Symbolic Perimeter Abstraction Heuristics for Cost-Optimal Planning

Álvaro Torralba[a,*], Carlos Linares López[b], Daniel Borrajo[b]

[a] *Saarland University, Saarland Informatics Campus, Saarbrücken, Germany*
[b] *Departamento de Informática, Universidad Carlos III de Madrid. Leganés (Madrid) Spain*

Abstract

In the context of heuristic search within automated planning, abstraction heuristics map the problem into an abstract instance and use the optimal solution cost in the abstract state space as an estimate for the real solution cost. Their flexibility in choosing different abstract mappings makes abstractions a powerful tool to obtain domain-independent heuristics. Different types of abstraction heuristics exist depending on how the mapping is defined, like Pattern Databases (PDBs) or Merge-and-Shrink (M&S).

In this paper, we consider two variants of PDBs, symbolic and perimeter PDBs, combining them to take advantage of their synergy. Symbolic PDBs use decision diagrams in order to efficiently traverse the abstract state space. Perimeter PDBs derive more informed estimates by first constructing a perimeter around the goal and then using it to initialize the abstract search. We generalize this idea by considering a hierarchy of abstractions. Our algorithm starts by constructing a symbolic perimeter around the goal and, whenever continuing the search becomes unfeasible, it switches to a more abstracted state space. By delaying the use of an abstraction, the algorithm derives heuristics as informed as possible. Moreover, we prove that M&S abstractions with a linear merge strategy can be efficiently represented as decision diagrams, enabling the use of symbolic search with M&S abstractions as well as with PDBs. Our experimental evaluation shows that symbolic perimeter abstractions are competitive with other state-of-the-art heuristics.

Keywords: Automated planning, Cost-optimal planning, Planning systems, Symbolic search, Abstraction heuristics

1. Introduction

Classical planning deals with the generation of plans, i.e., sequences of actions that an agent can execute to achieve a set of goals. A planning task consists of a set of variables that describe the current state of the world, a set of actions that the agent may perform to modify the value of such variables, the initial state, and the agent's goals. In cost-optimal planning, we are interested in finding a plan whose cost is minimal for any planning task given as input.

State-space search with A* [1] is a prominent approach for cost-optimal planning [2]. A* enumerates states reachable from the initial state until a plan is found and proven to be optimal. The order in which A* considers the states depends on a heuristic function that estimates the cost from each state to the goal, helping to focus the search effort on those states that are closer to the goal. Even though in most cases the size of the reachable state space grows exponentially in the size of the planning task, this approach has been very successful thanks to a broad research on domain-independent admissible heuristics [3, 4, 5] and pruning methods [6, 7, 8] that help to reduce the number of states that must be visited by A*.

Abstraction heuristics are a class of heuristics that map the state space to a smaller one, and use the cost of the optimal solution in the abstract state space as an admissible estimate of the cost in the original task. The cost from every abstract state to the goal is precomputed prior to the search by traversing the entire abstract state space with a backward uniform-cost search starting from the abstract goal. These costs are stored in a look-up table, so that the

*Corresponding author

Email addresses: torralba@cs.uni-saarland.de (Álvaro Torralba), carlos.linares@uc3m.es (Carlos Linares López), dborrajo@ia.uc3m.es (Daniel Borrajo)

evaluation of the heuristic during the search is very efficient. However, this requires the abstract state space to be small enough so that the look-up table fits into memory.

The key is how to derive "*good*" abstractions so that the abstract instance can be solved efficiently while providing useful estimates. For that reason there exist different types of abstraction heuristics. Pattern Databases (PDBs) select a subset of variables of the problem and ignore the rest [9, 10]. This is a simple way of defining abstractions that is able to produce good heuristics across many domains. However, a limitation of PDBs is that they always completely ignore at least one variable, which may render the heuristic uninformative in some domains. As an alternative, Merge-and-Shrink [11, 12] (M&S) is an algorithm to derive abstractions that take into account all variables.

Perimeter heuristics aim to obtain more accurate heuristics by constructing a perimeter around the goal states in the original state space. The perimeter corresponds to the set of states from which the goals can be achieved with a given cost. Perimeter heuristics estimate the cost from a state to the goal as the cost from the state to any state in the perimeter, plus the cost from the perimeter to the goal. The main drawback is that computing the heuristic becomes more expensive, since it requires to estimate the minimum cost to any state in the perimeter. Abstraction heuristics are a good match for this task, because one can estimate such minimum cost directly without any overhead; it just suffices to initialize the abstract state space exploration that precomputes all costs with all the abstract states in the perimeter, instead of only with the abstract states representing the goals.

We generalize the idea of perimeter PDBs by considering a hierarchy of PDBs, instead of only one. This hierarchy contains abstractions of different sizes, ranging from trivial PDBs with only one variable to the original state space (with all variables). Instead of traversing all these abstract state spaces completely, they are only partially explored. Our algorithm starts constructing a perimeter in the original state space and, whenever the search becomes unfeasible (according to some predefined time and memory limits), it switches to a coarser abstraction by abstracting away a new variable. That way, the search is conducted in an abstract state space as close to the original task as possible, deriving more informative heuristics.

Applying perimeter PDBs in domain-independent planning poses some challenges due to the difficulties of regression in large state spaces with multiple goal states [13]. We show how such challenges can be successfully addressed by using symbolic PDBs, which use symbolic search to traverse the abstract state space [14]. Symbolic search uses Binary Decision Diagrams [15] (BDDs) to compactly represent sets of states, often with an exponential gain in memory with respect to their explicit enumeration. Thanks to this, symbolic search has been shown to be a very efficient method for exhaustive exploration of state spaces, especially after recent improvements [16, 17]. Another advantage of representing the heuristic function with decision diagrams is that, by avoiding a lookup table with an entry per abstract state, we can handle larger abstract state spaces. This is important for our algorithm, which considers arbitrarily large abstractions whose entire state space does not fit in memory. To exploit the synergy between symbolic and perimeter PDBs, we define the BDD operations that can be used to effectively generate the set of abstract states that correspond to states in the perimeter without explicitly enumerating all the states involved.

We define a hierarchy of M&S abstractions, which we call Symbolic M&S (SM&S) in which variables are abstracted one by one, as in the case of PDBs. The difference with respect to the PDB hierarchies is that abstracted variables can be partially considered instead of being completely ignored. We prove that, if a linear merge strategy (that considers variables one by one in a linear ordering) is used, sets of M&S abstract states can be efficiently represented with BDDs. This bounds the size of any BDD involved in the abstract search with respect to the total number of abstract states, ensuring that the symbolic search will be tractable on small enough abstract state spaces. Therefore, we can use our algorithm in combination with hierarchies of PDB or M&S abstractions. We call the algorithm and the resulting heuristics Symbolic Perimeter Abstractions (SPA) in general or SPPDB/SPM&S, if we want to restrict the type of abstractions used to PDBs or M&S respectively.

We perform an in-depth experimental analysis comparing the performance of SPA when using different hierarchies of PDB and M&S abstractions. Our results show that symbolic abstractions are very informative heuristics. Despite the fact that they have not been widely used by explicit-state search planners, symbolic PDBs or symbolic perimeters guiding A* are already very competitive and outperform the popular LM-CUT heuristic [4] in many domains. SPA can further improve the results of the symbolic perimeter in a number of domains, especially when using PDB hierarchies. We provide insights into the types of domains, analyzing when a perimeter can be useful or detrimental to enhance symbolic PDBs.

A preliminary version of our work was presented in a conference paper [18].[1] In this article, we provide a more general and detailed description of our technique, as well as a more thorough experimental evaluation. The rest of the article is structured as follows. Section 2 describes the basics and nomenclature that we will use in the rest of the paper. Section 3 reviews the state of the art in abstraction heuristics. In Section 4 we present our definition of perimeter abstraction heuristics. The details for using symbolic search to traverse M&S abstractions are explained in Section 5. The overall SPA algorithm that puts together the different parts described in previous sections is presented in Section 6. Section 7 presents the evaluation of symbolic perimeter abstraction heuristics. The article concludes in Section 8 with a summary of the main contributions and conclusions.

2. Preliminaries

This section presents the basic definitions of classical planning, heuristic search, and symbolic search.

2.1. Classical Planning

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

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

The *state space* of a planning task Π is the LTS Θ_Π where: $\mathcal{S}$ is the set of all states; s_0 is the initial state I of Π; $s \in S_\star$ if and only if $G \subseteq s$; the labels L correspond to the actions A, and $s \xrightarrow{a} s'$ is a transition in T if $pre_a \subseteq s$, and $s'(v) = \mathit{eff}_a(v)$ for $v \in V(\mathit{eff}_a)$ while $s'(v) = s(v)$ for $v \in V \setminus V(\mathit{eff}_a)$. A *plan* for a state s is a sequence of actions that defines a path from s to any $s_G \in S_\star$. The cost of a plan π is defined as the sum of the costs of its actions, $cost(\pi) = \sum_{a_i \in \pi} cost(a_i)$. The cost of a cheapest plan for s is denoted as $h^*(s)$. A plan for s_0 is a plan for Π, and it is *optimal* if and only if its cost equals $h^*(s_0)$.

2.2. Planning as Heuristic Search

Most state-of-the-art cost-optimal planners employ heuristic search [2]. A *heuristic* is a function $h : \mathcal{S} \to \mathbb{R}_0^+ \cup \{\infty\}$ which estimates the optimal cost to reach a goal state from each state (remaining cost), or returns $+\infty$ if the goals are known to be unreachable from the state. A heuristic is *perfect* if it coincides with h^*. A heuristic is *admissible* if it never overestimates the remaining cost, that is, $\forall s : h(s) \leq h^*(s)$. A heuristic is *consistent* if for every transition $(s, a, s') \in T$, $h(s) \leq h(s') + c(a)$.

The most popular algorithm for cost-optimal planning is A* with an admissible heuristic [1]. A* keeps an open list, initialized with the initial state, and a closed list, initialized empty. At each step, A* selects a node m from the open list with lowest $f(n) = g(n) + h(n)$. $g(n)$ is the current best cost from the root to node n and $h(n)$ is the heuristic estimation. Then, A* expands m, inserting m into the closed list, generating all its successors and inserting each successor m' generated by applying an action a to the state in m into the open list with $g(m') = g(m) + c(a)$. The expansion order of A* guarantees that no node with $f(n) > h^*(I)$ is ever expanded. If the heuristic is consistent, A* will never re-expand a node because nodes are expanded with their optimal g-values.

A number of admissible heuristics for cost-optimal planning exist in the literature. They include critical-path heuristics (h^m) [3], landmark heuristics (LM-CUT) [4], flow-based heuristics [19, 20], and potential heuristics [5], among others. In this paper, we focus on abstraction heuristics which are reviewed in detail in Section 3. In practice, more than one heuristic may be used, either in a portfolio [21, 22] or other methods to combine multiple estimates in a single search [23, 24]. There also exist several search enhancements for A* that are orthogonal to the heuristics used. Some examples include partial-order pruning [7], symmetries [6, 25] or dominance pruning methods [26, 8].

[1] We originally called our technique symbolic M&S [18]. We rename it to Symbolic Perimeter M&S to highlight that it is a perimeter abstraction heuristic.

2.3. Symbolic Search

Symbolic search originated in the context of model checking and verification [27, 28, 29] where it has been widely used to determine reachability in transition systems [30]. It is also a successful technique for state space exploration both in classical [31] and non-deterministic planning [32, 33].

Symbolic search algorithms take advantage of succinct data-structures to efficiently represent sets of states. A set of states $S \subseteq \mathcal{S}$ is represented by its *characteristic function* $f_S(x_1 \dots x_n) : \mathcal{S} \to \{1, 0\}$ that represents whether a given state in $\mathcal{S}$ belongs to S (1) or not (0). The input of the function is the bit-vector description of a state represented by binary variables x_i, so that the function signature may be written as $f_S : \{0, 1\}^n \to \{1, 0\}$. Each finite-domain variable $v \in V$ with domain D_v is represented with $\lceil \log_2 |D_v| \rceil$ binary variables.[2] To simplify the notation, we use the same symbol, S, to denote a set of states and its characteristic function.

Binary Decision Diagrams (BDDs) [15] are data structures that can be used to compactly represent Boolean functions, such as the characteristic functions of sets of states. A BDD is a rooted labeled directed acyclic graph with two types of *terminal nodes* or *sinks*: $\top$ and $\bot$. *Non-terminal* or *inner* nodes are defined by a 3-tuple $p = \langle x_i, p_0, p_1 \rangle$, where x_i is a variable and p_0 and p_1 are either sink or inner nodes representing functions that do not depend on x_i. For a node $\langle x_i, p_0, p_1 \rangle$, p_0 corresponds to the case of assigning variable x_i the value false ($\bot$), thus giving the so-called *low* (or 0) successor; p_1 corresponds to the case of assigning variable x_i the value true ($\top$), giving the *high* (or 1) successor. For any assignment of variables in a path from the root to a sink, the represented function will be evaluated to the value labeling the sink. We consider reduced ordered BDDs, which have the additional requirement that variables on any path from the root to the sinks always appear in the same order. This allows the application of reduction rules that remove unnecessary and redundant parts from the BDD. Algebraic Decision Diagrams (ADDs) [34] are a useful extension of BDDs, which can have any number of terminal nodes. ADDs can describe integer functions, $f_S(x_1 \dots x_n) : \mathcal{S} \to \mathbb{Z}$, such as heuristic functions.

BDDs allow us not only to succinctly represent sets of states but also to operate with them through function transformations. For example, 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 ($\mathcal{S} \setminus S$) corresponds to the negation ($\neg$) of the characteristic function. Quantification of variables is another type of function transformation commonly found in symbolic search. The existential quantification of a variable $v \in V$ with respect to a function f removes the dependency of f on variable v such that $\exists v : f = f|_{v=1} \vee f|_{v=0}$, where $f|_{v=1}$ and $f|_{v=0}$ are the sets of assignments to variables $V \setminus \{v\}$ so that they make f true whenever $v = 1$ and $v = 0$, respectively. Thus, the result of the existential quantification corresponds to the projection of the set of states to the set of variables $V \setminus \{v\}$. Similarly, the universal quantification $\forall v : f$ removes the dependency of f on variable v such that $\forall v : f = f|_{v=1} \wedge f|_{v=0}$. This means that when a variable $v \in V$ is universally quantified away, the resulting function f' is satisfied only by those cases in which f is true for every possible assignment of v (i.e., $v = 1$ and $v = 0$).

In order to search the state space, the set of actions A is represented with one or more *Transition Relations* (TRs). A TR is a function $T_c(x, x')$ that represents one or more actions with the same cost, c. TRs are defined using two sets of variables, the *source*-set and the *target*-set of variables, denoted as x and x', respectively. 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 describe the states in which the action is applicable (the preconditions of the actions) and the variables of the *target*-set describe the states after the execution of the action (the effects of the actions). Given a set of states S and a TR T_i, the *image* operation is used to compute the set of successor states that can be reached from any state in S by applying any action represented by the TR. The image corresponds to the operation $image(S(x), T_i(x, x')) := (\exists x \, (S(x) \wedge T_i(x, x'))) \, [x'/x]$. The conjunction $S(x) \wedge T_i(x, x')$ corresponds to all pairs of states $\langle s, s' \rangle$ such that $s \in S$ and $s' = a(s)$ for an action a represented by T_i. 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.

Similarly, the *pre-image* operation computes the set of predecessor states, i. e., states that can reach some state in S through some action in the TR. Formally:

$$pre\text{-}image(S(x), T_i(x, x')) = \exists x' \, ((S(x)[x/x']) \wedge T_i(x, x'))$$

[2] Based on this transformation, when discussing the representation of sets of states as decision diagrams, we assume FDR variables to be binary without loss of generality.

3. Abstractions

In this section, we describe previous work on abstraction heuristics, starting with the basic definitions that we adopt, as well as techniques to generate domain-independent abstraction heuristics, such as PDBs, Merge-and-Shrink, and Perimeter PDBs.

3.1. Preliminary Definitions

Abstractions are simplifications of the task that transform the state space into a smaller one, called abstract state space. The corresponding abstract planning task is optimally solved and the cost of its solution is used to guide the search in the original state space.

Definition 1 (Abstraction [12])**.** *Let Π be a planning task with state space $\Theta_\Pi = (\mathcal{S}, L, T, s_0, S_\star)$. An* abstraction *of Θ is a surjective function $\alpha : \mathcal{S} \to \mathcal{S}^\alpha$ mapping $\mathcal{S}$ to a set of abstract states, $\mathcal{S}^\alpha$.*

An abstraction α defines an abstract state space $\Theta^\alpha = \langle \mathcal{S}^\alpha, L, T^\alpha, s_0^\alpha, \mathcal{S}_\star^\alpha \rangle$ where $\mathcal{S}^\alpha$ is the set of abstract states, L is the set of labels, $T^\alpha = \{(\alpha(s), l, \alpha(t)) \mid (s, l, t) \in T\}$, $s_0^\alpha = \alpha(s_0)$ and $\mathcal{S}_\star^\alpha = \{s^\alpha \mid \exists s \in \mathcal{S}_\star, s^\alpha = \alpha(s)\}$. The size of α, denoted as $|\alpha|$, is the number of abstract states, $|\mathcal{S}^\alpha|$. The transformation $\Theta \to \Theta^\alpha$ is a *homomorphism*, i. e., a structure-preserving mapping such that for all $s, t \in \mathcal{S}$ and $l \in L$, $(s, l, t) \in T$, implies $(\alpha(s), l, \alpha(t)) \in T^\alpha$.

Definition 2 (Abstraction heuristic [12])**.** *Let α be an abstraction with an associated abstract state space Θ^α. The induced* abstraction heuristic *$h^\alpha(s)$ is a heuristic function that returns the cost of the cheapest path from $\alpha(s)$ to $\mathcal{S}_\star^\alpha$ in Θ^α.*

Abstraction heuristics are admissible and consistent, since Θ^α is a homomorphism and, thus, paths in the original state space are preserved in the abstract state space. Every abstraction α induces an *equivalence relation* on $\mathcal{S}$, $\sim^\alpha$, defined as $s \sim^\alpha t$ if and only if $\alpha(s) = \alpha(t)$, i. e., any two states mapped to the same abstract state are considered to be equivalent by the abstract state space. Hence, each abstract state s_i^α may be interpreted as a set of states S_i^α such that it contains every state mapped to s_i^α, i. e., $S_i^\alpha = \{s \mid \alpha(s) = s_i^\alpha\}$. Therefore, we will freely refer to an abstract state as an equivalence class or a set of states of the original state space.

Definition 3 (Relevant variables of an abstraction [12, 35])**.** *Let α be an abstraction of a planning task with variable set V. We say that α depends on variable $v \in V$ if and only if there exist states s and t such that $\alpha(s) \neq \alpha(t)$ and $s[v'] = t[v']$ for all $v' \in V \setminus \{v\}$. The set of relevant variables for α, written V_α, is the set of variables in V on which α depends.*

Note that abstractions are *transitive*: if α is an abstraction of a transition system, Θ, and α' is an abstraction of Θ^α, then their composition, $\alpha' \circ \alpha$, is also an abstraction of Θ. Let α_1 and α_2 be two abstractions of a given state space Θ. We say that α_2 is a coarsening abstraction of α_1 if and only if for every pair of states $s, t \in \mathcal{S}$, $\alpha_1(s) = \alpha_1(t)$ implies $\alpha_2(s) = \alpha_2(t)$. In that case, α_1 is a refinement of α_2.

3.2. Pattern Databases

Pattern Databases (PDBs) are a kind of abstraction based on abstraction *patterns*. Patterns were originally defined as a selection of tiles in the sliding-tile puzzle [9] and later extended to other domains where more general definitions have been considered, shifting the focus from the mere selection of care variables to different state-space abstractions [36]. In planning, a *pattern* is usually defined as a selection of state variables, while the value of other variables is ignored [10]

Definition 4 (Projection [12, 35])**.** *A* projection *α of a planning task over a subset of variables $W \subseteq V$ is defined by restricting the initial state, goals and preconditions/effects of the actions to W such that $s \sim^\alpha t$ if and only if $s[v] = t[v]$ for all $v \in W$.*

The performance of PDBs greatly depends on the *patterns* chosen to create the abstractions. When PDBs are constructed using explicit search, few variables may be included in the pattern so that the size of the abstract state space is small enough to be enumerated. To compensate for the fact that each PDB only considers a few variables, multiple PDBs are generated and these are combined additively. A set of PDBs can be additively combined if they are

disjoint [37] (i. e., if no operator is relevant for more than one of them) or if the cost of the operators is divided with a cost-partitioning schema [10, 38]. Several algorithms for pattern selection of explicit PDBs have been proposed in the context of automated planning:

- iPDB [39] starts considering the patterns consisting of a single goal variable and uses a hill-climbing search. At each iteration, it attempts all possible ways to add a new variable to an existing pattern, such that the variable is relevant for the pattern (i. e. either it is a goal or it provides a precondition for an operator that modifies another variable in the pattern). The algorithm stops when the improvement is marginal. All the resulting PDBs are combined using the canonical heuristic method, that takes the maximum over all additive subsets of PDBs.
- gaPDB [40] uses a genetic algorithm to derive multiple PDBs using a 0-1 cost-partitioning so that they are additive.
- CPC [41] runs multiple instances of gaPDB in order to generate a set of complementary PDBs.

In symbolic search the abstract state space does not need to be explicitly enumerated, so an arbitrary number of variables may be considered in a single PDB. Symbolic PDBs have been mainly used by symbolic search planners, which use a BDD variant of A* [42], as the planner GAMER [43]. GAMER uses an automatic pattern selection algorithm based on iPDB to construct a single PDB. It starts considering the pattern of all goal variables. Then, it conducts a hill-climbing search, attempting to add all relevant variables to the current best pattern and selecting the best variable. To speed-up the process, in case that several variables provide the same improvement, the pattern containing all such variables is constructed. Kissmann and Edelkamp [43] also considered the creation of a symbolic perimeter and mentioned the possibility of combining the perimeter with abstraction heuristics. In this paper we explore this idea in full detail.

3.3. Merge-and-Shrink

Merge-and-shrink (M&S) is an algorithm that derives abstractions that take into account all variables, thus overcoming the major limitation of PDB abstractions. M&S was originally proposed in the context of directed model checking [44, 11] and later applied to planning [35, 12]. Formally, M&S abstractions are constructed using the following rules:

(A) *Atomic projections*: For $v \in V$, the *atomic projection* π_v of the task over a single variable v is an M&S abstraction over $\{v\}$.

(M) *Merge*: If α_1 and α_2 are M&S abstractions over disjoint sets of variables $W_1, W_2 \subset V, W_1 \cap W_2 = \varnothing$, then $\alpha_1 \otimes \alpha_2$ is an M&S abstraction over $W_1 \cup W_2$.The merged abstraction $\alpha_1 \otimes \alpha_2$ is defined by $(\alpha_1 \otimes \alpha_2)(s) := (\alpha_1(s), \alpha_2(s))$. In other words, the new abstraction has $|\alpha_1| \times |\alpha_2|$ states, one for each pair $s_{\alpha_1} \in \Theta^{\alpha_1}$, $s_{\alpha_2} \in \Theta^{\alpha_2}$. The state space of the merged abstraction corresponds to the *synchronized product* of the state spaces of α_1 and α_2, $\Theta^{\alpha_1 \otimes \alpha_2} = \Theta^{\alpha_1} \otimes \Theta^{\alpha_2} = (\mathcal{S}', L, T', s'_0, \mathcal{S}'_\star)$ where $\mathcal{S}' = \mathcal{S}^{\alpha_1} \times \mathcal{S}^{\alpha_2}$, $T' = \{((s_1, s_2), l, (s'_1, s'_2)) \mid (s_1, l, s'_1) \in T^{\alpha_1} \wedge (s_2, l, s'_2) \in T^{\alpha_2}\}$, $s'_0 = (I^{\alpha_1}, I^{\alpha_2})$ and $\mathcal{S}'_\star = \{(s_1, s_2) \mid s_1 \in \mathcal{S}^{\alpha_1}_\star \wedge s_2 \in \mathcal{S}^{\alpha_2}_\star\}$. The constraint $W_1 \cap W_2 = \varnothing$ ensures that $\Theta^{\alpha_1} \otimes \Theta^{\alpha_2}$ is isomorphic to $\Theta^{\alpha_1 \otimes \alpha_2}$.

(S) *Shrink*: If the size of any new abstraction β resulting from merging two abstractions goes over a given memory limit, the *Shrink* operation generates a smaller abstraction by applying an abstraction γ over Θ^β. If β is a M&S abstraction over a set of variables $W \subseteq V$ and γ is an abstraction from S^β to S^γ, then $\gamma \circ \beta$ is an M&S abstraction over W. This further abstracts an abstraction β by aggregating an arbitrary number of abstract states into the same abstract state.

Rule (A) allows the algorithm to start from *atomic projections*, one for each variable. Given that the size of the new abstraction can grow exponentially by repeatedly applying Rule (M), *Rule (S)* allows M&S to reduce the size of abstractions.

Algorithm 1 shows the pseudocode of the M&S algorithm. It takes as input a planning task and a parameter N that imposes a bound on the abstraction size, i. e., no abstraction in the process will have more than N abstract states. The algorithm initializes a pool of abstractions with the atomic projection with respect to every variable $v \in V$. While

Algorithm 1: Merge-and-shrink [12]

Input: Planning task Π, size bound N
Output: M&S abstraction α

$\mathcal{A} := \{\pi_v \mid v \in V\}$ // Rule (A)
while $|\mathcal{A}| > 1$ **do**
  Select $\alpha_1, \alpha_2 \in \mathcal{A}$
  Shrink α_1 and/or α_2 until $|\alpha_1| \times |\alpha_2| \leq N$ // Rule (S)
  $\alpha' := \alpha_1 \otimes \alpha_2$ // Rule (M)
  $\mathcal{A} := (\mathcal{A} \setminus \{\alpha_1, \alpha_2\}) \cup \{\alpha'\}$
return the only abstraction in $\mathcal{A}$

there is more than one abstraction left in the pool, the algorithm selects two abstractions and *merges* them, replacing them by their combination. If necessary for the merged abstraction to satisfy the size bound N, a shrinking step is applied to both selected abstractions, prior to every merging step. To implement M&S in practice, we need a *merge strategy* deciding which abstractions to merge by rule (M), and a *shrink strategy* deciding which (and how many) states to aggregate by rule (S). The performance of M&S greatly depends on the policies chosen for these steps.

M&S is a generalization of PDBs; for any PDB, we can construct an equivalent M&S abstraction just by merging the atomic abstractions that correspond to variables in the pattern. Values of variables not in the pattern are shrunk to a single abstract state, so that the abstraction does not distinguish the value of those variables, just as in the case of PDBs. However, using different shrink strategies M&S can derive abstract state spaces that PDBs cannot, e. g., considering all the variables of the task while keeping the abstract state space small.

We say that a merge strategy is *linear* if it always selects at least one atomic abstraction to be merged. Hence, linear merge strategies are characterized by the order in which variables are chosen, $v_1, \ldots, v_n$. Even though the original work of Dräger et al. in Model Checking used a non-linear strategy, in planning most works have considered linear merge strategies [12, 45, 46]. Non-linear merge strategies have recently been introduced in planning by Sievers et al. [47] and promising approaches have been proposed [48, 49]. A *shrink strategy* takes as input an abstract state space and decides which abstract states must be aggregated to keep the number of abstract states below N. Several shrink strategies have been defined, like fh-shrinking [35], or bisimulation-based shrinking [45, 46].

3.4. Cascading-Tables Representation of M&S

To represent the mapping from states to abstract states, M&S abstractions internally use the *cascading-tables* representation, first introduced by Dräger et al. [44] and discussed in detail by Helmert et al. [12]. This representation will be important for later describing our algorithm that transforms them to a symbolic ADD representation. The cascading-tables representation is built incrementally, representing every intermediate M&S abstraction by means of a table, as shown in the example of Figure 1. This example shows a logistics task in which a number of packages must be transported from a location L to another location R. The task has one variable per package to identify its position (at L, at R, or in the truck T) and one for the truck (at L or at R).

v_i	
L	0
T	1
R	2

(A) π_i

π_1 \ π_2	0	1	2
0	0	1	2
1	3	4	5
2	6	7	8

(M) $\pi_1 \otimes \pi_2$

π_1 \ π_2	0	1	2
0	0	1	2
1	1	3	4
2	2	4	5

(S) $\alpha_1 \coloneqq \gamma_{bisim} \circ (\pi_1 \otimes \pi_2)$

Figure 1: Example of *cascading-tables* M&S representation. (A), (M) and (S) correspond to the application of M&S rules in the *cascading-tables* representation [12].

Each table represents an intermediate abstraction with relevant variables $W \subseteq V$. These tables recursively define the mapping between states in Θ^W to abstract states. Each abstract state is identified with a number, from 0 to $|\alpha| - 1$,

where $|\alpha|$ is the number of abstract states in the abstraction.

Atomic abstraction tables (rule A) associate every possible value of the variable with an abstract state. In our example, we consider two atomic abstractions describing the position of two packages: π_1 and π_2. They map each value (L, R, and T) to a different abstract state. For example, abstract state 1 of π_1 corresponds to having package 1 in the truck.

In each iteration, the M&S algorithm merges two abstractions (rule M). The resulting abstraction has an abstract state for each pair of states of the input abstractions. This is represented with a two-dimensional table, in which rows and columns correspond to abstract states of the merged abstractions. In our example, table (M) shows the product of atomic abstractions π_1 and π_2, which associates each pair of abstract states in π_1 and π_2, $\langle s_i^{\pi_1}, s_j^{\pi_2} \rangle$, to a different ID. For example, abstract state 1 of $\pi_1 \otimes \pi_2$ represents states in which package 2 is in the truck and package 1 is at L. The abstraction in Figure 1 can be merged again with a new atomic abstraction π_3. The result will be a new table with one column for each abstract state in π_3 and one row for each abstract state in $\pi_1 \otimes \pi_2$, for a total of 27 cells.

The shrinking operation (rule S) applies a function γ that maps the abstract states to a new set of abstract states, producing a new abstraction α_1. To apply γ, the value of the cells in the cascading tables are changed accordingly. In our example, γ_{bisim} corresponds to bisimulation shrinking, which in this case considers that all packages are interchangeable so $\gamma_{bisim}(2) = \gamma_{bisim}(6) = 2$ and abstract state $s_2^{\alpha_1}$ corresponds to states in which $(v_1 = L \wedge v_2 = R) \vee (v_1 = R \wedge v_2 = L)$.

An additional lookup table stores the precomputed optimal cost for each abstract state in the final M&S abstraction α. During the search, the heuristic value of a particular state s is retrieved in two steps. The first step recursively traverses the tables returning the abstract state ID associated with the values of s to retrieve the ID of $\alpha(s)$. The second step retrieves the heuristic value associated to such ID in the lookup table. For example, consider the partial state $\langle L, R \rangle$ in which package 1 is at L and package 2 is at R. The retrieval algorithm first checks the tables related to π_1 and π_2, obtaining abstract states 0 and 2, respectively. Then, these values are used to perform a lookup in table α_1, retrieving the value 2. An additional lookup, not shown in our example, is needed to retrieve the heuristic value of abstract state number 2.

3.5. Perimeter Abstraction Heuristics

Perimeter PDBs (PPDBs) are another improvement of PDBs based on perimeter search to obtain more informed heuristics. Perimeter search is a form of bidirectional search independently devised by Manzini [50] and Dillenburg and Nelson [51] that operates in two phases: the backward phase and the forward phase, as represented in Figure 2. The backward phase generates a perimeter P of radius r that contains all states that can reach the goal with a cost of exactly r. The perimeter is computed by a uniform-cost search that generates all states with that g-value or less. The perfect heuristic value of each state in the perimeter is known, whereas states outside the perimeter have a cost to the goal strictly greater than r. Then, the forward phase performs a forward search from the initial state to the perimeter using any algorithm like A* or IDA* [52]. Since the goal of the forward search is any state in the perimeter P, the heuristic evaluation estimates the cost to the closest state in the perimeter[3] so that $h(s) = \min_{s' \in P} h(s, s')$.

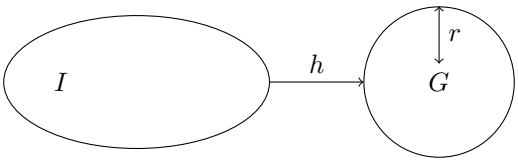


Figure 2: High-level diagram of perimeter search. First, the perimeter around the goal of radius r contains every state that can reach the goal with cost r or less. Then, the forward search is performed guided with heuristics that estimate the cost to reach the closest state in the perimeter frontier.

The major drawback of perimeter search is that the heuristic evaluation may become too expensive when the perimeter is large. In practice, some optimizations avoid to evaluate $h(s, s')$ for every state in the perimeter [50], but it is often not enough to deal with large perimeters. In those cases, a model can be used to predict the optimal radius

[3]Previous definitions of PPDBs assumed unit-cost domains so that all nodes in the perimeter have the same goal distance. Our definition in Section 4 supports general action costs.

r beforehand in order to avoid large overheads with respect to unidirectional search without a perimeter [53]. There are also other ways to use the perimeter while avoiding a large number of heuristic evaluations, such as taking the estimate to the goal and correcting it by considering the error made in the perimeter nodes [54].

Abstraction heuristics are a candidate to overcome this problem, because they precompute the heuristic value of every abstract state. Perimeter Pattern Databases (PPDBs) were first studied by Felner and Ofek in [55]. PPDBs store for each abstract state the minimum cost to any abstract state in the perimeter. Then, a single PDB lookup suffices to perform the heuristic evaluation independently of the size of the perimeter.

A perimeter PDB is constructed in two phases:

1. As in perimeter search, use a backward search to build a perimeter P of radius r around the goal states in the original search space.

2. Perform a second backward search in the abstract state space seeded with abstract states corresponding to states in the perimeter $\{\alpha(p) : p \in P\}$ with initial cost r. As noted by Felner and Ofek, seeding the abstract search with cost r is equivalent to adding r to each heuristic value. This means that the heuristic value of a state is an admissible estimate to the closest state in the perimeter plus the perimeter radius.

The PPDB is used then as a heuristic in a forward search. As the heuristic estimates the cost to the goal while going through one of the states in the perimeter, it is only admissible for nodes outside the perimeter. Nodes inside the perimeter have a path to the goal without going through any node in the perimeter. However, this is not a problem because the search halts whenever a state in the perimeter is chosen for expansion. PPDBs have great potential because states are easily evaluated during the search, avoiding the main drawback of perimeter search. Moreover, PPDBs produce heuristics at least as informed as standard PDBs. However, after performing a theoretical and empirical analysis, Felner and Ofek reported that PPDBs are not better than just taking the maximum between the PDB and the perimeter heuristic (correcting values in the PDB under $r + 1$ to $r + 1$). Alternatively, Linares López [56] proposed storing different PDBs for each state in the perimeter in order to enhance the heuristic whenever multiple patterns are used.

More recently, Eyerich and Helmert applied perimeter PDBs in the context of automated planning [13]. They show that perimeter search can enhance the performance of standard PDBs, contradicting the conclusions reached by Felner and Ofek and suggesting that perimeter PDBs should be revisited. Indeed, the analysis by Felner and Ofek only considered permutation domains with no spurious paths in the abstract state space, which is not a common case in planning domains. Eyerich and Helmert identified several challenges to extrapolate perimeter PDBs from the permutation puzzles commonly used to evaluate domain-dependent heuristic search solvers to planning, and addressed them with a explicit-state search approach. These challenges are useful to illustrate the synergy of perimeter PDBs and symbolic PDBs, since using symbolic search automatically addresses them:

- Perimeter search has been previously applied to domains with one goal state and a state space where all transitions are invertible. In planning, one has to deal with the difficulties of regression caused by having up to an exponential number of goal states. However, symbolic search is an efficient method to perform regression that often outperforms partial-state based regression [57].

- Since the PDB is computed for each individual planning instance, the time spent in generating the abstraction heuristic has to be amortized. This applies to all kinds of abstraction heuristics, but here it is aggravated by the time spent in matching states in the perimeter to abstract states to seed the open list of abstract searches. In symbolic search, this is done through symbolic operations, as we will detail in Section 5.3.

- The perimeter radius must be automatically set by the planner, without instance-specific parameter tuning. As Eyerich and Helmert, we rely on parameters that bound the time and memory resources invested in constructing the PPDBs. In our case, we use the number of BDD nodes in the search frontier to limit the memory and to estimate the time needed for the next step.

- In order to terminate the forward search as soon as a state in the perimeter is expanded, membership in the perimeter must be efficiently tested. This can be challenging, especially for large perimeters. However, if the perimeter is encoded as a BDD, membership operations only take linear time in the number of variables of the task.

4. Hiearchies of Perimeter Abstraction Heuristics

In this section, we generalize the definition of perimeter PDBs. Previous work on perimeter abstraction heuristics has considered only two different phases. In the first phase, a perimeter is constructed by a search on the original state space and in the second phase the search is completed in an abstract state space. Instead, we consider an arbitrary number of searches in a hierarchy of abstract state spaces.

Definition 5 (Abstraction Hierarchy)**.** *An abstraction hierarchy is a directed acyclic graph in which each node corresponds to an abstraction. There is an arc from node α_i to node α_j if and only if α_j is a coarsening abstraction of α_i.*

We will mostly use hierarchies in which each node has only one child, i. e., lists of an arbitrary number of abstractions, $\alpha_0, \alpha_1, \ldots, \alpha_k$ such that each α_i is an abstraction of $\Theta^{\alpha_{i-1}}$ for all $i > 0$. Note that α_0 is usually defined as the identity abstraction, $\alpha_0(s) = s$, so that the first search is performed on the original state space $\Theta^{\alpha_0} \equiv \Theta$. All other abstractions can be defined either using M&S or PDBs.

Figure 3 illustrates the main idea of perimeter abstraction heuristics. For each abstraction α_i, we perform a backward uniform-cost search UCS_i on its abstract state space Θ^{α_i}. We characterize the current status of a uniform-cost search UCS as a tuple $\langle open, closed \rangle$ where $open$ contains those states generated but not expanded and $closed$ represents the states that have already been expanded. We assume that duplicate detection is enabled so that $closed \cap open = \emptyset$. Both $open$ and $closed$ can be split into buckets $open_g$ and $closed_g$ that represent the open states generated with cost g and expanded states with cost g or less, respectively. We define the radius of a search $r(UCS)$ as the minimum g-value of any state in the open list, $r(UCS) = \min(g | open_g \neq \varnothing)$ or ∞ if open is empty.

After constructing a perimeter around the goal, the search frontier is relaxed into an abstract state space α_1. Abstract searches are used to explore the state space outside the perimeter. However, the abstraction will typically consider states in the perimeter equivalent to other states, so the information is not perfect anymore. This is reflected in Figure 3 by the irregular shape of the perimeters. The algorithm does not only build a single perimeter; whenever the abstract search in α_i becomes intractable, it is relaxed into a coarser abstract state space α_{i+1}. In the end, all the abstract perimeters are used as heuristics, to estimate the cost from states in the forward search to the perimeter.

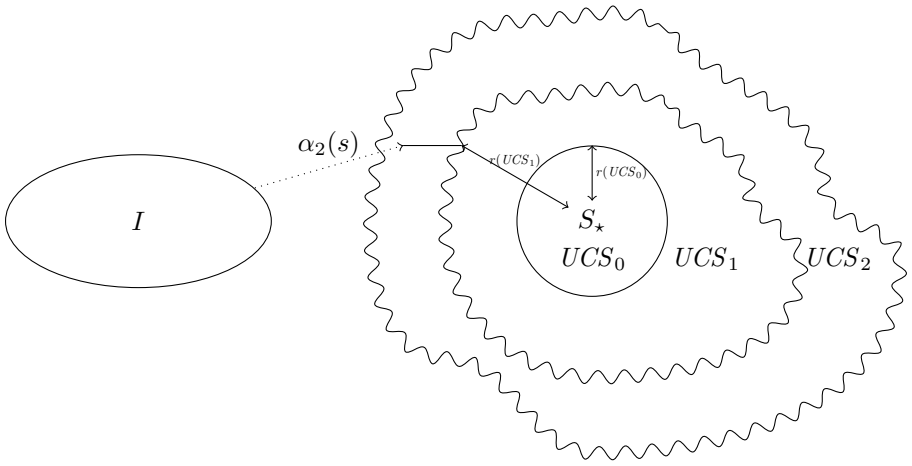


Figure 3: Example of a perimeter abstraction heuristic. Several abstract searches, each one in a coarser abstract state space, provide the cost estimates to states in the previous perimeter.

Algorithm 2 shows the precomputation phases of a PA heuristic. The algorithm initializes the first search in Θ^{α_0} with the set of goal states. Then, each phase i consists of a backward uniform-cost search in the abstract state space Θ^{α_i} initialized with the perimeter frontier of search $i-1$. A search may be stopped at any point and then the `get-h` function generates an ADD that represents the corresponding heuristic, as specified in Definition 7. If there are more state spaces to explore, we re-seed (line 9) $open$ and $closed$ with abstract states relative to the next abstraction as

Algorithm 2: Perimeter Abstraction Heuristic

Input: Planning problem $\Pi = \langle V, A, I, G \rangle$.
Input: Abstraction Hierarchy $\alpha_0, \ldots, \alpha_k$.
Output: List of heuristics $h_{UCS0}, h_{UCS1}, \ldots$.

```
open_0 ← α_0(G)
closed ← ⊥
foreach α_i ∈ {α_0, ..., α_k} do                          /* for each precomputation phase */
    ⟨open, closed⟩ ← Uniform-Cost-Search (open, closed, α_i)
    h_UCSi ← get-h(open, closed)                          /* gather heuristic (see Def. 7) */
    if open = ∅ then                                      /* if the search has sucessfully finished */
        return h_UCS0, ..., h_UCSi
    if i < k then                                         /* if there are more abstractions left */
        ⟨open, closed⟩ ← re-seed(open, closed, α_{i+1})   /* Init next search (see Def. 6) */
return h_UCS0, ..., h_UCSk
```

specified in Definition 6. The algorithm terminates returning the heuristic when all phases have been completed or if at some point there are no states left in $open$.

4.1. Initialization of Perimeter Abstract Searches

Abstract searches are initialized with the $open$ and $closed$ lists of the preceding search —so that search $UCS_i = \langle open', closed' \rangle$ continues from the perimeter generated by search $UCS_{i-1} = \langle open, closed \rangle$. To ensure that the resulting heuristics are consistent and admissible, states should not be removed from $open$ or added into $closed$ (see proof of Theorem 1 below). In other words, if we consider the sets of states associated with $open$, $open$', $closed$ and $closed$' (by interpreting the abstract states as the set of states that are mapped into them), they must meet the following constraints in the initialization of UCS_i: $open^{i-1} \subseteq open^i$ and $closed^i \subseteq closed^{i-1}$. The open list is then seeded with the abstract states of all the states in the perimeter, as in the case of common PPDBs. Basically, we iterate over all states in $open$ and insert the corresponding abstract state in $open$' with the same g-value it had. For the closed list, we follow a similar procedure, but inserting an abstract state in $closed$' only if all the corresponding states are present in $closed$.

Definition 6 (Perimeter search initialization)**.** *Let $UCS_i = \langle open', closed' \rangle$ be a search on Θ^{α_i} initialized with the perimeter of another search $UCS_{i-1} = \langle open, closed \rangle$ on $\Theta^{\alpha_{i-1}}$. For every abstract state on α_i, $s_j^{\alpha_i}$, let $S_j^{\alpha_{i-1}}$ denote the set of all states mapped to $s_j^{\alpha_i}$, i. e., $S_j^{\alpha_{i-1}} = \{s \in \Theta^{\alpha_{i-1}} \mid \alpha_i(s) = s_j^{\alpha_i}\}$. We initialize search UCS_i as:*

$$open'_g = \{s_j^{\alpha_i} \mid \exists_{s \in S_j^{\alpha_{i-1}}} s \in open_g\}$$

$$closed'_g = \{s_j^{\alpha_i} \mid \forall_{s \in S_j^{\alpha_{i-1}}} s \in closed_g\}$$

Figure 4 shows an example of the initialization of abstract searches. Consider the part of the original state space depicted in Figure 4a. There are seven states, including the goal state G, three states that form a path to the goal, A, B and C, and another three states that are in another part of the state space D, E and F. From our figure, one can infer the cost to the goal of A, B, C and G. But, since the dotted node may be substituted by any arbitrary directed graph, nothing can be said about the cost to the goal from D, E and F: they are arbitrarily far from the goal state and may even be dead-end states. The abstraction maps states C and D to the same abstract state (CD) and all other states are mapped onto themselves.

A perimeter of radius 3 is constructed in the original state space, in which case states G, C and B are expanded —assuming unitary costs. At that point, the open list contains $\{A\}$ and the closed list contains $\{G, B, C\}$. This frontier is used to initialize an abstract search in the abstract state space of Figure 4b. In that case, the open list of the abstract

(a) Original state space. (b) Abstract state space.

Figure 4: Original and abstract state spaces in our example of initialization of abstract searches. The abstraction maps states C and D to the same abstract state (CD) and all other states are mapped onto themselves.

search is initialized with $\{\alpha(A)\} = \{A\}$ and the closed list is initialized with $\{G, B\}$ (note that the abstract state CD is not inserted in closed since D has not been closed yet).

Previous definitions of PPDBs initialized the closed list to the empty list. While both approaches result in an admissible and consistent heuristic, the initialization proposed in Definition 6 includes more states into the closed list, which will be detected as duplicates. This may reduce the number of states expanded by the abstract search, hence reducing the search effort and pruning some spurious paths that do not exist in the original state space. Therefore, this initialization of the closed list helps to derive more informed heuristics while preserving admissibility. In our example, there is a path from A to CD of cost 2, passing by B. If the closed list of the search in the abstract state space is initialized to the empty list, the heuristic value of D will be $5 = 2 + 3$, i.e., the distance from CD to A in the abstract state space plus the distance from A to the goal in the original state space. However, if the closed list is initialized with states expanded by the perimeter search, states B and G are detected as duplicates by the search in the abstract state space. This is sound because they were already expanded by the search in the original state space. As B is not expanded in the abstract search, the spurious path between A and D is pruned and the heuristic value of D, E, F and other states will be greater.

4.2. Perimeter Abstraction Heuristic

Each uniform-cost search conducted by the algorithm produces an estimate h_{UCS_i}. If the search UCS_i on Θ^{α_i} was initialized with the tuple $\langle open, closed \rangle$, the corresponding heuristic $h_{UCS_i}(s)$ estimates the cost of reaching the goal from a state s through the perimeter in $open$. Since the searches are not always completed, the heuristic estimates follow the definition of partial abstraction heuristics [58]. Previous definitions of partial abstractions assumed uniform costs, so that states expanded or generated by UCS with lower cost than $r(UCS_i)$ take the value with which they were expanded/generated and all other states are assigned a value of $r(UCS_i) + 1$. The non-uniform cost case is slightly more complicated because not all the nodes in the open list have the same g-value. In particular, with certain cost functions, it is possible to have states in open with cost c such that $r(UCS_i) < c < r(UCS_i) + \min_{a \in A} c(a)$. For example, if actions have a cost of 2 and 3, there may be states in the open list with cost 2 and 3 so $r(UCS_i) = 2 < 3 < r(UCS_i) + 2 = 4$. In such an scenario, $r(UCS_i) + \min_{a \in A} c(a)$ is not an admissible estimate and, at the same time, $r(UCS_i)$ is not the most informed estimate that could be given for unseen states.

Definition 7 (Partial abstraction heuristic)**.** *Let $UCS = \langle open, closed \rangle$ be a backward uniform-cost search over an abstract state space Θ^{α}, then the* partial abstraction heuristic *of UCS is $h_{UCS}(s) = \min(g_{UCS}(\alpha(s)), r(UCS) + \min_{a \in A} c(a))$, where:*

$$g_{UCS}(\alpha(s)) = \begin{cases} \min(\{g \mid \alpha(s) \in closed_g \cup open_g\}) & \alpha(s) \in closed \cup open \\ \infty & otherwise \end{cases}$$

As UCS is a backward search, the g-value of abstract states in UCS corresponds to their goal distance. Then, the g-values of abstract states built in this pre-processing step will be used to compute the heuristic to be used later in the actual forward search. The heuristic value of a state is computed as the minimum value between the g-value of its corresponding abstract state (applying abstraction α) and the minimum cost for any state not in the perimeter (the

radius of the perimeter plus the minimum cost of any action). And, the g-value of the abstract state is infinite if it has not been generated during the backward search, or the minimum cost form the goals to generate it (minimum g when it was inserted in open or closed).

During the search, the perimeter abstraction heuristic, h_{PA}, combines the estimate of each search, h_{UCS_i}.[4] A common method to combine multiple heuristic estimates is to take the maximum of all of them. However, one must be careful to preserve admissibility when combining the heuristic estimates coming from searches initialized with different perimeters. Each h_{UCS_i} estimates the cost from each state s to the goal by passing through some state in the perimeter with which UCS_i was initialized. This is admissible for states outside such perimeter, but it may be inadmissible for states that were expanded during the perimeter computation because they can reach the goal directly without leaving the perimeter.

Therefore, to compute the heuristic value of an state s, we take the maximum of all estimates $h_{UCS_0}(s), \ldots, h_{UCS_i}$ until the search UCS_i where $\alpha_i(s)$ was first expanded. This is necessary to obtain an estimate as accurate as possible. Each search UCS_i is initialized with the perimeter of the previous one so the value of $r(UCS_i)$ increases monotonically with i. However, the value of h_{UCS_i} does not increase monotonically with i, because during the initialization of a new abstract search UCS_{i+1}, new states may be introduced into the open list (those that are mapped by the abstraction to the same abstract state as any node in the open list) with a g-value lower than $r(UCS_i) + \min_{a \in A} c(a)$. Therefore, adding the minimum action cost to the current radius is not admissible for those states anymore, since they are now considered equivalent to a state with lower cost to the goal. Consider again the example of Figure 4, where the abstraction function maps states C and D to the same abstract state. If the initial perimeter search, UCS_0, has only expanded G, then C is the only node in *open* and the heuristic values of h_{UCS_0} are 0 for G, 1 for C, and 2 for the all other states. However, as C and D are mapped to the same abstract state, $h_{UCS_1}(D) = 1 < h_{UCS_0}(D)$.

Definition 8 (Perimeter abstraction heuristic)**.** *Let $UCS_0, \ldots, UCS_k$ be the perimeter abstraction phases performed over an abstraction hierarchy $\alpha_0, \ldots, \alpha_k$. Let s be a state, and UCS_i be the first exploration in which $\alpha_i(s)$ was expanded, i. e., $i = \min\left\{j \mid \alpha_j(s) \in closed_{UCS_j}\right\}$. Then, we define the perimeter abstraction heuristic as:*

$$h_{PA}(s) = \max_{i=[0,\ldots,k]} h_{UCS_i}(s)$$

Theorem 1. *Perimeter abstraction heuristics are admissible and consistent.*

Proof. A heuristic h is consistent if and only if $h(s) \leq h(s') + c(l)\ \forall(s, l, s')$. Let $UCS_0, \ldots, UCS_k$ be the searches that define the perimeter abstraction heuristic, h_{PA}. We divide the proof in two cases, depending on whether s' was expanded by any abstract search UCS_i, or not.

1. Let UCS_i be the first search in which s' was expanded and let C denote the cost with which s' was expanded. Then $h_{UCS_i}(s') = C$. As no previous search expanded s', $h_{PA}(s') = \max_{j=[0,\ldots,i]} h_{UCS_j}(s') \geq C$. Thus, it suffices to show that $h_{PA}(s) \leq C + c(l)$. Let j be the first index for which $h_{PA}(s) = h_{UCS_j}(s)$. We consider two cases, depending on whether $j < i$, or $i \leq j \leq k$.

   (a) If $j < i$, then $r(UCS_j) \leq C$ because otherwise s' would have been expanded by UCS_j or an earlier search. As $h_{PA}(s) = h_{UCS_j}(s) \leq r(UCS_j) + \min_{a \in A} c(a)$, we obtain that $h_{PA}(s) \leq C + c(l)$.

   (b) If $i \leq j \leq k$, as s' was expanded by UCS_i, necessarily s was inserted in $open_{g+c(l)}$ in UCS_i. Then, according to Definition 7, $h_{UCS_i}(s) \leq C + c(l)$, no matter if s was expanded by UCS_i or not. Since states are preserved in the open list until they are expanded by some UCS_k, the same holds for any UCS_j with $i \leq j \leq k$.

2. In the second case, s' was not expanded by any search. Note that $h_{PA}(s) \leq r(UCS_k) + \min_{a \in A} c(a)$ for all states because for any given search $h_{UCS_i} \leq r(UCS_i) + \min_{a \in A} c(a)$ and the values of $r(UCS_i)$ monotonically

[4] This was not necessary in previous definitions of PPDBs reviewed in Section 3.5 because they consider a single abstract state space and stop the forward search when colliding with the perimeter. Thus, states in the inner part of the perimeter are never evaluated. Since we have several perimeters, the evaluation of states in the inner part of a perimeter cannot be avoided.

increase with larger values of i. Therefore, to prove the inequality $h_{PA}(s) \leq h_{PA}(s') + c(l)$ it suffices to show that $h_{PA}(s') \geq r(UCS_k)$. There are two possibilities, depending on whether s' has been generated by any search or not. If it has been generated but not expanded, then it must remain in the open list of the last search, UCS_k.[5] Since s' was not expanded, $r(UCS_k) \leq g_{UCS_k}(s)$ (recall that $r(UCS_k)$ is the minimum possible g-value in its open list). Hence, $h_{PA}(s') \geq h_{UCS_k}(s') \geq r(UCS_k)$. Finally, if s' has not been generated, $h_{PA}(s') = r(UCS_k) + \min_{a \in A} c(a) \geq r(UCS_k)$.

Admissibility is derived from consistency, given that the heuristic is goal-aware: $\forall s_\star \in S_\star,\ h_{PA}(s_\star) = 0$. □

5. Symbolic Abstraction Heuristics

Symbolic abstractions use symbolic search to precompute the heuristic values by traversing the abstract state space, and taking advantage of the efficiency of these techniques for exhaustive state space exploration. This has been previously done for symbolic PDBs [14, 43], but it can also be done for any other abstraction heuristics that can be efficiently represented as BDDs.

Symbolic search is very useful in the context of our hierarchy of perimeter abstractions because of its ability to perform partial searches on huge abstract state spaces that cannot be explicitly enumerated. In this section we focus on how to use symbolic search on linear M&S abstractions, by representing them as BDDs, in order to be able to use symbolic search to explore perimeter abstractions with SM&S hierarchies.

5.1. Symbolic Representation of Linear M&S

In order to use M&S abstractions in symbolic search, they must be represented as BDDs. The observation that linear M&S representations are essentially equivalent to BDDs was first made by Blai Bonet at ICAPS 2008 (Malte Helmert, personal communications). This connection was further studied for the purpose of using M&S heuristics in symbolic A* search [59]. The main result is that, when using a linear merge strategy, there exists a polynomial time transformation from the M&S representation to a BDD representation. Recently, it has been proved that this is not the case when using non-linear merge strategies [60].

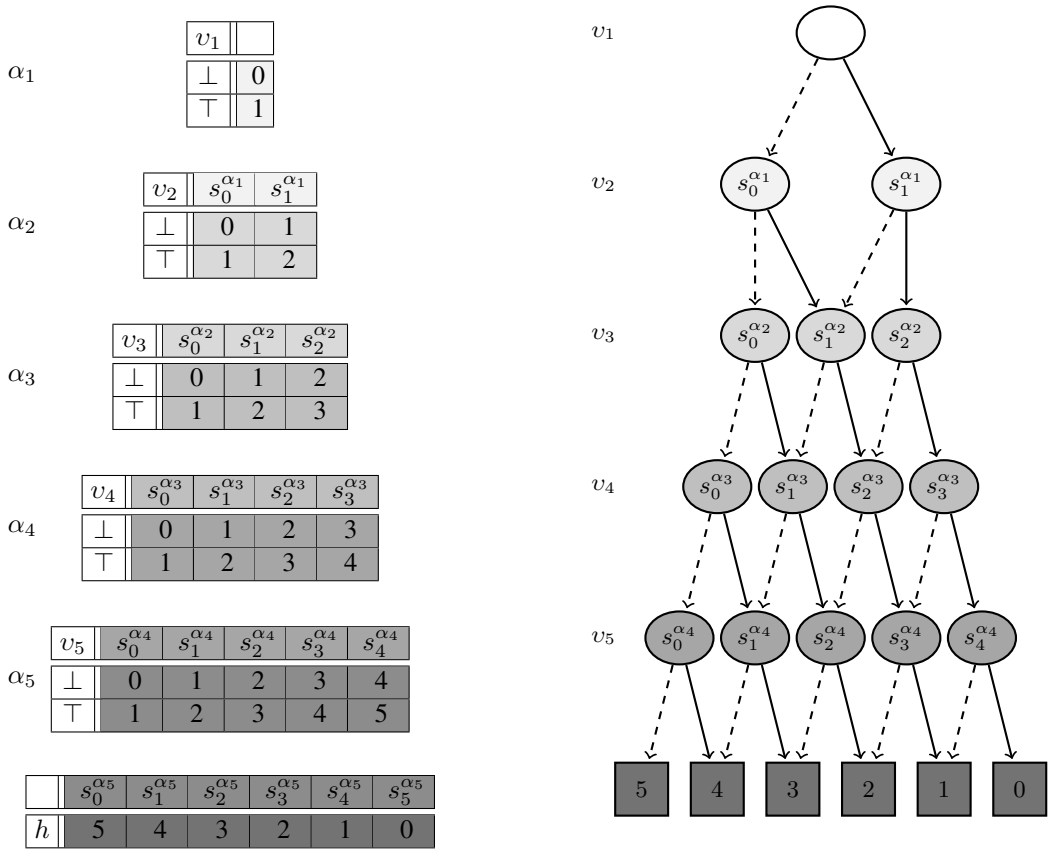
Here, we describe in detail the algorithm that transforms the *cascading-tables* representation with linear merge strategies to ADDs or BDDs and provide precise bounds on the size of the resulting data structures. The *cascading-tables* representation involves two mappings, the abstraction function $\alpha : \mathcal{S} \to \mathcal{S}^\alpha$ mapping states in $\mathcal{S}$ to *abstract states* in $\mathcal{S}^\alpha$ and the precomputed cost for the abstract states $h^*_\alpha : \mathcal{S}^\alpha \to \mathbb{R}^+_0 \cup \{\infty\}$. Combining both mappings results in the desired heuristic function, $h^\alpha : \mathcal{S} \to \mathbb{R}^+_0 \cup \{\infty\}$.

The key observation is that, when a linear merge strategy is used, the *cascading-tables* representation of an M&S abstraction can be cast as an ADD. We first develop the intuition behind this correspondence and then introduce an algorithm that computes the ADD representation of any given linear M&S abstraction in polynomial time and space in the size of the *cascading-tables*.

Figure 5 illustrates the correspondence between the *cascading-tables* representation of a linear M&S heuristic and the ADD representation with a simple example with five independent Boolean variables, v_1 to v_5 that must be made true. There is one action a_i for each variable that makes variable v_i true. The initial state is not relevant for us, since the heuristic estimates the cost to the goal from any state in the state space. We run linear M&S with bisimulation shrinking to derive the perfect heuristic, which in this case consists of counting the number of variables that remain false in the state.

Figure 5a depicts the *cascading-tables* representation of the M&S heuristic. As we are considering a linear merge ordering, at each step we add a new variable v_i. Each table corresponds to an abstraction α_i that considers i variables. Rows of the table directly correspond to different values of v_i, 1 and 0 (note that for simplicity we are omitting the tables related to atomic abstractions). Each column of a table corresponds to abstract states of previous abstractions. For example, the value 1 in the table that represents α_2 corresponds to the column $s_1^{\alpha_2}$ of the table that represents the next abstraction, α_3.

[5] Note again the relevance of Definition 6, where $open'$ preserves all the states in the open list and $closed'$ prevents states from being closed if they have not been expanded.



(a) Cascading tables M&S

(b) ADD M&S

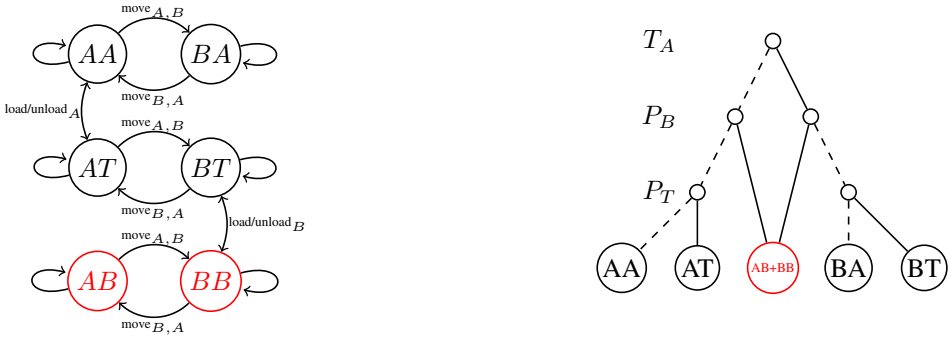
Figure 5: Relation between the *cascading-tables* and ADD representations for M&S with a linear merge strategy. Each ADD layer corresponds to one cascading table and each ADD node corresponds to a column of the table associated with its layer. The leaf nodes represent the heuristic value of the corresponding abstract state.

The table that represents each α_i encodes equivalences between pairs $\langle s_j^{\alpha_{i-1}}, v_i \rangle$. For example, in α_2, $\langle s_1^{\alpha_1}, v_2 = \bot \rangle \equiv \langle s_0^{\alpha_1}, v_2 = 1 \rangle \equiv s_1^{\alpha_2}$, i.e., $s_1^{\alpha_1}$ and $s_0^{\alpha_1}$ are considered equivalent by α_2. In our example, each state $s_i^{\alpha_j}$ corresponds to having i variables true and the remaining $(j - i)$ variables false. In the last layer, the heuristic values of each abstract state $s_i^{\alpha_5}$ correspond to counting the number of variables that remain false, $(5 - i)$.

Figure 5b depicts the corresponding ADD representing the heuristic h^{α_5}. The figure is organized to highlight the similarity between both representations. Every level of the ADD corresponds to an intermediate abstraction α_i. Each abstract state of the intermediate abstraction is represented with an ADD node on the corresponding ADD level. To highlight this correspondence, ADD nodes are labeled in the figure with the abstract state they correspond to. In the last level, terminal ADD nodes correspond to the heuristic value of abstract states of the final M&S abstraction, $s_i^{\alpha_5}$. The tables in the *cascading-tables* representation are just an alternative to represent the ADD edges, i. e., the mapping from ADD nodes in one layer to the next one. $s_1^{\alpha_2}$ has two incoming edges: a 1-edge from $s_0^{\alpha_1}$ and a 0-edge from $s_1^{\alpha_1}$. These two edges correspond to the two cells with value 1 in the table representing α_2, since 0-edges correspond to $v_i = 0$ and 1-edges to $v_i = 1$.

To build the analogy between the M&S construction process and an ADD, we can describe the two operations in M&S (merge and shrink) in terms of the modifications they imply in the ADD representation. As reflected in Figure 5, merging a new variable to a M&S abstraction corresponds to adding a new layer to the ADD. The shrinking operation that aggregates several abstract states into one, corresponds to the application of ADD reduction rules. Figure 6 shows an example where two abstract states, AB and BB, are shrunk. After applying shrinking, both AB and BB become

equivalent in the abstract state space, so that they will be assigned the same heuristic value. In the corresponding ADD representation, this means that the nodes that represented AB and BB will become equivalent according to the reduction rules, because they represent exactly the same function. In Figure 6b the assignments $T_A \wedge P_B$ and $\neg T_A \wedge P_B$ point both to the same node.



(a) Shrinking in M&S.

(b) Reduction rule in ADD.

Figure 6: Correspondence between shrinking in M&S and ADD reduction rules. A task with a package, a truck, and two locations, A and B. The two abstract states in which the package is at B, AB and BB, are shrunk and get represented by the same ADD node, $AB + BB$.

The example of Figure 5b is very convenient to highlight the correspondence between columns/cells of each cascading table and ADD nodes/edges, exploited by Theorem 2 to prove a bound on the number of ADD nodes required for representing the abstraction.

Theorem 2. *Let $\alpha_1, \ldots, \alpha_n$ be a list of M&S abstractions generated with a linear merge strategy $v_1, \ldots, v_n$ such that $\alpha_{i+1} = \gamma \circ (\alpha_i \otimes \pi_{i+1})$. Then, the symbolic ADD representation of the function $\alpha_n : S \to S^{\alpha_n}$ under variable ordering $v_1, \ldots, v_n$ has at most $\sum_{i=0}^{n-1} |\alpha_i|(|D_{v_{i+1}}| - 1)$ nodes.*

Proof. Our bound comes from the sum of the nodes of each layer l, which is bounded by $|\alpha_l|(|D_{v_{l+1}}| - 1)$. Layer l has at most one node for each abstract state in $|\alpha_l|$, plus intermediate nodes to connect with layer $l + 1$, if the variable v_{l+1} has more than two values.

To discern between all the values of a variable v_l, we need to build a binary tree with at most $|D_{v_l}| - 1$ nodes, so in the worst case the number of nodes employed in layer l is $|\alpha_l|(|D_{v_{l+1}}| - 1)$. □

Given the one-to-one correspondence between M&S abstract states and ADD nodes, the algorithm to obtain the ADD representation of an M&S heuristic with a linear merge strategy is straightforward. Algorithm 3 computes the ADD from the *cascading-tables* representation of an M&S heuristic. The algorithm computes an ADD node for each abstract state of all intermediate abstractions in a bottom-up approach (as decision diagrams are usually built). ADD nodes are built with two functions: `ADD-constant` and `ADD-value`. `ADD-constant` generates a terminal node associated with an integer constant. `ADD-value` receives a variable v_l and a value in the domain of the variable $d \in D_{v_l}$ and returns an ADD that represents a function f such that $f(d) = 1$ and $f(d') = 0$ for all $d' \in D_{v_l}, d' \neq d$.

The algorithm maintains a matrix named ADD with one entry per abstract state in every intermediate abstraction. Let $\text{ADD}_{i,j}$ denote the ADD node that corresponds to the abstract state $s_j^{\alpha_i}$. Since several abstract states may be represented with the same node, the ADD matrix stores only a reference to the nodes[6].

First, the algorithm generates one terminal ADD node per abstract state in the last layer, with their heuristic value (line 2). Again, the same node will be used for states with the same heuristic value (their entries in the ADD matrix reference to the same ADD node).

Then, nodes in the previous layer can be constructed pointing to nodes in layers previously computed. A loop iterates over all layers in a bottom-up fashion (line 3). To construct the next layer of the ADD, the algorithm iterates

[6]Note that we do not explicitly check the ADD reduction rules, since the CUDD BDD library [61] automatically applies them (e. g., all calls to `ADD-constant(`∞`)` will always return a reference to the same node).

Algorithm 3: M&S Heuristic to ADD

Input: A list of cascading tables $T_1, \ldots, T_n$ and heuristic values of α_n, $h^*_{\alpha_n}$
Output: The ADD that represents $h^*_{\alpha_n}$

1 **for** abstract state $s_i^{\alpha_n} \in \alpha_n$ **do**
2 $\quad \text{ADD}_{n,i} \leftarrow \texttt{ADD-constant}\,(h^*_{\alpha_n}(s_i^{\alpha_n}))$
3 **for** layer $l \in [n-1 \ldots 0]$ **do**
4 $\quad$ **for** abstract state $s_i \in \alpha_l$ **do**
5 $\quad\quad$ **if** $T_l[s_i] = \mathit{pruned}$ **then**
6 $\quad\quad\quad \text{ADD}_{l,i} \leftarrow \texttt{ADD-constant}\,(\infty)$
7 $\quad\quad$ **else**
8 $\quad\quad\quad \text{ADD}_{l,i} \leftarrow \sum_{d \in v_l} \left(\texttt{ADD-value}\,(v_l, d) \times \text{ADD}_{l+1,T_l[i,d]}\right)$
9 **return** $\text{ADD}_{0,0}$

over all abstract states in the corresponding abstraction (line 4). For every abstract state in the current layer, the cascading table is used to know which nodes must be pointed to from the newly generated node. Nodes corresponding to states pruned by M&S are directly assigned a constant heuristic value of ∞. Otherwise, the node representing the state corresponds to the function $\sum_{d \in v_l} \texttt{ADD-value}\,(v_l, d) \times \text{ADD}_{l+1,T_l[i,d]}$. This function just returns a node that points to the node indicated by the cascading table ($T_l[i, d]$) for every value d of the variable v_l.

In this expression, sum ($+$) and multiplication ($\times$) operations over ADDs that represent 0-1 functions are equivalent to disjunction ($\vee$) and conjunction ($\wedge$) over BDDs, respectively. Thus, the multiplication $\texttt{ADD-value}\,(v_l, d) \times \text{ADD}_{l+1,T_l[i,d]}$ sets the node that represents the abstract state indicated by the cascading table $T_l[i, d]$ as the value assigned to value d. The sum can be interpreted as a disjunction over all values of the variable. The simpler example is when v_l is a binary variable, since the result of `ADD-node` is a node whose 1-edge points to $\text{ADD}_{l+1,T_l[i,1]}$ and whose 0-edge points to $\text{ADD}_{l+1,T_l[i,0]}$. If v_l has more than two possible variables, intermediate auxiliary nodes are needed. When all layers have been constructed, the algorithm ends returning the reference to the root node of the ADD, $\text{ADD}_{0,0}$.

Algorithm 3 can be adapted to compute the ADD representation of the M&S abstraction function (that maps states in S to abstract states in S^α_n) by assigning to terminal nodes the id of each abstract state instead of their heuristic value.

Corollary 3. *Let $\alpha_1, \ldots, \alpha_n$ be a list of M&S abstractions generated with a linear merge strategy $v_1, \ldots, v_n$ such that $\alpha_{i+1} = \gamma \circ (\alpha_i \otimes \pi_{i+1})$. Let B be the bound on the number of ADD nodes established by Theorem 2, $B = \sum_{i=0}^{n-1} |\alpha_i|(|D_{v_{i+1}}| - 1)$. Then Algorithm 3 computes the ADD representation of the M&S heuristic, h^{α_n}, in time* $\mathrm{O}\,(B \log(B))$.

Proof. Algorithm 3 computes the ADD representation of the M&S heuristic by creating the nodes iteratively, so that it suffices to sum the time needed to generate each node. Every ADD node is constructed with a call to `ADD-constant` (which is performed in constant time) or with the standard ADD sum and product operations in the expression $\sum_{d \in v_l} \left(\texttt{ADD-value}\,(v_l, d) \times \text{ADD}_{l+1,T_l[i,d]}\right)$. Since the ADDs describing values d of a variable, $\texttt{ADD-value}\,(v_l, d)$, are all disjoint, these operations can be performed in time proportional to the number of created nodes. Thus, a constant number of operations is needed for each node, plus a lookup in the table of nodes to ensure the ADD reduction rules. Therefore, at most B nodes have to be constructed and each node takes $\mathrm{O}\,(\log(B))$ time. □

5.2. Symbolic Merge-and-Shrink Hierarchies

The objective of using symbolic search in M&S is to derive more accurate heuristics by searching larger abstract state spaces that do not need to be explicitly represented. However, these larger state spaces cannot be directly obtained with the M&S algorithm, because the size of M&S abstract state spaces must remain small enough to explicitly represent them. Instead, we define SM&S state spaces which are derived from M&S abstractions without requiring their explicit representation. SM&S abstractions result of merging several intermediate M&S abstractions.

In particular, we focus on linear merge strategies, where at any point during the M&S procedure there is a single non-atomic M&S abstraction plus the atomic-abstractions of the remaining variables.

Definition 9 (SM&S abstraction)**.** *Let Π be a planning task and let α be an abstraction with relevant variables $V_\alpha \subseteq V$. Let $\Pi_{V \setminus V_\alpha}$ be the projection of Π over the non-relevant variables of α, $V \setminus V_\alpha$. The SM&S abstraction $\alpha^{\text{SM\&S}}$ associated to α is defined as the synchronized product of α and $\Pi_{V \setminus V_\alpha}$: $\alpha^{\text{SM\&S}} = \alpha \otimes \Pi_{V \setminus V_\alpha}$.*

Intuitively, M&S abstractions partially consider a subset of variables (the *relevant* variables) and completely ignore the rest. The SM&S abstraction derived from an M&S abstraction considers the same information about the relevant variables, but instead of ignoring all the other variables, they are fully considered (no abstraction is made over those variables). In other words, the SM&S abstraction is a refinement of the M&S abstraction. For example, consider a typical logistics task with several packages and trucks. An M&S abstraction α that considers only two packages may reduce the abstraction size by considering some combinations to be equivalent. For example, $(p_1 = G \wedge p_2 = T) \equiv (p_1 = T \wedge p_2 = G)$ or $(p_1 = A \equiv p_1 = B)$. However, α completely ignores all the information regarding the location of the trucks or the other packages. $\alpha^{\text{SM\&S}}$, on the other hand, fully considers the location of all the trucks, yet it is still an abstraction of the original task since it also applies the same equivalences than α for packages p_1 and p_2.

Definition 9 can be applied for any intermediate abstraction in the M&S algorithm. Next, we consider the abstraction hierarchy that results from computing the SM&S abstraction of every intermediate M&S abstraction.

Definition 10 (SM&S abstraction hierarchy)**.** *Let $\alpha_0, \ldots, \alpha_n$ be all the intermediate abstractions generated by the M&S procedure, where α_0 is the empty abstraction without any variable and α_n is the final result of M&S. We define the corresponding SM&S hierarchy as the list of SM&S abstractions: $\alpha_0^{\text{SM\&S}}, \ldots, \alpha_n^{\text{SM\&S}}$.*

Figure 7 shows an example of how the SM&S hierarchy is obtained from an M&S procedure. The left part shows the intermediate abstractions generated during the M&S algorithm. The M&S algorithm is initialized with the atomic abstraction of each variable, π_i. In our example, there are five variables and, consequently, there are five atomic abstractions, $\pi_1, \ldots, \pi_5$. The M&S algorithm iteratively merges two abstractions by computing their synchronized product and, if necessary, applies shrinking. In our example, a linear merge strategy is used, so that variables are iteratively merged into the main abstraction. It starts merging variables π_1 and π_2 into α_1 and iteratively includes more variables into each intermediate abstraction α_i. The induced SM&S abstractions always consider all variables, as a result of merging all variables into each α_i without applying any additional shrinking. Whereas the α_i's usually become finer with increasing i, the $\alpha_i^{\text{SM\&S}}$'s become coarser with increasing i.

The SM&S hierarchy defines n state spaces, resulting in a trade-off between the size of the abstract state space and heuristic accuracy. $\alpha_0^{\text{SM\&S}}$ corresponds to the original non-abstracted state space, which has exponential size and is intractable to represent, but produces the perfect heuristic. On the other hand, $\alpha_n^{\text{SM\&S}}$ corresponds to the final M&S abstract state space, $\alpha_n^{\text{SM\&S}} \equiv \alpha_n$, whose state space can be explicitly represented (assuming a suitable value of the maximum number of M&S abstract states) possibly with a great loss of information. This is reflected in the example of Figure 7 since $\alpha_0^{\text{SM\&S}}$ is the merge of all the variables and $\alpha_4^{\text{SM\&S}}$ is α_4. All the rest of SM&S abstractions in the hierarchy, $\alpha_i^{\text{SM\&S}}, 0 < i \leq n$, enable the desired trade-off between $\alpha_0^{\text{SM\&S}}$ and $\alpha_n^{\text{SM\&S}}$. Each $\alpha_i^{\text{SM\&S}}$ is strictly more relaxed than the previous $\alpha_{i-1}^{\text{SM\&S}}$, so that the size of the abstract state spaces decreases at the expense of producing less informed heuristics $h^{\alpha_{i-1}^{\text{SM\&S}}}(s) \geq h^{\alpha_i^{\text{SM\&S}}}(s)$.

Note that the size of SM&S abstract state spaces may be of exponential size, so that they cannot be explicitly represented. This is not a problem, since we will directly traverse those state spaces using symbolic search. In order to perform symbolic search over the abstract state spaces, we need to represent sets of abstract states and to perform the successor generation efficiently. Below, we describe the encoding that we use and analyze its theoretical properties.

5.2.1. SM&S Abstract State Representation

In this section we study the symbolic representation of SM&S abstractions, i. e., how to represent sets of abstract states by means of BDDs. In the original state space, sets of states are represented as functions $f : \mathcal{S} \to \{1, 0\}$, in terms of binary variables $x = x_1, \ldots, x_n$. In the same way, sets of abstract states in $\alpha_i^{\text{SM\&S}} = \alpha_i \otimes \Pi_{v_{i+1}, \ldots, v_n}$ are represented as characteristic functions $f : \mathcal{S}^{\alpha_i^{\text{SM\&S}}} \to \{1, 0\}$. An important decision is what variables (and variable ordering) should be used to represent sets of abstract states in $\mathcal{S}^{\alpha_i^{\text{SM\&S}}}$. We decided to use the same set of variables that is used to represent the original state space. An alternative could be to design a new set of auxiliary variables y,

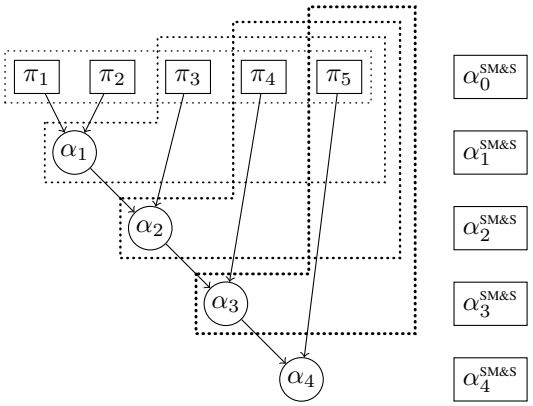


Figure 7: Hierarchy of Symbolic M&S state spaces. Atomic abstractions (π_i) are merged in a linear ordering to derive different M&S heuristics (α_i). Dotted lines show the combination of several abstractions that result in each $\alpha_i^{\text{SM\&S}}$.

optimized to represent abstract states, replacing variables $v_1, \ldots, v_i$ by another set of variables $y_1, \ldots, y_k$ specifically designed to represent the abstraction. While this alternative could help to perform the abstract search more efficiently, encoding abstract states with the same variables lets us use the same representation for searches in any state space. Thus, using the same set of variables for all the abstract states makes the conversion between original states and abstract states easier. Moreover, the same BDD can be interpreted as a set of abstract states or a set of concrete states; i. e., the set of all states mapped to some abstract state in the set.

Figure 8 shows an example of the symbolic M&S relaxation of a search and how a BDD can be interpreted both as a set of states and a set of abstract states. Given a set of states in the original state space, depicted in Figure 8a, and an intermediate M&S abstraction α_2, depicted in Figure 8b, we derive the corresponding set of states $\alpha_2^{\text{SM\&S}}(S)$. S is a set of four states, namely 00000, 01100, 10110 and 11010. α_2 is an M&S abstraction over variables v_1 and v_2 with two abstract states e_0 and e_1. In general, our M&S abstractions will have a larger (but bounded) number of abstract states. The ADD depicted in Figure 8b represents the mapping from partial states $\langle v_1, v_2 \rangle$ to one of the abstract states, e_0 or e_1. Each abstract state corresponds to an equivalence relation over variables v_1 and v_2. e_0 makes starting with 00 been equivalent to start with 11, so that if a set of abstract states contains state $00abc$, it automatically contains $11abc$ as well. Whenever any state starting with 00 is reached in the abstract search, the corresponding state starting with 11 will be automatically reached and vice versa. e_1 makes 10 and 01 equivalent in a similar way.

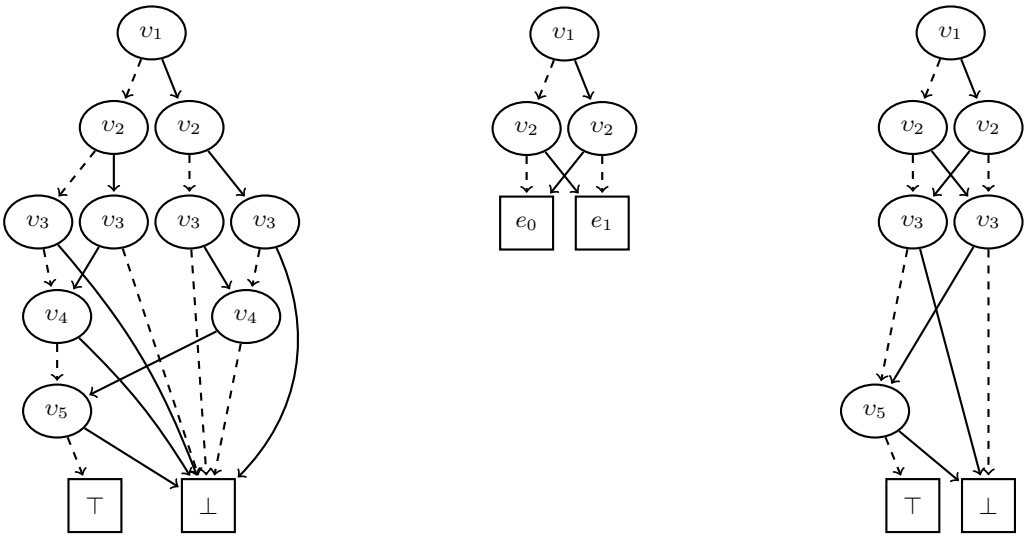
Figure 8c shows the BDD that represents the set of abstract states $\alpha_2^{\text{SM\&S}}(S)$, containing four abstract states ($000x0$, $011x0$, $110x0$, and $101x0$) or eight concrete states (after substituting x by 0 or 1), depending on the interpretation. An important point is that BDD nodes pointed to by the paths 00 and 11 are equivalent. A similar reasoning can be made for every abstract state e_i making the top part of any BDD describing abstract states at most as large as the ADD representation of the M&S abstraction.

The example shown in Figure 8 is an ideal case, where not only the layers corresponding to α_2 are reduced, but also other layers get simplified. Opposite examples can be defined where the number of BDD nodes required to represent $\alpha_i^{\text{SM\&S}}(S)$ is exponentially larger than the number of nodes to describe S. However, the correspondence between the ADD that describes the M&S abstraction α_2 and the upper levels of any BDD describing a set of abstract states lets us prove some bounds on the size of BDDs describing any set of abstract states.

Proposition 4. *Let α_k be an M&S abstraction over relevant variables $v_1, \ldots, v_k$ with M abstract states. Let $S^{\alpha_k^{SM\&S}}$ be a set of abstract states in $\Theta^{\alpha_k^{SM\&S}}$. Then, the layer $k+1$ of the BDD representation of $S^{\alpha_k^{SM\&S}}$, under variable ordering starting by $v_1, v_2, \ldots, v_k$, has at most M BDD nodes.*

Proof. Each node in layer $k+1$ corresponds to one or more abstract states in α_k. Suppose there is a node n' that does not correspond to any abstract state. Let $d_1, \ldots d_k$ be an assignment to variables $v_1, \ldots, v_k$ that leads from the root of the BDD to n'. By definition, all assignments are related to a unique abstract state, so we reach a contradiction.

Let $s^{\alpha_k^{\text{SM\&S}}}$ be the abstract state corresponding to the partial state $d_1, \ldots d_k$. By definition of the abstract state space, $d_1, \ldots d_k$ is indistinguishable of all the other assignments mapped to $s^{\alpha_k^{\text{SM\&S}}}$, i.e., they all have the same goal



(a) Original BDD S (b) Abstraction α_2 ADD (c) $\alpha_2^{\text{SM\&S}}(S)$ BDD

Figure 8: Symbolic M&S relaxation that takes an input BDD, S, and an ADD that represents a M&S heuristic and computes the BDD that represents the abstract states in S. The top part of $\alpha_2^{\text{SM\&S}}(S)$ BDD coincides with α_2 ADD.

distance in the abstract state space. Therefore, node n' represents not only $d_1, \dots d_k$ but also all the other assignments that are mapped to $s^{\alpha_k^{\text{SM\&S}}}$ by $\alpha_k^{\text{SM\&S}}$. Thus, the number of nodes in layer k is bounded by the number of abstract states, which is itself bounded by M.

□

Proposition 4 bounds the size of a single layer of the BDD describing a set of abstract states in $\Theta^{\alpha_k^{\text{SM\&S}}}$. However, this result can easily be extended to all the upper layers in the BDD. As we are assuming a linear merge strategy, all the M&S abstractions are built as the merge of an M&S abstraction and the atomic abstraction of another variable. Therefore, the bound of Proposition 4 is valid for all the upper levels of the BDD.

Theorem 5. *Let $M_1, \dots, M_k$ be the number of abstract states of k intermediate abstractions $\alpha_1, \dots, \alpha_k$ of an M&S algorithm with a linear merge strategy, such that $\alpha_i = \gamma_i \circ (\alpha_{i-1} \otimes \pi_i)$. Let $S^{\alpha_k^{SM\&S}}$ be a set of abstract states in $\Theta^{\alpha_k^{SM\&S}}$. Then, the BDD representation of $S^{\alpha_k^{SM\&S}}$, under variable ordering starting by $v_1, v_2, \dots, v_{k+1}$, is represented with at most $(\sum_{i\in[1,\dots,k]} M_i) + M_k 2^{n-k+1}$ BDD nodes.*

Proof. The BDD that represents $S^{\alpha_k^{\text{SM\&S}}}$ may be divided in its top and bottom parts. The top part includes layers 1 to k and the bottom part the remaining $(n-k)$ layers. From Proposition 4, we can bound the size of each top layer to M_i nodes, so that the top part of the BDD uses, at most, $\sum_{i\in[1,\dots,k]} M_i$ nodes. The bottom layers correspond to functions over the remaining $(n-k)$ variables. Each of these functions is described as a BDD with $(n-k)$ levels. In the worst case, they do not share any node, and thus each one has 2^{n-k+1} nodes. Since there are M_k different functions, the size of the bottom part of S_B is bounded by $M_k 2^{n-k+1}$. □

5.2.2. SM&S Transition Representation

Once we have defined the symbolic representation of sets of abstract states, in order to perform a symbolic search, we need a reliable way to perform successor generation. Planning actions are represented by means of one or more transition relations (TRs). As a brief recap, each TR is a function $f : S \times S' \to \{1, 0\}$ that relates predecessor to successor states, i. e., $f(s, s')$ is true if and only if there is a transition from s to s'. Predecessor states are represented with the standard set of variables x, and an auxiliary set of variables x' represents the successor states. The variable ordering is also very important for the performance of image computation. Usually, variables in x and x' are interleaved in the following manner: $x_1, x'_1, x_2, x'_2, \dots x_n, x'_n$. This variable ordering exploits the fact that actions usually affect a few variables so that most variables preserve their previous value.

In SM&S abstract state spaces, however, the transitions are different than in the original state space. Moreover, in a given $\alpha_i^{\text{SM\&S}}$, variables $v_1, \ldots, v_i$ are highly related and the TR representation can be very complex. Therefore, contrary to our state representation, it is not possible to preserve the same variable ordering as for the TRs of the original state space.

We avoid the problem of representing SM&S TRs by using the TRs of the original state space. This is possible since the set of abstract states may be interpreted as a set of non-abstracted states. Of course, the result of the image operation is a set of states of the original state space. Fortunately, we can apply the abstraction function to obtain the set of abstract states associated with them. The *image* and *pre-image* operations are applied as $image(S^{\alpha_i^{\text{SM\&S}}}, T^{\alpha_i^{\text{SM\&S}}}) = \alpha_i^{\text{SM\&S}}(image(S^{\alpha_i^{\text{SM\&S}}}, T))$. The details of this operation are described in the next section.

Thus, we do not need to compute a new set of TRs for every abstract state space we traverse. However, using the original TRs also has some drawbacks. In particular, the intermediate BDDs may induce a large overhead. Even in the case where the sets of abstract states are guaranteed to be efficiently representable, the intermediate set of original states does not have any guarantee with respect to its size.

5.3. Frontier Shrinking

One key aspect of perimeter abstraction heuristics is that, instead of restarting abstract searches from scratch, they restart the search from the current frontier by relaxing the open and closed lists. Thus, in order to initialize the abstract search, they have to perform the mapping from states in the original state space to abstract states, according to Definition 6. In this section, we consider how to initialize abstract searches efficiently when using a symbolic representation of the sets of states involved in the searches. We call this operation "shrinking the frontier" because we expect the frontier of the abstract state space to be more compact according to Theorem 5.

According to Definition 6, there are two different types of frontier shrinking operations. *Existential shrinking* is used to shrink the open list and *universal shrinking* is the operation applied to the closed list. Both take as input a set of states S_B and an abstraction function α and compute the set of abstract states that will be used to initialize the open and closed list, $S_B^{\exists\alpha}$ and $S_B^{\forall\alpha}$, respectively. Next, we describe how to implement both shrinking operations when the sets of states are described with BDDs and our abstraction is a PDB or a SM&S abstraction.

5.3.1. Frontier Shrinking in PDBs

PDB abstractions are characterized by a set of variables V_α, so that the value of the remaining variables is completely ignored. Implementing the existential and universal shrinking operations in this setting, reduces to applying the standard existential/universal quantification over variables that are not in V_α, as formulated in Equations 1 and 2.

$$S_B^{\exists\alpha} = \exists_{V \setminus V_\alpha} S_B \tag{1}$$

$$S_B^{\forall\alpha} = \forall_{V \setminus V_\alpha} S_B \tag{2}$$

5.3.2. Frontier Shrinking in SM&S

In SM&S abstractions, like in PDBs, we can distinguish between relaxed and non-relaxed variables. However, instead of completely ignoring the relaxed variables, SM&S uses an M&S abstraction over the relaxed variables to keep some information about them. Therefore, we describe the SM&S abstraction in terms of the set of relaxed variables V_α (the relevant variables of the M&S abstraction) and the set of equivalences induced by the M&S abstraction, $\sim^\alpha$. Recall that each abstract state s^α is a set of states such that every pair of states $s_1, s_2 \in s^\alpha$ are equivalent, $s_1 \sim^\alpha s_2$. Equations 3 and 4 show how to compute the existential and universal shrinking of a set of states S_B given a BDD representation of S_B and each s^α:

$$S_B^{\exists\alpha} = \bigvee_{s^\alpha \in S^\alpha} ((\exists V_\alpha \, (S_B \wedge s^\alpha)) \wedge s^\alpha) \tag{3}$$

$$S_B^{\forall\alpha} = \bigvee_{s^\alpha \in S^\alpha} ((\forall V_\alpha \, ((S_B \wedge s^\alpha) \vee \neg s^\alpha)) \wedge s^\alpha) \tag{4}$$

As in the case of PDBs, both operations use the existential/universal BDD quantification of the abstracted variables. However, in this case, we must iterate over all the abstract states in S^α to keep their information. For each abstract state, s^α, we compute its existential shrinking in three consecutive steps:

1. $S_B \wedge s^\alpha$ is the subset of S_B which corresponds to any state in s^α. However, in our final result all partial states mapped to s^α must be indistinguishable.

2. Existential quantification of V_α gets all the assignments to variables $V \setminus V_\alpha$ that have been reached for any $s \in s^\alpha$. That way, we keep the values of other variables while ignoring the relaxed variables, just like in the existential shrinking of PDB abstractions.

3. Finally, we compute the conjunction with s^α to reset the value of V_α to only states in s^α.

As an example, take Figure 8 on page 20. Abstract state e_0 represents the partial states 00 and 11. Our first step, gets all the states in S_B that fit that description: 00000 and 11010. Then, the existential quantification forgets the value of the relaxed variables, obtaining the set of states $xx000$ and $xx010$. Finally, those assignments are valid for e_0, so we end with four states after our relaxation: 00000, 00010, 11000 and 11010.

The case of universal shrinking is similar, though in this case the second step uses universal quantification to get only those assignments to non-relaxed variables that are true for all partial assignments mapped to s^α. Moreover, a disjunction with $\neg s^\alpha$ is necessary to ignore the values reached with other abstract states in the universal quantification.

6. The SPA Heuristic

In the previous sections, we have presented an abstraction hierarchy based on M&S abstractions and a new model of perimeter abstraction heuristics that can exploit it. In this section, we mix those ingredients up and propose the Symbolic Perimeter Abstraction heuristic, SPA. The main purpose of this section is to describe the design and implementation decisions that remain unsettled and are needed in order to have a practical implementation of the theoretical results of the previous sections.

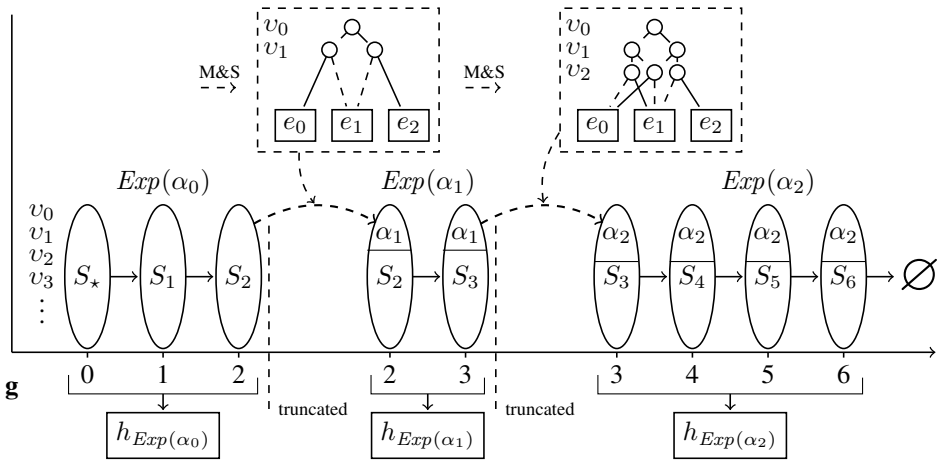
6.1. The SPA Algorithm

SPA performs symbolic regression and uses M&S or PDB abstractions to progressively relax the search whenever it becomes unfeasible. Figure 9 depicts a high level view of the interaction between symbolic search and abstractions, highlighting the differences between M&S and PDBs.

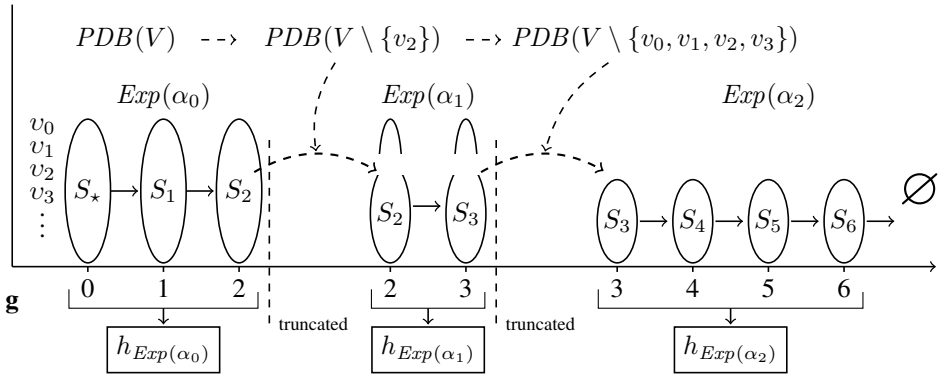
Each diagram is divided into four parts: the upper part depicts the M&S abstractions that are used to relax the search; the middle part depicts the BDDs involved in the search; the x-axis represents the search progress with different layers labeled with their g-value; and below the axis we show the resulting heuristics of each search.

SPA starts computing a symbolic perimeter, $Exp(\alpha_0)$. This perimeter corresponds to a regression search in the original state space, so that no abstraction is used to relax the search yet. The first BDD with $g = 0$ contains the goal states and by successive $pre\text{-}image$ operations SPA performs a standard symbolic regression search. As this search is intractable for general planning domains, the search will surpass in most instances the predetermined memory or time bounds given for the precomputation of the heuristic. Then, the search is truncated (at $g = 2$ in Figure 9). The minimum cost to the goal of the expanded state sets is stored, transforming the list of BDDs representing the search to an ADD representing the heuristic, $h_{Exp(\alpha_0)}$. Up to this point, we have just generated a symbolic perimeter with symbolic regression search, and no abstractions have been used yet. Then, an abstraction is derived by using either M&S or PDBS. SPA initializes $Exp(\alpha_1)$ by relaxing the $open$ and $closed$ lists of the previous search, using the existential and universal shrinking operations described in Section 5.3. The diagrams show the main differences between SPM&S and SPPDB, as described below.

In Figure 9a, M&S merges variables, applying shrinking if needed to fit the maximum number of abstract states (with a limit of six abstract states, α_1 must have at most three abstract states before merging the next variable). The top levels of the BDDs in the $open$ or $closed$ list of the new relaxed search, $Exp(\alpha_1)$, are at most as large as the ADD that represents the M&S abstraction, α_1, depicted in the upper part of the diagram. This is due to the fact that all partial states related to the same abstract state are considered equivalent, so that when one is reached, all of them are, as explained in Section 5.2.1. For example, in Figure 9a, abstract state e_1 represents partial states 00 and 10. During the exploration, if state $10010\ldots$ is reached, then state $00010\ldots$ is also reached and vice versa. Hence, BDD nodes pointed to by 00 and 10 are equivalent, making the top part of any BDD in the exploration equal to the ADD representation of α_1. Also, M&S abstractions are cumulative, so the top levels of α_2 coincide with those of α_1.



(a) SPM&S with a limit of six abstract states for M&S. In abstract searches, the top part of each BDD corresponds to the ADD that represents the M&S used to relax the search.



(b) SPPDB with pattern selection following variable ordering $v_2, v_0, v_1, v_3, v_4, \ldots$.

Figure 9: SPA example with M&S and PDB abstractions. Unit action costs are assumed. Ellipses represent sets of states in the symbolic regression searches.

In Figure 9b, v_2 is removed from the pattern. As PDBs completely ignore variables outside the pattern, BDDs in the search of SPPDB will only contain nodes related to variables in the pattern, so BDDs in $Exp(\alpha_1)$ do not have any node related to v_2. However, SPM&S is forced to relax variables from the top of the PDBs one by one, whereas SPPDB may choose any variable to relax.

With the relaxation induced by the abstraction the search continues but, after a number of steps, it may become unfeasible again (according to the specified time/memory bounds). SPA continues then interleaving symbolic explorations and M&S/PDB iterations until the exploration is completed or the time/memory bounds are reached again. When finished, it returns the list of ADDs representing the heuristic values computed so far. Once the heuristic is computed as a list of ADDs, it can be used in the search as stated in Definition 8 in order to solve the planning instance.

Algorithm 4 shows the SPM&S algorithm, i. e., the version of SPA that uses M&S abstractions to relax the search frontier. It receives as input a planning task Π and some parameters to bound the memory and time resources. The output is a heuristic H represented as a list of ADDs. Each ADD is the result of a backward symbolic exploration over an M&S abstraction α. If SPM&S exceeds the time limit $T_{\text{SPM\&S}}$ a last ADD is included (line 16). This last ADD is the standard M&S heuristic, computed as usual with an explicit traversal of the abstract state space. SPM&S starts initializing the symbolic backward search to $(open = S_\star, closed = \varnothing, d = 0)$ and α is empty. Symbolic search progresses following the relaxation imposed by α (line 7).

Several parameters bound the memory and time used by the algorithm. Memory is controlled by the maximum

Algorithm 4: Symbolic Perimeter Merge-and-Shrink

Input: Planning problem: $\Pi = \langle V, A, I, G \rangle$
Input: Memory bounds: N, N_F
Input: Time bounds: $T_{SM\&S}$, T_{Sym}, T_{Exp}, T_I
Output: List of ADDs: H

1 $H \leftarrow \emptyset$
2 $abs \leftarrow \{\pi_v \mid v \in V\}$
3 $\alpha \leftarrow \emptyset$
4 $(open, closed) \leftarrow (S_\star, \emptyset)$
5 **while** $open \neq \emptyset$ and $t_s < T_{SM\&S}$ **do**
6 **if** $|frontier(open)| \leq N_F$ and $t_s < T_{Sym}$ **then**
7 `BackwardUniformCostSearch` $(\alpha, open, closed, N_F, T_{Exp}, T_I)$
8 $H \leftarrow H \cup$ `ADD`$(closed, open)$ /* Insert heuristic ADD in H */
9 `Select` $(\pi_v \in abs)$ /* Select next atomic abstraction */
10 $abs \leftarrow abs \setminus \pi_v$ /* Remove atomic abstraction from abs */
11 $\alpha \leftarrow$ `Shrink`$(\alpha, \frac{N}{size(\pi_v)})$ /* Get shrinking equivalence */
12 $open \leftarrow \{(g, open_g^{\exists\alpha})$ for all $open_g \in open\}$ /* Shrink search frontier */
13 $closed \leftarrow \{(g, closed_g^{\exists\alpha})$ for all $closed_g \in closed\}$ /* Shrink closed list */
14 $\alpha \leftarrow \alpha \otimes \pi_v$ /* Merge next variable */
15 **if** $open \neq \emptyset$ **then**
16 $H \leftarrow H \cup$ `Explicit-Search`(α)
17 **return** H

number of M&S abstract states, N, and the maximum number of nodes, N_F, to represent the search frontier. Four different parameters bound the time allotted to the algorithm. $T_{\text{M\&S}}$ aborts the heuristic computation to guarantee termination. T_{Sym} prevents SPA from performing more symbolic explorations to focus on completing the M&S abstraction. T_{Exp} fixes a maximum time for each individual exploration to avoid consuming all the time in one single exploration. Finally, T_I limits the maximum time employed in one $pre\text{-}image$. If T_I is exceeded, not only the $image$ but the whole exploration is halted. In order to avoid starting another $image$ as hard as the halted one, the maximum number of nodes in the frontier search is reduced to $N_F = \frac{|S_e|}{2}$. All these parameters are set independently of the domain and only depend upon the memory and time resources available to the planner.

The `BackwardUniformCostSearch` procedure performs a symbolic backward uniform-cost search over $\alpha^{\text{SM\&S}}$, stopping when *open* is empty, the search frontier is larger than N_F, an image takes longer than T_I, or the time limit of T_{Exp} seconds has been exceeded. The heuristic derived from this search is stored in H as an ADD (line 8). When the search is truncated because one of the bounds was reached, the current abstraction is substituted, merging a new variable and shrinking if needed. After shrinking the abstraction, we also shrink the search frontier, $open$ and $closed$, in order to continue the search. Note that, after shrinking the search frontier with the new abstraction, the search only continues if the relaxed frontier has an acceptable size. SPM&S keeps including variables into the abstraction until the search is small enough, i. e., the relaxed frontier is smaller than the parameter N_F. The use of perimeter search prevents SPA from starting a new exploration per merged variable, which might lead to redundant work in case that most searches were truncated at the same frontier cost.

SPPDB follows the same pseudocode, except that variables are completely abstracted away in line 11 and they can be selected in any order in line 14.

6.2. Theoretical Properties

We conclude with a recapitulation of the theoretical results that apply to the h_{SPA} heuristic.

Corollary 6. *The h_{SPA} heuristic is admissible and consistent.*

Proof sketch. It follows from Theorem 1, since h_{SPA} is a particular instance of perimeter abstraction heuristic. □

One of the advantages of M&S and PDBs is that it is possible to control its time and memory usage by appropriately setting the maximum number of abstract states, M. Assuming that M is polynomially bounded, computing the abstraction has polynomial complexity [35]. SPA does not ensure that the full symbolic exploration over an abstraction has a more concise BDD representation than another over the original task. Even in the case of PDBs, relaxing the search frontier with existential shrinking could actually enlarge it. Fortunately, we can bound the number of BDD nodes needed to represent any set of states in the exploration. This is straightforward for the case of PDBs, since the size of the BDDs is bounded, if only a constant number of variables is selected.

Corollary 7. *Let α be an M&S abstraction with relevant variables V_α, generated with a maximum number of abstract states M. Let S_B be a BDD describing a set of states on $\Theta^\alpha \otimes \Pi_{V \setminus V_\alpha}$ using a variable order whose first/top variables correspond to V_α. Then, the size of S_B is bounded by:*

$$|S_B| \leq M \left(|V_\alpha| + 2^{|V \setminus V_\alpha|+1} \right)$$

Proof. It follows from Theorem 5, just assigning the bound M to the number of abstract states in every intermediate M&S abstraction. □

Corollary 7 ensures that, as variables are merged into the abstraction, the complexity of its full symbolic exploration decreases. The bound on the top part of the BDD grows linearly with M, while the bound on its bottom part is exponentially reduced. Thus, the full symbolic exploration of the abstraction will be eventually tractable. In the limit, the exploration is performed over a linear sized state space that can be explicitly explored, just as the original M&S algorithm does. Theorem 7 requires relevant variables for the abstraction to be placed in the top levels of exploration BDDs. For this to hold in every symbolic exploration, the symbolic search must use the same variable ordering as the M&S merge strategy. Since changing the variable ordering of a BDD may cause an exponential blow-up, we use the same variable ordering in order to guarantee an efficient computation of SPM&S.

7. Experiments

In this section, we evaluate the performance of SPA heuristics, and compare them against the two baseline methods: abstraction heuristics and a symbolic perimeter, as well as to other state-of-the-art heuristics for cost-optimal planning. We also evaluate the influence of different parameters of our heuristic. In particular, we analyze multiple types of abstraction strategies to relax the perimeter, including PDB and SM&S abstraction hierarchies. We compare the performance of A* search with SPA against SymBA*, the winner of the last IPC, and show that they obtain similar coverage to the state of the art. This is remarkable, especially given that our analysis does not include other possible orthogonal enhancements to search algorithms using SPA (e.g. partial-order pruning [7] or the combination with other heuristics), which are not directly applicable in SymBA*. In the following we mostly focus on the convenience of using symbolic perimeter abstractions, discussing those cases where they are advantegous and providing explanations if other techniques improve over them in some domains.

SPA has been integrated in the FAST DOWNWARD planning system [62]. All configurations use the h^2 reachability analysis [3] in order to compute state invariants and remove actions from the planning task [63]. The state invariants are used to prune the abstract searches, like in constrained PDBs [64, 65]. All the symbolic searches, for the construction of the perimeter and the exploration of the abstract state spaces, use the enhancements for image computation [66].

In our experiments we measure the coverage, i. e., number of instances solved, on the optimal-track STRIPS planning instances from the International Planning Competitions from IPC'98 until IPC'11. Since some domains have been used in multiple IPCs and not all domains have the same number of instances, besides the total coverage we report a final score that gives the same weight to every domain. This score function normalizes the coverage of every planner by the number of instances in that domain, considering all versions of the same domain in different competitions as a single entry, giving a total of 36 different domains.

All experiments were conducted on a cluster of Intel E5-2660 machines running at 2.20 GHz, with time (memory) cut-offs of 30 minutes (4 GB).

7.1. Parameterization of Symbolic Perimeter Abstractions

The SPA algorithm takes as input different parameters that can be classified into three categories:

Memory and time bounds. SPA heuristics, as other abstraction heuristics like PDBs or M&S, rely on a heavy precomputation phase in order to efficiently compute the heuristic value for each state in the search. We defined parameters that limit the time and memory to invest in the preprocessing phase in order to guarantee that it successfully terminates. Larger bounds will likely result in more informed heuristics at the expense of devoting more preprocessing time. Of course, the optimal value of these parameters directly depends on the total resources allotted to the planner. SPA parameters were manually set to fit the International Planning Competition (IPC) setting: the maximum number of nodes to represent the state set to expand is $N_F = 10{,}000{,}000$. A maximum time is set to each individual $image$ ($T_I = 30s$), exploration ($T_{Exp} = 300s$) and symbolic search ($T_{Sym} = 900s$). Finally, the heuristic preprocessing is interrupted after $T_{SPA} = 1200s$ seconds to ensure that the search is always started.

Variable ordering. All the symbolic searches use a static variable ordering. We consider two different variable orderings: GAMER and FAST DOWNWARD orderings. The variable ordering used by GAMER is optimized for symbolic search and the one used in FAST DOWNWARD is suitable for the merge strategy of M&S. However, we always use the same ordering for the symbolic search and merging variables in M&S since otherwise there is no guarantee about the BDD sizes.

Abstraction hierarchy. SPA can be used in combination with different abstraction strategies that select which abstract state spaces will be traversed in order to compute the heuristic estimates. In our experiments we used two different types of abstraction hierarchies: the SM&S hierarchies we presented in Section 5.2 and PDB hierarchies.

We define PDB hierarchies by starting with the pattern that contains all the variables (equivalent to the original task) and abstracting away one variable at a time. Thus, the PDB hierarchies are defined in terms of a variable ordering. In order to select variable orderings for the construction of the PDB hierarchies, we use the preexisting merge linear strategies for M&S:

- *Level* (*lev*): Follows the BDD variable ordering, so that it results in the same abstraction layers than SM&S.
- *Reverse level* (*rev*): Reversed version of level, following the BDD variable ordering in reverse order.
- *Cggoal-lev* (*cgl*): Always selects a variable related in the causal graph [67, 68, 69, 70] to an already selected variable. If there is none, it selects a goal variable. Ties are broken according to the level criterion.
- *Cggoal-rnd* (*cgr*): As CGGoalLevel but breaking ties randomly.
- *Goalcg-lev* (*gcl*): First it selects all goal variables and then variables related to them in the causal graph, breaking ties according to the level criterion.
- *Random* (*rnd*): selects a random variable ordering. Useful as a baseline approach.

SM&S abstraction hierarchies are defined with the merge and shrink strategies. The merge strategy is always set to exactly the same linear ordering as the one used for the BDDs. Only this ordering guarantees that the resulting BDDs will eventually be tractable after abstracting away enough variables, as explained in Section 5.2.1. Regarding shrink strategies, we use the ones that have been previously defined in the literature [12, 35]: bisimulation (b), greedy bisimulation (g) and f-preserving shrinking (fh). All the shrinking strategies reduce the abstract state space enough to ensure that no M&S abstraction has more than 10,000 abstract states.

Table 1 shows the total coverage of all the combinations of configurations of three different parameters: the abstraction selection strategy, the variable ordering employed by the BDDs (GAMER or FAST DOWNWARD ordering) and whether the perimeter is used to initialize the abstract searches ($\mathcal{P}$) or not ($\neg\mathcal{P}$). The virtual best solver configurations consider that the best configuration is selected for every instance, i. e., an instance is considered solved if it is solved by any of the configurations using PDBs, SM&S, or both. These are not real planners and they cannot be interpreted as configurations of our technique. Their purpose is to give an idea of how different are the sets of instances solved by each configuration, as well as the maximum potential of SPA when the best abstraction hierarchy for each instance is known.

		GAMER ORDERING		FD ORDERING	
		$\mathcal{P}$	$\neg\mathcal{P}$	$\mathcal{P}$	$\neg\mathcal{P}$
SP		**16.83 (800)**	0.00 (0)	15.89 (756)	0.00 (0)
SPPDB	*rev*	17.55 (825)	17.29 (812)	16.43 (784)	16.21 (769)
	lev	17.72 (826)	17.50 (818)	16.72 (783)	16.62 (779)
	cgl	17.57 (823)	17.33 (813)	16.41 (775)	16.18 (765)
	cgr	17.39 (813)	17.31 (810)	16.31 (772)	16.40 (770)
	gcl	**17.82 (828)**	17.74 **(828)**	16.90 (785)	16.96 (785)
	rnd	17.28 (814)	17.08 (812)	16.36 (775)	16.22 (768)
SPM&S	b	17.25 (806)	17.36 (810)	16.55 (774)	16.71 (773)
	g	17.38 (809)	**17.39 (812)**	16.29 (769)	16.17 (766)
	fh	16.52 (762)	16.54 (764)	15.35 (710)	15.35 (709)
Virtual $best$	SPPDB	18.42 (852)	18.09 (841)	17.31 (808)	17.29 (802)
	SPM&S	17.60 (818)	17.69 (822)	16.68 (779)	16.88 (780)
	SPA	**18.57 (857)**	18.31 (847)	17.61 (816)	17.46 (806)

Table 1: Coverage score and total coverage of SPA with different abstraction hierarchies. The symbolic perimeter, SP, which does not make use of any abstraction, is compared against PDB and SM&S hierarchies. We report the results of the virtual best solver using PDBs, SM&S, and both. All the versions are ran with perimeter abstractions ($\mathcal{P}$) and with standard abstraction heuristics ($\neg\mathcal{P}$). The best configurations for each type of abstraction hierarchy are highlighted in bold.

What is the impact of the abstraction hierarchy?. The first observation is that most abstraction strategies help to improve the symbolic perimeter, SP. The only strategy that does not produce better results than a simple perimeter is the SM&S abstractions with *fh* shrinking. Of course, the bad performance is not due to having a less informed heuristic, but because the M&S heuristic exceeded the available memory in some domains.

Overall, the best strategy is PDBs with *gcl*, with a total coverage of 828. Other PDB strategies such as *lev* or *rev* have a similar performance and the difference is not deemed as statistically significant. Interestingly, the performance of PDBs with a random selection of variables is a very competitive approach, leading to better results than some M&S approaches or other PDB strategies. This shows the lack of good abstraction strategies for our setting, where finding them is not straightforward.

The results of the virtual best configurations confirm that there is a lot of margin for improvement in the abstraction selection strategies. When selecting the best hierarchy for every instance, SPA solves up to 857 instances, 29 more than the best configuration with a single hierarchy and 16 more than the configuration using multiple abstraction hierarchies reported in Table 3. With current merge and shrink strategies, SM&S strategies are helpful only in a few instances, solving 5-8 instances that PDB configurations cannot solve.

How useful is to use the perimeter to initialize the abstract searches?. We ran an ablation analysis, running all the configurations of SPA but starting abstract searches from the abstract goal instead of doing it from the perimeter ($\neg\mathcal{P}$). Note, however, that even though $\neg\mathcal{P}$ ignores the perimeter for the initialization of abstract searches, it uses the perimeter for other purposes such as selecting one abstraction from the hierarchy (the one in which the perimeter is reduced enough) and to return the maximum between the heuristics derived from all the searches (including the perimeter and one or more abstract searches).

In general, using the perimeter is beneficial, though it may depend on the abstraction strategy. Interestingly, using the perimeter to initialize abstract searches is especially useful in combination with strategies that do not distinguish goal from non-goal variables such as *lev* or *rev*. In those cases, the perimeter derives information from the goal variables, so that they can be abstracted away without a significant information loss. Similarly, if all the goal variables are included in the abstraction (*gcl*), using the perimeter is not that important.

In combination with SM&S hierarchies, the results with and without perimeter are similar, so SPA is not taking advantage of the perimeter initialization in those cases. The shrink strategies in the literature are specifically focused on preserving the cost of states close to the goal, which may not be the best when using a perimeter that already provides information for such region of the state space.

What is the impact of variable ordering?. The variable ordering is used in the BDD representation, as well as for the abstraction strategies underlying the PDB selection and merge strategy. As expected, M&S results are better with the FAST DOWNWARD ordering while symbolic search (e.g., SP) benefits from the GAMER ordering. In total, SPA performs better with the GAMER ordering, though it may vary depending on the domain. Apart from that, the conclusions obtained with GAMER ordering seem to be independent of the variable ordering chosen for the BDDs.

7.2. Removing Spurious States

The results shown in the previous section contradict previously published results, which reported SPA being better with SM&S hierarchies than when using only PDBs [18]. One important difference in the experimental setting is that in this article we use constrained abstractions [64], encoding state-invariant constraints into the transition relations in order to avoid the generation of invalid states in the symbolic searches [65]. torralba2015phd

The use of mutexes greatly improves the performance of symbolic regression search. This is not different in SPA, where symbolic regression is used both in the original and the abstract state spaces.

	GAMER ORDERING		FD ORDERING	
	$\mathcal{M}_\varnothing$	$\mathcal{M}$	$\mathcal{M}_\varnothing$	$\mathcal{M}$
SP	14.44 (715)	16.83 (800)	14.34 (689)	15.89 (756)
SPPDBs	14.84 (730)	**17.45 (822)**	14.82 (708)	16.37 (783)
SPM&S-b	15.23 (735)	17.25 (806)	15.42 (717)	16.55 (774)
SPM&S-g	15.14 (732)	**17.38** (809)	15.12 (719)	16.29 (769)

Table 2: Coverage score and total coverage of different configurations of SPA without state-invariant pruning ($\mathcal{M}_\varnothing$) and using it ($\mathcal{M}$). The best configurations and those deviating only 1% are highlighted in bold.

Table 2 compares the performance of our approaches when disabling the removal of spurious states during the search. When spurious states are not removed from the search ($\mathcal{M}_\varnothing$), the performance of all our symbolic perimeter approaches decreases significantly. However, the benefits of removing spurious states are greater in the case of SP and SPPDB than for any SPM&S version. Interestingly, M&S abstractions obtain better results than PDBs when no state invariants are used. This can be partially explained by the ability of M&S heuristics to prune spurious states, whose impact is heavily reduced when those states are being removed anyway thanks to the state invariants. Therefore, even though SPPDB with state-invariant pruning outperforms SPM&S with the current M&S strategies, further enhancements for M&S like irrelevance pruning [71] or shrink strategies tailored for dead-end detection [72] or exploiting symmetries [73] are promising approaches to revert this trend.

7.3. Coverage of A* Search with Symbolic Perimeter Abstraction Heuristics

In this section we compare the coverage of different configurations of SPA against the two combined approaches: symbolic perimeter (SP) and abstraction heuristics, including M&S, the symbolic PDBs of GAMER (our re-implementation using the same code as SPA, with the image computation and state-invariant pruning enhancements of cGAMER [57]), and two additive PDB methods (iPDB and gaPDB). Additionally, we report the coverage of LM-CUT for comparison. Finally, we include the winner of the last IPC, SymBA*, which also makes use of perimeter abstraction heuristics, but in a bidirectional symbolic search.

Table 3 shows the result of six different configurations of SPA, using PDB or SM&S abstractions to relax the perimeter. The SPA algorithm, as presented in Section 6, uses a single abstraction hierarchy (either *cgr*, SM&S-b, or SM&S-g in Table 3) to relax the perimeter. Since our abstraction hierarchy strategies do not perform any search on the space of abstract state spaces, they are not competitive with GAMER's abstraction selection, which uses an optimization procedure trying multiple PDBs before selecting the best one. Hence, we include two configurations that construct multiple abstraction heuristics (SP-multi), using multiple PDB hierarchies plus SM&S-b. We also include a version of GAMER's procedure that uses the perimeter to initialize all abstract searches.

The results of SPA show that it is a strong heuristic. The construction of a symbolic perimeter (SP) suffices to generate a heuristic competitive with strong heuristics commonly used as baseline in cost-optimal planning, such as iPDB, gaPDB, M&S, and LM-CUT. This is especially true when considering the coverage score that gives MICONIC the same weight as other domains. Using abstraction strategies on top of the symbolic perimeter further improves

	SP	SPA						Abstractions					LM-CUT	SymBA*
		SPPDB	SPM&S		SP-multi		SPPDB	SPDB	M&S		add-PDB			
		cgl	*b*	*g*	*b* + PDB	PDB	GAMER	GAMER	*b*	*g*	iPDB	gaPDB		
AIRPORT (50)	25	27	25	25	26	25	26	26	23	23	28	24	**29**	27
BARMAN (20)	8	8	8	8	9	9	9	8	4	4	4	4	4	**11**
BLOCKSWORLD (35)	30	32	30	30	31	**33**	31	31	25	28	28	22	28	30
DEPOT (22)	7	8	7	7	7	7	7	7	7	6	**11**	7	7	7
DRIVERLOG (20)	12	13	13	13	**14**	13	12	**14**	13	13	13	13	13	**14**
ELEVATORS08 (30)	23	23	23	23	**24**	**24**	23	22	14	15	21	20	22	**24**
ELEVATORS11 (20)	18	18	18	18	**19**	**19**	17	17	12	13	17	17	18	**19**
FLOORTILE11 (20)	14	14	14	14	14	14	**15**	14	8	8	3	2	14	14
FREECELL (80)	22	31	21	21	34	37	28	**40**	19	19	21	20	15	23
GRID (5)	2	3	3	3	3	3	**4**	**4**	2	2	3	2	2	3
GRIPPER (20)	**20**	**20**	**20**	**20**	**20**	**20**	**20**	13	8	8	8	8	7	**20**
LOGISTICS 00 (28)	16	17	16	18	19	19	16	20	16	16	**21**	20	20	20
LOGISTICS 98 (35)	4	5	5	5	5	5	**6**	**6**	5	4	4	5	**6**	5
MICONIC (150)	107	107	107	107	108	108	107	106	60	52	58	60	**141**	107
MPRIME (35)	22	22	22	22	23	23	22	22	22	23	24	23	23	**25**
MYSTERY (30)	15	15	15	15	15	15	15	16	**17**	**17**	**17**	**17**	**17**	15
NOMYSTERY11 (20)	13	16	14	14	17	18	16	14	16	14	19	**20**	14	14
OPENSTACKS08 (30)	**30**	**30**	**30**	**30**	**30**	**30**	**30**	**30**	21	21	22	22	21	**30**
OPENSTACKS11 (20)	**20**	**20**	**20**	**20**	**20**	**20**	**20**	**20**	16	16	17	17	16	**20**
OPENSTACKS06 (30)	11	10	10	11	10	10	11	9	7	7	7	7	7	**20**
PARCPRINTER08 (30)	**23**	**23**	**23**	**23**	**23**	**23**	**23**	**23**	20	20	13	16	22	21
PARCPRINTER11 (20)	**18**	**18**	**18**	**18**	**18**	**18**	**18**	**18**	15	15	9	12	17	16
PARKING11 (20)	1	4	1	1	4	4	2	1	1	1	**7**	1	2	1
PATHWAYS-NONEG (30)	4	4	4	4	4	4	**5**	4	4	4	4	4	**5**	**5**
PEG-SOLITAIRE08 (30)	29	29	29	29	**30**	**30**	29	29	27	27	27	28	28	29
PEG-SOLITAIRE11 (20)	19	19	19	19	**20**	**20**	19	19	17	17	17	18	18	19
PIPESWORLD-NT (50)	15	15	14	15	14	14	**24**	20	16	15	20	19	17	15
PIPESWORLD-T (50)	12	16	13	13	12	15	**18**	17	14	17	17	13	12	16
PSR-SMALL (50)	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	**50**	49	**50**
ROVERS (40)	12	**13**	**13**	**13**	**13**	12	**13**	12	8	6	8	7	7	**13**
SATELLITE (36)	**9**	**9**	**9**	**9**	**9**	**9**	**9**	**9**	6	6	6	6	7	**9**
SCANALYZER08 (30)	12	12	12	12	12	12	12	12	12	12	13	12	**15**	12
SCANALYZER11 (20)	9	9	9	9	9	9	9	9	9	9	10	9	**12**	9
SOKOBAN08 (30)	28	29	28	28	28	28	**30**	**30**	29	29	**30**	24	**30**	28
SOKOBAN11 (20)	20	20	20	20	20	20	20	20	20	20	20	20	20	20
TIDYBOT11 (20)	14	12	15	16	15	11	**17**	14	12	13	14	13	**17**	**17**
TPP (30)	8	8	8	8	8	8	8	8	6	6	6	6	7	**9**
TRANSPORT08 (30)	12	12	12	12	13	12	12	**14**	11	11	11	11	11	13
TRANSPORT11 (20)	7	8	8	8	9	9	8	**11**	6	6	7	7	6	9
TRUCKS (30)	10	10	10	10	11	13	12	**15**	8	8	9	8	10	12
VISITALL (20)	11	12	12	10	12	12	12	**16**	10	**16**	**16**	12	10	12
WOODWORKING08 (30)	**28**	25	**28**	27	25	25	26	**28**	14	14	9	10	22	25
WOODWORKING11 (20)	**20**	**20**	**20**	**20**	19	19	19	**20**	9	9	4	5	15	19
ZENOTRAVEL (20)	10	12	10	11	12	12	11	11	10	10	12	12	**13**	11
TOTAL COVERAGE (1396)	800	828	806	809	838	841	841	**849**	649	650	685	653	796	838
SCORE COVERAGE (36)	16.83	17.82	17.25	17.38	18.17	18.14	18.35	18.16	14.11	14.37	15.82	14.47	16.31	**18.44**

Table 3: Coverage of symbolic perimeter abstractions (SPA) using PDB hierarchies (SPPDB), SM&S hierarchies (SPM&S) with bisimulation (b) and greedy bisimulation (g), multiple abstractions (SP-multi), and GAMER's PDB selection. Results compared against a symbolic perimeter (SP), and other non-perimeter abstraction heuristics like M&S, the symbolic PDBs of GAMER and additive PDBs (iPDB and gaPDB). Results of the state-of-the-art LM-CUT heuristic and SymBA*, which uses symbolic bidirectional A* search, are included for comparison. Score coverage normalizes total coverage dividing by the number of instances in each domain.

the results of SPA. Nevertheless, the heuristics are shown to be complementary since there are some domains were LM-CUT clearly performs better, such as AIRPORT, MICONIC or SCANALYZER.

The dominance of SPM&S over SP and M&S is quite clear. Even though there are a few cases where the pre-processing phase of SPM&S does not terminate causing it to be outperformed by the symbolic perimeter alone, this is compensated by combining the strengths of both algorithms in many other instances. Moreover, in some cases SPM&S obtains better results than any of the two techniques (as in GRID or ROVERS, for example). On the other hand, even though there are some domains where SPM&S beats SPPDB (e.g., TIDYBOT), the use of M&S abstractions does not pay off in general over using simpler abstractions such as PDBs.

Another interesting result is the comparison with the winner of the optimal-track of the last IPC, SymBA*. While SymBA* has higher coverage score, SPPDBs have similar score and even higher total coverage. This is remarkable, especially when taking into account that the performance of A* with SPA heuristics could easily be boosted by using other complementary heuristics or search-enhancements, such as partial-order pruning [7], symmetries [6, 25], or

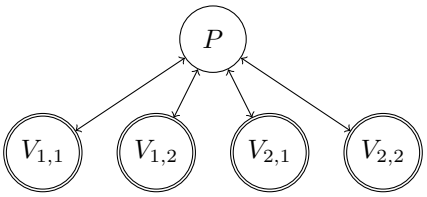


Figure 10: Causal graph of the VISITALL domain. Goal variables represented with a double line.

dominance pruning methods [8, 26].

The most surprising result is that GAMER's procedure to generate symbolic PDBs produces remarkably strong heuristics, having the largest coverage among all the configurations, including SPA and SymBA*. This is partially due to a big edge in FREECELL so in the normalized score it has similar performance to SP-multi. These results confirm that finding a good pattern is important, so the versions trying multiple patterns perform better. In order to know whether the perimeter may be helpful in combination with the pattern selection of GAMER, we ran an additional version that generates a perimeter and uses it to initialize the abstract searches during GAMER's procedure.

The use of a perimeter is beneficial in most settings (see Section 7.1). This is not surprising since using a perimeter to initialize any abstract search can only make the corresponding abstraction heuristic more informed. Also, the search is initialized with a subset of the states that are reachable in the abstract state space as well, so the size of the abstract state space that is explored can only decrease. However, there are several domains where it is harmful for GAMER's PDB selection. A close inspection to those cases shows that, in symbolic search, the use of a perimeter makes the search exponentially harder in certain situations.

Consider the VISITALL domain, where a robot must traverse an entire grid. We have a variable, P, that defines the current position of the robot in the grid and a variable for each cell, $V_{i,j}$, which represents whether the cell (i, j) has been visited or not. The only action, move, as the name suggest moves the robot to an adjacent location, setting the corresponding visited variable to true. The causal graph is depicted in Figure 10. The relevant property in this case is that, when abstracting the non-goal variables (P), all goal variables are completely independent. This is precisely the first pattern that is attempted by Gamer's abstraction strategy. This pattern produces a heuristic that counts the number of non-visited cells, which is a very informed estimate and coincides with the perfect heuristic for the initial state of most instances. A symbolic search on this abstract state space takes polynomial time in the size of the task [74, 75]. When using the perimeter, however, the variables are not independent anymore since we can only have visited a given location in the perimeter if we have visited one of its adjacent locations. Therefore, if we use the perimeter to initialize the abstract search, we get a more informed heuristic but the abstract search takes exponential time and memory to be completed.

The same causal graph structure appears in many other domains as well. In particular, those domains where there are a substantial ($> 20\%$) number of goal variables which are independent after relaxing the rest. These domains include many of the cases where the perimeter is detrimental like LOGISTICS ($\approx 60\%$), TRUCKS ($\approx 40\%$), and TRANSPORT ($\approx 30-65\%$). In PARCPRINTER ($\approx 25-50\%$) and WOODWORKING ($\approx 30-50\%$) goal variables are not completely independent but they can be separated into several independent clusters. There are also other domains with this structure in which the perimeter is beneficial like OPENSTACKS ($\approx 50\%$), ELEVATORS ($\approx 30-50\%$), BLOCKSWORLD ($\approx 30-50\%$), MICONIC ($\approx 30-50\%$), ROVERS ($\approx 15-25\%$), or GRIPPER ($\approx 90\%$). In these cases, either the search with all variables is efficient as well (GRIPPER and MICONIC), or the goal pattern does not produce a sufficiently well-informed heuristic (e.g., in ELEVATORS all actions achieving the goals have zero cost, so the only-goals pattern produces a heuristic that always returns zero).

In the other domains, using the perimeter is not harmful, except in DRIVERLOG, GRID, FREECELL, or MYSTERY where either the search of the only-goals variables is still easy even if they are not completely independent (DRIVERLOG) or the use of the perimeter changes Gamer's pattern selection, showing that there is still room for improvement in studying pattern selection algorithms for symbolic PDBs.

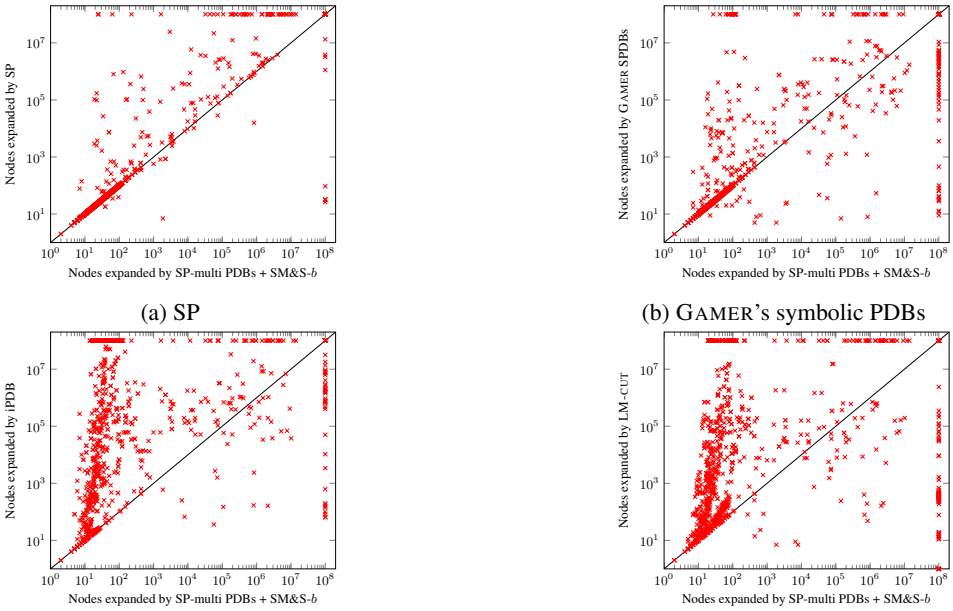
Therefore, it is not by chance that the symbolic PDBs of GAMER beat our perimeter abstractions with the same abstraction strategy in some of these domains. Even though this may be read as a negative result against the use of a perimeter, two things must be noted. First, it is straightforward to check the causal graph of a planning task to quickly

identify whether the perimeter should be used. In domains that do not exhibit the structure mentioned above, using the perimeter is recommended since it usually increases the performance. Second, when there is so much independence, additivity could be exploited. In fact, in most of these domains, the additive PDB configuration of iPDB is very strong as well. Finally, note that the SM&S abstractions with a suitable shrink strategy (e.g., an abstraction like the one in Figure 5 of Section 5.1) may simplify the perimeter making the symbolic search on the abstract state space tractable again.

7.4. Informativeness of SPA Heuristics

The coverage comparisons of Section 7.3 show that SPA is a state-of-the-art heuristic, outperforming other heuristics such as M&S or additive PDBs and being competitive with LM-CUT. However, there are two main factors that influence the performance of heuristics: informativeness and computational effort. In this subsection we analyze how well informed the SPA heuristics are with respect to the state of the art. From the heuristics in the literature, we chose iPDB because it also uses PDBs but in an additive setting, and LM-CUT because it is known to be an expensive but very well-informed heuristic [4]. As representative of the SPA method, we chose the configuration using SM&S-b and multiple PDBs that got the best coverage score among the versions using our abstraction hierarchies.

In order to compare the informativeness of the heuristics, we report two different metrics: expanded nodes and initial h-value. The initial h-value is a direct comparison of the value of each heuristic for particular states, while the number of expansions measures the ability of the heuristic to reduce the overall search effort. Figures 11 and 12 show the comparisons in both metrics against different competitors in all commonly solved instances. When comparing the heuristic values, we omit PARCPRINTER because the large heuristic values of that domain hinder the visualization.



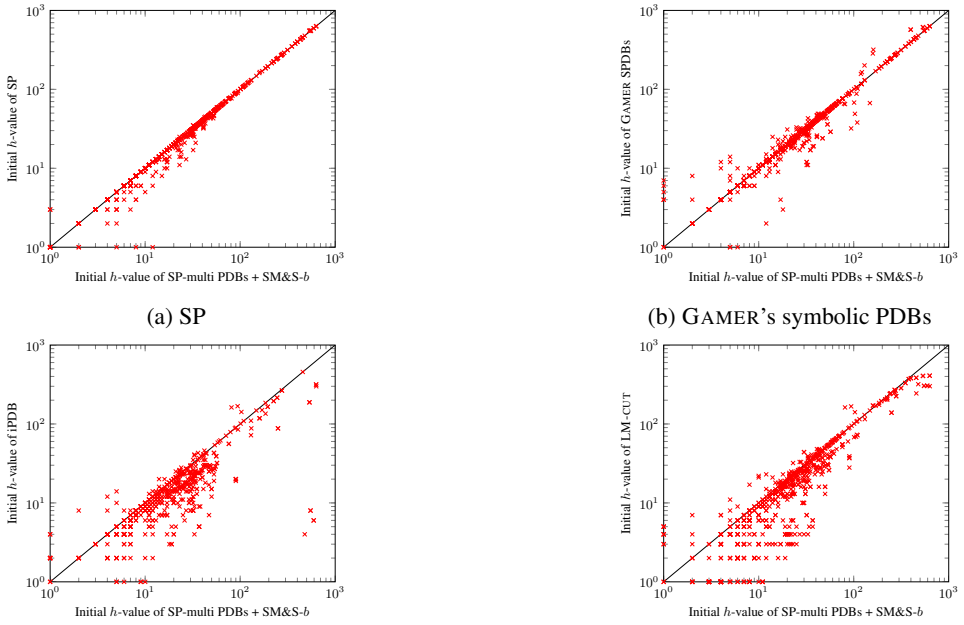
(a) SP (b) GAMER's symbolic PDBs

(c) iPDB (d) LM-CUT

Figure 11: Number of expanded nodes by SPA (x-axis) against other heuristics (y-axis). Unsolved instances are assigned a number of 100 million expansions. Points above the main diagonal are instances solved with fewer node expansions by SPA.

The comparison of SPA against SP and SPPDB reveals that the main strength of the heuristic is due to the perimeter search. Using abstraction heuristics to extend the information of the perimeter derives strictly more informed heuristics. Therefore, both SPPDB and SPA obtain more accurate heuristics than SP and, in some cases, significantly reduce the number of expanded nodes by several orders of magnitude. However, in most cases, SPA and SPPDB obtain the same heuristic value than SP, so the symbolic perimeter is already a strong heuristic.

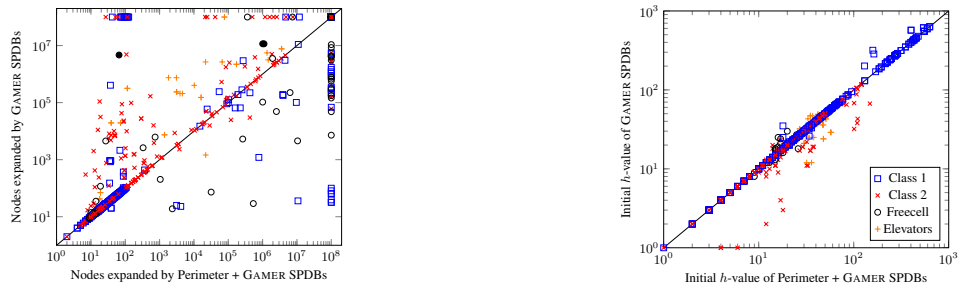
Perhaps more surprisingly, the comparison with iPDB and LM-CUT shows that SPA is a well-informed heuristic compared to the current state of the art in cost-optimal planning. It does not only derive the perfect heuristic for many



(a) SP (b) GAMER's symbolic PDBs

(c) iPDB (d) LM-CUT

Figure 12: Initial state heuristic value of SPA (x-axis) against other heuristics (y-axis) in commonly solved instances of all domains except PARCPRINTER. Points below the main diagonal are instances in which SPA is more informed.



(a) Expanded nodes (b) Initial h-value.

Figure 13: Comparison of expanded nodes and initial h-value with (x-axis) and without (y-axis) perimeter in GAMER's symbolic PDBs. Class 1 represents all domains with more than 20% of causally-independent goals except ELEVATORS. Class 2 represents all other domains except FREECELL. ELEVATORS (class 1) and FREECELL (class 2) are separated for being exceptions to our general rule.

instances (for most of those solved with less than 1,000 expansions), but also derives a heuristic competitive with LM-CUT in the harder tasks. Differences in heuristic value of the initial state show that SPA is more precise in many tasks, being a well-informed heuristic across the entire state space and not only close to the goal.

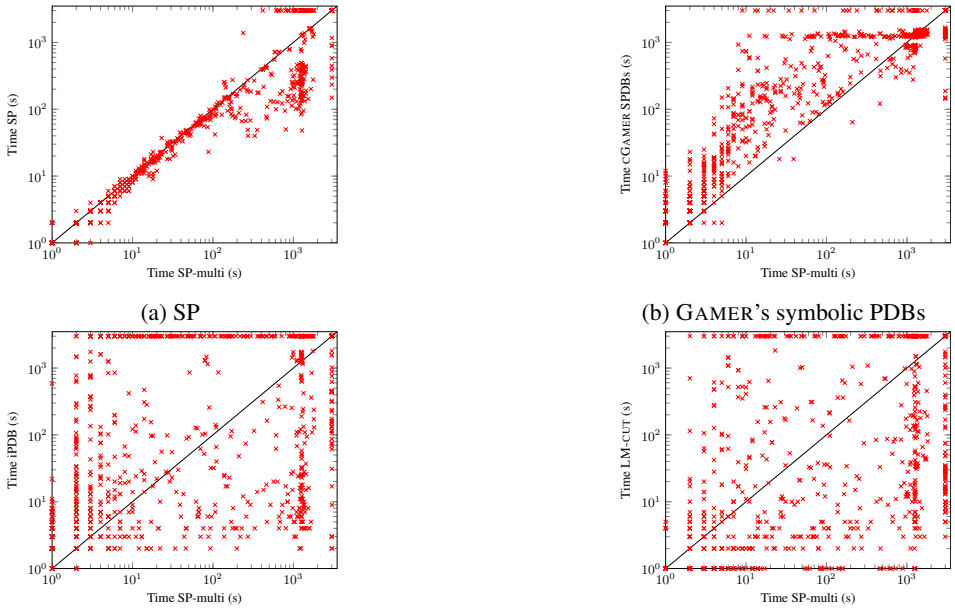
The comparison against Gamer's symbolic PDBs reveals that both methods are complementary, having strengths in many different tasks. In order to analyze the impact of the perimeter in Gamer's pattern selection we compare the number of expansions and initial h-value on the two classes of domains described in Section 7.3. In general, using the perimeter shows good results except in the domains already identified and FREECELL. This confirms our hypothesis that the main cause for the perimeter to be harmful is when domains have causally independent goals.

Figure 14 complements the data above with a comparison of the solving time of the four planners. As expected, the data is completely biased by the precomputation phase of abstraction heuristics. Since SP-multi and GAMER's SPDBs have a limit of 1200 seconds for the precomputation of the heuristics, a huge amount of instances are solved around that time. Hence, configurations with strictly lower precomputation time such as SP generally beat configurations

spending more time. On the other hand, the precomputation time pays off in coverage, as reflected by the results of Table 3.

SPA is generally faster than GAMER SPDBs. One reason is that many instances are solved directly during the construction of the perimeter. More importantly, GAMER is expending the 1200 seconds of precomputation time in more instances, which means that SPA could improve its results with better pattern selection strategies that explore more abstract state spaces.

The comparison with iPDB and LM-CUT show that they have different strengths and weaknesses. SPA solves many instances quickly when constructing the perimeter that are not always easy for explicit-search planners. But, again, LM-CUT and iPDB beat SPA in many cases where it spends 1200 seconds to precompute the heuristic.



(a) SP

(b) GAMER's symbolic PDBs

(c) iPDB

(d) LM-CUT

Figure 14: Time in seconds of SPA (x-axis) against other heuristics (y-axis). Unsolved instances are assigned a time of 3000s. Points above the main diagonal are instances solved faster by SPA.

8. Discussion

Symbolic Perimeter Abstractions (SPA) are a novel kind of admissible heuristics for cost-optimal planning that extends previous definitions of perimeter abstraction heuristics in order to consider multiple perimeters in different abstract state spaces, instead of only one. Applying perimeter abstractions in planning poses some challenges, such as how to perform efficient regression search or how to obtain the set of abstract states that correspond to all states in the perimeter to initialize the abstract search. SPA addresses these challenges by using symbolic search in order to traverse the abstract state spaces.

In summary, SPA uses abstractions to relax a symbolic backward search, keeping only partial information about some variables of the task. An abstraction hierarchy determines a set of abstract state spaces to search, ranging from the original non-abstracted state space to one small enough to be explored by explicit-state search. Every abstraction reduces the size of the abstract state space, so that there is a trade-off between the abstract search complexity and the heuristic informativeness. Searching all these abstractions is clearly redundant, since heuristics of less relaxed abstractions are strictly more informed than more relaxed ones. Predicting which abstraction has the best benefit/effort relation is a hard task, which SPA addresses by using perimeter abstraction heuristics. The search is performed in the less relaxed state spaces while it is feasible and coarser state spaces are used to continue the search afterwards.

SPA can be used both with hierarchies of PDB and M&S abstractions. M&S represents abstract state spaces explicitly, which limits the size of M&S abstractions. In order to obtain hierarchies with arbitrarily large M&S abstractions, we define SM&S abstractions as larger M&S abstractions based on the smaller explicitly represented M&S abstractions. SM&S abstractions can be searched with symbolic search, in an attempt to combine the strengths of symbolic PDBs and the flexibility of M&S abstractions. We prove that, for any abstraction derived using M&S with a linear merge strategy, it is possible to represent any set of abstract states as a decision diagram of polynomial size in the number of abstract states used to derive such abstraction. Not only this bounds the size of BDDs in the symbolic search of M&S abstractions, but it may also have other uses such as extracting polynomial certificates of unsolvability in the form of BDDs [76].

Symbolic abstraction heuristics have been used mainly by symbolic search planners we advocate their use in explicit-state search planners. Experimental results show that the symbolic perimeter heuristic is competitive with other state-of-the-art heuristics such as LM-CUT even without any abstraction strategy. SPA successfully uses abstractions to relax the perimeter in order to derive more informed heuristics, improving the performance even further. Despite the greater generality of M&S abstractions, the simplicity of PDBs allows for more efficient symbolic operations and thus better heuristics. Our algorithm is not necessarily limited to PDB or M&S abstractions, and could also potentially be used in combination with Cartesian abstractions [77, 78], or abstractions induced by BDD minimization methods that have been successfully used for model-checking [79].

Finding good abstraction hierarchies for the task at hand is still an open problem. We partially address this problem by using multiple hierarchies. SPA with multiple abstraction hierarchies is competitive with (and complementary to) symbolic PDB approaches that perform a search on the space of possible patterns, like the one used by the symbolic search planner GAMER. We also analyzed the potential of using the perimeter to initialize abstract searches in GAMER's pattern selection algorithm, substantially improving the results in many domains, but being detrimental in those domains where additivity is indirectly exploited by the symbolic search. Analyzing the use of additive ensembles of perimeter PDBs is a promising line of research for future work. Finally, another interesting avenue for future research is the study of symbolic perimeter abstractions outside optimal planning, e.g. in the context of proving unsolvability [80].

Acknowledgements

We would like to thank the reviewers for their valuable comments and effort to improve the manuscript. The second and third researchers have been supported by MICINN project TIN2011-27652-C03-02 and TIN2014-55637-C2-1-R.

Bibliography

[1] 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.
[2] B. Bonet, H. Geffner, Planning as heuristic search, Artificial Intelligence Journal 129 (2001) 5–33.
[3] 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), 2000, pp. 140–149.
[4] 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), 2009, pp. 162–169.
[5] F. Pommerening, M. Helmert, G. Röger, J. Seipp, From non-negative to general operator cost partitioning, in: B. Bonet, S. Koenig (Eds.), Proceedings of the AAAI Conference on Artificial Intelligence (AAAI), 2015, pp. 3335–3341.
[6] N. Pochter, A. Zohar, J. S. Rosenschein, Exploiting problem symmetries in state-based planners, in: W. Burgard, D. Roth (Eds.), Proceedings of the AAAI Conference on Artificial Intelligence (AAAI), 2011, pp. 1004–1009.
[7] M. Wehrle, M. Helmert, Efficient stubborn sets: Generalized algorithms and selection strategies, in: S. Chien, M. B. Do, A. Fern, W. Ruml (Eds.), Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS), 2014, pp. 323–331.
[8] Á. Torralba, J. Hoffmann, Simulation-based admissible dominance pruning, in: Q. Yang, M. Wooldridge (Eds.), Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), 2015, pp. 1689–1695.
[9] J. C. Culberson, J. Schaeffer, Pattern databases, Computational Intelligence 14 (1998) 318–334.
[10] 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, 2001, pp. 13–34.
[11] K. Dräger, B. Finkbeiner, A. Podelski, Directed model checking with distance-preserving abstractions, Journal on Software Tools for Technology Transfer 11 (2009) 27–37.

[12] 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.
[13] P. Eyerich, M. Helmert, Stronger abstraction heuristics through perimeter search, in: D. Borrajo, S. Kambhampati, A. Oddi, S. Fratini (Eds.), Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS), 2013, pp. 303–307.
[14] 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), 2002, pp. 274–283.
[15] R. E. Bryant, Graph-based algorithms for Boolean function manipulation, IEEE Transactions on Computers 35 (1986) 677–691.
[16] Á. Torralba, V. Alcázar, P. Kissmann, S. Edelkamp, Efficient symbolic search for cost-optimal planning, Artificial Intelligence Journal 242 (2017) 52–79.
[17] 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), 2015, pp. 4320–4321.
[18] 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), 2013, pp. 2394–2400.
[19] B. Bonet, An admissible heuristic for SAS+ planning obtained from the state equation, in: F. Rossi (Ed.), Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), 2013, pp. 2268–2274.
[20] B. Bonet, M. van den Briel, Flow-based heuristics for optimal planning: Landmarks and merges, in: S. Chien, M. B. Do, A. Fern, W. Ruml (Eds.), Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS), 2014, pp. 47–55.
[21] 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.
[22] S. