Efficient Symbolic Search for Cost-Optimal Planning

Álvaro Torralba[a], Vidal Alcázar[b], Peter Kissmann[a], Stefan Edelkamp[c]

[a] *Saarland University, Saarbrücken, Germany*
[b] *Universidad Carlos III de Madrid, Madrid, Spain*
[c] *Universität Bremen, Bremen, Germany*

Abstract

In cost-optimal planning we aim to find a sequence of operators that achieve a set of goals with minimum cost. Symbolic search with Binary Decision Diagrams (BDDs) performs efficient state space exploration in terms of time and memory. This is crucial in optimal settings, in which large parts of the state space must be explored in order to prove optimality. However, the development of accurate heuristics for explicit-state search in recent years have left symbolic search techniques in a secondary place.

In this article we propose two orthogonal improvements for symbolic search planning. On the one hand, we analyze and compare different methods for image computation in order to efficiently perform the successor generation on symbolic search. Image computation is the main bottleneck of symbolic search algorithms so an efficient computation is paramount for efficient symbolic search planning. On the other hand, we study how to use state-invariant constraints to prune states in symbolic search. This is essential in regression search but it is yet to be exploited in symbolic search planners.

Experiments with symbolic bidirectional uniform-cost search and symbolic A* search with PDBs show remarkable performance improvements on most IPC benchmark domains. Overall, with the help of our improvements, symbolic bidirectional search outperforms explicit-state search with state-of-the-art heuristics such as LM-CUT across many different domains.

Keywords: Cost-Optimal Planning, Symbolic Search, Image Computation, State Invariants

1. Introduction

Classical planning consists of finding a sequence of operators, commonly called a plan, that achieves a set of goals from a given initial state. The objective of optimal planners is to find the plan of minimum cost, defined by the sum of the cost of its operators. A prominent method for cost-optimal planning is state-space search. Dijkstra search [1] is a classical algorithm also known as "Uniform-Cost Search", which is arguably a more precise name [2]. Uniform-cost search always expands a reachable state with least cost among those yet unexpanded, thus guaranteeing an optimal solution when a goal state is expanded. A* search [3] uses *admissible heuristics* that provide a lower bound on the distance from a state to the goal to reduce the number of expanded states. State-space search algorithms can be classified by the direction in which they traverse the search space: progression or regression. In progression, search is performed in forward direction, from the initial state towards the goal. In regression, backward search is performed from the set of goal states towards the initial state. Additionally, bidirectional algorithms search in both directions simultaneously.

Forward A* with admissible heuristics is the most popular approach for cost-optimal planning, thanks to effective domain-independent admissible heuristics such as Merge-and-Shrink [4] or LM-cut [5]. On the other hand, regression and bidirectional search have lost importance due to the shortcomings associated with regression in planning [6]. Regression in planning is commonly thought to be less robust than progression

Email addresses: torralba@cs.saarland-uni.de (Álvaro Torralba), vidal.alcazar.saiz@gmail.com (Vidal Alcázar), peter.kissmann@googlemail.com (Peter Kissmann), edelkamp@tzi.de (Stefan Edelkamp)

due to (a) the existence of multiple goal states, which requires considering partially-defined states (i. e., in which the value of some variables is unknown) and (b) the impact that states unreachable from the initial state have on backward search [7].

The main shortcoming of state-space search is that the number of states that must be explored to find an optimal solution may be exponential on the size of the problem. *Symbolic search* aims to alleviate this problem by reasoning about sets of states, represented by efficient data-structures like *Binary Decision Diagrams* (BDDs) [8], instead of individual states. BDDs can sometimes represent sets of states with exponentially less memory than their explicit enumeration so symbolic search with BDDs usually lessens in practice the memory requirement of explicit-state search algorithms. This means that symbolic versions of common search algorithms, such as symbolic uniform-cost search and symbolic A* [9] (also known as BDDA*) are often able to solve problems that explicit-state algorithms are unable to solve due to memory problems. Moreover, BDDs also provide efficient operations to manipulate sets of states and perform search.

Symbolic planning with BDDs was brought over from symbolic model checking [10] and has been pioneered by Cimatti et al. [11] to develop non-deterministic planners. The Model-Checking Integrated Planning System MIPS [12] was the first classical planning system based on symbolic search with BDDs. Later on, state-set branching [13] was proposed as an improvement over regular BDDA*. Finally, Gamer [14, 15] is an evolution of MIPS with several improvements like symbolic pattern database heuristics to guide BDDA* and an optimization of the BDD variable ordering. However, and despite the advantages that symbolic search has over explicit-state search and its relative success in different AI competitions, the area as a whole remains relatively unexplored. In this paper, we aim to improve symbolic search for cost-optimal planning with two orthogonal lines of research:

Image computation. We experiment with different ways to perform the image computation, used to perform successor generation in symbolic search and one of the main computational bottlenecks in symbolic planning. We discuss different methods to represent the *transition relations* (TRs), i.e., the BDD encoding of planning operators. Planning operators are intelligently grouped in order to decide which operators should be represented together in the same TR and speed-up image computation.

State-invariant constraints. We study the exploitation of constraints such as mutexes and invariant groups in combination with BDDs, both in symbolic search and symbolic abstractions. We show that constraints can be encoded as BDDs efficiently, and that they can also be encoded in the TRs.

In order to evaluate the benefit of these advances in state-of-the-art symbolic planning algorithms, we implemented them in Gamer. We also extended Gamer symbolic bidirectional breadth-first search to uniform-cost search, which is optimal in domains with non-uniform operator costs. We call the resulting planner *constrained* Gamer, cGamer for short.

Experimental results show that our proposed refinements in image computation and state-invariant pruning lead to a more efficient search. These improvements are orthogonal and, when combined, they get an important leap in performance for symbolic planning algorithms. With these enhancements, the best variant of cGamer, which uses bidirectional uniform-cost search, outperforms current explicit-state heuristic search in most domains.

Some previous conference papers contain preliminary results of our work, both the analysis of image computation [16] and state-invariant pruning in symbolic planning [17]. In this article, we extend our previous work by considering more detailed aspects such as additional strategies for image computation or the use of relevance invariants that detect dead-end states in forward search. The article at hand is structured as follows. Sections 2 formally introduces cost-optimal planning. Section 3 introduces BDDs and symbolic search. Sections 4 and 5 present our analysis of image computation and state-invariant pruning in symbolic search, respectively. Finally, Section 6 shows the experimental results of the proposed techniques in multiple settings and Section 7 discusses the conclusions of this work.

2. Preliminaries

A finite-domain variable planning task is defined as a tuple $\Pi = \langle \mathcal{V}, \mathcal{O}, \mathcal{I}, \mathcal{G} \rangle$. $\mathcal{V}$ is a set of *state variables*, and every variable $v \in \mathcal{V}$ has an associated *finite domain* D_v. A *partial state* p is a function on a subset

of variables $\mathcal{V}_p \subseteq \mathcal{V}$ that assigns each variable $v \in \mathcal{V}_p$ a value in its domain, $p[v]$. A *state* s is a complete assignment to all the variables. We denote $\mathcal{S}$ to the set of all states. A *fact* is an assignment to a single variable and is usually identified as a variable-value pair, $\langle v, d \in D_v \rangle$. Thus, a partial state p can be defined as a set of facts and is associated with the set of states that satisfy the partial assignment, $\{s \mid p[v] = s[v] \forall v \in \mathcal{V}_p\}$. We denote as $p|'_{\mathcal{V}}$ the projection of p to the set of variables $\mathcal{V}' \subseteq \mathcal{V}_p$, i. e., as the partial assignment over $\mathcal{V}'$ having the same value as p for variables it is defined for. $\mathcal{I}$ is the *initial state* and $\mathcal{G}$ is the partial state that defines the goals. $\mathcal{O}$ is a set of operators, where each operator is a tuple $o = \langle (pre(o), \mathit{eff}(o), w(o)) \rangle$, where $pre(o)$ and $\mathit{eff}(o)$ are partial assignments over $\mathcal{V}_{pre(o)}$ and $\mathcal{V}_{\mathit{eff}(o)}$ that represent the *preconditions* and *effects* of the operator, respectively, and $w(o) \mapsto \mathbb{R}_0^+$ is the non-negative cost of o. The set of preconditions, $pre(o)$ can be split depending on whether they are affected by the operator effects or not. The *prevail* conditions of the operators are the subset of preconditions that are not affected by the effects of the operator, $prev(o) = \{f_i = \langle v, d \rangle \mid f_i \in pre(o) \text{ and } v \notin \mathcal{V}_{\mathit{eff}(o)}\}$.

An operator $o \in \mathcal{O}$ is applicable (in progression) in a state s if $pre(o) \subseteq s$. The state $o(s)$ resulting from the application of o in s is defined as $o(s) = s|_{\mathcal{V} \setminus \mathcal{V}_{\mathit{eff}(o)}} \cup \mathit{eff}(o)$. While applicability of operators in progression is defined over complete states, in order to perform regression from the goals of the problem we define the reversed applicability over partial states. An operator $o \in \mathcal{O}$ is applicable in a partial state p over $\mathcal{V}_p$ in regression if p is consistent with the operator effects and prevails, $\forall v \in \mathcal{V}_p : (v \notin \mathcal{V}_{\mathit{eff}(o)}$ or $p[v] = \mathit{eff}(o)[v])$ and $(v \notin \mathcal{V}_{prev(o)}$ or $p[v] = prev(o)[v])$ and is *relevant* to p, $V_p \cap \mathcal{V}_{\mathit{eff}(o)} \neq \emptyset$. The resulting partial state p' obtained from the application in regression of o in p is defined as $p' = (p \cup \mathit{eff}(o))|_{\mathcal{V} \setminus \mathcal{V}_{pre(o)}} \cup pre(o)$.

A solution *plan* is a sequence of operators, $\pi = (o_1, \ldots, o_n)$ related to a sequence of states $(s_0, s_1, \ldots, s_n)$ such that s_0 is the initial state and $s_n \in \mathcal{S}_\star$, i.e., the set of all goal states, and s_i results from executing the operator o_i in the state s_{i-1}, $\forall i = 1..n$ in progression. The cost of a plan is the sum of the cost of its operators, $w(\pi) = \sum_{0 \leq i \leq n} w(o_i)$. A plan is *optimal* if no other plan of lower cost exists. Most state-of-the-art cost-optimal planners employ heuristic search. A *heuristic* is a function h: $\mathcal{S} \to \mathbb{R}_0^+$ which estimates the cost to reach $\mathcal{S}_\star$ from a state $s \in \mathcal{S}$. The heuristic is *perfect* if it equals the optimal cost $\mathrm{h}^*(s)$ $\forall s \in \mathcal{S}$. It is *admissible* if $h(s) \leq \mathrm{h}^*(s)$ for all $s \in \mathcal{S}$.

3. Symbolic Planning with BDDs

Symbolic search takes advantage of succinct data structures to represent sets of states through their *characteristic functions*. Given a set of states S, its *characteristic function* f_S is a Boolean function $f_S(x_1 \cdots x_n) : \mathcal{S} \to \{\top, \bot\}$ that represents whether a given state belongs to S, i. e., $f_S(s) = \top$ iff $s \in S$. The input of the function is the bit-vector description of a state represented by binary variables x_i such that each finite-domain variable $v \in \mathcal{V}$ with domain D_v is represented with $\lceil \log_2 |D_v| \rceil$ binary variables.[1] To simplify the notation, we use the same symbol, S, to denote a set of states and its characteristic function.

We also use characteristic functions to operate with sets of states. The union ($\cup$) and intersection ($\cap$) of sets of states are derived from the disjunction ($\vee$) and conjunction ($\wedge$) of their characteristic functions, respectively. Also, the complement set (S^{c}) corresponds with the negation ($\neg$) of the characteristic function. Quantification of variables is another type of function transformation commonly encountered in symbolic search. The existential quantification of a variable v, $\exists_v S$, computes the projection of S to $\mathcal{V} \setminus \{v\}$. Formally, $\exists_v S := S|_{v=\top} \vee S|_{v=\bot}$, where $f_S|_{v=x} := (f_S \wedge \{s \mid s[v] = x\})|_{\mathcal{V} \setminus \{v\}}$ is the sets of assignments to variables $\mathcal{V} \setminus \{v\}$ so that they make f_S true whenever $v = x$. Thus, the result of the existential quantification corresponds to the projection of the set of states to the set of variables $\mathcal{V} \setminus \{v\}$.

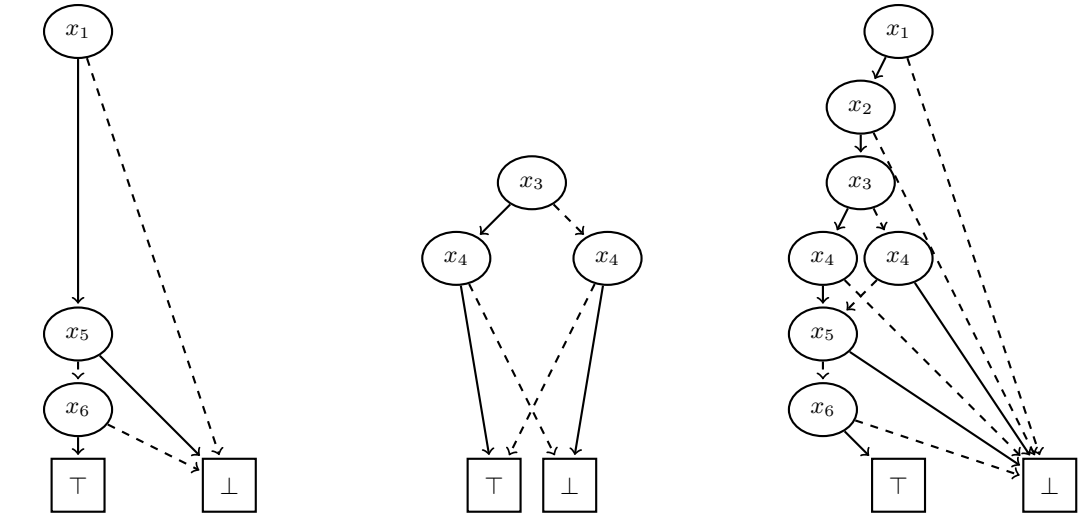
The set of operators $\mathcal{O}$ is represented with one or more *Transition Relations* (TRs). A TR is a function $T_c(x, x')$ that represents one or more operators with the same cost, c. TRs are defined using two sets of variables, the *source*-set, x and the *target*-set x'. Both sets of variables have the same cardinality as the set of variables of the characteristic function, so that each variable in the *source*- and *target*-sets corresponds to a variable in the characteristic function. The variables of the *source*-set correspond to the preconditions of the operators of the TR and the variables of the *target*-set correspond to the effects.

[1] Based on this transformation, in the rest of the paper we assume SAS$^+$ variables to be binary without loss of generality.

$image(S, T) := (\exists x\,(S \wedge T))\,[x'/x]$ computes the set of successor states that can be reached from any state in S by applying any operator represented by T. Similarly, $pre\text{-}image(S, T) := \exists x'\,((S(x)[x/x']) \wedge T(x, x'))$ computes the set of predecessor states that can reach a state in S by applying an operator in T. Image computation is discussed in detail in Section 4.

3.1. Binary Decision Diagrams

Reduced Ordered Binary Decision Diagrams, BDDs for short, are the most popular data structure to represent the logical formulæ employed in symbolic search [8]. BDDs are decision diagrams in which each internal node decomposes the function according to the possible values of a variable, following a fixed variable ordering on every path from the root to a sink. This variable ordering allows the application of two reduction rules that eliminate any unnecessary or redundant nodes. Figure 1 shows an example of two BDDs and their intersection. The definition of BDDs ensures their canonicity [8]: for any Boolean function f_S and variable ordering, there exists a unique BDD representing f_S. The application of reduction rules provides up to exponential memory gains in the number of variables when representing some functions with respect to the number of states it represents. Theoretical analyses on the complexity of representing those states in some planning domains and games have proved them to have polynomial upper bounds and exponential lower bounds on the number of nodes needed to represent different relevant sets of states on different domains [18, 19, 20].



(a) $BDD_1 = (x_1 \wedge \neg x_5 \wedge x_6)$ (b) $BDD_2 = (x_3 \Leftrightarrow x_4)$ (c) $BDD_1 \wedge BDD_2$

Figure 1: Example of two BDDs and their conjunction. BDD_1 and BDD_2 may be interpreted as the sets of states such that $(x_1 \wedge \neg x_5 \wedge x_6)$ and $(x_3 \Leftrightarrow x_4)$, respectively. Their conjunction is then interpreted as the intersection of those sets of states.

For certain kinds of functions there is an exponential gap between the complexity of describing them with one variable ordering or another [8, 21]. We follow GAMER's strategy to select a static ordering through domain analysis [15]. GAMER uses a local search optimization procedure that places variables related in the causal graph as close as possible. Recent research has shown empirically that GAMER's criteria is among the best known strategies to select static orderings and is significantly better than a random ordering though still arbitrarily far from optimal [22, 23]. A promising alternative is to perform dynamic variable reordering [24].

Tight bounds can be proved on the complexity of BDD operations as long as all the involved BDDs share the same variable ordering [8]. Equivalence comparison and negation are performed in constant time. The *apply* operation of two BDDs $B_1 \circ B_2$, used to compute the disjunction, conjunction (e.g., see Figure 1) and other binary operations over two Boolean functions, has quadratic space and time complexity on the size of the BDDs, $|B_1| \times |B_2|$ — the conjecture that the time complexity is linear on $|B_1| + |B_2| + |B_1 \circ B_2|$

was recently disproved [25, 26]. However, not all operations involved in symbolic search are polynomial. Disjunction or conjunction of several BDDs has exponential complexity on the number of BDDs. Finally, existential and universal quantification have exponential complexity in the number of quantified variables.

3.2. Symbolic Bidirectional Uniform-Cost Search

Symbolic search uses BDDs to represent and operate with set of states. As usual, symbolic uniform-cost search uses an *open* list that contains the states that have been reached but not expanded and a *closed* list that stores the states that have already been expanded. States are classified by the cost g with which they have been reached from the initial state of the search. Therefore, *Open* and *Closed* are lists of BDDs where $Open_i$ and $Closed_i$ represent the sets of states reached or expanded with $g = i$, respectively. $Closed_* := \bigvee_i Closed_i$ is the set of all expanded states. At each step, the algorithm expands the set of states with lowest g-value in *Open* that are not yet in $Closed_*$, inserting them in *Closed* and all their successors in *Open*.

Bidirectional uniform-cost search performs a forward and a backward uniform-cost searches. The forward search starts at the initial state, $\mathcal{I}$, and advances towards the goal states, $S_\star$. The backward search performs regression from $S_\star$ towards $\mathcal{I}$. The two searches are performed in an interleaved manner so that at each step the algorithm decides whether to continue the backward or forward search. Newly generated states are compared with the set of expanded states from the other search direction. In case of a match, a solution plan has been found, though it is not necessarily optimal. Rather, it is necessary to continue until the plan is proved to be optimal [27].

The symbolic version of bidirectional uniform-cost search is detailed in Algorithm 1. The *Step* procedure performs a step in an uniform-cost search. If there are 0-cost operators, it first performs a breadth-first search only using 0-cost operators in order to retrieve all states with the same g-value (line 14). Then, the selected states are inserted into the closed list and expanded. The image operator (described in detail in Section 4) generates the successor states and they are inserted into *Open*.

The *UpdatePlan* procedure checks whether a new better plan has been found over an expanded state (line 16) or a newly generated state (line 20). The check for generated states is not strictly needed, but it helps to decrease the cost of the best plan found so far, w_{total}, as soon as possible. This reduces the number of image computations by skipping those that cannot possibly lead to a better plan (line 18). Also, after checking whether a plan exists, states in $Closed'_*$ can be removed from *succ* since they have already been expanded in the opposite direction so that their optimal distance to the goal is known. Since the check ignores states in the *Open* list of the opposite frontier, *UpdatePlan* must be called again when states are expanded in order to ensure that no plan is missed. To check if a new plan has been found, *UpdatePlan* compares the newly expanded/generated states with the closed list of the opposite search. In case of a match, a new plan has been found. The cost of the new plan is the sum of the g-values of the intersected states in both searches (line 31). If this cost is smaller than the smallest cost found so far (w_{total}), then w_{total} is updated and the corresponding solution path π can be created. Solution reconstruction works differently than in explicit-state search, because symbolic search algorithms do not keep track of the parents of generated states. *ConstructSolution* retrieves a path from any state to the initial state or the goal using the closed list that contains all the expanded states classified by their g-value. The complete plan is the concatenation of a path from $\mathcal{I}$ to a state s in the intersection of both frontiers, and a path from s to the goal. Both paths are retrieved with two calls to the *ConstructSolution* algorithm. The algorithm may stop when the sum of the minimum g values of generated states for the forward and backward searches is at least w_{total}, the cost of the cheapest solution path found so far. This stopping condition guarantees that the current plan, π, is optimal [27, 28].

At each step, the algorithm decides whether to perform a forward or backward step. A typical criterion is Ira Pohl's cardinality principle that picks the direction with fewer frontier states [29]. In symbolic search, however, fewer states does not necessarily imply less search effort. Therefore, GAMER estimates the time needed to perform the next step in each direction and selects the one that is easier. This greedy policy works well under the assumption that, as the search progresses, sets of states are usually harder to represent. To compute the estimated time for step k, t_k, we take into account the time spent on the previous step, t_{k-1}, and the BDD sizes for the state set to be expanded in the next and the previous step, s_k and s_{k-1}, respectively.

Algorithm 1: Symbolic Bidirectional Uniform-Cost Search

Input: Planning problem: $\Pi = \langle \mathcal{V}, \mathcal{O}, \mathcal{I}, \mathcal{G} \rangle$
Input: Transition relations: $\mathcal{T}$
Output: Cost-optimal plan or "no plan"

$fOpen_0 \leftarrow \{\mathcal{I}\}$
$bOpen_0 \leftarrow S_\star$
$fClosed \leftarrow bClosed \leftarrow \bot$
$g_f \leftarrow g_b \leftarrow 0$
$w_{total} \leftarrow \infty$
$\pi \leftarrow$ "no plan"
while $g_f + g_b < w_{total}$ **and not** $fOpen$ is empty **and not** $bOpen$ is empty **do**
  if `NextStepDirection`$(fOpen, bOpen) = Forward$ **then**
    $fOpen, fClosed, g_f, \pi, w_{total} \leftarrow$ `Step`$(fOpen, fClosed, g_f, bClosed, \mathcal{T}, \pi, w_{total})$
  else
    $bOpen, bClosed, g_b, \pi, w_{total} \leftarrow$ `Step`$(bOpen, bClosed, g_b, fClosed, \mathcal{T}^{-1}, \pi, w_{total})$
return π

Procedure `Step`$(Open, Closed, g_{min}, Closed', \mathcal{T}, \pi, w_{total})$
  $Open_{g_{min}} \leftarrow$ `BreadthFirstSearch`$(Open_{g_{min}} \wedge \neg Closed_*, \{T_0\}, Closed_*, \emptyset)$
  $Closed_{g_{min}} \leftarrow Open_{g_{min}}$
  $\pi, w_{total} \leftarrow$ `UpdatePlan`$(\pi, w_{total}, Open_{g_{min}}, g_{min}, Closed')$
  for all $T_c \in \mathcal{T}, c > 0$ **do**
    if $g_{min} + c < w_{total}$ **then**
      $succ \leftarrow image(Open_{g_{min}}, T_c) \wedge \neg Closed_*$
      $\pi, w_{total} \leftarrow$ `UpdatePlan`$(\pi, w_{total}, succ, g_{min} + c, Closed')$
      $Open_{g_{min}+c} \leftarrow Open_{g_{min}+c} \vee (succ \wedge \neg Closed'_*)$
  $Open_{g_{min}} \leftarrow \bot$
  $g_{min} \leftarrow \min\{g \mid Open_g \neq \bot\}$
  return $Open, Closed, g_{min}, \pi, w_{total}$

Procedure `UpdatePlan`$(\pi, w_{total}, S_g, g, Closed')$
  if $S_g \wedge Closed'_* \neq \bot$ **then**
    for all $i \in \{0, \ldots \mid Closed'_i \neq \bot\}$ **do**
      if $g + i \geq w_{total}$ **then**
        break
      if $S_g \wedge Closed'_i \neq \bot$ **then**
        $w_{total} \leftarrow g + i$
        $\pi_f \leftarrow$ `ConstructSolution`$(s_0, S_g \wedge Closed'_i, \mathcal{O}, fClosed)$
        $\pi_b \leftarrow$ `ConstructSolution`$(s_\star, \pi_f(s_0), \mathcal{O}^{-1}, bClosed)$
        $\pi \leftarrow \pi_f \parallel \pi_b^{-1}$
  return π, w_{total}

In general, we assume a linear relation between BDD size and image computation time. However, this might not work well at the beginning of the search, where t_k is too low and the BDD size ratio is too large. Thus, we estimate t_k as shown in Equation 1, using the time of the previous layer whenever it was below one second and the linear estimation for all the other cases.

$$t_k = \begin{cases} 0, & \text{if } k = 0 \\ t_{k-1}, & \text{if } k > 0 \wedge t_{k-1} \leq 1s \\ t_{k-1}\frac{s_k}{s_{k-1}}, & \text{if } k > 0 \wedge t_{k-1} > 1s \end{cases} \quad (1)$$

This estimation, though not perfect, often balances the search effort made by both frontiers. However, in some domains the time may grow exponentially in the frontier size. To avoid exhausting all the available time in a single step, we interrupt any step if it takes more than twice the estimated time for the opposite direction. In that case, we re-estimate the time needed for the failed step to double the time spent before failure, so that if the planner retries it, its allotted time will be at least doubled.

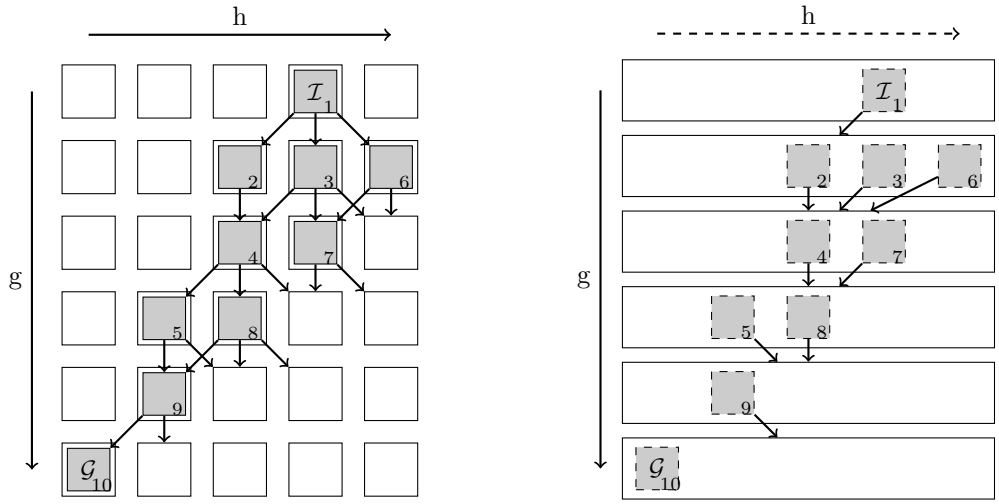
3.3. Symbolic A Search*

Symbolic A* (alias BDDA*) was first introduced by Edelkamp and Reffel [9] and integrated in the planning context by Edelkamp and Helmert [12] in the model checking integrated planning system MIPS. As usual with A*, BDDA* expands states in ascending order of $f = g + h$. The difference is that BDDA* groups sets of states with the same g and h-value into g, h-buckets, and expands all states in a bucket at once (see Figure 2a). The heuristic function in symbolic search is precomputed prior to the search and represented as a list of BDDs, *heur*, one per possible heuristic value. The heuristic evaluation is done with a conjunction: given a set of states S and the $heur_i$ BDD, $S \wedge heur_i$ corresponds to the subset of states that have a heuristic value equal to i.

Different tie-breaking criteria can be used to select which node will be expanded next among those with minimum f-value. In explicit-state search nodes with larger g-value (and therefore smaller h-value) are preferred in order to expand a goal state as soon as possible, terminating the algorithm. In symbolic search, however, buckets with minimum g-value are preferred, i. e., the opposite criterion to that used in explicit-state search. This expansion order may be detrimental on the last f-diagonal (e. g., buckets 6-10 in Figure 2a) because all the states with $f = f^*$ are expanded. The advantage is that this avoids re-expanding any bucket because once a g, h-bucket has been expanded, no new states with such g, h-values will be generated. For example, in Figure 2a, after having expanded buckets 1 and 2, buckets 3 and 4 have the same f-value. When bucket 3 is expanded, new states may be inserted in bucket 4, so if bucket 4 is expanded first, then it will be re-expanded twice.

Multiple variants of BDDA* exist across the literature, varying the representation of state sets involved in the search. ADDA* [30] is an alternative implementation with ADDs, while Set A* [31] refines the partitioning in a matrix representation of g- and h-buckets in the open list. GAMER uses the Matrix BDDA* implementation [32]. In this article, we use List BDDA*, which is a variant of Matrix BDDA* in which the open list buckets are only distinguished by their g-value [33, 34]. Figure 2b shows how List BDDA* expands the states in exactly the same way as Matrix BDDA*. The difference is that all the buckets in the same row of the g, h-matrix are represented by a single BDD. The heuristic evaluation (conjunction of the sets of states with each heuristic BDD) is deferred until a set of states is going to be expanded, avoiding to compute the precise h value of states that will never be expanded anyway. The advantage of List BDDA* is that it reduces the number of conjunctions with the heuristic BDDs, having a positive impact specially when there are many different heuristic values. For example, consider states in the first row of Figure 2 with $g = 1$. In Figure 2a, in order to know in which bucket the states have to be inserted, one conjunction is needed for every possible value of h. In Figure 2b only one conjunction is needed in order to extract bucket 9.

To inform BDDA*, GAMER uses symbolic partial pattern databases. Pattern databases (PDBs) map the original state space to a smaller abstract state space. PDBs use backward uniform-cost search to precompute the optimal solution cost from each abstract state to the set of abstract goal states, and use it as an estimation for the original problem [35, 36]. In the planning literature, this abstract problem is usually



(a) Matrix BDDA* (b) List BDDA*

Figure 2: Example of BDDA* implementations. The greyed cells are expanded in the order specified by the numbers. Arrows denote successor buckets.

defined by the projection of the planning task over a subset of variables [37]. A *partial PDB* [38] is a PDB that is not fully computed, but rather its calculation is stopped at some point, e.g., when the pre-defined allotted time has expired. If all abstract states with a solution cost of d or less have been expanded, then $d+1$ is an admissible estimation for all non-expanded states. *Symbolic PDBs* [39] are PDBs constructed with symbolic backward uniform-cost search. GAMER uses an automatic pattern selection procedure to generate a well-informed PDB. This is similar to the one proposed for explicit-search planners [40], though instead of constructing a collection of PDBs, it aims to construct a single bigger PDB. Details can be found in the literature [15, 32].

4. Image Computation

As introduced in Section 3, the image operation is used to compute the set of successor states. This is often the most time-consuming process in symbolic search, so it is important to perform it as efficiently as possible. In this section we study the related work on image computation and propose three different image computation methods for symbolic search in planning. Our first method, $\mathbf{TR}^{1+}$, avoids using the set of auxiliary variables x' when using a TR per operator. Then, we present the **CT** method whose goal is to improve how the operator preconditions are matched. Finally, we discuss the **UT** method, other disjunctive partitioning criteria to represent the TRs.

4.1. Basics of Image Computation

In symbolic search planning, operators are described in the *transition relations* (TRs). A TR is a relation between predecessor and successor states, i. e., it represents all the pairs of states $\langle s, s' \rangle$ such that $s' = o(s)$ for an operator o represented by the TR. Thus, if sets of states are described as functions over the set of variables x, TRs also use a set of auxiliary variables, x', to represent successor states.[2] In the BDD variable ordering variables from x and x' are alternated $(x_1, x'_1, x_2, x'_2, \ldots, x_n, x'_n)$ as proposed, e. g., by Burch et al. [41]. This ordering is logical because x_i and x'_i are closely related since for any operator o with $v_i \notin \mathcal{V}_{\mathit{eff}(o)}$, x'_i is assigned the value of x_i.

[2]As explained in Section 3, we assume WLOG that x and x' have a direct correspondence with $\mathcal{V}$: $|x| = |x'| = |\mathcal{V}|$ and each v_i has an associated x_i and x'_i.

The image operation of a set of states with respect to a given transition relation can be decomposed into basic logical operations over BDDs: $image(S, T) := (\exists x\,(S \wedge T))\,[x'/x]$. The conjunction $S \wedge T$ corresponds tow all pairs of states $\langle s, s' \rangle$ such that $s \in S$ and $s' = o(s)$ for an operator o represented by T. Then, the existential quantification ignores predecessor states and the variable swapping, $[x'/x]$, represents the resulting states s' with the standard set of variables x. In practice, $\exists x\,(S \wedge T)$ is a single BDD operation, the so-called relational product, which is often implemented in a more efficient way than the sequential application of conjunction and quantification [41]. The related *pre-image* operation, used to perform regression, is decomposed into similar operations. In this case, variables are swapped first and the existential quantification is performed over x' instead of x. $pre\text{-}image(S, T) := \exists x'\,((S[x/x']) \wedge T)$. Even though we focus our discussion in this section on image computation, the same conclusions can be obtained about pre-image computation for all purposes.

The representation of the TRs plays a key role in image computation. As we must distinguish the cost of each transition, different TRs have to be used for each operator cost. The monolithic representation uses a single TR, T_c, to represent all operators of cost c. However, this is often unfeasible because, in the worst case, a TR uses exponential memory in the number of operators that it represents. A traditional solution is to use a conjunctive or disjunctive partitioning of the transition relations [42]. Instead of having a single T_c that represents all operators with cost c, we have a list of TRs $\{T_{c,1}, \ldots, T_{c,k}\}$ such that $T_c = \bigvee_{i=1}^{k} T_{c,0}$ (disjunctive partitioning) or $T_c = \bigwedge_{i=1}^{k} T_{c,0}$ (conjunctive partitioning). Then, the images with respect to the individual TRs can be combined to obtain the same result, i. e., $image(S, T_c) = \bigvee_{i=1}^{k} image(S, T_{c,i})$ or $image(S, T_c) = \bigwedge_{i=1}^{k} image(S, T_{c,i})$, respectively.

Whether it is better to use a conjunctive or a disjunctive partitioning depends on the characteristics of the problem being modeled. In symbolic model checking, a conjunctive partitioning is often better because most studied systems are described in terms of rules that are applied in parallel. For this reason, research has generally been focused on the problem of scheduling the conjunction and quantification operations during image computation [43, 44], with a few works combining both types of partitioning [45]. In planning, though, disjunctive partitioning is more natural, since planning operators are applied sequentially. Jensen et al. [46, 13] first proposed the partitioning of TRs so that each partition contains operators that change the f-value of the states by the same amount. The precomputation of Δf was obtained from the cost of the operators of the problem and the increase or decrease of the heuristic value. However, this method is limited to problems and heuristics where Δh can be precomputed from the problem description.

Due to the intractability of using a single T_c, GAMER uses a TR per operator. Given an operator $o = \langle (pre(o), \mathit{eff}(o), w(o)) \rangle$, its associated transition relation T_o is obtained as the conjunction of its literals, describing the preconditions with variables from x and the effects with variables from x'. Any variable not modified by o must also be explicitly encoded so that it keeps its value after the application of o. This is encoded in the TR as a bi-implication of the form $biimp\,(x_i, x'_i) = (x_i \wedge x'_i) \vee (\overline{x}_i \wedge \overline{x}'_i)$. Thus, if we assume that the function $fBDD(v, val, x)$ returns the BDD that corresponds to the fact $\langle v, val \rangle$ represented with the set of variables x, the TR of a single operator is computed as follows:

$$T_o = \bigwedge_{\langle v, val \rangle \in pre(o)} fBDD(v, val, x) \wedge \bigwedge_{\langle v, val \rangle \in \mathit{eff}(o)} fBDD(v, val, x') \wedge \bigwedge_{k \in \mathcal{V} \setminus \mathcal{V}_{\mathit{eff}(o)}} biimp(x_k, x'_k)$$

The size of a TR representing a single operator is linear in the number of variables of the planning task. First, the preconditions and effects are simple conjunctions of facts, which require a BDD whose size is linear in the number of relevant variables. Second, each bi-implication is represented by just three BDD nodes as long as the variables from x and x' that correspond to the same variable $v \in \mathcal{V}$ are adjacent in the variable ordering, as they are in our ordering schema. Finally, as all the bi-implication, precondition and effect BDDs are independent, T_o is just a concatenation of these BDDs.

When a disjunctive partitioning is used, the image is computed in two steps. First, the image is computed for each TR, and then the results of TRs associated to operators of the same cost, c, are aggregated computing their disjunction $\bigvee_{i=1}^{k} image(S, T_{c,i})$. This is not a single operation, but rather $k - 1$ disjunctions of two BDDs at a time. Even though the result will always be the same, the order in which the individual BDDs are merged has an impact on the performance of image computation. The computational cost of most BDD

operations critically depends on the BDD size, so is important to keep the intermediate BDDs as small as possible. GAMER uses an iterative algorithm. Given a list of BDDs to be aggregated, $S_1, \ldots, S_k$, it aggregates pairs of BDDs $(S_1 \vee S_2, \ldots, S_{k-1} \vee S_k)$ and repeats the process until only one BDD remains. To model the order in which disjunctions are applied, we represent them in a binary tree, called *disjunction tree*. Each internal node applies the disjunction of the result of its left and right branches. Each leaf node is associated with an operator, so that it represents the image result with respect to that operator's TR. GAMER's procedure corresponds to a balanced tree like the one in Figure 3.

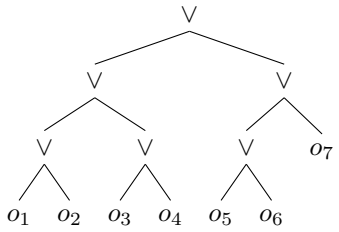


Figure 3: Disjunction tree corresponding to GAMER's procedure for $k = 7$. Operators are ordered by the instantiation procedure so operators coming from the same action schema will be placed together.

4.2. Image without Auxiliary Variables ($\mathbf{TR^{1+}}$*)*

Our first method for image computation, $\mathbf{TR^{1+}}$, leverages the particular structure of planning operators for faster image computation when a TR T_o encodes a single operator o. First, as all variables not appearing in the effects retain their values, the existential quantification and variable swapping only need to be applied over variables modified by $\mathit{eff}(o)$, which we call x_o. Therefore, the bi-implication term may be omitted from T_o, using instead $\widetilde{T_o}$, which is defined as: $\widetilde{T_o} = \bigwedge_{\langle v,val\rangle \in pre(o)} fBDD(v, val, x) \wedge \bigwedge_{\langle v,val\rangle \in \mathit{eff}(o)} fBDD(v, val, x')$. With this encoding, the image is computed as $image(S, T_o) = (\exists x_o \ (S \wedge T_o))\,[x'_o/x_o]$.

Moreover, since according to the semantics of a planning operator the effect does not depend on the predecessor state, T_o can be divided into a precondition T_o^{pre} over x and an effect T_o^{eff} over x' such that $T_o = T_o^{pre} \wedge T_o^{\mathit{eff}}$. In fact, once preconditions and effects are encoded separately there is no need to relate the sets x and x' anymore. Thus, the TRs can be represented using only the set of predecessor variables x. This way the variables representing the successor states x' as well as the swap operation are no longer needed, reducing the size of intermediate BDDs and speeding up image computation. The new image operation is:

$$image_{\mathbf{TR}^{1+}}(S, T_o) = (\exists x_o \ (S \wedge T_o^{pre})) \wedge T_o^{\mathit{eff}}$$

4.3. Conjunction Trees ($\mathbf{CT}$*)*

When considering multiple TRs it is advisable to avoid checking their applicability individually. This is especially important when $\mathbf{TR^{1+}}$is used, as the number of grounded operators can become very large in some planning problems. Some explicit-state planners employ speed-up techniques to filter non-applicable operators efficiently, such as the successor generator used by Fast Downward [47]. In Fast Downward the operators are organized in a decision tree similar to the structures used by RETE networks to detect the triggering of a rule [48]. Figure 4 shows our conjunction tree, which resembles these decision trees in which each leaf node contains a set of operators that have the same preconditions. Every internal node is associated with a variable $v \in \mathcal{V}$ and has an edge for every value i of v and an additional "don't care" edge. An operator $o \in \mathcal{O}$ is propagated down the i edge if and only if $\langle v = i\rangle \in pre(o)$ and down the "don't care" edge if $v \notin \mathcal{V}_{pre(o)}$. To compute the successors of a state s, the tree is traversed, omitting branches labeled with unsatisfied preconditions and always following "don't care" edges. An operator is applicable in s if and only if it is in a leaf reached by that traversal.

This approach carries over to BDDs as follows. As all the variables are binary, each internal node only has three children c_0, c_1 and $c_\star$, dividing the operators into three sets: those that require $\overline{v}$ as a precondition, those that require v as a precondition and those whose applicability does not depend on v

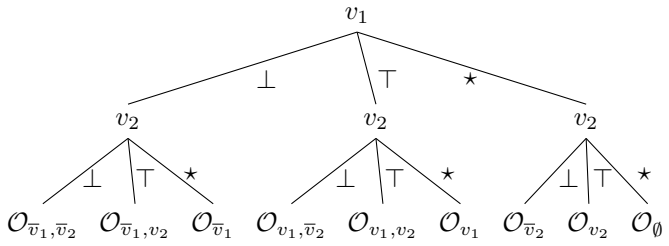


Figure 4: Conjunction tree for applying an operator's precondition on a set of states. Each internal node corresponds to one binary variable, v_i, classifying the operators depending on whether they have precondition $v_i = \top$, $v_i = \bot$, or no precondition on v_i ($v_i = \star$).

Algorithm 2: CT-image: Image using the conjunction tree

Input: *node*: **CT** node. S_g: Set of states to be expanded.
Output: Set of pairs $\langle c, S_c \rangle$ where S_c is a set of successor states generated with an operator of cost c.

```
1 if S_g = ⊥ then return ∅
2 if node is leaf then
3     return ⋃_{o∈node.O} {⟨w(o), image(S_g, T_o)⟩}
4 v ← node.variable
5 r_0 ← CT-image(node.c_0, S_g ∧ ¬v)
6 r_1 ← CT-image(node.c_1, S_g ∧ v)
7 r_⋆ ← CT-image(node.c_⋆, S_g)
8 return r_0 ∪ r_1 ∪ r_⋆
```

at all. Applicability of operators is different for progression and regression search, so different conjunction trees are needed. Conjunction trees for forward search take into account the preconditions of operators. Conjunction trees for backward search take into account the preconditions of the inverted operators, i. e., their effects and prevail conditions. In the presence of zero-cost operators, all algorithms studied in Section 3 apply breadth-first search using only the subset of zero-cost operators until a fix-point is reached. Since regular and zero-cost operators are applied over different sets of states, two different conjunction trees are needed: one with the zero-cost operators and another one with the rest. Therefore, in order to apply bidirectional search in domains with zero-cost operators up to four different conjunction trees are needed: fw-zero, fw-cost, bw-zero, and bw-cost.

In symbolic search the operators are not applied over a single state but rather over sets of states. Nevertheless, operators only need to be applied over states that satisfy the corresponding conditions. We take advantage of this by precomputing the subset of states relevant to a given TR prior to the image operation. This is done by splitting the original set of states into subsets as the successor tree is traversed. This way, the subset on which a TR, T_o is applied once a leaf node is reached is the subset of states that satisfy $pre(o)$. We call this method **CT**.

Algorithm 2 shows how to compute the image of a BDD using **CT**. It takes as input a set of states S and the root node of the conjunction tree and returns the sets of successor states, associated with the cost of the operators that generated them. At every inner node one recursive call is made for each child node, applying the corresponding conjunction between the set of states S and the precondition $v \in \mathcal{V}$ associated with the node. Moreover, if there are no states satisfying the preconditions of a branch, it is not traversed. When a leaf node is reached, the image is computed with respect to $S' = S \wedge pre(o)$ so here we should only account for the effects. Thus, in the leaf nodes the image is computed as $image_{\mathbf{CT}}(S', T_o) = (\exists x_o S') \wedge T_o^{\mathit{eff}}$.

CT has several advantages over the baseline approach: first, if S does not contain any state matching the conditions of a branch of the tree, all the operators regarding that branch are ignored, which reduces

the number of individual image operations that are needed. Second, if several operators share the same preconditions, the conjunction of S with that set of preconditions is computed only once. Finally, the conjunction with the preconditions is done with BDDs whose size usually decreases as we go deeper in the conjunction tree, so the time required to do the individual conjunctions will be shorter.

An overhead may occur due to the computation and storage of intermediate BDDs in memory, though. The conjunction with a precondition should only be used when the benefits compensate the overhead, i. e., when a precondition is shared between many operators. Thus, **CT** can be parameterized with a parameter *min operators conjunction*, so that the intermediate conjunction with a partial precondition is only computed when needed for at least that number of operators. If the number of operators that should go down some edge is less than *min operators conjunction*, they are propagated down the *don't care* branch $c_\star$ instead, and marked so that we know that not all their preconditions have been checked. When computing the image of marked operators, the conjunction with their preconditions is needed. When *min operators conjunction* is set to 1 we have the full tree strategy and when it is set to ∞ the conjunction tree consists of only one leaf node containing all the operators, which is equivalent to not having a tree at all. Setting the parameter to intermediate values produces intermediate strategies.

The performance of **CT** may depend on the order in which it checks the conditions. We check the preconditions in the same ordering as in the BDD representation. This aims at making the conjunctions as simple as possible, since a conjunction with the first variable of a BDD is trivial. A comparison with different heuristic criteria showed that this does not have a large impact and our simple criterion works well in practice [34].

*4.4. Unions of Transition Relations (***UT***)*

Even though having a monolithic TR per operator cost is often unfeasible due to its size, computing the union of TRs of only a subset of operators may be beneficial. The union of a set of TRs $\mathcal{T}$ is a new TR, $Union(\mathcal{T})$, such that the image with respect to $Union(\mathcal{T})$ is equivalent to the disjunction of the images of the individual TRs in $\mathcal{T}$, $image(S, Union(\mathcal{T})) = \bigvee_{T\in\mathcal{T}} image(S, T)$. $Union(\mathcal{T})$ cannot be represented with separated preconditions and effects as described in Section 4.2 though, as the bi-implications are needed to ensure that the generated successors are correct. However, not all the bi-implications need to be explicitly included: a TR $T_o \in \mathcal{T}$ must include a bi-implication related to some variables x_i only if x_i is modified by some other $T'_o \in \mathcal{T}$ (and not by T_o). Thus, $Union(\mathcal{T})$ may be computed as shown in Equation 2, where X_T and $X_\mathcal{T}$ are the sets of variables in the effects of operators represented by the transition relation T and any transition relation in the set $\mathcal{T}$, respectively. The image is computed with the standard method presented in Section 4.1.

$$Union(\mathcal{T}) = \bigvee_{T\in\mathcal{T}} \left(T \wedge biimp_{x_i\in X_\mathcal{T}\setminus X_T}(x_i, x'_i) \right) \tag{2}$$

A critical decision is which operators should be joined together in a single TR while ensuring that the resulting $Union(\mathcal{T})$ is tractable to compute. Algorithm 3 is a simple algorithm for creating a disjunctive partitioning starting from the set of one TR per each operator. The algorithm aggregates TRs using equation 2 until a monolithic TR is created or the time and memory (in terms of BDD nodes) bounds are exceeded. Recall that, as the cost of reaching a state must be preserved, only operators with the same cost w may be aggregated. Therefore, we call the algorithm once per different cost, i, with $\mathcal{T}_i = \{ T_o \mid o \in \mathcal{O}, w(o) = i\}$.

The most important aspect of the algorithm is the selection of the TRs to be aggregated in line 3. This is important, not only because it may impact the algorithm performance, but also because, if a single $Union(\mathcal{T}_c)$ cannot be computed for some cost c, it is important to have a set of TRs as balanced in size as possible. As we impose a limit on the maximum size of a TR, balancing the size of the TRs will often lead to a better partitioning with a lower number of TRs. We define three strategies, based on different criteria, to select which TRs should be unified: $\mathbf{UT^{DT}}$, $\mathbf{UT^{SM}}$ and $\mathbf{UT^{CT}}$.

- The $\mathbf{UT^{DT}}$ strategy employs the balanced disjunction tree shown in Figure 3. Instead of computing the union of the successors generated with the TRs at the leaf nodes, the actual TRs are merged. TRs

Algorithm 3: Aggregate

Input: $\mathcal{T}$: Set of elements to aggregate.
Input: *maxTime, maxNodes*: Time and node bounds.
Output: Set of elements aggregated.

$\mathcal{T}_{res} = \emptyset$
while $|\mathcal{T}| > 1$ and $currentTime < maxTime$ **do**
  Select $T_i, T_j \in \mathcal{T}$
  $\mathcal{T} \leftarrow \mathcal{T} \setminus \{T_i, T_j\}$
  $T' \leftarrow Union(\{T_i, T_j\}, maxTime - currentTime, maxNodes)$
  if Union *suceeded* **then**
    $\mathcal{T} \leftarrow \mathcal{T} \cup \{T'\}$
  else
    $\mathcal{T}_{res} \leftarrow \mathcal{T}_{res} \cup \{T_i, T_j\}$
return $\mathcal{T}_{res} \cup \mathcal{T}$

that share the same parent node are merged in a bottom-up fashion until *maxNodes* is exceeded in that branch or the algorithm runs out of time.

- The $\mathbf{UT^{SM}}$ strategy aggregates the two smallest TRs at every step. With $\mathbf{UT^{DT}}$, if a small TR is merged with another TR whose size is close to *maxNodes* and the union of both TRs is bigger than *maxNodes*, then the first TR will not be considered for merging anymore. If this occurs, it may happen that the set of final TRs contains individual TRs that could have been merged without exceeding *maxNodes*.

- The $\mathbf{UT^{CT}}$ strategy aims to leverage the synergy between operators, using the conjunction tree described in Section 4.3. The intuition is that the union of two TRs whose operators have similar preconditions and effects tends to be more succinct. $\mathbf{UT^{CT}}$ aggregates TRs of operators present in the same branch of the conjunction tree just like $\mathbf{UT^{DT}}$ does. This ensures that there will be some degree of similarity between the TRs because at least one subset of the preconditions will be shared between all the operators. Also, this allows us to use **CT** in combination with a disjunctive partitioning in a seamless way: TRs are merged up the tree from the leaves until Algorithm 3 finishes, in which case the merged TRs will be the new leaf nodes of the conjunction tree. This keeps the property of **CT** that only states that can be expanded by some operator are propagated down the tree even if the TRs of the individual operators are merged.

5. State-Invariant Constraints in Symbolic Search

The most successful uses of symbolic search in planning so far have been bidirectional blind search and symbolic abstraction heuristics [9, 49]. A common point of these methods is that they require performing regression on the goals of the problem. Regression in planning is considered to be less robust than progression, mainly due to the existence of numerous spurious states in regression. Spurious states are defined as states that are not reachable from s_0 [7], although other definitions exist [50]. To alleviate the impact of spurious states, constraints obtained from state invariants of the problem have been employed by explicit-state planners [7, 51, 6].

In this section we consider how to exploit state-invariant constraints in a symbolic setting. We consider two different approaches: encoding the constraints in a separate BDD ($\boldsymbol{\mathcal{M}_{BDD}}$) or encoding them directly in the transition relation (**e-del**). We conclude the section discussing the how constraints can be used in symbolic abstractions to generate more informed heuristics.

5.1. Background on State-Invariant Constraints

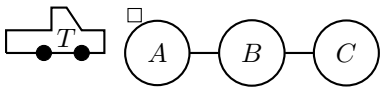
A *state-invariant constraint* is a logical formula that must hold in every state that can be part of a plan from the initial state to the goal.[3]

Definition 1 (State-invariant constraint)**.** *Let c be a logical formula and let $S_{fw}, S_{bw} \subseteq \mathcal{S}$ be the set of states reachable from the initial state and from which a goal state can be reached, respectively. Then, c is a* reachability constraint *iff for any s, $s \not\models c \implies s \notin S_{fw}$. c is a* relevance constraint *iff for any s, $s \not\models c \implies s \notin S_{fw}$. c is a* state-invariant constraint *iff for any s, $s \not\models c \implies s \notin S_{fw} \cap S_{bw}$. Any reachability or relevance constraint is a state-invariant constraint, though the opposite does not hold in general.*

A state is *valid* if and only if satisfies all the constraints. *Invalid* states that violate a constraint can be pruned during the search since they are either unreachable or dead ends. By definition, reachability constraints cannot possibly prune any state in forward search. Similarly, relevance constraints are not useful in backward search. An operator is *valid* if it is consistent with the constraints, i. e., exist $s, s' s.t. o(s) = s$ and $s, s' \models c$. Invalid operators are removed in a preprocessing step so we assume all operators to be valid.

We consider two types of state-invariant constraints: mutexes and *"exactly-1"* invariant groups. *Mutexes* are pairs of mutually exclusive facts[4]. A pair of facts $M = \langle f_1, f_2 \rangle$ is a mutex if there is no state s that may belong to a solution path such that $M \subseteq s$. Invariant groups are invariants defined over a set of facts. An *"at-most-1"* invariant group is a set $M_a = \{f_0, f_1, \ldots, f_n\}$ such that $\forall p_m \in \{\langle f_i, f_j \rangle \mid f_i, f_j \in M_a \text{ and } f_i \neq f_j\}$ p_m is mutex. Thus, *a priori* these invariant groups do not offer more information than taking into account individual binary mutexes. An *"exactly-1"* invariant group is an *"at-most-1"* invariant group with an additional constraint: at least one of the facts in the set must hold in any state.

State invariants are usually computed prior to the search using a variety of methods. Monotonicity analysis, which is generally employed to generate a finite-domain representation of the problem [53], detects reachability mutexes and *"exactly-1"* invariant groups. Another common method to find mutexes is the h^2 heuristic [54]. h^2 performs a reachability analysis in P^2 [55], a version of the original problem in which the atoms are actually pairs of facts. Every pair of facts that is unreachable in P^2 is a mutex. h^2 can be computed backwards too in order to find relevance mutex constraints [56, 57]. In this article we use the invariants detected by the monotonicity analysis and the preprocessing algorithm presented by Alcázar and Torralba (2015), which iteratively computes h^2 in forward and backward direction in order to discover reachability and relevance mutexes.



$v_p = \langle A, B, C, T \rangle$
$v_T = \langle A, B, C \rangle$
$v_t = \langle 0, 1, 2, 3, 4 \rangle$

$\mathcal{I} = \langle v_p = A, v_T = A, v_t = 0 \rangle$
$\mathcal{G} = \langle v_p = C, v_t = 4 \rangle$

Figure 5: Mutex examples in the *Trucks* domain. Operators are *load*, *unload*, and *drive*, defined in the straightforward manner and taking a time step.

In the example of Figure 5, there are reachability and relevance mutexes. An example of reachability mutex is that at time 1 the truck cannot be at location C. It is impossible since the truck is at location A at time 0 so at location 1 it can only be at A or B. Similarly, an example of relevance mutex is that, at time 3, the package cannot be at B, since there would be no remaining time to deliver the package.

5.2. Encoding State Invariants as BDDs ($\mathcal{M}_{BDD}$)

Pruning spurious states has been considered essential for backward search since long ago [7]. The use of mutexes in explicit-state search is straightforward: simply prune every (partial) state s such that facts

[3] Other works consider only reachability invariants. Our definition is slightly more general because it considers dead-end states as well.

[4] Previous works have also considered not only pairs but sets of $m \geq 2$ facts that cannot simultaneously appear in the same state [52], even though experimental results are usually restricted to $m = 2$. In this paper, for simplicity, we consider mutex pairs even though our results can easily be extended to the general case.

$f_i, f_j \in s$ are mutex. Despite the impact of state invariants in explicit-state regression, they have not been employed in symbolic search. Although it is obvious that a *per state* application of mutexes in symbolic search is not practical, there are alternatives. In particular, we propose creating a BDD that represents in a succinct way all the states that are valid according to the state invariants. This BDD, that we call the *constraint BDD* (*cBDD*), can be used to discard all the invalid states that have been generated during the search.

The *cBDD* is created in the following way. Every binary mutex is a pair of facts $\langle f_i, f_j \rangle$ ($f_i \neq f_j$) such that if $f_i, f_j \in s$, then state s is invalid. This corresponds to the logical formula $\neg f_i \wedge \neg f_j$. Constraints derived from *"exactly-1"* invariant groups are encoded in a similar way. Given an *"exactly-1"* invariant group $\theta = \{f_1, \ldots, f_k\}$, two types of constraints may be deduced: first, the set of all the mutexes of the form $f_i \wedge f_j$ if $f_i \neq f_j$; second, the fact that at least one fact $f_i \in \theta$ must be true in every valid state. The first constraint overlaps with the mutex constraints, so it is not necessary to consider it again. The latter however can be encoded as an additional *"at-least-1"* constraint of the form $(f_1 \vee f_2 \vee \cdots \vee f_k)$.

All constraints must hold in valid states so *cBDD* results from the conjunction of all the constraints. Formally, if M is the set of binary mutexes found by h^2 and I_g the set *"exactly-1"* invariant groups:

$$cBDD = \left(\bigwedge_{\langle f_i, f_j \rangle \in M} \neg f_i \vee \neg f_j \right) \wedge \left(\bigwedge_{\langle f_1, \ldots, f_k \rangle \in I_g} f_1 \vee \cdots \vee f_k \right)$$

Even though each mutex and *"at-least-1"* constraint is efficiently representable with only one node per fact, the size of *cBDD* is exponential in the number of encoded constraints in the worst case [20]. To ensure that we can represent *cBDD*, we use a conjunctive partitioning, dividing *cBDD* into k BDDs: $cBDD = cBDD_1 \wedge cBDD_2 \wedge \cdots \wedge cBDD_k$.

To obtain an efficient partitioning, we follow Algorithm 3, *Aggregate*, described on page 13 that computes a partitioning of a set of BDDs. In this case, the algorithm is initialized with the BDDs of individual constraints and *Union* corresponds to the conjunction of the individual BDDs. These are aggregated until the remaining BDDs are larger than a given threshold. The order in which these conjunctions are applied affects the efficiency of the procedure and the number of BDDs used to represent *cBDD*. To optimize it, we compute a BDD for each variable $v_i \in \mathcal{V}$ that describes all mutexes relative to v_i and any v_j with $j > i$ and use those BDDs to initialize Algorithm 3.

Invalid states are pruned by computing the difference $S_g \setminus \neg cBDD$ of a newly generated set of states S_g with the set of invalid states, $\neg cBDD$. In terms of BDD manipulation this is simply $S_g \wedge cBDD$. Extending the operation to the case where we have more than one *cBDD* is straightforward: $S_g \wedge cBDD_1 \wedge \cdots \wedge cBDD_k$. In the rest of the section we assume that *cBDD* is represented in a single BDD, without loss of generality.

An important remark about the usefulness of pruning invalid states is necessary, though. In general, the size of a BDD is not proportional to the number of states it represents. This means that there is no guarantee that pruning invalid states will help in symbolic search, as opposed to the explicit-state case. Indeed, pruning mutexes may cause an exponential blow-up on the BDD representation. There exist alternatives to the computation of the difference $S_g \setminus cBDD$ such as BDD "don't care" minimization methods [58, 59, 60]. However, an experimental analysis showed that in practice they are consistently worse than the approach presented in this article [34].

Finally, a clarification about the source of the constraints is needed. As described in Section 5.1, there are reachability and relevance constraints, which can be violated only by backward and forward search, respectively. Consequently, it is not useful to encode both types of constraints in the same BDD, as they should be exploited only in the appropriate direction. Hence, we use two cBDDs, $cBDD^{fw}$ for forward and $cBDD^{bw}$ for backward search.

5.3. Encoding Constraints in the TRs

Encoding the state invariants in the *cBDD* allows the search to prune invalid states after they are generated. In partial-state regression, *e-deletion* [61] modifies the definition of applicability in regression [6] in order to avoid the generation of spurious partial states. We apply the same idea in symbolic search,

encoding the state-invariant constraints in the TRs to avoid the generation of invalid states. Our encoding differs from previous work because we eliminate all invalid states from the search. In partial-state regression this is not possible because partial states implicitly contain states that violate invariants related to undefined variables.

We take as input the TR of an operator and a set of reachability and relevance constraints to define a new TR that does not generate invalid states. However, not all constraints need to be encoded in every operator. Replicating all the constraints in the TRs is possible, but may lead to a great degree of redundancy. We thus identify which constraints must be encoded in each operator in order to guarantee that no invalid state is generated. By assuming that the set of states to be expanded does not contain invalid states, we can safely consider only a subset of constraints in each TR, still avoiding the generation of invalid states. We say that a constraint is relevant for an operator if it may become violated after its application.

Definition 2. *(Relevant Constraint) A reachability constraint c is* relevant in regression *for an operator o iff exists s s.t. $o(s) \models c$ and $s \not\models c$. A relevance constraint c is* relevant in progression *for an operator o iff exists s s.t. $s \models c$ and $o(s) \not\models c$.*

As an example, consider Figure 5 on page 14. $\neg(v_t = 1 \wedge v_p = A)$ is relevant in progression for $drive(A, B, 0)$, which drives the truck from A to B at time step 0, because if we apply the operator in the initial state, $\langle v_p = A, v_T = A, v_t = 0\rangle$, we reach a state in which $v_p = A$ and $v_t = 1$. On the other hand, $\neg(v_t = 2 \wedge v_p = A)$ is irrelevant for $drive(A, B, 0)$ because this operator does not produce either $v_t = 2$ or $v_p = A$ so the successor state cannot violate the constraint unless the predecessor does.

Reachability constraints are encoded as additional preconditions to avoid the generation of invalid states in regression. This does not affect the applicability of the operator in forward search, since any reachable state necessarily satisfies these constraints. Relevance constraints are encoded in a similar way, as *postconditions* that must hold after applying the operator.

Definition 3. *(Constrained Operator) Let T_o be the TR of $o \in \mathcal{O}$ and let $cBDD_o^{bw}[x]$ and $cBDD_o^{fw}[x']$ be BDDs representing the reachability and relevance constraints relevant for o in terms of variables x and x'. Then the constrained TR is $T_o^c = T_o \wedge cBDD_o^{bw}[x] \wedge cBDD_o^{fw}[x']$.*

Theorem 1. *Let $S \subseteq \neg cBDD$ be a state set that does not contain states detected as invalid, $o \in \mathcal{O}$ be an operator, and T_o^c be the constrained TR derived from o. Let S_p, S_r be the resulting state sets from the image and pre-image of S with T_o^c, respectively. Then $S_p, S_r \subseteq cBDD$, i. e., they do not contain states that can be detected as invalid.*

Proof. Let c be a constraint, and s' a state that violates c. Suppose that $s' \in S_r$. As c was satisfied by every $s \in S$, c is relevant for o in regression. Then $c \in pre(o^c)$, so o^c is not applicable on s', reaching contradiction.

Suppose that $s' \in S_p$. As c was satisfied by every $s \in S$, c is relevant for o in progression. Then c is a postcondition of o^c, so $s' \neq o^c(s)$ for all $s \in S$, reaching contradiction.

□

Using T_o^c with relevant constraints suffices to guarantee that the successor set does not contain invalid states as long as the source set does not either. In progression this is always the case, as $\mathcal{I}$ is single valid state in any solvable instance. In regression, we compute the difference $S_\star \setminus cBDD$ to remove all invalid states from the goal description. The advantage is that this difference is only computed once, before starting the search. This avoids further overhead in using $cBDD$, which can be discarded to free memory.

Relevant Constraints in Progression. Proposition 1 shows the sufficient and necessary conditions for mutexes to be relevant in progression. In our case *"at-least-1"* groups are not relevant in progression, since we only infer them with a forward reachability analysis.

Proposition 1. *(Relevant mutex in progression) Let $M = \neg(f_1 \wedge f_2)$ be a relevance mutex and $o \in \mathcal{O}$ be a valid operator. Then M is relevant for o in progression if and only if:*

1. *M is not implied by $prev(o) \cup eff(o)$.*

2. *For some fact* $f_i \in \{f_1, f_2\}$, $f_i \in \mathit{eff}(o)$.

Proof. Assume WLOG that $f_i = f_1$. To prove the *if* part of the statement, let s be any state s.t. $f_1 \notin s$, $f_2 \in s$ and all other variables have values compatible with $\mathit{pre}(o)$. Such state exists because otherwise, either o is not valid (e.g., if $f_2 \in \mathit{eff}(o)$) or condition (1) does not hold contradicting the premises. Then s satisfies the three statements:

i $s \models M$: because $f_1 \notin s$.

ii o is applicable in s: By the construction of s. $f_1 \notin \mathit{pre}(o)$ because $f_1 \in \mathit{eff}(o)$. If $f_2 \in \mathit{eff}(o)$, then $f_2 \notin \mathit{pre}(o)$. If $f_2 \notin \mathit{eff}(o)$, then f_2 is compatible with $\mathit{prev}(o)$ by condition (1).

iii $o(s) \not\models M$: $f_1 \in \mathit{eff}(o)$ so $f_1 \in o(s)$. $f_2 \in o(s)$ because either $f_2 \in \mathit{eff}(o)$ or $f_2 \in s$. Condition (1) ensures that f_2 is not deleted by o.

To prove the *only if* case, note that if (1) does not hold, then there does not exist s such that $o(s) \not\models M$, so M is not relevant for o in progression. We may deduce condition (2) assuming that M is relevant for o in progression, that is, M is satisfied in s but not in $o(s)$. Then, for some fact $f_1 = \langle v_1, x\rangle$, $f_1 \notin s$, $f_1 \in o(s)$. Therefore, $s[v_1] \neq o(s)[v_1]$, which implies $v_1 \in \mathcal{V}_{\mathit{eff}(o)}$. As $o(s)$ is the result of applying o in s, $\mathit{eff}(o)[v_1] = x$ and $f_1 \in \mathit{eff}(o)$. □

Relevant Constraints in Regression. Next, Propositions 2 and 3 show the sufficient and necessary conditions for mutexes and *"at-least-1"* groups to be relevant in regression. Essentially the relevant constraints are those that contain facts that may be *added* or *removed* when o is applied in regression.

Proposition 2. *(Relevant mutex in regression) Let* $M = \neg(f_1 \wedge f_2)$ *be a reachability mutex and* $o \in \mathcal{O}$ *be a valid operator. Let* $\mathcal{V}_u(o) = \mathcal{V}_{\mathit{eff}(o)} \setminus \mathcal{V}_{\mathit{pre}(o)}$ *be the* set of undefined preconditions *of* o*. Then* M *is relevant in regression for* o *if and only if:*

1. M *is not implied by* $\mathit{pre}(o)$.
2. *For some fact* $f_i = \langle v_i, x\rangle \in \{f_1, f_2\}$*, either (2a)* $f_i \in \mathit{pre}(o) \setminus \mathit{prev}(o)$ *or (2b)* $v_i \in \mathcal{V}_u(o) \wedge f_i \notin \mathit{eff}(o)$.

Proof. Assume WLOG that $f_i = f_1$. To prove the *if* part of the statement, let s be any state s.t. $f_1, f_2 \in s$ and the rest of facts are compatible with $\mathit{pre}(o)$. Such an assignment exists when the operator is valid and condition (1) holds. Then s satisfies the three statements:

i $s \not\models M$ since $f_1, f_2 \in s$.

ii o is applicable in s: By the construction of s. Condition (1) ensures that f_1 and f_2 do not contradict $\mathit{pre}(o)$.

iii $o(s) \models M$: Due to rule (2), $f_1 \notin o(s)$. We have two cases (2a) or (2b). If (2b) holds, this is automatic, since $f_1 \notin \mathit{eff}(o)$ and $v_1 \in \mathcal{V}_{\mathit{eff}(o)}$. If (2a) holds, $f_1 \in \mathit{pre}(o)$ implies that o deletes f_1 (otherwise f_1 would be in $\mathit{prev}(o)$). Therefore, $f_1 \notin o(s)$ so $o(s) \models M$.

To prove the *only if* case, note that if (1) does not hold, then there does not exist s such that $s \not\models M$ and o is applicable in s, so M is not relevant for o. We may deduce condition (2) assuming that M is relevant for o in regression, i. e., there exists a state s such that $M \models o(s)$ and $M \not\models s$. Then, for some fact $f_1 = \langle v_1, x\rangle$, $f_1 \notin o(s)$, $f_1 \in s$. Therefore, $s[v_1] \neq o(s)[v_1]$, which implies $v_1 \in \mathcal{V}_{\mathit{eff}(o)}$ and $f_1 \notin \mathit{eff}(o)[v_1]$. Two cases are possible with respect to the preconditions of o, that correspond to conditions (2a) and (2b):

(a) $v_1 \in \mathcal{V}_{\mathit{pre}(o)}$. As o must be applicable in s, $f_1 \in \mathit{pre}(o)$.

(b) $v_1 \notin \mathcal{V}_{\mathit{pre}(o)}$ and $v_1 \in \mathcal{V}_u(o)$ immediately follows. □

Proposition 3. *(Relevant* "at-least-1" *invariant in regression) Let* $M_{inv} = f_1 \vee \cdots \vee f_k$ *be a reachability* "at-least-1" *invariant, and* $o \in \mathcal{O}$ *be a valid operator. Then* M_{inv} *is relevant in regression for* o *if and only if:*

1. M_{inv} *is not implied by the preconditions of the operator* $\mathit{pre}(o)$.

2. *For some fact* $f_i = \langle v_i, x \rangle \in M_{inv}$, $f_i \in \mathit{eff}(o)$.

Proof. To prove the *if* part of the statement, let s be a state s.t. $f_j \notin s$ for all $f_j \in M_{inv}$ and all other facts are compatible with $pre(o)$. Such an assignment exists when o is valid and condition (1) holds. Then s satisfies:

i $s \not\models M_{inv}$ since all $f_j \in M_{inv}$ are false in s.
ii o is applicable in s. By the construction of s since, by condition (1), $f_j \in M_{inv}$ does not contradict $pre(o)$.
iii $o(s) \models M_{inv}$: Necessarily $f_i \in o(s)$ since $f_i \in \mathit{eff}(o)$ due to rule (2).

To prove the *only if* case, note that if (1) does not hold, then there does not exist s such that $s \not\models M_{inv}$ and o is applicable in s, so M_{inv} is not relevant for o. We may deduce condition (2) from M_{inv} being relevant in regression for o, i. e., M_{inv} is satisfied in $o(s)$ but not in s. Then, for some fact $f_i = \langle v_i, x \rangle \in M_{inv}$, $f_i \notin s$, $f_i \in o(s)$. Therefore, $s[v_i] \neq s'[v_i] = x$, which implies $\mathit{eff}(o)[v_i] = x$ and, therefore, $f_i \in \mathit{eff}(o)$. □

5.4. Constrained Symbolic Abstraction Heuristics

The use of regression is not limited to backward search in the original state space. For instance, PDBs [35] usually perform regression in an abstraction, α, of the original problem to precompute the distance from every abstract state to the goal. PDBs in explicit-state search that make use of mutexes are known as *constrained PDBs* [51], cPDBs for short. cPDBs perform two types of pruning in the regression search over the abstract state space:

1. Prune all abstract states that violate a state invariant of the problem.
2. Prune transitions $s^\alpha \xrightarrow{o} t^\alpha$ when the abstract states, s^α and t^α, are incompatible with the semantics of the operator,[5] i. e., when the operator can only be applied on invalid states in s^α or its application on a valid state from s^α leads to an invalid state in t^α.

The use of pruning in cPDBs may simplify the regression search by reducing the size of the abstract state space. More importantly, cPDBs may potentially prune spurious paths, i. e., solution plans in the abstract state space that do not have a corresponding plan in the original problem. Thus, cPDBs are able to derive heuristics more informed than standard PDBs and are always as good as them. As mentioned in Section 3.3, GAMER uses a symbolic version of PDBs [39, 15]. We propose *symbolic constrained PDBs*, a constrained version of symbolic PDBs that exploits constraints as studied in previous sections, by encoding them in a *cBDD* or in the TRs. There are some subtleties related to the usage of constraints in PDBs that make it different from backward search in the original problem. We classify constraints into three classes depending on whether their variables are abstracted or not. *Full constraints* are entirely in the pattern, *partial constraints* have some facts in the pattern and some abstracted away and *null constraints* are those without facts in the pattern. These types of constraints are related with the pruning methods mentioned above:

1. Full constraints can be used to prune invalid abstract states. Partial or null constraints cannot be used for these purposes because some of their facts are abstracted so that they are never violated. As an example, take a mutex constraint $m = \neg(f_1 \wedge f_2)$. If we ignore the variable related to f_2, then we do not know whether it is true so that no abstract state will violate the constraint m.
2. Partial constraints can be used to prune transitions in which the semantics of the operator are incompatible with the abstract states.
3. Finally, null constraints are never useful for pruning the search. They could help to infer other constraints, but here we are assuming that the set of constraints is provided as input. Thus, we may ignore null constraints and assume that all the constraints are either full or partial.

[5] The original work of [51] considered only the case where o cannot be applied in regression on t^α. We extend here the definition to consider incompatibilities with s^α too.

Another remarkable difference (applicable to explicit-state search too) is that, while relevance constraints are redundant in regression and do not provide additional pruning in backward search, in the abstract space those constraints may be violated. Therefore, when working with abstractions both reachability and relevance constraints should be used.

Encoding constraints as BDDs. As studied in Section 5.2, one can encode the constraints in a BDD, *cBDD*, that corresponds to the conjunction of all the constraints and represents the set of valid states. Only full constraints are considered because the rest are never violated by abstract states. This corresponds to the pruning method proposed by Haslum et al. (2005) in explicit cPDBs. Pruning transitions with a *cBDD* is more complicated because it depends of the operator being applied, so that one has to individually check each operator. We do not contemplate this possibility since it would require maintaining different *cBDDs*, one per operator, and it cannot be combined with methods of image computation that represent multiples operators in the same TR as explained in Section 4.4. Therefore, unlike the case of search in the original state space, $\boldsymbol{\mathcal{M}_{BDD}}$ does not perform all the pruning possible.

Encoding constraints in the TRs. In Section 5.3, we identified which subset of constraints must be encoded in the TR of each operator to guarantee that no invalid state is generated, given that the predecessor set of states does not contain invalid states. For symbolic PDBs, full constraints are treated as in the case of the original search, getting the same guarantees of not generating any invalid abstract states. This suffices to perform the first type of pruning, as with the $\boldsymbol{\mathcal{M}_{BDD}}$ approach. However, in the abstract search the predecessor set of states may contain invalid states. In PDBs the value of abstracted variables is undefined, so abstract states correspond to partial states. Consider the set of states associated to an abstract state, S_i^α, that corresponds to all the possible assignments that the abstracted variables may take. Partial constraints may be violated for some states in S_i^α but not for others. S_i^α cannot be pruned without loss of admissibility because there exist valid states associated to it. As an example, take the constraint of our trucks example, $\neg(v_t = 2 \wedge v_p = A)$. If we ignore the time, an abstract state is composed of the position of the package and the position of the truck. Consider the abstract state $\mathcal{I}^\alpha$, $\langle v_p = A, v_T = A\rangle$. Obviously we cannot prune $\mathcal{I}^\alpha$. And still, for some values of the time variable v_t, the state would be invalid. Hence, this identifies abstract transitions $s^\alpha \xrightarrow{o} t^\alpha$ that are incompatible with the state invariants.

