Proceedings of the Twenty-Fourth International Joint Conference on Artificial Intelligence (IJCAI 2015)

Simulation-Based Admissible Dominance Pruning

Álvaro Torralba and **Jörg Hoffmann**
Saarland University
Saarbrücken, Germany
torralba@cs.uni-saarland.de, hoffmann@cs.uni-saarland.de

Abstract

In optimal planning as heuristic search, admissible pruning techniques are paramount. One idea is *dominance pruning*, identifying states "better than" other states. Prior approaches are limited to simple dominance notions, like "more STRIPS facts true" or "higher resource supply". We apply *simulation*, well-known in model checking, to compute much more general dominance relations based on comparing transition behavior across states. We do so effectively by expressing state-space simulations through the *composition* of simulations on orthogonal projections. We show how simulation can be made more powerful by intertwining it with a notion of *label dominance*. Our experiments show substantial improvements across several IPC benchmark domains.

1 Introduction

Heuristic search is the predominant approach to cost-optimal planning. But the number of states that must be explored to prove optimality often grows exponentially even when using extremely well-informed heuristics [Helmert and Röger, 2008]. Therefore, recent years have seen substantial effort devoted to identifying and exploiting structure allowing to prune redundant parts of the state space. Known techniques of this kind pertain to *symmetries* (e. g. [Fox and Long, 1999; 2002; Domshlak *et al.*, 2012]), *partial-order reduction* based methods like expansion-core [Chen and Yao, 2009] or strong stubborn sets [Valmari, 1989; Wehrle and Helmert, 2012; Wehrle *et al.*, 2013; Wehrle and Helmert, 2014], and *dominance pruning* [Hall *et al.*, 2013]. We follow up on the latter here.

Dominance pruning is based on identifying states "better than" other states. For example, consider a Logistics task where one truck must carry several packages to location G. Consider the position of any one package p. All other state variables having equal values, the best is to have p at G, and it is better for p to be in the truck than at any location other than G. We refer to this kind of relation between states as a *dominance relation*. (Hall et al. use the term "partial-order", which we change here to avoid ambiguity with, e. g., partial-order reduction.)

Two main questions need to be answered: *(1) How to discover the dominance relation? (2) How to simplify the search (and/or planning task) given a dominance relation?* Hall et al. answered (2) in terms of an admissible pruning method, pruning state s if a dominating state t, with an at-most-as-costly path, has already been seen. We follow that idea here, contributing a BDD implementation. Our main contribution regards (1). Hall et al. use dominance relations characterized by consumed resources: state t dominates state s if s and t are identical except that $t(r) \geq s(r)$ for all resources r.[1] Herein, we instead find the dominance relation through *simulation*, used in model checking mainly to compare different system models [Milner, 1971; Gentilini *et al.*, 2003].

A simulation is a relation $\preceq$ on states where, whenever $s \preceq t$, for every transition $s \to s'$ there exists a transition $t \to t'$ using the same action, such that $s' \preceq t'$. In words, t simulates s if anything we can do in s, we can do also in t, leading to a simulating state. (For the reader familiar with the use of bisimulation in merge-and-shrink [Helmert *et al.*, 2014]: simulation is "one half of" bisimulation.) A simulation clearly qualifies as a dominance relation. But how to find a simulation on the state space?

We employ a *compositional* approach, obtaining our simulation relation on the state space from simulation relations on *orthogonal projections*, i. e., projections whose variable subsets do not overlap. We enhance simulation with a concept of *label* (action) *dominance*, in addition to states. In our Logistics example above, e. g., for each package this detects the described relation (G is better than being in the truck is better than being at any location other than G). This yields a very strong dominance relation that allows to ignore any state in which a package is unnecessarily unloaded at an irrelevant location. Empirically, we find that indeed our pruning method often substantially reduces the number of expanded nodes.

For space reasons, we omit some proofs. Full proofs, and more examples, will be made available in a TR.

[1] Precisely, Hall et al. consider numeric state variables r and analyze whether higher r is always good, or is always bad, or neither. In Metric-FF's [Hoffmann, 2003] linear normal form, this is equivalent to the formulation above. Hall et al. also handle STRIPS facts, as variables with domain $\{0, 1\}$. But, there, their notions trivialize to "t dominates s if $t \supseteq s$".

2 Background

A **planning task** is a 4-tuple $\Pi = (V, A, I, G)$. V is a finite set of **variables** v, each $v \in V$ being associated with a finite domain D_v. A **partial state** over V is a function s on a subset $V(s)$ of V, so that $s(v) \in D_v$ for all $v \in V(s)$; s is a **state** if $V(s) = V$. The **initial state** I is a state. The **goal** G is a partial state. A is a finite set of **actions**, each $a \in A$ being a pair (pre_a, eff_a) of partial states, called its **precondition** and **effect**. Each $a \in A$ is also associated with its non-negative **cost** $c(a) \in \mathbb{R}_0^+$.

A **labeled transition system (LTS)** is a tuple $\Theta = (S, L, T, s_0, S_G)$ where S is a finite set of **states**, L is a finite set of **labels** each associated with a **label cost** $c(l) \in \mathbb{R}_0^+$, $T \subseteq S \times L \times S$ is a set of **transitions**, $s_0 \in S$ is the **start state**, and $S_G \subseteq S$ is the set of **goal states**.

The **state space** of a planning task Π is the LTS Θ_Π where: S is the set of all states; s_0 is the initial state I of Π; $s \in S_G$ iff $G \subseteq s$; the labels L are the actions A, and $s \xrightarrow{a} s'$ is a transition in T if s complies with pre_a, and $s'(v) = \mathit{eff}_a(v)$ for $v \in V(\mathit{eff}_a)$ while $s'(v) = s(v)$ for $v \in V \setminus V(\mathit{eff}_a)$. A **plan** for a state s is a path from s to any $s_G \in S_G$. The cost of a cheapest plan for s is denoted $h^*(s)$. A plan for s_0 is a plan for Π, and is **optimal** iff its cost equals $h^*(s_0)$. As defined by Wehrle and Helmert [2014], a plan for s is **strongly optimal** if its number of 0-cost actions is minimal among all optimal plans for s. We denote by $h^{0*}(s)$ the number of 0-cost actions in a strongly optimal plan of s.

Abstractions and abstract state spaces are quite common in planning (e. g. [Helmert *et al.*, 2007]). We build on this work, but only indirectly. We use merge-and-shrink abstractions as the basis from which our simulation process starts. That process itself will be described in a generic form not relying on these specific constructs. Hence, in what follows, we provide only a summary view that suffices to present our contribution.

Say we have a task $\Pi = (V, A, I, G)$ with state space $\Theta_\Pi = (S, A, T, I, S_G)$, and a variable subset $W \subseteq V$. The **projection** onto W is the function $\pi^W : S \mapsto S^W$ from the states S over V into the states S^W over W, where $\pi^W(s)$ is the restriction of s to W. The **projected state space** Θ_Π^W is the LTS $(S^W, A, T^W, \pi^W(I), S_G^W)$, where $T^W := \{(\pi^W(s), l, \pi^W(s')) \mid (s, l, s') \in T\}$ and $S_G^W := \{\pi^W(s_G) \mid s_G \in S_G\}$. Given two variable subsets W and U, (the projections onto) W and U are **orthogonal** if $W \cap U = \emptyset$. Orthogonality ensures the following **reconstruction** property: $\Theta_\Pi^W \otimes \Theta_\Pi^U = \Theta_\Pi^{W \cup U}$ for orthogonal W and U, where $\otimes$ is the **synchronized product** operation. Namely, given any two labeled transition systems $\Theta^1 = (S^1, L, T^1, s_0^1, S_G^1)$ and $\Theta^2 = (S^2, L, T^2, s_0^2, S_G^2)$ that share the same set L of labels, $\Theta^1 \otimes \Theta^2$ is the labeled transition system with states $S^1 \times S^2$, labels L, transition $(s_1, s_2) \xrightarrow{l} (s_1', s_2')$ iff $s_1 \xrightarrow{l} s_1' \in T^1$ and $s_2 \xrightarrow{l} s_2' \in T^2$, start state (s_0^1, s_0^2), and goal states $\{(s_G^1, s_G^2) \mid s_G^1 \in S_G^1, s_G^2 \in S_G^2\}$.

Merge-and-shrink abstractions [Helmert *et al.*, 2007; 2014] construct more general abstraction functions, and the corresponding abstract state spaces, by starting from *atomic abstractions* (projections onto single state variables), and interleaving *merging steps* (replacing two abstractions with their synchronized product) with *shrinking steps* (replacing an abstraction with an abstraction of itself). It has been shown that, if every shrinking step replaces the selected abstraction with a bisimulation of itself, then the final abstraction is a bisimulation of the overall state space Θ_Π. It has also been shown that such shrinking can be combined with *exact label reduction* [Sievers *et al.*, 2014]. A **label reduction** is a function τ from the labels L into a set L^τ of reduced labels preserving label cost i. e. $c(l) = c(\tau(l))$. Given an LTS Θ, denote by $\tau(\Theta)$ the LTS identical to Θ except that all labels l have been replaced by $\tau(l)$. Given a set $\{\Theta^1, \ldots, \Theta^k\}$ of LTSs sharing labels L, a label reduction τ is **exact** if $\tau(\Theta^1) \otimes \cdots \otimes \tau(\Theta^k) = \tau(\Theta^1 \otimes \cdots \otimes \Theta^k)$. Reducing labels in this way, and using bisimulation shrinking, merge-and-shrink delivers a bisimulation of $\tau(\Theta_\Pi)$.

3 Simulation Relations

Given a planning task with states S, a **dominance relation** is a binary relation $\preceq \subseteq S \times S$ where $s \preceq t$ implies $h^*(t) \leq h^*(s)$ and, if $h^*(t) = h^*(s)$ then $h^{0*}(t) \leq h^{0*}(s)$. This is exactly what is needed for admissible pruning during search, as discussed in the next section.

To find dominance relations in practice, we focus on the special case of *simulation relations*. These are well known in model-checking (e. g. [Grumberg and Long, 1994; Loiseaux *et al.*, 1995]). Here we use a variant adapted to planning (only) in making explicit the distinction between goal states and non-goal states:

Definition 1 (Simulation) *Let $\Theta = (S, L, T, s_0, S_G)$ be an LTS. A binary relation $\preceq \subseteq S \times S$ is a **simulation** for Θ if, whenever $s \preceq t$ (in words: t **simulates** s), for every transition $s \xrightarrow{l} s'$ there exists a transition $t \xrightarrow{l} t'$ s.t. $s' \preceq t'$. We call $\preceq$ **goal-respecting** for Θ if, whenever $s \preceq t$, $s \in S_G$ implies that $t \in S_G$.*

*We call $\preceq$ the **coarsest** goal-respecting simulation if, for every goal-respecting simulation $\preceq'$, we have $\preceq' \subseteq \preceq$.*

A unique coarsest goal-respecting simulation always exists and can be computed in time polynomial in the size of Θ [Henzinger *et al.*, 1995]. Note that the coarsest simulation is always *reflexive*, i. e., $s \preceq s$; the same is true of all simulation relations considered here. Intuitively, every state "dominates itself".

Observe that $s \preceq t$ implies $h^*(t) \leq h^*(s)$, because any plan for s can be also applied to t. Hence a goal-respecting simulation over states is a dominance relation. But, for obtaining that property, goal-respecting simulation is unnecessarily strict. It suffices to preserve, not the label l of the transition $s \to s'$, but only its cost:

Definition 2 (Cost-Simulation) *Let $\Theta = (S, L, T, s_0, S_G)$ be an LTS. A binary relation $\preceq \subseteq S \times S$ is a **cost-simulation** for Θ if, whenever $s \preceq t$, $s \in S_G$ implies that $t \in S_G$, and for every transition $s \xrightarrow{l} s'$ there exists a transition $t \xrightarrow{l'} t'$ s.t. $s' \preceq t'$ and $c(l') \leq c(l)$.*[2]

[2] Hall et al. [2013] include an equivalent definition, calling it "compatibility" and not relating it to simulation.

A cost-simulation still is a dominance relation, and any goal-respecting simulation is a cost-simulation but not vice versa. However, in our compositional approach where individual dominance relations are computed on orthogonal projections, we need to preserve labels for synchronization across these projections. Hence, to ensure we obtain a dominance relation of the state space, we must use goal-respecting simulation (rather than cost-simulation) on each projection.

For our notion of label dominance, it will be important to consider LTSs with **NOOPs added**. For any LTS Θ, we denote by Θ_{noop} the same LTS but with a new additional label $noop$ where $c(noop) = 0$ and, for every state s, a new transition $s \xrightarrow{noop} s$. Obviously, any dominance relation over Θ_{noop} is a dominance relation over Θ. So for our purposes it suffices to find a dominance relation over the state space $(\Theta_\Pi)_{noop}$ with NOOPs added.

4 Admissible Dominance Pruning

By **A* with dominance pruning**, we refer to the following modification of A*: *Whenever a node N with state $s(N)$ and path cost $g(N)$ is generated, check whether there exists a node N' in the open or closed lists, with state $s(N')$ and path cost $g(N')$, so that $s(N) \preceq s(N')$ and $g(N') \leq g(N)$. If so, **prune** N, i. e. do not insert it into the open list, nor into the closed list.*

As $s \preceq t$ implies $h^*(t) \leq h^*(s)$, any plan through N costs at least as much as an optimal plan through N', so A* with dominance pruning guarantees optimality. In presence of 0-cost actions, one must be careful to not prune s if this eliminates all possible plans for t. However, s cannot belong to the strongly optimal plan of t because $s \preceq t$ and $h^*(t) = h^*(s)$ implies $h^{0*}(t) \leq h^{0*}(s)$.

Dominance pruning can reduce A*'s search space, but comes with a computational overhead. First, **checking cost**, the runtime required for checking whether there exists a node N' in the open or closed lists, with the mentioned properties. Second, **maintenance cost**, the runtime and memory required for maintaining whichever data structure is used to keep checking cost (which is excessive in a naïve implementation) at bay. Depending on which of these two costs tend to be higher, variants of dominance pruning make sense.

Hall et al.'s [2013] dominance relations are characterized by resources r. They maintain, for each r, the set $S(r)$ of seen states with a positive value for r. Given a new state s, their check iterates over the states in the intersection of $S(r)$ for those r where $s(r)$ is positive. This implementation has high checking cost but low maintenance cost. Hence Hall et al. perform the check not at node-generation time, but at node-expansion time, reducing the number of checks that will be made.

To deal with our much more general dominance relations, we developed a BDD-based [Bryant, 1986] implementation. This has low checking cost but high maintenance cost. Hence we perform the check at node-generation time, but only against the closed list, reducing the number of maintenance operations needed. We maintain a BDD $\mathcal{B}_g$ for the set of states simulated by any $s(N')$ where N' is a previously expanded node with $g(N') = g$. This is done for every g-value of the expanded nodes so far. Every time a node N' is expanded, we determine the set of states $S_{\preceq s(N')}$ simulated by $s(N')$, and add $S_{\preceq s(N')}$ into $\mathcal{B}_{g(N')}$. The checking operation then is very fast: when a node N is generated, test membership of $s(N)$ in $\mathcal{B}_g$ for all $g \leq g(N)$. Each such test takes time linear in the size of the state.

5 The Compositional Approach

As hinted, our approach is *compositional*, constructing the dominance relation over the state space Θ_Π as the composition of simulation relations over orthogonal projections thereof. Stating this in a generic manner (and simplifying to the atomic case of two orthogonal projections), we have an LTS Θ^{12} which equals the synchronized product $\Theta^1 \otimes \Theta^2$ of two smaller LTSs. We obtain a simulation for Θ^{12} from simulations for Θ^1 and Θ^2:

Definition 3 (Relation Composition) *Let $\Theta^1 = (S^1, L, T^1, s^1_0, S^1_G)$ and $\Theta^2 = (S^2, L, T^2, s^2_0, S^2_G)$ be LTSs sharing the same labels. For binary relations $\preceq_1 \subseteq S_1 \times S_1$ and $\preceq_2 \subseteq S_2 \times S_2$, the **composition** of $\preceq_1$ and $\preceq_2$, denoted $\preceq_1 \otimes \preceq_2$, is the binary relation on $S_1 \times S_2$ where $(s_1, s_2)(\preceq_1 \otimes \preceq_2)(t_1, t_2)$ iff $s_1 \preceq_1 t_1$ and $s_2 \preceq_2 t_2$.*

Proposition 1 *Let $\Theta^{12} = \Theta^1 \otimes \Theta^2$, and let $\preceq_1$ and $\preceq_2$ be goal-respecting simulations for Θ^1 and Θ^2 respectively. Then $\preceq_1 \otimes \preceq_2$ is a goal-respecting simulation for Θ^{12}.*

The proof is direct by definition, and is almost identical to that of a similar result concerning bisimulation, stated by Helmert et al. [2014].

Our basic idea can now be described as follows. Say we have a planning task $\Pi = (V, A, I, G)$ with state space Θ_Π, a partition $V_1, \ldots, V_k$ of the task's variables, and a goal-respecting bisimulation abstraction α_i of each $\tau(\Theta^{V_i}_\Pi)$ where τ is an exact label reduction. This is precisely the input we will get from merge-and-shrink abstraction. We will henceforth refer to this input as our *initial abstractions*. Say we construct a goal-respecting simulation $\preceq_i$ for each abstract state space Θ^{α_i}. Because bisimulation is a special case of simulation, $\preceq_i$ is a goal-respecting simulation for $\tau(\Theta^{V_i}_\Pi)$. Applying Definition 3 and Proposition 1 iteratively, $\bigotimes_i \preceq_i$ is a goal-respecting simulation for $\bigotimes_i \tau(\Theta^{V_i}_\Pi)$. Because τ is exact, $\bigotimes_i \tau(\Theta^{V_i}_\Pi) = \tau(\bigotimes_i \Theta^{V_i}_\Pi)$, and by the reconstruction property $\tau(\bigotimes_i \Theta^{V_i}_\Pi) = \tau(\Theta_\Pi)$. Because the label reduction is cost-preserving, $\bigotimes_i \preceq_i$ is a cost-simulation for Θ_Π, and hence a dominance relation as desired.

One can use this result and method as-is, obtaining a new dominance pruning method as a corollary of suitably assembling existing results and methods. However, empirically, this method's ability to find interesting dominance relations is quite limited. We now extend the simulation concept to overcome that problem.

6 Label-Dominance Simulation

Sievers et al. [2014] introduce label "subsumption", where l' subsumes l if it labels all transitions labeled by l. To intertwine dominance between labels with dominance between states, we extend that concept as follows:

Definition 4 (Label Dominance) *Let Θ be an LTS with states S, let $\preceq \subseteq S \times S$ be any binary relation on S, and let l, l' be labels. We say that l' **dominates** l **in** Θ **given** $\preceq$ if $c(l') \leq c(l)$, and for every transition $s \xrightarrow{l} s'$ there exists a transition $s \xrightarrow{l'} t'$ s.t. $s' \preceq t'$.*

The relation $\preceq$ here is arbitrary, but will be a simulation in practice. Hence, intuitively, a label dominates another one if it "applies to the same states and always leads to an at least as good state". To give a simple example, consider the LTS corresponding to a single vehicle's position, and say we have a 0-cost "beam" action which takes us from any position to the vehicle's goal. Provided that every position is, per $\preceq$, simulated by the goal position, "beam" dominates all other labels.

In IPC benchmarks, typically Definition 4 is important not for regular actions, but for NOOPs. For illustration, say we have a truck variable v_T, two locations A and B, and a package variable v_P whose goal is to be at B. Our variable partition is the trivial one, $V_1 = \{v_T\}$ and $V_2 = \{v_P\}$. Bisimulation using exact label reduction will return LTSs as shown in Figure 1. The "load" and "unload" actions get reduced in a way allowing to synchronize with the correct truck position; the distinction between the truck "drive" actions is irrelevant so these are reduced to the same label. Clearly, no label dominates any other in either of these two LTSs. However, consider Θ^1 and Θ^2 with NOOPs added. The new label *noop* dominates the load/unload actions in Θ^1, and dominates the drive actions in Θ^2, provided $\preceq_1$ and $\preceq_2$ are reflexive as will be the case in practice.

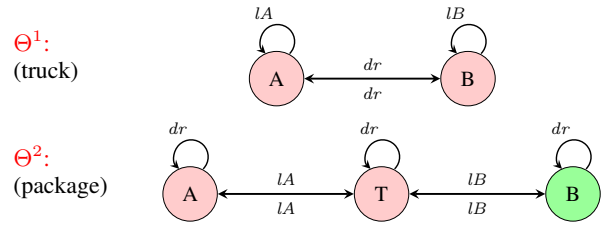

Figure 1: Label-reduced bisimulations, i. e. the input to our simulation process, in the Logistics example.

This behavior allows us, e. g., to conclude that, in Θ^2, B dominates T: While T has an outgoing transition to B, labeled lB, B itself has no such outgoing label. However, B has the outgoing label *noop* leading to B. The transition $B \xrightarrow{noop} B$ simulates $T \xrightarrow{lB} B$, except that it uses label $l' = noop$ instead of label $l = lB$. This is admissible (only) if l' dominates l in all other LTSs involved. In our case here, the only other LTS is Θ^1, and indeed the label $l' = noop$ dominates $l = lB$ in that LTS. We exploit this kind of information as follows:

Definition 5 (Label-Dominance Simulation) *Let $\mathcal{T} = \{\Theta^1, \ldots, \Theta^k\}$ be a set of LTSs sharing the same labels. Denote the states of Θ^i by S_i. A set $\mathcal{R} = \{\preceq_1, \ldots, \preceq_k\}$ of binary relations $\preceq_i \subseteq S_i \times S_i$ is a **label-dominance simulation** for $\mathcal{T}$ if, whenever $s \preceq_i t$, $s \in S_i^G$ implies that $t \in S_i^G$, and for every transition $s \xrightarrow{l} s'$ in Θ^i, there exists a transition $t \xrightarrow{l'} t'$ in Θ^i such that $c(l') \leq c(l)$, $s' \preceq_i t'$, and, for all $j \neq i$, l' dominates l in Θ^j given $\preceq_j$.*

*We call $\mathcal{R}$ the **coarsest** label-dominance simulation if, for every label-dominance simulation $\mathcal{R}' = \{\preceq'_1, \ldots, \preceq'_k\}$ for $\mathcal{T}$, we have $\preceq'_i \subseteq \preceq_i$ for all i.*

A unique coarsest label-dominance simulation always exists, and can be computed in time polynomial in the size of $\mathcal{T}$. We will prove this in the next section as a corollary of specifying our algorithm for doing this computation.

In the example, $T \preceq_2 B$ holds because $s \xrightarrow{l} s'$ in Θ^2 is $T \xrightarrow{lB} B$, and the simulating $t \xrightarrow{l'} t'$ in Θ^2 is $B \xrightarrow{noop} B$, which works because $c(noop) = 0 \leq 1 = c(lB)$, $B \preceq_2 B$, and *noop* dominates lB in Θ^1. In the same fashion, provided that $A \preceq_2 B$, the transition $T \xrightarrow{lA} A$ is simulated by $B \xrightarrow{noop} B$. Note that neither of these two inferences could be made with the standard concept of simulation (even after exact label reduction), because that concept insists on using the same labels, not dominating ones.

We now prove soundness of label-dominance simulation, i. e., that label-dominance simulations $\mathcal{R}$ yield cost-simulations of the original state space. Similarly to before, we iteratively compose $\mathcal{R}$'s element relations, as captured by the following lemma:

Lemma 1 *Let $\mathcal{T} = \{\Theta^1, \ldots, \Theta^k\}$ be a set of LTSs sharing the same labels, and let $\mathcal{R} = \{\preceq_1, \ldots, \preceq_k\}$ be a label-dominance simulation for $\mathcal{T}$. Then $\{\preceq_1 \otimes \preceq_2, \preceq_3, \ldots, \preceq_k\}$ is a label dominance simulation for $\{\Theta^1 \otimes \Theta^2, \Theta^3, \ldots, \Theta^k\}$.*

Proof Sketch: The claim regarding $\preceq_3, \ldots, \preceq_k$ is simple. For $\preceq_1 \otimes \preceq_2$, consider states $(s_1, s_2) \preceq_{12} (t_1, t_2)$ and a transition $(s_1, s_2) \xrightarrow{l} (s'_1, s'_2)$. We identify a dominating transition $(t_1, t_2) \xrightarrow{l'} (t'_1, t'_2)$ as follows: 1. As $s_1 \preceq_1 t_1$, obtain a transition $t_1 \xrightarrow{l^{tmp}} t_1^{tmp}$ dominating $s_1 \xrightarrow{l} s'_1$ in Θ^1. 2. As l^{tmp} dominates l in Θ^2, obtain a transition $s_2 \xrightarrow{l^{tmp}} s_2^{tmp}$ dominating $s_2 \xrightarrow{l} s'_2$ in Θ^2. 3. As $s_2 \preceq_2 t_2$, obtain a transition $t_2 \xrightarrow{l'} t'_2$ dominating $s_2 \xrightarrow{l^{tmp}} s_2^{tmp}$ in Θ^2. 4. As l' dominates l^{tmp} in Θ^1, obtain a transition $t_1 \xrightarrow{l'} t'_1$ dominating $t_1 \xrightarrow{l^{tmp}} t_1^{tmp}$ in Θ^1. □

Theorem 1 *Let $\mathcal{T} = \{\Theta^1, \ldots, \Theta^k\}$ be a set of LTSs sharing the same labels, and let $\mathcal{R} = \{\preceq_1, \ldots, \preceq_k\}$ be a label-dominance simulation for $\mathcal{T}$. Then $\bigotimes_i \preceq_i$ is a cost-simulation for $\bigotimes_i \Theta^i$.*

Proof: Applying Lemma 1, we get that $\bigotimes_i \preceq_i$ is a label-dominance simulation for $\{\bigotimes_i \Theta^i\}$. Now, for such a singleton set of LTSs, the requirements on the transition $t \xrightarrow{l'} t'$ replacing $s \xrightarrow{l} s'$ are that $c(l') \leq c(l)$, and $s' \preceq_i t'$. Hence label-dominance simulation simplifies to cost-simulation, and the claim follows. □

To clarify the overall process, assume now again the initial abstractions provided by merge-and-shrink abstraction, i. e.,

a partition $V_1, \ldots, V_k$ of the variables, and a goal-respecting bisimulation abstraction α_i, with abstract state space Θ^{α_i}, of each $\tau(\Theta_\Pi^{V_i})$ where τ is an exact label reduction. We compute the coarsest label-dominance simulation $\mathcal{R} = \{\preceq_1, \ldots, \preceq_k\}$ for $\mathcal{T} := \{\Theta^{\alpha_1}_{noop}, \ldots, \Theta^{\alpha_k}_{noop}\}$. As adding NOOPs does not affect bisimulation and is interchangeable with the synchronized product, with Theorem 1 we have that $\bigotimes_i \preceq_i$ is a cost-simulation for $[\bigotimes_i \tau(\Theta_\Pi^{V_i})]_{noop}$, and hence a cost-simulation for Θ_Π as desired.

7 Computing $\mathcal{R}$

We now show how to operationalize Definition 5: Given $\mathcal{T} = \{\Theta^1, \ldots, \Theta^k\}$, how to compute the coarsest label-dominance simulation $\mathcal{R}$ for $\mathcal{T}$?

It is well known that the coarsest simulation can be computed in time polynomial in the size of the input LTS [Henzinger *et al.*, 1995]. The algorithm starts with the most generous relation $\preceq$ possible, then iteratively removes pairs $s \preceq t$ that do not satisfy the simulation condition. When no more changes occur, the unique coarsest simulation has been found. This method extends straightforwardly to label-dominance simulation.

Proposition 2 *Let $\mathcal{T} = \{\Theta^1, \ldots, \Theta^k\}$ be a set of LTSs sharing the same labels. Then a unique coarsest label-dominance simulation for $\mathcal{T}$ exists.*

Proof: The identity relation is a label-dominance simulation. If $\mathcal{R} = \{\preceq_1, \ldots, \preceq_k\}$ and $\mathcal{R}' = \{\preceq'_1, \ldots, \preceq'_k\}$ are label-dominance simulations, then $\{\preceq_1 \cup \preceq'_1, \ldots, \preceq_k \cup \preceq'_k\}$, also is a label-dominance simulation. □

Denote the states of Θ^i by S_i. Define the Boolean function $\mathbf{Ok}(i, s, t)$, where $s \preceq_i t$, to return **true** iff the condition for label-dominance simulation holds, i. e., iff $s \in S_i^G$ implies that $t \in S_i^G$, and for every transition $s \xrightarrow{l} s'$ in Θ^i there exists a transition $t \xrightarrow{l'} t'$ in Θ^i such that $c(l') \leq c(l)$, $s' \preceq_i t'$, and, for all $j \neq i$, l' dominates l in Θ^j given $\preceq_j$. Our algorithm proceeds as follows:

For all i, set $\preceq_i := \{(s,t) \mid s, t \in S_i, s \notin S_G^i \text{ or } t \in S_G^i\}$
while ex. (i, s, t) s.t. not $\mathbf{Ok}(i, s, t)$ **do**
  Select one such triple (i, s, t)
  Set $\preceq_i := \preceq_i \setminus \{(s,t)\}$
endwhile
return $\mathcal{R} := \{\preceq_1, \ldots, \preceq_k\}$

Proposition 3 *Let $\mathcal{T} = \{\Theta^1, \ldots, \Theta^k\}$ be a set of LTSs sharing the same labels. Our algorithm terminates in time polynomial in the size of $\mathcal{T}$, and returns the coarsest label-dominance simulation for $\mathcal{T}$.*

Proof: Each iteration reduces one $\preceq_i$ by one element. This gives a polynomial bound on the number of iterations, and every iteration takes polynomial time.

The returned $\mathcal{R}$ is a label-dominance simulation as that is the termination condition. $\mathcal{R}$ is coarsest as every label-dominance simulation must refine the initial relations $\{(s,t) \mid s, t \in S_i, s \notin S_G^i \text{ or } t \in S_G^i\}$, and every time we remove a pair (s,t) we know that $s \not\preceq_i t$ in any label-dominance simulation. □

8 Experiments

Our techniques are implemented in Fast Downward (FD) [Helmert, 2006]. We ran all optimal-track STRIPS planning instances from the international planning competitions (IPC'98 – IPC'14). All experiments were conducted on a cluster of Intel E5-2660 machines running at 2.20 GHz, with time (memory) cut-offs of 30 minutes (4 GB). We run A* with FD's blind heuristic, and with LM-cut [Helmert and Domshlak, 2009]. We perform an ablation study of label-dominance simulation, vs. standard simulation (neither NOOPs nor label-dominance), vs. bisimulation as computed by merge-and-shrink (not doing any work on top of merge-and-shrink, just using its output for the pruning). To represent the state of the art in alternative pruning methods, we include the best-performing partial-order reduction based on strong stubborn sets, which dominates other partial-order pruning approaches such as expansion-core [Wehrle *et al.*, 2013].

Our initial abstractions are obtained using merge-and-shrink with exact label reduction, bisimulation shrinking, and the non-linear merge DFP strategy [Dräger *et al.*, 2006; 2009; Sievers *et al.*, 2014]. We impose two bounds on this process, namely a time limit of 300 seconds, as well as a limit M on the number of abstract transitions. When either of these limits is reached, the last completed abstractions form the starting point for our simulation process, i. e., are taken to be the initial abstractions. With this arrangement of parameters, the trade-off between merge-and-shrink overhead incurred vs. benefits gained is relatively easy to control. The bound on transitions works better than the more usual bound on abstract states, because the same number of abstract states may lead to widely differing numbers of transitions and thus actual effort. A reasonably good "magic" setting for M, in our current context, is $100k$. For $M = 0$, i. e. computing the component simulations on individual state variables only, performance is substantially worse. For $M = 200k$, the overhead becomes prohibitive. In between, overall coverage undergoes relatively small changes only (in the order of 5 instances).

Consider Table 1. With our pruning method, nodes are first generated and then checked for pruning, so the evaluated states are exactly the non-pruned generated ones. Hence the number of evaluated states assesses our pruning power, and the ratio between generated nodes and search time assesses the average time-per-node. The "safety belt" disables pruning if, after 1000 expansions, no node has been pruned. This is a simple yet effective method to avoid runtime overhead in cases where no or not much pruning will be obtained.

Compared to partial-order reduction, simulation-based pruning tends to be "stronger on its own, but less complementary to LM-cut". Consider first the blind heuristic, which assesses the pruning power of each technique "on its own". Simulation-based pruning typically yields much stronger evaluation reductions, the only clear exception being ParcPrinter where partial-order reduction excels. This results in much better coverage in many domains and overall. With LM-cut, on the other hand, while simulation-based pruning still applies more broadly – there are 14 test suites where it reduces evaluations but partial-order reduction does not – the extent of the reduction is dramatically diminished.

		Blind												LM-cut											
		Coverage						Evaluations				Gen/sec.		Coverage						Evaluations				Gen/sec.	
Domain	#	A	L_0	L	S	B	P	L	S	B	P	L	P	A	L_0	L	S	B	P	L	S	B	P	L	P
Airport	50	**22**	-7	-7	-7	**0**	-1	1.2	1.2	1	4.4	341	11.3	28	-3	-1	-1	-1	**+1**	1	1	1	4.7	1.1	2
Driverlog	20	7	**+2**	**+2**	0	0	0	15.8	2	2	1	4.8	2.8	13	0	0	0	0	0	1.9	1.2	1.2	1	1.1	1.2
Elevators08	30	**14**	-1	**0**	**0**	**0**	**0**	1	1	1	1.1	0.9	4.1	22	0	0	0	0	0	1	1	1	1.3	1	1.2
Elevators11	20	**12**	-1	**0**	**0**	**0**	**0**	1	1	1	1.1	1	4.2	18	0	0	0	0	0	1	1	1	1.2	1	1.2
Floortile11	20	2	**+4**	**+4**	**+4**	0	0	177	177	1.8	1.3	5.7	3.7	7	**+1**	**+1**	**+1**	0	0	6.4	6.4	1	1	1.1	1.1
Floortile14	20	0	**+5**	**+5**	**+5**	0	0	–	–	–	–	–	–	6	**+2**	**+2**	**+2**	0	0	6.3	6.3	1	1	1.3	1.1
FreeCell	80	**20**	-7	**0**	**0**	**0**	-6	1	1	1	1	1	31.2	**15**	-1	**0**	**0**	**0**	**0**	1	1	1	1	0.9	1.4
Gripper	20	8	**+6**	**+6**	**+6**	**+6**	0	53968	53968	28353	1	292	3.1	7	**+7**	**+7**	**+7**	**+7**	0	14662	14662	10049	1	31.9	1.3
Hiking14	20	**11**	**0**	**0**	**0**	**0**	-3	2.4	1.9	1.8	1	3.1	30.5	9	0	0	0	0	0	1.7	1.5	1.5	1	1.9	1.8
Logistics00	28	10	**+7**	+6	0	0	0	32.7	3.1	1.2	1	9.3	3	20	0	0	0	0	0	1.9	1.1	1.1	2.9	0.8	1.4
Logistics98	35	2	**+1**	**+1**	0	0	0	6.7	1.2	1.2	1.5	4	4.4	6	0	0	0	0	0	1.3	1	1	4.3	0.9	1.3
Miconic	150	55	**+6**	**+6**	-1	0	-5	58.3	8.7	3.4	1	15.6	5.5	141	0	0	0	0	0	2.1	1.5	1.1	1	0.6	1.1
Mprime	35	**20**	-1	-1	**0**	**0**	-1	1.1	1	1	1	18	18.5	22	0	0	0	0	0	1.1	1	1	1	1	1.1
Mystery	30	**15**	-3	-3	-3	**0**	**0**	1.9	1.9	1	1.1	29.2	14.2	17	0	0	0	0	0	3.5	3.5	1	1.4	3.4	1.8
NoMystery	20	8	**+10**	**+10**	+1	+1	0	2497	128	29.1	1.1	46.4	23	14	**+6**	**+6**	+3	0	0	6.5	3.1	1	1	0.6	1.2
OpenStack08	30	22	**+2**	**+2**	**+2**	+1	0	2.1	2	1.8	2	8.1	9.3	21	0	0	0	0	0	2.5	2.4	2.1	1.8	2.3	2
OpenStack11	20	17	**+2**	**+2**	**+2**	+1	0	2.1	2	1.8	2	7.8	9.3	16	0	0	0	0	0	2.5	2.4	2.1	1.8	2.3	2.1
OpenStack14	20	3	0	0	0	0	**+1**	2.8	2.8	2.5	1.8	7	7.8	3	0	0	0	0	0	2.9	2.8	2.5	1.8	2.4	2.1
ParcPrint08	30	10	+6	+5	+3	+1	**+20**	862	10	1.5	18349	13.6	532	18	0	0	0	0	**+12**	5	1.2	1.1	1028	2.2	20.3
ParcPrint11	20	6	+6	+5	+3	+1	**+14**	869	10	1.5	21826	11.2	371	13	0	0	0	0	**+7**	5	1.2	1.1	1246	2.4	17.8
PegSol08	30	27	0	0	0	0	0	1	1	1	1	1	3.8	**28**	-1	**0**	**0**	**0**	-1	1	1	1	1	1	1.4
PegSol11	20	17	0	0	0	0	0	1	1	1	1	1	3.8	**18**	-1	**0**	**0**	**0**	-1	1	1	1	1	1	1.4
PipesNoTank	50	**17**	-8	-1	-1	**0**	-3	1	1	1	1	1.1	10.3	**17**	-3	**0**	**0**	**0**	**0**	1	1	1	1	1	1.1
PipesTank	50	**12**	-1	**0**	**0**	**0**	-3	1.1	1.1	1.1	1	16.8	25.2	**12**	**0**	**0**	**0**	**0**	-1	1.8	1.8	1.8	1	1.1	1.2
Rovers	40	6	**+2**	**+2**	+1	0	+1	33.4	9.6	1.7	2	20.6	3	7	**+2**	**+2**	+1	+1	**+2**	6.1	3.8	1.2	4.4	1.8	1.8
Satellite	36	6	0	0	0	0	0	72.9	35.3	9.9	10.7	8.4	3.8	7	+3	+3	+3	+3	**+4**	4.8	1.8	1.7	21.5	0.9	2.3
Scanalyzer08	30	**12**	**0**	**0**	**0**	**0**	-4	1	1	1	1	1	8.7	**15**	-1	-1	-1	**0**	**0**	1	1	1	1	1	1.2
Scanalyzer11	20	**9**	**0**	**0**	**0**	**0**	-4	1	1	1	1	1	8.7	**12**	-1	-1	-1	**0**	**0**	1	1	1	1	1	1.2
Sokoban08	30	**22**	-9	**0**	**0**	**0**	-1	1	1	1	1	1.7	8.2	**29**	-7	-1	**0**	**0**	**0**	1	1	1	1	1.1	1.2
Sokoban11	20	**19**	-9	**0**	**0**	**0**	-1	1	1	1	1	1.6	8.1	**20**	-2	**0**	**0**	**0**	**0**	1	1	1	1	1.1	1.2
Tetris	17	9	-6	**-1**	**-1**	**-1**	-4	1	1	1	1	5.2	52.2	6	-3	-2	-2	-2	**-1**	1	1	1	1	1	1.3
Tidybot11	20	9	-8	-7	-7	**-1**	-2	5.5	5.5	1	1.8	59.4	8.5	**14**	-2	-2	-2	**0**	**0**	6.8	6.8	1	1.5	2.6	1.3
Tidybot14	20	2	-2	-2	-2	**-1**	-2	–	–	–	–	–	–	9	-7	-7	-7	**-1**	**-1**	3.9	3.9	1	1.7	3.1	1.4
TPP	30	6	0	0	0	0	0	6.5	3.4	1	1	22.7	3.3	6	**+1**	**+1**	**+1**	**+1**	0	1.2	1.1	1	1	1.3	1.1
Transport14	20	**7**	**0**	**0**	**0**	**0**	-1	1	1	1	1	1	9.1	6	0	0	0	0	0	1.4	1.4	1.4	1	1.4	1.2
Trucks	30	6	**+2**	**+2**	0	0	0	24.8	21.9	2.8	1	13.8	6	10	0	0	0	0	0	2.7	2.3	1	1	1.2	1.2
VisitAll11	20	9	0	0	0	0	0	30	25.5	1	1	104	3.5	10	**+1**	**+1**	**+1**	0	0	7	6.8	1	1	1.5	1.1
VisitAll14	20	3	**+1**	**+1**	**+1**	0	0	27.8	23.4	1	1	92.8	3.5	5	0	0	0	0	0	5.2	5.1	1	1	1.6	1.1
Woodwork08	30	8	**+10**	**+10**	+5	+4	+7	981	112	87.8	488	7.6	7.5	17	+7	+7	+5	+5	**+10**	91.4	23.7	16.9	762	1.8	3.1
Woodwork11	20	3	**+9**	**+9**	+5	+4	+6	1059	116	92.2	514	6.7	6.3	12	+5	+5	+4	+4	**+7**	91.6	23.8	17	772	1.8	2.9
Zenotravel	20	8	**+1**	**+1**	0	0	0	41.6	1.5	1.1	1	4.3	6.2	13	0	0	0	0	0	3.6	1.6	1	1	1	1.2
$\sum$	1271	605	+19	**+57**	+16	+16	+8							833	+3	+20	+14	+17	**+38**						

Table 1: Experiments. "A": A^* without pruning. "L_0", "L": label-dominance simulation; "S": simulation; "B": bisimulation; "L_0" is without safety belt (see text), all others with safety belt. "P": partial-order reduction. Domains where no changes in coverage occur anywhere are omitted. "Evaluations" is the factor by which the per-domain summed-up number of evaluated states, relative to "A", decreases. "Gen/sec." is the factor by which the per-node runtime (summed-up number of generated nodes divided by summed-up search time), relative to "A", increases.

Partial-order reduction suffers from this as well, but retains much of its power in ParcPrinter and Woodworking, and consistently causes very little runtime overhead relative to this slow heuristic function. Thus partial-order reduction has better overall coverage. It does not dominate simulation-based pruning though, which yields better coverage in Floortile, Gripper, NoMystery, TPP, and VisitAll.

Label-dominance simulation clearly pays off against standard simulation as well as bisimulation. The latter already is very helpful in some domains, like Gripper and Woodworking. Simulation does add over this, but suffers in some domains, like Tidybot, from the runtime overhead. Label-dominance simulation has such issues as well, but makes up for them by more pronounced gains on other domains.

The per-node runtime overhead in simulation-based pruning is almost consistently outweighed by the search space size reduction (compare the respective "Gen/sec." vs. "Evaluations" columns in Table 1). The most substantial runtime overhead stems from computing the simulation relations. Our current implementation of that process is largely naïve. We experimented with ideas from model checking for doing this more effectively, but with limited success due to the different context (especially, label-dominance). It remains an important open topic to improve this part of our machinery.

9 Conclusion

The idea of pruning states based on some form of "dominance" is old, but has previously been incarnated in planning with simple special cases ("more facts true", "more resources available") only. Simulation relations are *the* natural framework to move beyond this. Our work constitutes a first step towards leveraging the power of simulation relations in, as well as extending them for, admissible pruning in planning. The method is orthogonal to existing pruning methods, and empirically exhibits complementary strengths relative to partial-order reduction, so there is potential for synergy. A major challenge in our view is how to intelligently control initial-abstraction size, investing a lot of overhead where simulation pruning is promising and, ideally, avoiding any overhead altogether where it is not.

Acknowledgments. We thank the anonymous reviewers, whose comments helped to improve the paper. This work was partially supported by the EU FP7 Programme under grant agreement 295261 (MEALS).

References

[Bryant, 1986] Randal E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Transactions on Computers*, 35(8):677–691, 1986.

[Chen and Yao, 2009] Yixin Chen and Guohui Yao. Completeness and optimality preserving reduction for planning. In *Proc. 21st International Joint Conference on Artificial Intelligence (IJCAI 2009)*, pages 1659–1664, 2009.

[Domshlak *et al.*, 2012] Carmel Domshlak, Michael Katz, and Alexander Shleyfman. Enhanced symmetry breaking in cost-optimal planning as forward search. In *Proc. 22nd International Conference on Automated Planning and Scheduling (ICAPS'12)*, 2012.

[Dräger *et al.*, 2006] Klaus Dräger, Bernd Finkbeiner, and Andreas Podelski. Directed model checking with distance-preserving abstractions. In *Proceedings of the 13th International SPIN Workshop (SPIN 2006)*, volume 3925 of *Lecture Notes in Computer Science*, pages 19–34, 2006.

[Dräger *et al.*, 2009] Klaus Dräger, Bernd Finkbeiner, and Andreas Podelski. Directed model checking with distance-preserving abstractions. *STTT*, 11(1):27–37, 2009.

[Fox and Long, 1999] Maria Fox and Derek Long. The detection and exploitation of symmetry in planning problems. In *Proc. 16th International Joint Conference on Artificial Intelligence (IJCAI-99)*, pages 956–961, 1999.

[Fox and Long, 2002] Maria Fox and Derek Long. Extending the exploitation of symmetries in planning. In *Proc. 6th International Conference on Artificial Intelligence Planning and Scheduling (AIPS-02)*, pages 83–91, 2002.

[Gentilini *et al.*, 2003] Raffaella Gentilini, Carla Piazza, and Alberto Policriti. From bisimulation to simulation: Coarsest partition problems. *Journal of Automated Reasoning*, 31(1):73–103, 2003.

[Grumberg and Long, 1994] Orna Grumberg and David E. Long. Model checking and modular verification. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 16(3):843–871, 1994.

[Hall *et al.*, 2013] David Hall, Alon Cohen, David Burkett, and Dan Klein. Faster optimal planning with partial-order pruning. In *Proc. 23rd International Conference on Automated Planning and Scheduling (ICAPS'13)*, 2013.

[Helmert and Domshlak, 2009] Malte Helmert and Carmel Domshlak. Landmarks, critical paths and abstractions: What's the difference anyway? In *Proc. 19th International Conference on Automated Planning and Scheduling (ICAPS'09)*, pages 162–169, 2009.

[Helmert and Röger, 2008] Malte Helmert and Gabriele Röger. How good is almost perfect? In *Proc. 23rd National Conference of the American Association for Artificial Intelligence (AAAI-08)*, pages 944–949, 2008.

[Helmert *et al.*, 2007] Malte Helmert, Patrik Haslum, and Jörg Hoffmann. Flexible abstraction heuristics for optimal sequential planning. In *Proc. 17th International Conference on Automated Planning and Scheduling (ICAPS'07)*, pages 176–183, 2007.

[Helmert *et al.*, 2014] Malte Helmert, Patrik Haslum, Jörg Hoffmann, and Raz Nissim. Merge & shrink abstraction: A method for generating lower bounds in factored state spaces. *Journal of the Association for Computing Machinery*, 61(3), 2014.

[Helmert, 2006] Malte Helmert. The Fast Downward planning system. *Journal of Artificial Intelligence Research*, 26:191–246, 2006.

[Henzinger *et al.*, 1995] Monika Rauch Henzinger, Thomas A. Henzinger, and Peter W. Kopke. Computing simulations on finite and infinite graphs. In *36th Annual Symposium on Foundations of Computer Science.*, pages 453–462, 1995.

[Hoffmann, 2003] Jörg Hoffmann. The Metric-FF planning system: Translating "ignoring delete lists" to numeric state variables. *Journal of Artificial Intelligence Research*, 20:291–341, 2003.

[Loiseaux *et al.*, 1995] Claire Loiseaux, Susanne Graf, Joseph Sifakis, Ahmed Bouajjani, and Saddek Bensalem. Property preserving abstractions for the verification of concurrent systems. *Formal Methods in System Design*, 6(1):11–44, 1995.

[Milner, 1971] Robin Milner. An algebraic definition of simulation between programs. In *Proc. 2nd International Joint Conference on Artificial Intelligence (IJCAI-71)*, pages 481–489, 1971.

[Sievers *et al.*, 2014] Silvan Sievers, Martin Wehrle, and Malte Helmert. Generalized label reduction for merge-and-shrink heuristics. In *Proc. 28th AAAI Conference on Artificial Intelligence (AAAI'14)*, pages 2358–2366, 2014.

[Valmari, 1989] Antti Valmari. Stubborn sets for reduced state space generation. volume 483 of *Lecture Notes in Computer Science*, pages 491–515, 1989.

[Wehrle and Helmert, 2012] Martin Wehrle and Malte Helmert. About partial order reduction in planning and computer aided verification. In *Proc. 22nd International Conference on Automated Planning and Scheduling (ICAPS'12)*, 2012.

[Wehrle and Helmert, 2014] Martin Wehrle and Malte Helmert. Efficient stubborn sets: Generalized algorithms and selection strategies. In *Proc. 24th International Conference on Automated Planning and Scheduling (ICAPS'14)*, 2014.

[Wehrle *et al.*, 2013] Martin Wehrle, Malte Helmert, Yusra Alkhazraji, and Robert Mattmüller. The relative pruning power of strong stubborn sets and expansion core. In *Proc. 23rd International Conference on Automated Planning and Scheduling (ICAPS'13)*, 2013.