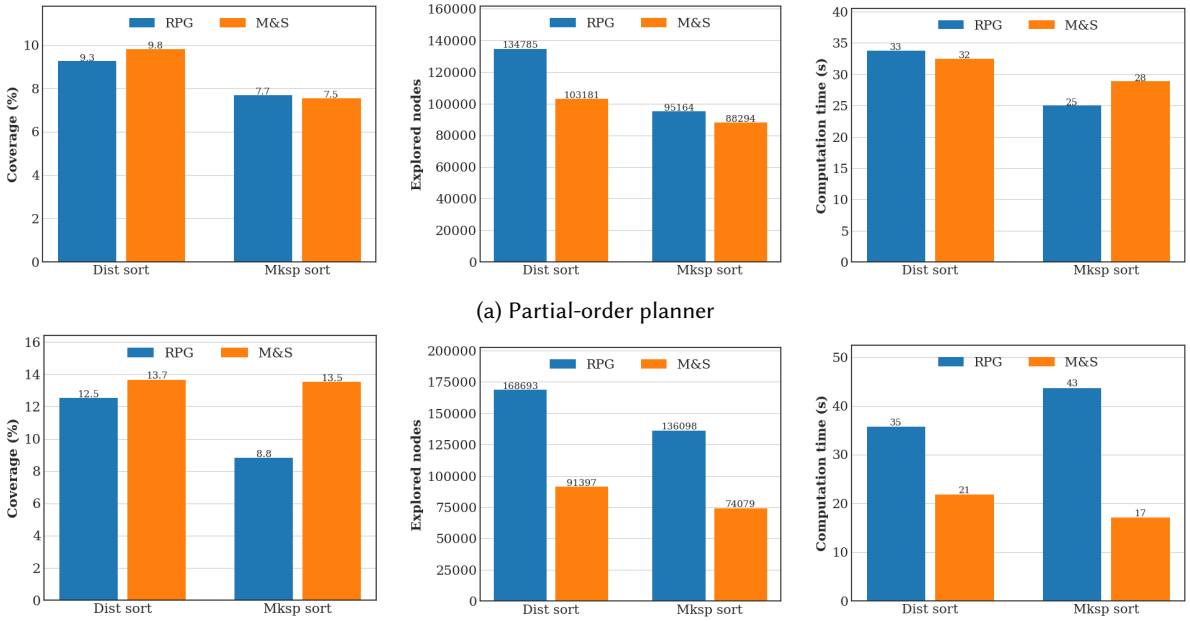


Fig. 4. Comparison of the makespan estimates obtained by TRPG vs M&S (total-order planner with CGGR500 and hshrink) on a rovers (left) and driverlog problem (right).



(a) Partial-order planner

(b) Total-order planner

Fig. 5. Performance on optimal problems in two scenarios: open-list sorting by distance or by makespan. Optimality coverage (number of problems solved to optimality), expanded nodes, and computation times are shown for both TRPG and merge-and-shrink-based time estimates. Nodes and computation time are averages over problems that both methods can solve. M&S results are for CGGR500 with hshrink.

the method described in (Eifler et al. 2020) to smartly exhaust the state space: we keep track of the maximal subsets of goal facts reached by expanded states, and then prune states when they cannot reach any yet-unseen combinations of goal facts. We refer the interested reader to (Eifler et al. 2020) for details. Once more, we compare the use of TRPG vs M&S as heuristics to obtain goal-fact makespan.

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

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)**

Figure 6 shows the results for global deadlines at 74%, 86%, and 98% of the optimal makespan. The figure shows that the tighter the deadline (i.e. the more difficult the problem) the more advantageous the use of M&S. The use of a CGGL merge strategy in this case is even faster than CGGR, leading to a computation time up to 29% lower than achieved with RPG. The figure shows only results on problems that RPG can solve. However, M&S leads to solving 7 more MUGS problems than RPG (total 78 vs 71).

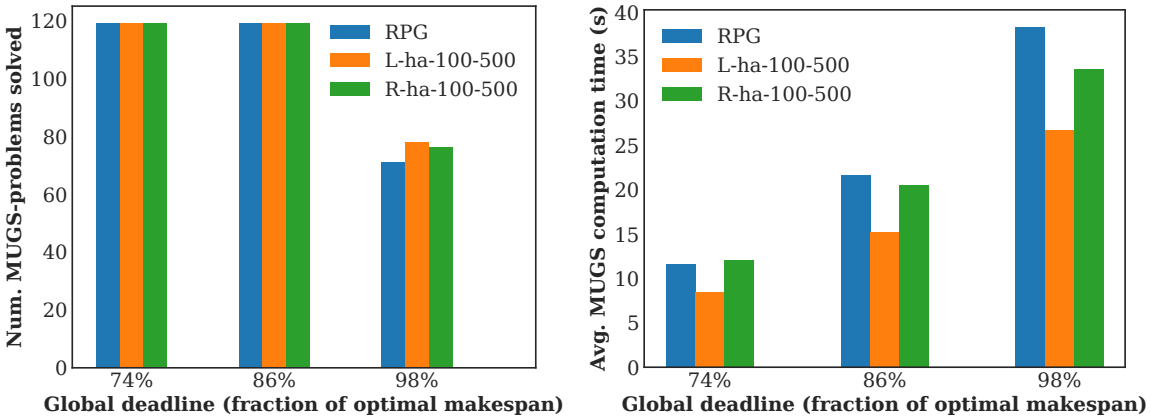


Fig. 6. Performance of M&S versus RPG on MUGS problems.

5.4 Discussion

Across all types of problems (optimal, deadlines, unfeasibility proof, MUGS) the use of a merge-and-shrink heuristic with our proposed CGGR merge strategy (using 10 abstractions limited to 500 edges) consistently outperformed the TRPG heuristic. The results also show that the effectiveness of CGGR depends on the size-limit used, since a 5000-edge limit was worse than TRPG in several settings (optimality coverage on a total-order-planner and optimality computation time on both planners). This shows the need to tune this parameter, or use small conservative values to guarantee low computation overheads.

Merge-and-shrink heuristics were not competitive when using a traditional CGGL merging strategy, which is popular in classical planning (Helmert, Haslum, Hoffmann, et al. 2007). M&S with CGGL performed worse than TRPG in optimality coverage and computation time, and coverage in deadline problems. In both cases, the lower coverage was associated with either larger state spaces or larger number of transitions in each abstraction—leading

the computation time budget to be exhausted within merge-and-shrink pre-computation of abstractions and goal-makespan formulas. However, merge-and-shrink with CGGL still performed performed better than TRPG on unsolvability proofs and MUGS, particularly in small problems where abstractions could be efficiently computed. Our results thus indicate CGGL to only be effective in small problems, and thus CGGR to be preferred.

One limitation of our study is the lack of implementation of label reduction, due to implementation complexity. While we prune dominated transitions, which also reduces the number of transitions, label reduction was left outside the scope of this paper, and should be part of future work. Such methods could further improve our results, since they potentially reduce bisimulation size exponentially (Nissim et al. 2011), and can therefore improve efficiency of merge-and-shrink heuristics.

6 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 building a classical surrogate task, yielding a classical transition graph capturing the logical constraints; then overlaying temporal information onto this, to pre-compute goal-makespan *formulas*, to then be used to provide heuristic makespan estimates at search time. Our method is applicable both to partial- and total-order temporal planners, and the formalism is applicable to temporal planning tasks with and without required concurrency. Another contribution of this paper is CGGR, which is a merge-and-shrink strategy consisting of obtaining 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, obtaining unfeasibility proofs, and computing MUGS. On average the approach led to better pruning, coverage and computation times both in total-order and partial-order planners, though they were especially helpful in the total-order case: leading to 43-60% speed improvements in optimal problems.

We believe that faster implementations of the abstraction building and evaluation processes are possible, which could lead to further coverage and computation time improvements. This is one avenue of future work. Another direction is the implementation and evaluation of label reduction for temporal planning problems, and the exploration of different merging and shrinking strategies specifically designed for temporal planning, and the integration with landmarks-based approaches (Marzal et al. 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 projects THuMP (EP/R033722/1) and COHERENT (EP/V062506/1). The authors would like to thank Joerg Hoffmann for helpful discussions and comments on this work.

References

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

S. Bernardini, F. Fagnani, and D. E. Smith. 2018. "Extracting Mutual Exclusion Invariants from Lifted Temporal Planning Domains." *Artificial Intelligence*, 258.

M. Brandao, A. Coles, A. Coles, and J. Hoffmann. June 2022. "Merge and Shrink Abstractions for Temporal Planning." In: *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*. (June 2022). doi:10.1609/icaps.v32i1.19781.

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

M. Cashmore, D. Magazzeni, and P. Zehtabi. 2020. "Planning for Hybrid Systems via Satisfiability Modulo Theories." *Journal of Artificial Intelligence Research*, 67, 235–283.

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

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

A. J. Coles, A. Coles, A. G. Olaya, S. J. Celorrio, C. L. López, S. Sanner, and S. Yoon. 2012. "A Survey of the Seventh International Planning Competition." *AI Magazine*, 33, 1, 83–88.

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

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

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

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

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

R. Eifler, M. Steinmetz, A. Torralba, and J. Hoffmann. 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.

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

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

M. Fox and D. Long. 2006. "Modelling Mixed Discrete-Continuous Domains for Planning." *Journal of Artificial Intelligence Research*, 27, 235–297. http://www.jair.org.

M. Fox and D. Long. 2003. "PDDL2.1: An Extension to PDDL for Expressing Temporal Planning Domains." *Journal of Artificial Intelligence Research (JAIR)*, 20, 61–124. doi:10.1613/jair.1129.

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

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

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

M. Helmert, P. Haslum, J. Hoffmann, and R. Nissim. 2014. "Merge-and-shrink abstraction: A method for generating lower bounds in factored state spaces." *Journal of the ACM (JACM)*, 61, 3, 1–63.

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

S. Jiménez, A. Jonsson, and H. Palacios. 2015. "Temporal Planning With Required Concurrency Using Classical Planning." In: *Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS)*.

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

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

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

R. Nissim, J. Hoffmann, and M. Helmert. 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)*.

S. Panjkovic and A. Micheli. 2024. "Abstract Action Scheduling for Optimal Temporal Planning via OMT." In: *Proceedings of the AAAI Conference on Artificial Intelligence (AAAI)*.

W. Piotrowski, M. Fox, D. Long, D. Magazzeni, and F. Mercorio. 2016. "Heuristic Planning for Hybrid Systems." In: *Proceedings of the AAAI Conference on Artificial Intelligence*.

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

E. Scala, P. Haslum, S. Thiébaux, and M. Ramírez. 2020. "Subgoaling Techniques for Satisficing and Optimal Numeric Planning." *Journal of Artificial Intelligence Research*, 68, 691–752.

S. Sievers and M. Helmert. 2021. "Merge-and-shrink: A compositional theory of transformations of factored transition systems." *Journal of Artificial Intelligence Research*, 71, 781–883.

M. Vallati, L. Chrpa, M. Grzes, T. L. McCluskey, M. Roberts, and S. Sanner. 2015. "The 2014 International Planning Competition: Progress and Trends." *AI Magazine*, 36, 3, 90–98.

V. Vidal. 2011. "CPT4: An optimal temporal planner lost in a planning competition without optimal temporal track." In: *The Seventh International Planning Competition: Description of Participant Planners of the Deterministic Track.*

A: All Actions for the Worked Example

$$
\begin{array}{lll}
\text{(drive A B):} & pre_{\vdash}(\text{drive A B}) = \{v_T \mapsto A\} & eff_{\vdash}(\text{drive A B}) = \{v_T \mapsto \bot\} \\
& pre_{\leftrightarrow}(\text{drive A B}) = \{v_D \mapsto D\} & \\
& pre_{\dashv}(\text{drive A B}) = \{\} & eff_{\vdash}(\text{drive A B}) = \{v_T \mapsto B\} \\
& \delta^-(\text{drive A B}) = 10, \delta^+(\text{drive A B}) = 10 & \\
\text{(drive B A):} & pre_{\vdash}(\text{drive B A}) = \{v_T \mapsto B\} & eff_{\vdash}(\text{drive B A}) = \{v_T \mapsto \bot\} \\
& pre_{\leftrightarrow}(\text{drive B A}) = \{v_D \mapsto D\} & \\
& pre_{\dashv}(\text{drive B A}) = \{\} & eff_{\vdash}(\text{drive B A}) = \{v_T \mapsto A\} \\
& \delta^-(\text{drive B A}) = 10, \delta^+(\text{drive B A}) = 10 & \\
\text{(drive A C):} & pre_{\vdash}(\text{drive A C}) = \{v_T \mapsto A\} & eff_{\vdash}(\text{drive A C}) = \{v_T \mapsto \bot\} \\
& pre_{\leftrightarrow}(\text{drive A C}) = \{v_D \mapsto D\} & \\
& pre_{\dashv}(\text{drive A C}) = \{\} & eff_{\vdash}(\text{drive A C}) = \{v_T \mapsto C\} \\
& \delta^-(\text{drive A C}) = 10, \delta^+(\text{drive A C}) = 10 & \\
\text{(drive C A):} & pre_{\vdash}(\text{drive C A}) = \{v_T \mapsto C\} & eff_{\vdash}(\text{drive C A}) = \{v_T \mapsto \bot\} \\
& pre_{\leftrightarrow}(\text{drive C A}) = \{v_D \mapsto D\} & \\
& pre_{\dashv}(\text{drive C A}) = \{\} & eff_{\vdash}(\text{drive C A}) = \{v_T \mapsto A\} \\
& \delta^-(\text{drive C A}) = 10, \delta^+(\text{drive C A}) = 10 & \\
\text{(drive B C):} & pre_{\vdash}(\text{drive B C}) = \{v_T \mapsto B\} & eff_{\vdash}(\text{drive B C}) = \{v_T \mapsto \bot\} \\
& pre_{\leftrightarrow}(\text{drive B C}) = \{v_D \mapsto D\} & \\
& pre_{\dashv}(\text{drive B C}) = \{\} & eff_{\vdash}(\text{drive B C}) = \{v_T \mapsto C\} \\
& \delta^-(\text{drive B C}) = 10, \delta^+(\text{drive B C}) = 10 & \\
\text{(drive C B):} & pre_{\vdash}(\text{drive C B}) = \{v_T \mapsto C\} & eff_{\vdash}(\text{drive C B}) = \{v_T \mapsto \bot\} \\
& pre_{\leftrightarrow}(\text{drive C B}) = \{v_D \mapsto D\} & \\
& pre_{\dashv}(\text{drive C B}) = \{\} & eff_{\vdash}(\text{drive C B}) = \{v_T \mapsto B\} \\
& \delta^-(\text{drive C B}) = 10, \delta^+(\text{drive C B}) = 10 &
\end{array}
$$

Fig. 7. All 'drive' actions for the Worked Example

(unload A):	$pre_{\vdash}(\text{unload A}) = \{v_P \mapsto T\}$	$eff_{\vdash}(\text{unload A}) = \{v_P \mapsto \bot\}$
	$pre_{\leftrightarrow}(\text{unload A}) = \{v_T \mapsto A, v_D \mapsto L\}$	
	$pre_{\dashv}(\text{unload A}) = \{\}$	$eff_{\dashv}(\text{unload A}) = \{v_P \mapsto A\}$
	$\delta^-(\text{unload A}) = 1, \delta^+(\text{unload A}) = 1$	
(load A):	$pre_{\vdash}(\text{load A}) = \{v_P \mapsto A\}$	$eff_{\vdash}(\text{load A}) = \{v_P \mapsto \bot\}$
	$pre_{\leftrightarrow}(\text{load A}) = \{v_T \mapsto A, v_D \mapsto L\}$	
	$pre_{\dashv}(\text{load A}) = \{\}$	$eff_{\dashv}(\text{load A}) = \{v_P \mapsto T\}$
	$\delta^-(\text{load A}) = 1, \delta^+(\text{load A}) = 1$	
(unload B):	$pre_{\vdash}(\text{unload B}) = \{v_P \mapsto T\}$	$eff_{\vdash}(\text{unload B}) = \{v_P \mapsto \bot\}$
	$pre_{\leftrightarrow}(\text{unload B}) = \{v_T \mapsto B, v_D \mapsto L\}$	
	$pre_{\dashv}(\text{unload B}) = \{\}$	$eff_{\dashv}(\text{unload B}) = \{v_P \mapsto B\}$
	$\delta^-(\text{unload B}) = 1, \delta^+(\text{unload B}) = 1$	
(load B):	$pre_{\vdash}(\text{load B}) = \{v_P \mapsto B\}$	$eff_{\vdash}(\text{load B}) = \{v_P \mapsto \bot\}$
	$pre_{\leftrightarrow}(\text{load B}) = \{v_T \mapsto B, v_D \mapsto L\}$	
	$pre_{\dashv}(\text{load B}) = \{\}$	$eff_{\dashv}(\text{load B}) = \{v_P \mapsto T\}$
	$\delta^-(\text{load B}) = 1, \delta^+(\text{load B}) = 1$	
(unload C):	$pre_{\vdash}(\text{unload C}) = \{v_P \mapsto T\}$	$eff_{\vdash}(\text{unload C}) = \{v_P \mapsto \bot\}$
	$pre_{\leftrightarrow}(\text{unload C}) = \{v_T \mapsto C, v_D \mapsto L\}$	
	$pre_{\dashv}(\text{unload C}) = \{\}$	$eff_{\dashv}(\text{unload C}) = \{v_P \mapsto C\}$
	$\delta^-(\text{unload C}) = 1, \delta^+(\text{unload C}) = 1$	
(load C):	$pre_{\vdash}(\text{load C}) = \{v_P \mapsto C\}$	$eff_{\vdash}(\text{load C}) = \{v_P \mapsto \bot\}$
	$pre_{\leftrightarrow}(\text{load C}) = \{v_T \mapsto C, v_D \mapsto L\}$	
	$pre_{\dashv}(\text{load C}) = \{\}$	$eff_{\dashv}(\text{load C}) = \{v_P \mapsto T\}$
	$\delta^-(\text{load C}) = 1, \delta^+(\text{load C}) = 1$	

Fig. 8. All ‘load’ and ‘unload’ actions for the Worked Example

Received 29 January 2025; accepted 23 January 2026