A simple way to consider all the possible inferences is to encode all the constraints in the TR representing the original operator and then abstract away the variables not in the pattern. Variables are abstracted away with existential quantification: there is a transition between two abstract states s^α, t^α if and only if a transition $s \xrightarrow{o} t$ exists in the original state space such that $\alpha(s) = s^\alpha$ and $\alpha(t) = t^\alpha$. This encoding ensures that all the possible pruning is performed. The original TR represents all the pairs of states s, t such that there exists a transition $s \xrightarrow{o} t$ in the original state space. Including all the constraints, we remove all the pairs related to invalid states. The result of the existential quantification is just the transitions between abstract states s^α, t^α such that a valid pair of states s, t supports the transition.

6. Experiments

In this section, we empirically evaluate the new techniques proposed in this paper. As the basis for the experiments we take the symbolic search planner GAMER [15, 32]. All planners used in our experiment use the h^2 invariant analysis to remove operators and simplify the planning task before starting the search [57]. This is very beneficial even for configurations that do not use the state invariant constraints. Originally, GAMER used unidirectional uniform-cost search and BDDA* with symbolic Pattern Databases. On top of that, we implemented the symbolic bidirectional uniform-cost search as described in Section 3.2. We also implemented the image computation refinements and state-invariant pruning techniques. We call the resulting planner cGAMER, standing for *constrained Gamer*.

We ran the experiments on a core of an Intel(R) Xeon(R) X3470 CPU @2.93 GHz with a time limit of 30 minutes and maximal memory usage of 4 GB. For BDD manipulation, the planner uses version 2.5.0 of the CUDD library [62]. All planners and libraries were compiled for 32-bit architectures. We evaluate our algorithms in all the STRIPS benchmarks in the optimal tracks of the International Planning Competition

(IPC) from 1998 to 2011, without conditional effects or derived predicates. They include 1396 tasks divided into 44 domains. The main metric to compare cost-optimal planners is coverage, i. e., the number of problems solved.

A caveat of the IPC benchmark set is that the number of problems in each domain is not uniform and there is overlapping between the domains used in different competitions, even reusing some problems. In order to get a total score comparison, we report a total metric that normalizes the score in every domain with respect to the number of problems, so that all the domains have the same weight (1 point per domain). In this metric, we consider the repeated problems and domains as a single entry, so that no domain or problem is counted twice.

In some cases, coverage is not fine-grained enough since, due to the exponential gap between problems, all versions have the same coverage even if there is a huge performance gap. Thus, in cases where the running time is not biased (e. g., by an expensive preprocessing phase when using PDBs), we use the time score metric used in the learning track of IPC-2011. This metric assigns a score between 0 and 1 to each planner for each problem solved, depending on the time it took to solve the problem (t) with respect to the time of the best planner (t^*). The fastest planner receives 1 point, and others receive less points according to a logarithmic scale:

$$\text{time-score}(t, t^*) = \begin{cases} 1, & \text{if } t \leq 1s \\ \frac{1}{1+\log_{10}(t/t^*)}, & \text{if } t > 1s \end{cases}$$

An important drawback of this time metric is that the results depend on the planners under consideration because t^* may be different. Adding a new planner may change the relative score of others, i. e., the winner between two planners may depend on whether a third planner is included, biasing the results. Hence, in general we will compare planners based on coverage metrics, using the time score to reveal differences between planners in domains where coverage is the same.

Of all symbolic algorithms, bidirectional uniform-cost search is the most suitable for an analysis on the impacts of our contributions because it is a state-of-the-art algorithm and the results do not depend on more parameters like the PDB generation procedure. Thus, we start evaluating the impact of image computation and state-invariant pruning in symbolic bidirectional uniform-cost search and, only later, we extend our analysis to other algorithms. We conclude our experiments by comparing our results to explicit-state search planners, showing that our improvements allow cGamer to outperform other state-of-the-art planners across a variety of domains.

6.1. Image Computation

Table 1 shows the results of different image computation methods:

- $\mathbf{TR^1}$ is our baseline, Gamer's original image computation.
- $\mathbf{TR^{1+}}$ (see Section 4.2) separately represents preconditions and effects to avoid using auxiliary variables.
- **CT** and $\mathbf{CT_{20}}$ use the conjunction tree (see Section 4.3). We report results with $\mathbf{CT_{20}}$ because it gave the best results, though other parameter configurations of the conjunction tree have similar results [34].
- $\mathbf{UT^{DT}_{100k}}$ unifies TRs (see Section 4.4) using up to 100 000 nodes and limiting the TR aggregation time to 60 seconds.

All the new image computation methods outperform the baseline. They are better not only in terms of total coverage and score, but also on a domain per domain basis. The time score results show that the new image computation methods provide an improvement in almost all domains (except in PSR-small, where all the problems are solved, and Parking, where no configuration solves any problem). This increase in efficiency allows the planner to solve more problems in 21 out of 44 domains. Given the exponential growth in problem complexity in most domains this reveals significant performance gains. Next, we analyze the results of each technique.

	Coverage					Time score				
	$\mathbf{TR^1}$	$\mathbf{TR^{1+}}$	**CT**	$\mathbf{CT_{20}}$	$\mathbf{UT^{DT}_{100k}}$	$\mathbf{TR^1}$	$\mathbf{TR^{1+}}$	**CT**	$\mathbf{CT_{20}}$	$\mathbf{UT^{DT}_{100k}}$
Airport (50)	23	23	22	23	**24**	20.82	20.96	18.80	21.24	**23.97**
Barman (20)	8	8	8	8	8	5.80	5.92	5.65	5.98	**8.00**
Blocksworld (35)	21	**23**	21	**23**	21	19.83	**22.60**	19.01	**22.77**	20.10
Depot (22)	5	**6**	5	**6**	5	4.08	**5.71**	4.13	**5.68**	4.70
Driverlog (20)	12	13	12	13	**14**	9.42	11.11	9.98	11.22	**14.00**
Elevators08 (30)	24	**25**	**25**	**25**	**25**	16.85	20.39	20.02	20.84	**25.00**
Elevators11 (20)	19	19	19	19	19	12.79	15.34	15.04	15.52	**19.00**
Floortile11 (20)	10	11	10	11	**12**	7.39	9.39	8.13	9.37	**12.00**
Freecell (80)	14	14	14	16	**20**	9.52	9.70	10.78	12.93	**20.00**
Grid (5)	2	2	2	2	2	1.57	1.57	1.63	1.67	**2.00**
Gripper (20)	20	20	20	20	20	15.91	17.38	17.55	**19.54**	18.21
Logistics 00 (28)	20	20	20	20	20	16.44	18.37	17.34	18.42	**19.96**
Logistics 98 (35)	5	5	5	5	5	3.23	4.00	3.44	3.76	**5.00**
Miconic (150)	86	105	105	106	**113**	61.79	82.40	81.71	82.51	**113.00**
MPrime (35)	19	21	22	**23**	21	16.25	17.61	18.43	**20.27**	**20.36**
Mystery (30)	**14**	**14**	13	13	**14**	12.56	12.65	12.08	12.56	**13.33**
NoMystery11 (20)	14	15	15	15	**16**	10.67	13.63	13.25	13.89	**15.68**
Openstacks08 (30)	30	30	30	30	30	21.95	23.70	23.74	24.49	**30.00**
Openstacks11 (20)	20	20	20	20	20	13.69	14.88	14.47	15.01	**20.00**
Openstacks06 (30)	11	12	11	12	**20**	9.40	9.81	9.19	10.05	**19.98**
PARC-Printer08 (30)	21	22	20	22	**24**	17.91	19.69	17.67	18.85	**23.62**
PARC-Printer11 (20)	16	17	15	17	**18**	12.73	14.54	12.67	14.40	**17.32**
Parking11 (20)	0	0	0	0	0	0.00	0.00	0.00	0.00	0.00
Pathways-noneg (30)	5	5	5	5	5	4.77	4.82	4.50	4.88	**5.00**
Peg-Solitaire08 (30)	29	**30**	**30**	**30**	**30**	25.09	29.43	28.01	**29.88**	**29.70**
Peg-Solitaire11 (20)	19	**20**	**20**	**20**	**20**	15.52	19.33	17.68	**19.89**	**19.70**
Pipesworld-nt (50)	**15**	**15**	**15**	**15**	14	12.90	12.36	13.69	12.86	**13.85**
Pipesworld-t (50)	16	15	15	16	**17**	11.64	10.69	10.82	11.97	**17.00**
PSR-small (50)	50	50	50	50	50	**49.56**	**49.45**	**49.45**	**49.51**	**49.74**
Rovers (40)	14	14	14	14	14	10.84	11.87	11.68	11.91	**14.00**
Satellite (36)	7	**9**	**9**	**9**	**9**	5.99	7.66	7.62	7.65	**9.00**
Scanalyzer08 (30)	12	12	12	12	12	8.87	10.02	10.50	10.54	**11.68**
Scanalyzer11 (20)	9	9	9	9	9	5.96	7.05	7.67	7.71	**8.94**
Sokoban08 (30)	**28**	**28**	27	**28**	**28**	21.40	21.70	19.19	21.71	**27.97**
Sokoban11 (20)	20	20	20	20	20	15.01	15.38	14.07	15.38	**19.97**
Tidybot11 (20)	**12**	9	**12**	**12**	11	10.76	7.83	**11.41**	**11.31**	10.95
TPP (30)	8	8	8	8	8	7.18	7.62	7.28	7.62	**8.00**
Transport08 (30)	12	**14**	13	**14**	**14**	10.10	12.19	11.60	12.72	**14.00**
Transport11 (20)	8	**10**	7	**10**	**10**	5.74	7.96	5.84	8.54	**10.00**
Trucks (30)	10	10	10	10	**11**	8.32	9.18	8.79	9.16	**11.00**
VisitAll (20)	12	12	12	12	12	10.42	11.65	11.34	11.66	**11.94**
Woodworking08 (30)	22	22	22	22	22	17.15	18.90	17.83	18.91	**22.00**
Woodworking11 (20)	16	16	16	16	16	11.45	13.10	12.04	13.01	**16.00**
Zenotravel (20)	10	**11**	10	**11**	**11**	7.31	9.81	9.04	9.58	**11.00**
Total (1396)	748	784	770	792	**814**	596.58	669.35	644.76	687.37	**806.67**
Score (36)	16.00	16.51	16.25	16.74	**17.13**	12.88	14.25	13.77	14.79	**16.91**

Table 1: Results of bidirectional uniform-cost search with different image computation methods. The best configurations per domain and those deviating 1% are highlighted.

$\mathbf{TR^{1+}}$ *dominates the baseline.* It has equal or better performance in all domains except Tidybot and the two versions of Pipesworld. In total, it solves 36 more problems than $\mathbf{TR^1}$ across 17 domains.

$\mathbf{CT_{20}}$ *dominates* $\mathbf{TR^{1+}}$. Even though **CT** has slightly worse total score than $\mathbf{TR^{1+}}$, it performs well in domains that have lots of operators with many shared preconditions. This is the case in Freecell and, especially, Tidybot where **CT** is the best method. However, in other domains (e.g., Blocksworld, Sokoban, or Transport) the coverage decreases due to the additional overhead. $\mathbf{CT_{20}}$ limits this overhead by setting the *min operators conjunction* parameter to 20 operators. In most domains, the time score of $\mathbf{CT_{20}}$ is just the maximum of $\mathbf{TR^{1+}}$ and **CT**, so the parameter is successfully able to control when it is useful to use the conjunction tree approach. Moreover, there are some cases in which $\mathbf{CT_{20}}$ is better than both $\mathbf{TR^{1+}}$ and **CT** like in Freecell and Gripper.

Overall, $\mathbf{UT^{DT}_{100k}}$ *is the best image computation method.* Even though all the new image approaches improve the results of our base planner, GAMER, the clear winners among our image computation variants are the approaches that aggregate TRs. The total time score of $\mathbf{UT^{DT}_{100k}}$ is 806.67, very close to the total coverage of 814. This means that $\mathbf{UT^{DT}_{100k}}$ is reliably the fastest image computation method in almost every domain. In

the few domains where it is not the fastest one, its performance is close to the fastest. Figure 6 reflects the overwhelming superiority of $\mathbf{UT^{DT}_{100k}}$ over the baseline. $\mathbf{UT^{DT}_{100k}}$ is better in the vast majority of problems, solving problems up to two orders of magnitude faster than the baseline and being only clearly worse on four problems of the whole benchmark set. Moreover, $\mathbf{UT^{DT}_{100k}}$ obtains the best coverage in many domains, solving 66 more problems than the original image computation of GAMER (27 of those are in Miconic). Other methods outperform $\mathbf{UT^{DT}_{100k}}$ only in a few domains and by a small margin.

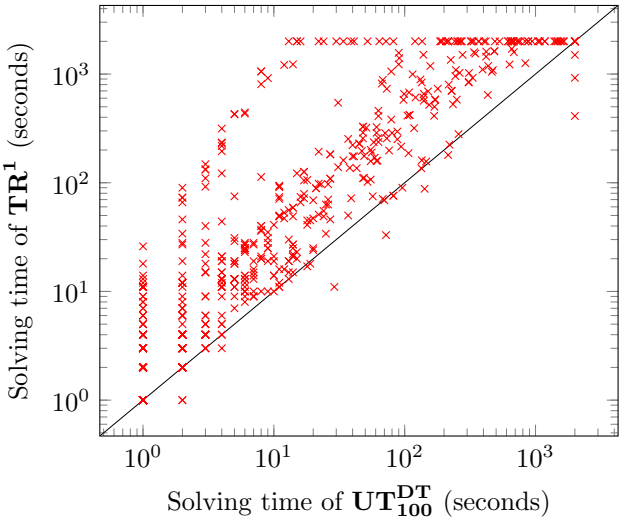


Figure 6: Comparison of solving time of the new $\mathbf{UT^{DT}_{100k}}$ image computation versus $\mathbf{TR^1}$. Unsolved problems are assigned a time of 2000 seconds.

What is the best parameter configuration of TR aggregation?. Our aggregation algorithm (see Algorithm 3), which automatically derives a disjunctive partitioning of the TR, has two different parameters to control the resulting partitioning. On the one hand, in Section 4.4, we defined three criteria to select which TRs to merge in each algorithm iteration, $\mathbf{UT^{DT}}$, $\mathbf{UT^{SM}}$ and $\mathbf{UT^{CT}}$. On the other hand, the maximum TR size controls the memory used to represent the TRs. If the maximum TR size is set to 1, no TRs are aggregated and the algorithm behaves as the original $\mathbf{TR^1}$. If the maximum TR size is set to ∞, the planner will use a monolithic TR for each operator cost, but may exceed the available memory in the initialization. Given the memory limit of 4GB, we set the parameter to different values ranging from 1000 nodes (1k) to one million (1M) plus ∞ which corresponds to the monolithic TR approach. Figure 7 shows the time score and total coverage of each parameter configuration.

The configuration without any TR aggregation only solves 748 instances, 73 less than the best configuration in terms of coverage. However, setting the maximum TR size to 1000 nodes is enough to solve 818 instances, only 3 behind the best achieved. The fastest configurations are reliably those setting the maximum size of TRs in 10 000 or 100 000. The coverage slightly decreases with larger values, mainly because the TRs use too much memory.

Regarding the aggregation criteria, $\mathbf{UT^{SM}}$ performs clearly worse than the other criteria, but the differences between the rest are not significant. Thus, any of these strategies which aggregate "similar" TRs with respect to different definitions are all valid strategies to decide which TRs should be aggregated. In the rest of the experiments we will use the $\mathbf{UT^{DT}_{100k}}$ image computation. Even though there are other configurations with slightly better coverage, $\mathbf{UT^{DT}_{100k}}$ is a simpler criterion and has the best time score.

6.2. Constrained Symbolic Search

Table 2 shows the impact of using state-invariant constraints to prune symbolic bidirectional uniform-cost search. Our baseline is the original GAMER planner that does not use state-invariant pruning. The

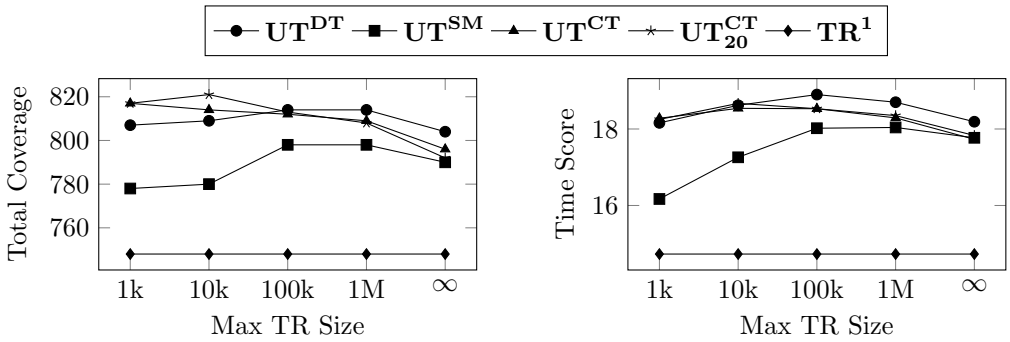


Figure 7: Total time score and coverage of bidirectional uniform-cost search with different configurations of TR aggregation. The baseline, $\mathbf{TR^1}$, is included for comparison.

rest of the configurations prune all the states that violate any invariant found by the preprocessor and they only differ on how the constraints are encoded. $\mathcal{M}^{1k}_{BDD}$ and $\mathcal{M}^{100k}_{BDD}$ encode the constraints as BDDs with a maximum number of nodes of 1 000 and 100 000, respectively. In **e-del** and **e-del**$^+$ constraints are encoded directly in the TRs. **e-del**$^+$ include all discovered constraints in the TR of each operator and **e-del** encodes only the necessary constraints for each operator.

State-invariant pruning dramatically improves the performance. The best configuration solves up to 862 problems, 48 more than the baseline. Given that the difficulty of the instances increases exponentially in many domains, such increase in performance is very significant. Even though there is no theoretical guarantee that pruning states increases the performance of BDD-based search, this seems to be the case for the set of benchmarks considered in our experiments. The only cases where the search performance slightly decreases are Grid, PARC-Printer and VisitAll. This may be due to many different factors, though it can be observed that in Grid and VisitAll relatively few mutexes are found with respect to other domains so there is less potential for pruning. PARC-Printer is characterized for having a large number of action costs, so that not too many states have the same g-value and BDDs are never too large. On the other hand, performance increases dramatically in many cases, such as Barman, Floortile, Freecell, etc. As shown by Figure 8 the use of state invariants is almost always beneficial and the speed-up is of up to two orders of magnitude.

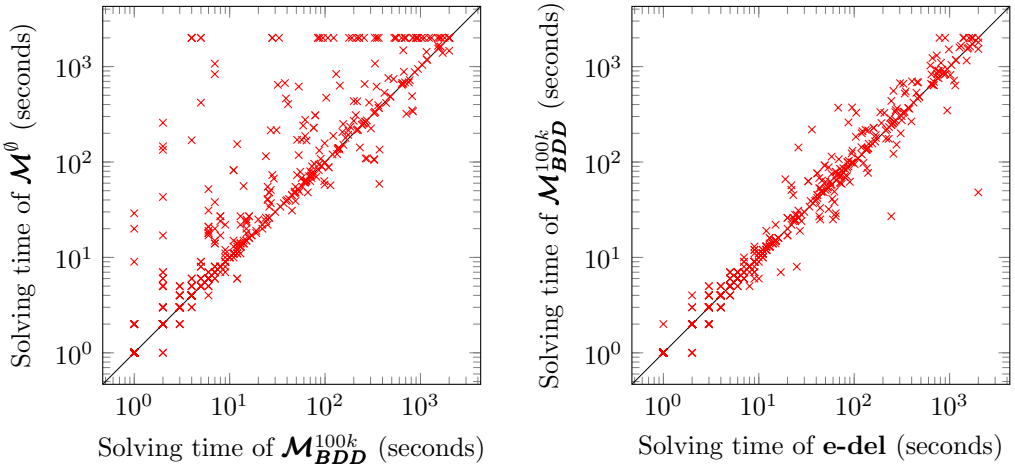


Figure 8: Comparison of time to solve problems with approaches using state-invariant constraints. Unsolved problems are assigned a time of 2000 seconds.

	Coverage					Time score				
	$\mathcal{M}^{\emptyset}$	$\mathcal{M}^{1k}_{BDD}$	$\mathcal{M}^{100k}_{BDD}$	**e-del**	**e-del$^+$**	$\mathcal{M}^{\emptyset}$	$\mathcal{M}^{1k}_{BDD}$	$\mathcal{M}^{100k}_{BDD}$	**e-del**	**e-del$^+$**
Airport (50)	24	25	25	**27**	23	23.56	24.00	24.51	**25.67**	19.75
Barman (20)	8	10	11	**12**	8	5.26	8.31	9.75	**12.00**	7.06
Blocksworld (35)	21	31	31	**33**	21	16.87	30.34	29.99	**31.83**	15.13
Depot (22)	5	**8**	**8**	**8**	5	3.60	7.36	7.49	**7.84**	3.47
Driverlog (20)	14	14	14	14	14	13.38	13.76	**14.00**	13.77	13.75
Elevators08 (30)	25	25	25	25	25	**25.00**	**24.88**	**24.91**	**24.91**	**24.95**
Elevators11 (20)	19	19	19	19	19	**18.97**	**18.97**	**18.92**	**18.92**	**18.82**
Floortile11 (20)	12	**14**	**14**	**14**	**14**	8.11	13.69	13.72	**14.00**	13.41
Freecell (80)	20	25	25	**27**	14	16.14	23.79	24.28	**26.23**	10.11
Grid (5)	2	2	2	2	2	**2.00**	1.68	1.67	1.52	0.95
Gripper (20)	20	20	20	20	20	17.47	17.26	17.21	19.68	**19.93**
Logistics 00 (28)	20	20	20	20	20	**19.99**	**19.95**	**19.95**	**19.96**	**19.98**
Logistics 98 (35)	5	5	5	5	5	**4.99**	**5.00**	**5.00**	**5.00**	**5.00**
Miconic (150)	113	113	113	113	113	**112.82**	**112.88**	**112.91**	**112.53**	**112.98**
MPrime (35)	21	**22**	**22**	21	20	20.97	**21.47**	**21.46**	20.42	18.99
Mystery (30)	**14**	**14**	**14**	13	10	12.98	**13.49**	**13.47**	11.61	9.25
NoMystery11 (20)	16	16	16	16	16	**15.74**	**15.82**	**15.81**	15.58	15.31
Openstacks08 (30)	30	30	30	30	30	29.16	29.44	29.48	**30.00**	**29.78**
Openstacks11 (20)	20	20	20	20	20	19.31	19.54	19.54	**20.00**	19.76
Openstacks06 (30)	**20**	**20**	**20**	**20**	14	18.24	18.23	**19.30**	18.66	11.79
PARC-Printer08 (30)	**24**	23	23	22	23	**23.35**	21.25	21.58	19.59	20.78
PARC-Printer11 (20)	**18**	**18**	**18**	**18**	17	**17.12**	16.65	16.42	16.10	14.85
Parking11 (20)	0	**1**	**1**	**1**	0	0.00	0.98	**1.00**	0.98	0.00
Pathways-noneg (30)	5	5	5	5	5	4.69	**4.96**	**5.00**	4.68	4.80
Peg-Solitaire08 (30)	30	30	30	30	30	**29.98**	**30.00**	**30.00**	**29.77**	24.34
Peg-Solitaire11 (20)	20	20	20	20	20	**19.98**	**19.99**	**20.00**	19.65	15.17
Pipesworld-nt (50)	14	15	15	**17**	10	13.15	14.54	14.17	**15.89**	6.50
Pipesworld-t (50)	**17**	**17**	**17**	**17**	10	15.80	16.08	15.72	**16.42**	9.66
PSR-small (50)	50	50	50	50	50	**49.59**	**49.51**	**49.77**	**49.36**	49.22
Rovers (40)	14	14	14	14	**15**	13.61	13.67	13.68	13.59	**14.45**
Satellite (36)	9	11	11	11	**13**	8.09	10.81	10.80	10.64	**12.70**
Scanalyzer08 (30)	12	12	12	12	12	11.26	10.72	10.77	**11.38**	11.20
Scanalyzer11 (20)	9	9	9	9	9	8.27	7.95	8.00	**8.61**	8.44
Sokoban08 (30)	**28**	**28**	**28**	**28**	27	22.52	26.89	27.36	**27.77**	22.41
Sokoban11 (20)	**20**	**20**	**20**	**20**	19	16.54	19.40	19.49	**19.77**	15.74
Tidybot11 (20)	11	**17**	**17**	**17**	12	9.73	15.77	15.80	**16.95**	10.29
TPP (30)	8	8	8	8	8	**8.00**	**8.00**	**8.00**	**8.00**	7.66
Transport08 (30)	14	14	14	14	14	**14.00**	**13.97**	**14.00**	**14.00**	**14.00**
Transport11 (20)	10	10	10	10	10	**9.99**	**9.99**	9.67	**10.00**	9.89
Trucks (30)	11	**13**	**13**	12	**13**	9.46	12.54	12.53	11.02	**12.85**
VisitAll (20)	12	12	12	12	12	**12.00**	11.74	**11.97**	11.74	11.75
Woodworking08 (30)	22	**26**	**26**	**26**	**26**	19.02	24.68	24.58	**25.93**	**25.69**
Woodworking11 (20)	16	**19**	**19**	**19**	**19**	13.27	17.64	17.65	**18.93**	18.65
Zenotravel (20)	11	11	**12**	11	**12**	10.92	10.97	**11.97**	10.99	**12.00**
Total (1396)	814	856	858	**862**	799	764.90	828.56	833.30	**841.89**	743.21
Score (36)	17.13	**18.45**	**18.55**	**18.62**	16.92	15.82	17.67	17.86	**18.10**	15.61

Table 2: Results of bidirectional uniform-cost search with different state-invariant pruning methods. The best configurations per domain and those deviating 1% are highlighted.

What is the best encoding for state-invariant constraints?. Whenever we use the encoding of constraints in separated BDDs of different sizes, $\mathcal{M}^{1k}_{BDD}$ and $\mathcal{M}^{100k}_{BDD}$, we obtain similar results, though slightly favoring the version with a larger limit of nodes. As concluded in the experiments regarding the TR representation in the previous section, using larger bounds for the BDD size leads to faster computation at the expense of using more memory. Encoding constraints in the TRs, **e-del**, is the most efficient way to use the constraints in the search. The BDDs involved in the search are not affected, but **e-del** reduces the overhead in pruning the invalid states and even gets speed-ups in the image computation. The advantage of e-deletion is especially noticeable in Barman, Freecell or Pipesworld, in which there are many mutexes, though the performance decreases in a few domains like Mystery or Zenotravel. The right plot of Figure 8 shows that the advantage of **e-del** over $\mathcal{M}^{100k}_{BDD}$ is more moderate than our other comparisons.

Encoding only the necessary constraints is important. **e-del** encodes the necessary constraints, according to our theoretical results in Section 5.3. In order to show the importance of identifying which constraints are relevant, we compare the results with **e-del$^+$**, which encodes all the constraints in the TRs. Even though in Rovers and Satellite encoding all the constraints simplifies the search, in most cases it is unfeasible to do so, such that the overall results are worse than the configuration not performing pruning at all. This

highlights the importance of not including constraints in the operators if they are completely unrelated to the variables affected by the operator.

What is the impact of **e-del** *on TR representation?.* Figure 9 plots the relative number of BDD nodes used to represent each transition relation. Cases where a monolithic TR could not been generated were excluded, since they are not really comparable. The size of the TRs is expected to increase when encoding state-invariant constraints and, indeed, that is the general trend. However, the increase is in most cases below an order of magnitude and there even are a number of cases where the constraints simplify the TRs. This explains how **e-del** is able to provide an advantage over representing the constraints in separated BDDs.

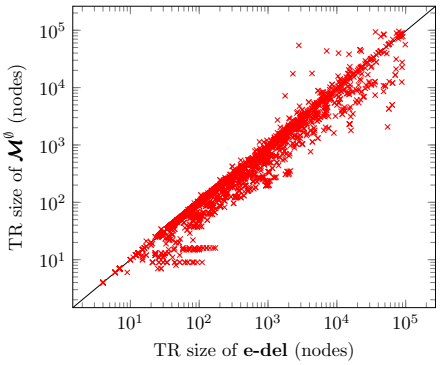


Figure 9: Number of BDD nodes of the TR excluding cases where a single TR could not be created with 100 000 nodes or less.

6.3. Symbolic Unidirectional Uniform-Cost Search

The results of previous subsections have shown that the new image computation and state invariant pruning methods greatly enhance the performance of bidirectional search. In this section, we compare the results in unidirectional search and examine whether the new image techniques have more impact in forward or backward search.

Image computation in forward and backward search. Table 3 shows the summary scores of the image computation methods on unidirectional uniform-cost search in forward and backward direction. The results are similar to those in the bidirectional search case, with $\mathbf{UT^{DT}_{100k}}$ being the best configuration for image computation in almost every domain and the new image computation approaches outperforming the baseline, $\mathbf{TR^1}$. The directionality of the search does not affect the comparison of the different image approaches. This confirms that the same principles apply to image and pre-image computation as introduced in Section 4.1.

	Forward				Backward			
	$\mathbf{TR^1}$	$\mathbf{TR^{1+}}$	$\mathbf{CT_{20}}$	$\mathbf{UT^{DT}_{100k}}$	$\mathbf{TR^1}$	$\mathbf{TR^{1+}}$	$\mathbf{CT_{20}}$	$\mathbf{UT^{DT}_{100k}}$
Total Time (1396)	595.05	618.50	645.25	**740.36**	306.73	390.25	396.19	**467.42**
Score Time (36)	14.79	15.22	15.98	**18.09**	7.39	9.12	9.22	**10.43**
Total Cov (1396)	722	734	749	**768**	453	534	538	**541**
Score Cov (36)	17.63	17.73	18.13	**18.58**	10.71	12.22	12.29	**12.34**

Table 3: Time and coverage scores of symbolic forward and backward uniform-cost search using different image computation methods. Best results in each direction are highlighted.

State-invariant pruning in forward and backward search. In IPC benchmarks, there are more reachability than relevance state-invariant constraints [57]. Thus, invariant constraints can be expected to have a greater impact in backward search than forward search. Table 4 reports the results of different state-invariant pruning methods in forward and backward search. The impact of pruning invalid states during the search is much larger in backward than in forward search. While invariant pruning increases the coverage of forward

search in 8 problems, it allows backward search to solve 165 more instances. This is mainly due to the number of mutexes found in each direction:

- In forward search, not enough mutexes are found to improve the performance. Mutexes are reliably found for all the problem instances only in Floortile, Mystery, PARC-Printer and Trucks. The results in these domains vary a lot. Mutex pruning helps greatly in Floortile, moderately in Trucks, does not have an impact in Mystery and decreases performance in PARC-Printer. The case of PARC-Printer is an exception caused by the structure of the domain, since the search performs many cheap steps and the overhead of mutex pruning might decrease performance. This overhead is almost completely eliminated with e-deletion, though. On the other hand, it is remarkable that e-deletion may increase the performance of forward search, even in domains where constraints provide no pruning such as Blocksworld, Depot, Pipesworld or Tidybot. In those domains, encoding the constraints in the TRs helps to simplify image computation and to moderately speed up search.
- In backward search, however, there are many available constraints in most domains and they can be successfully exploited to improve performance in most of them. In several domains where regression without pruning fails completely such as Barman or Blocksworld, using state-invariant pruning not only makes regression possible, but makes it outperform forward search.

The results of unidirectional search also shed more light on what is the best encoding for state-invariant constraints. **e-del** outperforms $\boldsymbol{\mathcal{M}}^{100k}_{\boldsymbol{BDD}}$ both in forward and backward search, but its impact is larger in the case of backward search, where there are more state invariants to take advantage of. Since bidirectional uniform-cost search combines takes advantage of the best search direction, the advantages of **e-del** are only truly reflected in domains where backward search is not inferior to forward search, e. g., Barman, Blocksworld, or Freecell.

Do state invariants affect the directionality of the domains?. In general, the results depict a great advantage of forward over backward search. However, the use of state-invariant constraints helps to reduce the gap. Without the use of invariants, backward search is only clearly better in Miconic, whereas using state-invariants, it is also better in Barman, Blocksworld, NoMystery, Satellite, and Woodworking. This is also important to improve the performance of bidirectional search, as we have analyzed in Section 6.2.

*6.4. Symbolic BDDA**

Table 5 evaluates our symbolic search enhancements in BDDA* with symbolic PDBs. The symbolic PDBs are generated with GAMER's hill climbing method [15] in a precomputation phase that is terminated after 900 seconds. All our enhancements are used both in the symbolic PDB generation and in the symbolic A* algorithm. Time score metrics are omitted because they are not representative of A* performance, given that all the algorithms spend a maximum of 900 seconds in the PDB generation before starting the search.

Do our techniques improve BDDA?.* Image computation methods improve BDDA* as much as bidirectional symbolic uniform-cost search. This reveals that the advantage of the new image computation methods is not limited to a particular algorithm. State-invariant constraints also help, but less than in bidirectional symbolic uniform-cost search. The overall results again show that the use of state invariants helps to improve results of A* as well. However, the benefits are not as stable as in the case of uniform-cost search. Even though pruning invalid states ($\boldsymbol{\mathcal{M}}^{100k}_{\boldsymbol{BDD}}$) increases total coverage, it decreases the coverage in 9 domains with respect to the version without invalid state pruning ($\boldsymbol{\mathcal{M}}^{\emptyset}$). In order to find out whether the performance loss is caused by the abstract searches that generate the heuristic or the A* search, we analyze the quality of the heuristic used. One typical way to measure such quality is the heuristic value of the initial state.

Figure 10 shows that the our techniques can be used for the generation of more informed PDB heuristics. The image computation methods improve the performance of the abstract searches accross all domains, allowing the planner to explore more informed abstractions, so they are almost always beneficial. State-invariant pruning, however, decreases performance in some cases, obtaining worse heuristic estimates. Obviously,

	Forward			Backward		
	$\mathcal{M}^{\emptyset}$	$\mathcal{M}^{100k}_{BDD}$	e-del	$\mathcal{M}^{\emptyset}$	$\mathcal{M}^{100k}_{BDD}$	e-del
Airport (50)	***23.58**	*23.18	*22.80	9.29	21.90	21.73
Barman (20)	6.76	6.76	6.88	0.00	6.48	***9.96**
Blocksworld (35)	19.02	18.93	20.05	10.34	19.43	***22.75**
Depot (22)	3.79	3.79	***5.00**	1.51	1.45	2.91
Driverlog (20)	***10.95**	*10.73	*10.77	7.24	7.45	7.48
Elevators08 (30)	***18.98**	***19.00**	***19.00**	4.95	4.95	4.95
Elevators11 (20)	***15.99**	***16.00**	***15.99**	3.51	3.51	3.51
Floortile11 (20)	5.48	*13.58	***14.00**	5.64	*13.03	*13.11
Freecell (80)	*19.03	*18.75	*19.12	4.86	16.69	***19.42**
Grid (5)	***2.00**	***2.00**	***1.99**	0.32	0.45	0.41
Gripper (20)	*16.92	*16.94	***20.00**	8.72	10.95	*18.44
Logistics 00 (28)	**16.00**	**16.00**	**16.00**	13.80	13.80	13.80
Logistics 98 (35)	*3.09	*4.64	***5.00**	2.57	2.57	2.57
Miconic (150)	79.77	79.63	79.85	***113.84**	***113.57**	***113.66**
MPrime (35)	***25.87**	***25.90**	*25.30	8.31	9.05	9.00
Mystery (30)	***15.00**	***14.89**	*14.43	6.08	6.73	6.58
NoMystery11 (20)	10.07	10.08	10.10	11.21	10.85	**11.66**
Openstacks08 (30)	***29.68**	***29.91**	***29.93**	25.39	*26.44	*27.32
Openstacks11 (20)	**19.91**	**19.91**	**20.00**	17.88	18.38	18.98
Openstacks06 (30)	*19.49	*19.40	***19.85**	6.70	10.47	13.22
PARC-Printer08 (30)	***23.73**	19.66	*23.27	10.96	18.67	19.47
PARC-Printer11 (20)	***17.80**	14.45	***17.72**	7.18	14.61	*15.18
Parking11 (20)	0.00	0.00	0.00	0.00	0.00	0.00
Pathways-noneg (30)	***5.00**	***5.00**	*4.89	3.39	3.65	3.72
Peg-Solitaire08 (30)	***28.99**	***28.81**	*28.34	10.93	13.83	13.56
Peg-Solitaire11 (20)	***18.98**	*18.65	*18.38	2.83	5.55	5.50
Pipesworld-nt (50)	14.32	14.06	***16.22**	1.75	5.82	6.67
Pipesworld-t (50)	*16.46	*16.45	***16.77**	3.86	4.74	4.83
PSR-small (50)	**49.77**	49.16	49.08	47.44	48.80	48.24
Rovers (40)	***12.82**	***12.82**	***12.82**	10.92	10.93	10.94
Satellite (36)	6.57	6.34	6.57	7.22	*9.68	***10.00**
Scanalyzer08 (30)	11.34	10.56	10.94	**11.52**	**11.63**	11.25
Scanalyzer11 (20)	8.31	8.06	8.41	8.52	8.49	**8.71**
Sokoban08 (30)	*26.16	***26.85**	***27.12**	3.67	22.83	23.65
Sokoban11 (20)	*19.11	***19.44**	***19.51**	0.82	16.50	16.93
Tidybot11 (20)	*14.89	*14.81	***15.92**	1.00	6.70	7.06
TPP (30)	**8.00**	**8.00**	**8.00**	6.75	6.83	6.88
Transport08 (30)	***12.00**	*11.86	***12.00**	7.05	7.00	7.00
Transport11 (20)	***7.00**	*6.89	*6.90	2.04	2.04	2.04
Trucks (30)	*9.74	***10.75**	***10.74**	6.84	8.77	9.50
VisitAll (20)	**9.00**	**9.00**	**9.00**	8.73	8.74	8.79
Woodworking08 (30)	16.37	16.41	15.98	15.08	22.08	***25.00**
Woodworking11 (20)	10.96	10.94	10.67	9.71	16.38	***19.00**
Zenotravel (20)	*8.89	*8.88	***9.00**	7.24	7.23	7.23
Total Time (1396)	717.59	717.87	**734.31**	457.61	599.65	632.61
Score Time (36)	15.13	15.51	**15.91**	8.76	11.85	12.91
Total Cov (1396)	768	771	**776**	541	687	706
Score Cov (36)	16.23	**16.48**	**16.56**	10.63	13.80	14.42

Table 4: Time score of unidirectional uniform-cost search with different configurations of constraints encoding. The best configurations per domain and those deviating in only 1% are highlighted in bold and, in domains where there are differences in coverage, the configurations with best coverage are marked with *.

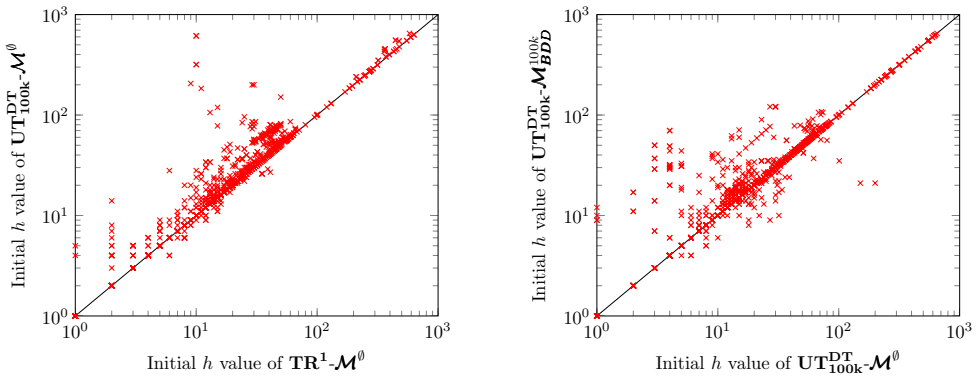


Figure 10: Heuristic value of the initial state in each instance (cases under 1000).

		$\mathbf{TR^1}$	$\mathbf{TR^{1+}}$	$\mathbf{CT}$ $\mathcal{M}^{\emptyset}$	$\mathbf{CT_{20}}$	$\mathbf{UT^{DT}_{100k}}$	$\mathbf{UT^{DT}_{100k}}$ $\mathcal{M}^{1k}_{BDD}$	$\mathcal{M}^{100k}_{BDD}$
Total Cov	BDDA*	730	744	711	753	792	807	810
(1396)	Bidir	748	784	770	792	814	856	**858**
Score Cov	BDDA*	16.25	16.28	16.08	16.60	17.48	17.86	17.87
(36)	Bidir	16.00	16.51	16.25	16.74	17.13	**18.45**	**18.55**

Table 5: Coverage of BDDA* and bidirectional uniform-cost search with different image computation and state-invariant pruning techniques.

using state invariants cannot decrease the values of a given abstraction so the reason is that different abstractions are being explored. Even though state-invariant constraints can only increase the heuristic value for a given pattern, this may change which patterns are preferred by the hill-climbing search in the space of possible patterns. Another reason is that, for some patterns, the abstract searches become much harder when using state-invariant constraints. This is not completely unexpected since, as argued before, there are no theoretical guarantees of the search being simpler when using state-invariants constraints. However, it may be surprising that this only happens in abstract state spaces and not in the whole search space. The reason is that the abstract symbolic search may have lower complexity than the original search, especially if the abstract problem consists of independent subproblems, i. e., the causal graph of the abstract task has separated components. In that case, the symbolic abstract search complexity depends on the size of the components instead of the number of variables in the abstraction. Introducing the state-invariant constraints of the original state space breaks the independence of the variables, significantly increasing the complexity of the abstract search.

6.5. Symbolic versus Explicit-State Search

In this section, we compare symbolic and explicit-state search algorithms, emphasizing the relevance of our improvements for symbolic search.

Symbolic vs explicit uniform-cost search. The left part of Table 6 shows the benefits of the symbolic representation in blind search, especially when using our improvements. For forward search we used the A* implementation of the Fast Downward planning system [47] with the *blind* heuristic (*FD*). For backward search we used the *FDR* planner [6], implemented on top of Fast Downward. *FDR* runs partial-state regression with state-invariant pruning.

In forward uniform-cost search, the symbolic variant is better in most domains, solving 154 more problems even without our improvements. With cGamer, the advantage gets increased to 208 problems, making symbolic search at least as good as the explicit version in all domains. In backward search, the performance of the symbolic baseline, Gamer, is similar to that of *FDR*. Gamer has better total coverage but *FDR* has better coverage score. Moreover *FDR* is faster when the solving times are considered. Nonetheless, note that *FDR* already uses state-invariant pruning. When our symbolic search enhancements are used, we consistently beat *FDR* in all domains except Mystery. In total, the advantage of symbolic uniform-cost search is notable, obtaining better results than the explicit version in all but 6 domains in which they are tied. Forward search is usually better than backward search in both variants, though its good results in some domains suggest that backward search should not be abandoned.

Symbolic versus heuristic state-of-the-art planners. To compare against state-of-the-art planners, we use A* with LM-cut [5] and two configurations of the merge-and-shrink heuristic [4]. Fast Downward Stone Soup [63], the winner of the optimal-track of IPC-2011 [64], was a portfolio sequentially running A* with these three heuristics. To compare against the best portfolio of that kind, we use an optimistic overapproximation of the best possible results using the three planners (*best*), which considers a problem solved if it was solved by any of the individual planners. Gamer and cGamer use bidirectional uniform-cost search and BDDA* with symbolic PDBs.

The right part of Table 6 shows that symbolic algorithms outperform explicit-state planners across a variety of domains. Regarding total performance, explicit-state search planners beat both variants of

	Forward			Backward			A*				Gamer		cGamer	
	FD	Gam	cGam	*FDR*	Gam	cGam	M&S^b	M&S^g	LMcut	*best*	Bidir	BDDA*	Bidir	BDDA*
Airport (50)	23	23	24	22	11	23	23	23	**29**	**29**	23	20	27	21
Barman (20)	4	8	8	4	0	10	4	4	4	4	8	4	**12**	8
Blocksworld (35)	18	21	21	18	12	23	20	28	28	28	21	27	**33**	26
Depot (22)	4	4	5	2	2	4	6	6	7	7	5	7	**8**	**8**
Driverlog (20)	7	11	11	6	7	9	13	12	13	13	12	**14**	**14**	**14**
Elevators08 (30)	13	19	19	4	6	10	14	1	22	22	24	20	**25**	18
Elevators11 (20)	11	16	16	2	4	8	12	0	18	18	**19**	17	**19**	15
Floortile11 (20)	8	9	**14**	**14**	8	**14**	8	4	**14**	**14**	10	12	**14**	**14**
Freecell (80)	16	14	20	8	6	20	4	19	15	19	14	20	**27**	25
Grid (5)	1	2	2	0	1	1	**3**	2	2	**3**	2	2	2	**3**
Gripper (20)	8	**20**	**20**	8	11	**20**	**20**	8	7	**20**	**20**	**20**	**20**	18
Logistics 00 (28)	10	16	16	10	16	16	20	16	20	20	20	18	20	**22**
Logistics 98 (35)	2	5	5	2	2	3	5	4	**6**	**6**	5	**6**	5	**6**
Miconic (150)	50	82	96	65	85	114	77	53	**141**	**141**	86	79	113	108
MPrime (35)	19	23	26	10	9	10	12	23	23	24	19	25	21	**28**
Mystery (30)	15	15	15	11	7	8	8	**17**	**17**	**17**	14	**17**	13	**17**
NoMystery11 (20)	8	13	12	9	9	12	**19**	14	14	**19**	14	14	16	15
Openstacks08 (30)	21	**30**	**30**	8	20	**30**	21	9	21	21	**30**	29	**30**	26
Openstacks11 (20)	16	**20**	**20**	3	15	**20**	16	4	16	16	**20**	19	**20**	19
Openstacks06 (30)	7	11	**20**	7	7	16	7	7	7	7	11	7	**20**	15
PARC-Printer08 (30)	20	22	**24**	20	15	23	20	20	22	23	21	13	22	19
PARC-Printer11 (20)	15	17	**18**	15	11	**18**	15	15	17	**18**	16	9	**18**	15
Parking11 (20)	0	0	0	0	0	0	0	0	**3**	**3**	0	1	1	0
Pathways-noneg (30)	4	**5**	**5**	4	4	4	4	4	**5**	**5**	**5**	**5**	**5**	**5**
Peg-Solitaire08 (30)	27	29	29	11	12	24	29	6	29	29	29	28	**30**	29
Peg-Solitaire11 (20)	17	19	19	3	2	14	19	0	19	19	19	18	**20**	19
Pipesworld-nt (50)	14	15	17	8	4	9	9	16	**18**	**18**	15	15	17	16
Pipesworld-t (50)	11	15	17	2	4	6	8	17	12	17	16	16	17	**18**
PSR-small (50)	49	**50**	**50**	46	49	**50**	**50**	**50**	49	**50**	**50**	**50**	**50**	**50**
Rovers (40)	5	**14**	13	6	11	12	8	6	7	8	**14**	13	**14**	13
Satellite (36)	5	7	7	6	7	10	7	6	7	7	7	7	**11**	10
Scanalyzer08 (30)	12	12	12	12	9	12	14	9	15	**17**	12	10	12	12
Scanalyzer11 (20)	9	9	9	9	6	9	11	6	12	**14**	9	7	9	9
Sokoban08 (30)	**27**	28	28	17	4	25	29	3	**30**	**30**	28	**30**	28	**30**
Sokoban11 (20)	**20**	**20**	**20**	14	1	18	**20**	1	**20**	**20**	**20**	**20**	**20**	19
Tidybot11 (20)	12	12	16	3	1	9	1	13	**17**	**17**	12	15	**17**	**17**
TPP (30)	6	**8**	**8**	5	**8**	**8**	7	6	7	7	**8**	**8**	**8**	**8**
Transport08 (30)	11	11	12	8	8	9	11	11	11	11	12	11	**14**	11
Transport11 (20)	6	6	7	3	3	4	7	6	6	7	8	6	**10**	6
Trucks (30)	7	10	11	9	6	10	8	8	10	10	10	11	**12**	**12**
VisitAll (20)	9	9	9	9	9	9	9	**16**	10	**16**	12	11	12	11
Woodworking08 (30)	9	20	21	9	20	25	13	14	22	22	22	22	**26**	25
Woodworking11 (20)	4	14	15	4	14	**19**	8	9	15	15	16	16	**19**	18
Zenotravel (20)	8	8	9	7	7	8	12	10	**13**	**13**	10	11	11	12
Total Cov (1396)	568	722	776	443	453	706	631	506	800	844	748	730	**862**	810
Score Cov (36)	11.67	15.35	16.56	9.18	9.19	14.42	13.70	12.23	16.40	18.09	16.00	16.25	**18.62**	17.87

Table 6: Coverage of symbolic (Gamer and cGamer) vs. explicit search (*FD*, *FDR*, and A*). Gamer is the baseline symbolic planner and cGamer uses our improvements. They use bidirectional uniform-cost search (Bidirectional) and BDDA* with symbolic PDBs. Explicit A* planners use M&S with full (M&S^b) and greedy (M&S^g) bisimulation and LM-cut. *best* gets the best results of the three A*-based planners in each problem.

Gamer, mainly due to the accuracy of the LM-cut heuristic. However, the total performance of cGamer is superior to explicit-state search planners in this set of benchmarks. The case of bidirectional uniform-cost search is clear, beating even the portfolio approaches. BDDA* is also superior when considering standalone planners, but it is still behind the optimistic results of the explicit-state search portfolio. The results highlight the importance of symbolic search not only in terms of average performance but also in a per-domain basis. Symbolic planners are required to obtain the best results in 23 domains. Comparatively, heuristic planners only get better results than symbolic search algorithms in 11 cases.

Analysis of coverage with different runtime limits. Figure 11 shows the cumulative number of instances solved by every planner at every second, in logarithmic scale. In general, the final score of the planners after 30 minutes is representative of the results with other time limits. However, there are some remarkable conclusions related to the behavior of these planners.

Blind search starts faster than other planners, but it converges quickly because the memory limit is exceeded after approximately 5 minutes.Symbolic unidirectional search is not only faster than explicit-state blind search (solving more problems in the first seconds), but also has a much better convergence rate due to the memory savings of BDDs.

Our symbolic search enhancements increase the score of Gamer uniformly over time. This speedup makes cGamer-fw faster than Gamer, even though it does not perform bidirectional search. However, the lack of regression search causes a faster convergence and it is not clear whether cGamer-fw would beat Gamer for larger timeouts.

Well-informed heuristics, such as LM-cut, increase the performance of explicit-state search and the total performance is slightly better to that of symbolic forward search and similar to symbolic BDDA* with PDB

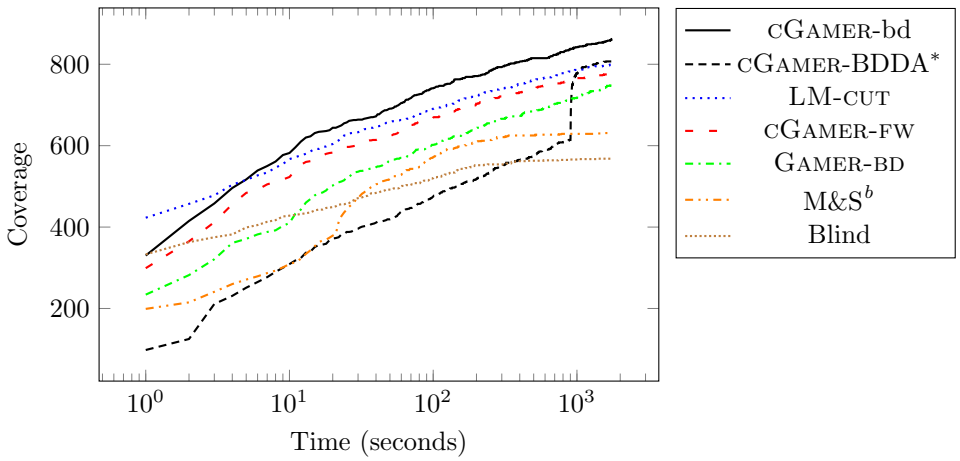


Figure 11: Cumulative coverage of symbolic and explicit-state search planners. Total coverage of each planner at each time in logscale. The curve of cGamer-BDDA* reflects that it uses up to 900 seconds for precomputing the heuristic before starting the search.

heuristics. However, symbolic bidirectional search with our improvements is able to beat LM-cut except for very short timeouts, probably due to the time spent in BDD initialization and the instances for which LM-cut is perfectly informed.

IPC-2014 analysis. The results of the last IPC in 2014[6] confirm that symbolic bidirectional search is a leading approach for cost-optimal planning. In the sequential-optimal track, the symbolic bidirectional uniform-cost version of cGamer was the runner-up, beating other symbolic search, explicit-state search, and portfolio planners. The winner of the sequential-optimal track, SymBA*, is another symbolic planner that uses the improvements presented in this paper, combining bidirectional search with abstraction heuristics [65, 66]. In total, symbolic planners outperformed explicit-state search planners in 10 out of 14 domains. From the seven domains that had already been used in previous IPCs, symbolic search planners are better in 5 of them. In the new domains, we see a similar trend with symbolic planners being better in 5 out of 7 domains. This supports the idea that our analysis is not biased by the benchmark set used in this paper and complements the results of our experiments to affirm that symbolic search planners are once again competitive with the state of the art [67].

Further experiments on SymBA* have shown that, while abstraction heuristics may improve the performance of bidirectional search on particular domains, symbolic bidirectional uniform-cost search is yet a state-of-the-art planner [66]. Thus, the race for the state of the art in cost-optimal planning is not over. Currently, both symbolic approaches (blind and guided) are about on par on the existing benchmark set. If abstraction heuristics can be improved slightly (maybe with some better selection policy) the table may turn in favor to heuristic search approaches, but at the moment to the best of the authors' knowledge bidirectional blind search as presented in this article is competitive with bidirectional heuristic search approaches, even though it does not make use of any heuristic and hence has the advantage of being simpler.

7. Discussion

Heuristic and symbolic search are two leading methods for cost-optimal planning. There is no doubt that symbolic search has the effectiveness to explore large state sets, while the explicit-state heuristics are often more informed. While the result of IPC 2011 suggested a clear advantage for heuristic explicit-state

[6]2014 International Planning Competition webpage: `https://helios.hud.ac.uk/scommv/IPC-14/`

search, in this paper we introduced two orthogonal improvements to symbolic search planners that change this picture.

On the one hand, we have presented and analyzed image computation methods for symbolic search planning. Our starting point was the state-of-the-art symbolic search planner GAMER that computes the image using a transition relation for each planning operator. We proposed three different alternatives for image computation:

1. $\mathbf{TR}^{\mathbf{1+}}$: Instead of representing each planning operator by means of a single BDD, represent the preconditions and effects independently. That way, no auxiliary set of variables is needed in order to represent the successor states.
2. **CT**: Apply the preconditions of operators at the same time using the conjunction tree, an approach similar to the one used in explicit-state search planning.
3. **UT**: Aggregate several operators in a single transition relation, controlling the memory used and avoiding to exceed a given memory limit.

Our analysis has shown that the three approaches improve the previous image computation of GAMER in terms of coverage and time score. Moreover, the experimental results show a dominance across most domains, so that the new image computation methods can replace the previous image computation of GAMER. We observed that it is better to avoid the usage of an auxiliary set of variables, representing conditions and effects separately. Unfortunately, this cannot be done when more than one operator is represented in the same TR, so the best image computation techniques still need the auxiliary set of variables to represent the relation between the precondition and the effect of the operators. In terms of time-efficiency, it is best to use a single monolithic TR to describe all the operators. However, as already noticed by previous works, the TR that describes all the planning operators can easily exceed the available memory (that was the main reason to split it into one TR per operator). Thus, one of the main conclusions of our analysis is that for efficient image computation, it is best to represent the planning operators with as few TRs as possible, using a monolithic TR for all operators with the same cost whenever possible. When constructing a monolithic TR within a reasonable memory limit is impossible, a disjunctive partitioning of the TR is necessary. Previous work in model checking had already suggested similar partitionings. Here, we proposed an algorithm to derive the disjunctive partitioning automatically, according to several heuristic criteria. Our analysis shows the importance of keeping the size of the TRs as balanced as possible, so we recommend the $\mathbf{UT}^{\mathbf{DT}}_{\mathbf{100k}}$ method for future work.

On the other hand, we used state-invariant constraints in order to prune symbolic search. We interpret state invariants as properties that must hold in any state that is part of a plan for the task, so that they can be used to prune forward and backward search. Our main contribution is to study different methods to encode and use these constraints:

- $\boldsymbol{\mathcal{M}_{BDD}}$: Encodes the constraints as BDDs and, during the search, uses a conjunction to compute the subset of valid states.
- **e-del**: Encodes the constraints as additional conditions in the operators, i. e., in the TRs. We proved which subset of constraints must be encoded in each TR to ensure that no invalid states are generated during the search.

Empirical results show that state-invariant pruning is tremendously effective in symbolic search, especially in regression. According to our evaluation, **e-deletion** is recommended for taking advantage of constraints in symbolic search. This has impact specially in regression search where there are more state-invariant constraints, and may also be relevant for encoding constraints from other sources, such as novel dead-end detection methods [68].

Our enhanced version of GAMER, CGAMER, not only outperforms GAMER in most domains but also has superior performance compared to explicit-state search approaches. The good results of symbolic blind search are especially surprising as after many years of research of finding refined heuristics for AI planning, this form of blind search still outperforms heuristic search planners on many domains, while being close on most others. The impact of the directionality of the domains [69] explains why this configuration fares

so well, as a simple alternating strategy allows choosing the direction in which the problem may be more easily solved, compensating the use of heuristics. This highlights the importance of regression. Progression is usually believed to be more robust than regression for search-based planning. While the results on unidirectional search somewhat confirm the superiority of forward search, our improvements help to reduce the gap in symbolic search, beating the results of partial-state backward search. An important conclusion to be drawn from our results is that symbolic backward search is a useful technique that can be complementary to standard heuristic search techniques.

In the last IPC in 2014 cGamer made the second place, outperforming other symbolic search based planners as well as explicit-state search based planners. The winner of the competition, SymBA* was also built upon the ideas of cGamer, using the enhancements proposed in this paper with a combination of symbolic bidirectional search and abstraction heuristics. This reflects the relevance of our enhancements, unifying operators based on the balanced disjunction tree up to a maximum memory limit of 100 000 nodes ($\mathbf{UT^{DT}_{100k}}$) for image computation and **e-deletion** for state-invariant pruning, to make symbolic search a state-of-the-art technique for cost-optimal planning.

A more detailed analysis of the impact of abstraction heuristics in bidirectional search shows that, while abstraction heuristics may be helpful in particular domains, the overall performance of SymBA* is similar to that of symbolic bidirectional uniform-cost search [66]. Bidirectional heuristic search has seen some possibilities [70] and limitations [71]. Insights have lead to variants like MM [72, 73] with a set-based parallel implementation on external memory [74]. Given the success of BDD-based bidirectional search documented in this work, the implementation of MM with BDDs and its application to AI planning domains is one tempting option for future research.

8. Acknowledgments

We would like to thank Carlos Linares López, Daniel Borrajo, Malte Helmert, Anders Jonsson, and the AIJ anonymous reviewers for their useful advice to improve this work. This work has been supported by a grant of Universidad Carlos III de Madrid, the DFG excellence cluster EXC 284 "Multimodal Computing and Interaction", and a FPI grant associated to MICINN project TIN2008-06701-C03-03.

Bibliography

[1] E. W. Dijkstra, A note on two problems in connexion with graphs, Numerische Mathematik 1 (1959) 269–271.
[2] A. Felner, Position paper: Dijkstra's algorithm versus uniform cost search or a case against Dijkstra's algorithm, in: D. Borrajo, M. Likhachev, C. Linares López (Eds.), Proceedings of the Symposium on Combinatorial Search (SoCS), AAAI Press, 2011, pp. 47–51.
[3] P. E. Hart, N. J. Nilsson, B. Raphael, A formal basis for the heuristic determination of minimum cost paths, IEEE Transactions on Systems Science and Cybernetics 4 (1968) 100–107.
[4] M. Helmert, P. Haslum, J. Hoffmann, R. Nissim, Merge-and-shrink abstraction: A method for generating lower bounds in factored state spaces, Journal of the ACM 61 (2014) 16:1–16:63.
[5] M. Helmert, C. Domshlak, Landmarks, critical paths and abstractions: What's the difference anyway?, in: A. Gerevini, A. E. Howe, A. Cesta, I. Refanidis (Eds.), Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS), AAAI Press, 2009, pp. 162–169.
[6] V. Alcázar, D. Borrajo, S. Fernández, R. Fuentetaja, Revisiting regression in planning, in: F. Rossi (Ed.), Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), IJCAI/AAAI, 2013, pp. 2254–2260.
[7] B. Bonet, H. Geffner, Planning as heuristic search, Artificial Intelligence Journal 129 (2001) 5–33.
[8] R. E. Bryant, Graph-based algorithms for boolean function manipulation, IEEE Transactions on Computers 35 (1986) 677–691.
[9] S. Edelkamp, F. Reffel, OBDDs in heuristic search, in: O. Herzog, A. Günter (Eds.), Proceedings of the German Conference on Artificial Intelligence (KI), volume 1504 of *Lecture Notes in Computer Science*, Springer, 1998, pp. 81–92.
[10] K. L. McMillan, Symbolic model checking, Kluwer Academic publishers, 1993. doi:10.1007/978-1-4615-3190-6.
[11] A. Cimatti, F. Giunchiglia, E. Giunchiglia, P. Traverso, Planning via model checking: A decision procedure for AR, in: S. Steel, R. Alami (Eds.), Proceedings of the European Conference on Planning (ECP), volume 1348 of *Lecture Notes in Computer Science*, Springer, 1997, pp. 130–142.
[12] S. Edelkamp, M. Helmert, MIPS: The model-checking integrated planning system, AI Magazine 22 (2001) 67–72.
[13] R. M. Jensen, M. M. Veloso, R. E. Bryant, State-set branching: Leveraging BDDs for heuristic search, Artificial Intelligence Journal 172 (2008) 103–139.

[14] S. Edelkamp, P. Kissmann, Optimal symbolic planning with action costs and preferences, in: C. Boutilier (Ed.), Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), 2009, pp. 1690–1695.
[15] P. Kissmann, S. Edelkamp, Improving cost-optimal domain-independent symbolic planning, in: W. Burgard, D. Roth (Eds.), Proceedings of the AAAI Conference on Artificial Intelligence (AAAI), AAAI Press, 2011, pp. 992–997.
[16] Á. Torralba, S. Edelkamp, P. Kissmann, Transition trees for cost-optimal symbolic planning, in: D. Borrajo, S. Kambhampati, A. Oddi, S. Fratini (Eds.), Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS), AAAI Press, 2013, pp. 206–214.
[17] Á. Torralba, V. Alcázar, Constrained symbolic search: On mutexes, BDD minimization and more, in: M. Helmert, G. Röger (Eds.), Proceedings of the Symposium on Combinatorial Search (SoCS), AAAI Press, 2013, pp. 175–183.
[18] W. Hung, Exploiting Symmetry for Formal Verification, Master's thesis, University of Texas at Austin, 1997.
[19] S. Edelkamp, P. Kissmann, Limits and possibilities of BDDs in state space search, in: D. Fox, C. P. Gomes (Eds.), Proceedings of the AAAI Conference on Artificial Intelligence (AAAI), AAAI Press, 2008, pp. 1452–1453.
[20] S. Edelkamp, P. Kissmann, On the complexity of BDDs for state space search: A case study in connect four, in: W. Burgard, D. Roth (Eds.), Proceedings of the AAAI Conference on Artificial Intelligence (AAAI), AAAI Press, 2011, pp. 18–23.
[21] R. E. Bryant, Symbolic boolean manipulation with ordered binary-decision diagrams, ACM Computing Surveys 24 (1992) 293–318.
[22] P. Kissmann, J. Hoffmann, What's in it for my BDD? On causal graphs and variable orders in planning, in: D. Borrajo, S. Kambhampati, A. Oddi, S. Fratini (Eds.), Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS), AAAI Press, 2013, pp. 327–331.
[23] P. Kissmann, J. Hoffmann, BDD ordering heuristics for classical planning, Journal of Artificial Intelligence Research (JAIR) 51 (2014) 779–804.
[24] P. Kissmann, S. Edelkamp, J. Hoffmann, Gamer and dynamic-gamer – symbolic search at IPC 2014, in: Proceedings of the International Planning Competition (IPC), 2014, pp. 77–84.
[25] R. Yoshinaka, J. Kawahara, S. Denzumi, H. Arimura, S. Minato, Counterexamples to the long-standing conjecture on the complexity of BDD binary operations, Information Processing Letters 112 (2012) 636–640.
[26] B. Bollig, A simpler counterexample to a long-standing conjecture on the complexity of Bryant's apply algorithm, Information Processing Letters 114 (2014) 124–129.
[27] T. A. J. Nicholson, Finding the shortest route between two points in a network, The Computer Journal 9 (1966) 275–280.
[28] A. V. Goldberg, R. F. F. Werneck, Computing point-to-point shortest paths from external memory, in: C. Demetrescu, R. Sedgewick, R. Tamassia (Eds.), Proceedings of the Workshop on Algorithm Engineering and Experiments and the Workshop on Analytic Algorithmics and Combinatorics (ALENEX/ANALCO), SIAM, 2005, pp. 26–40.
[29] I. Pohl, Bi-directional and heuristic search in path problems, Technical Report, Department of Computer Science, Stanford University, 1969.
[30] E. A. Hansen, R. Zhou, Z. Feng, Symbolic heuristic search using decision diagrams, in: S. Koenig, R. C. Holte (Eds.), Proceedings of the Symposium on Abstraction, Reformulation and Approximation (SARA), volume 2371 of *Lecture Notes in Computer Science*, Springer, 2002, pp. 83–98.
[31] R. M. Jensen, R. E. Bryant, M. M. Veloso, SetA*: An efficient BDD-based heuristic search algorithm, in: R. Dechter, R. S. Sutton (Eds.), Proceedings of the AAAI Conference on Artificial Intelligence (AAAI), AAAI Press / The MIT Press, 2002, pp. 668–673.
[32] P. Kissmann, Symbolic Search in Planning and General Game Playing, Ph.D. thesis, Universität Bremen, 2012.
[33] S. Edelkamp, P. Kissmann, Á. Torralba, Symbolic A* search with pattern databases and the merge-and-shrink abstraction, in: L. De Raedt, C. Bessière, D. Dubois, P. Doherty, P. Frasconi, F. Heintz, P. J. F. Lucas (Eds.), Proceedings of the European Conference on Artificial Intelligence (ECAI), volume 242 of *Frontiers in Artificial Intelligence and Applications*, IOS Press, 2012, pp. 306–311.
[34] Á. Torralba, Symbolic Search and Abstraction Heuristics for Cost-Optimal Planning, Ph.D. thesis, Universidad Carlos III de Madrid, 2015.
[35] J. C. Culberson, J. Schaeffer, Pattern databases, Computational Intelligence 14 (1998) 318–334.
[36] S. Edelkamp, S. Schrödl, Heuristic Search – Theory and Applications, Academic Press, 2012.
[37] S. Edelkamp, Planning with pattern databases, in: A. Cesta, D. Borrajo (Eds.), Proceedings of the European Conference on Planning (ECP), Lecture Notes in Computer Science, Springer, 2001, pp. 13–34. Volume lost due to September 11th.
[38] K. Anderson, R. Holte, J. Schaeffer, Partial pattern databases, in: I. Miguel, W. Ruml (Eds.), Proceedings of the Symposium on Abstraction, Reformulation and Approximation (SARA), volume 4612 of *Lecture Notes in Computer Science*, Springer, 2007, pp. 20–34.
[39] S. Edelkamp, Symbolic pattern databases in heuristic search planning, in: M. Ghallab, J. Hertzberg, P. Traverso (Eds.), Proceedings of the Conference on Artificial Intelligence Planning Systems (AIPS), AAAI Press, 2002, pp. 274–283.
[40] P. Haslum, A. Botea, M. Helmert, B. Bonet, S. Koenig, Domain-independent construction of pattern database heuristics for cost-optimal planning, in: Proceedings of the AAAI Conference on Artificial Intelligence (AAAI), AAAI Press, 2007, pp. 1007–1012.
[41] J. R. Burch, E. M. Clarke, D. E. Long, K. L. McMillan, D. L. Dill, Symbolic model checking for sequential circuit verification, IEEE Transactions on CAD of Integrated Circuits and Systems 13 (1994) 401–424.
[42] J. R. Burch, E. M. Clarke, D. E. Long, Symbolic model checking with partitioned transistion relations, in: Proceedings of the International Conference on Very Large Scale Integration (VLSI), 1991, pp. 49–58.
[43] J. R. Burch, E. M. Clarke, D. E. Long, Representing circuits more efficiently in symbolic model checking, in: Proceedings

of the Design Automation Conference (DAC), 1991, pp. 403–407.
[44] P. Chauhan, E. M. Clarke, S. Jha, J. H. Kukula, T. R. Shiple, H. Veith, D. Wang, Non-linear quantification scheduling in image computation, in: Proceedings of the International Conference on Computer-Aided Design (ICCAD), IEEE Press, 2001, pp. 293–298.
[45] I.-H. Moon, J. H. Kukula, K. Ravi, F. Somenzi, To split or to conjoin: the question in image computation, in: Proceedings of the Design Automation Conference (DAC), 2000, pp. 23–28.
[46] R. M. Jensen, E. A. Hansen, S. Richards, R. Zhou, Memory-efficient symbolic heuristic search, in: D. Long, S. F. Smith, D. Borrajo, L. McCluskey (Eds.), Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS), 2006, pp. 304–313.
[47] M. Helmert, The Fast Downward planning system, Journal of Artificial Intelligence Research (JAIR) 26 (2006) 191–246.
[48] C. Forgy, RETE: A fast algorithm for the many patterns/many objects match problem, Artificial Intelligence Journal 19 (1982) 17–37.
[49] Á. Torralba, C. Linares López, D. Borrajo, Symbolic merge-and-shrink for cost-optimal planning, in: F. Rossi (Ed.), Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), IJCAI/AAAI, 2013, pp. 2394–2400.
[50] S. Zilles, R. C. Holte, The computational complexity of avoiding spurious states in state space abstraction, Artificial Intelligence Journal 174 (2010) 1072–1092.
[51] P. Haslum, B. Bonet, H. Geffner, New admissible heuristics for domain-independent planning, in: M. M. Veloso, S. Kambhampati (Eds.), Proceedings of the AAAI Conference on Artificial Intelligence (AAAI), AAAI Press / The MIT Press, 2005, pp. 1163–1168.
[52] V. Alcázar, Generation and Exploitation of Intermediate Goals in Automated Planning, Ph.D. thesis, Universidad Carlos III de Madrid, 2014.
[53] M. Helmert, Concise finite-domain representations for PDDL planning tasks, Artificial Intelligence Journal 173 (2009) 503–535.
[54] P. Haslum, H. Geffner, Admissible heuristics for optimal planning, in: S. Chien, S. Kambhampati, C. A. Knoblock (Eds.), Proceedings of the Conference on Artificial Intelligence Planning Systems (AIPS), AAAI, 2000, pp. 140–149.
[55] P. Haslum, $h^m(p) = h^1(p^m)$: Alternative characterisations of the generalisation from $h^{\max}$ to h^m, in: A. Gerevini, A. E. Howe, A. Cesta, I. Refanidis (Eds.), Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS), AAAI Press, 2009, pp. 354–357.
[56] P. Haslum, Additive and reversed relaxed reachability heuristics revisited, in: Proceedings of the International Planning Competition (IPC), 2008.
[57] V. Alcázar, Á. Torralba, A reminder about the importance of computing and exploiting invariants in planning, in: R. Brafman, C. Domshlak, P. Haslum, S. Zilberstein (Eds.), Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS), AAAI Press, 2015, pp. 2–6.
[58] O. Coudert, J. C. Madre, A unified framework for the formal verification of sequential circuits, in: Proceedings of the International Conference on Computer-Aided Design (ICCAD), 1990, pp. 126–129.
[59] K. L. McMillan, A conjunctively decomposed boolean representation for symbolic model checking, in: R. Alur, T. A. Henzinger (Eds.), Computer Aided Verification (CAV), volume 1102 of *Lecture Notes in Computer Science*, Springer, 1996, pp. 13–25.
[60] Y. Hong, P. A. Beerel, J. R. Burch, K. L. McMillan, Safe BDD minimization using don't cares, in: Proceedings of the Design Automation Conference (DAC), 1997, pp. 208–213.
[61] V. Vidal, H. Geffner, Solving simple planning problems with more inference and no search, in: P. van Beek (Ed.), Principles and Practice of Constraint Programming (CP), volume 3709 of *Lecture Notes in Computer Science*, Springer, 2005, pp. 682–696.
[62] F. Somenzi, CUDD: CU decision diagram package release 2.5.0, 2012.
[63] M. Helmert, G. Röger, J. Seipp, E. Karpas, J. Hoffmann, E. Keyder, R. Nissim, S. Richter, M. Westphal, Fast downward stone soup, in: Proceedings of the International Planning Competition (IPC), 2011.
[64] C. Linares López, S. J. Celorrio, A. G. Olaya, The deterministic part of the seventh international planning competition, Artificial Intelligence Journal 223 (2015) 82–119.
[65] Á. Torralba, V. Alcázar, D. Borrajo, P. Kissmann, S. Edelkamp, SymBA*: A symbolic bidirectional A* planner, in: Proceedings of the International Planning Competition (IPC), 2014, pp. 105–109.
[66] Á. Torralba, C. Linares López, D. Borrajo, Abstraction heuristics for symbolic bidirectional search, in: Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), AAAI Press, 2016, pp. 3272–3278.
[67] S. Edelkamp, P. Kissmann, Á. Torralba, BDDs strike back (in AI planning), in: B. Bonet, S. Koenig (Eds.), Proceedings of the AAAI Conference on Artificial Intelligence (AAAI), AAAI Press, 2015, pp. 4320–4321.
[68] M. Steinmetz, J. Hoffmann, Towards clause-learning state space search: Learning to recognize dead-ends, in: D. Schuurmans, M. P. Wellman (Eds.), Proceedings of the AAAI Conference on Artificial Intelligence (AAAI), AAAI Press, 2016, pp. 760–768.
[69] B. Massey, Directions In Planning: Understanding The Flow Of Time In Planning, Ph.D. thesis, Computational Intelligence Research Laboratory, University of Oregon, 1999.
[70] M. N. Rice, V. J. Tsotras, Bidirectional A* search with additive approximation bounds, in: D. Borrajo, A. Felner, R. E. Korf, M. Likhachev, C. Linares López, W. Ruml, N. R. Sturtevant (Eds.), Proceedings of the Symposium on Combinatorial Search (SoCS), AAAI Press, 2012.
[71] J. K. Barker, R. E. Korf, Limitations of front-to-end bidirectional heuristic search, in: B. Bonet, S. Koenig (Eds.), Proceedings of the AAAI Conference on Artificial Intelligence (AAAI), AAAI Press, 2015, pp. 1086–1092.

[72] R. C. Holte, A. Felner, G. Sharon, N. R. Sturtevant, Bidirectional search that is guaranteed to meet in the middle, in: D. Schuurmans, M. P. Wellman (Eds.), Proceedings of the AAAI Conference on Artificial Intelligence (AAAI), AAAI Press, 2016, pp. 3411–3417.

[73] G. Sharon, R. C. Holte, A. Felner, N. R. Sturtevant, Extended abstract: An improved priority function for bidirectional heuristic search, in: J. A. Baier, A. Botea (Eds.), Proceedings of the Ninth Annual Symposium on Combinatorial Search, SOCS 2016, Tarrytown, NY, USA, July 6-8, 2016., AAAI Press, 2016, pp. 139–140.

[74] N. R. Sturtevant, J. Chen, External memory bidirectional search, in: S. Kambhampati (Ed.), Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016, IJCAI/AAAI Press, 2016, pp. 676–682.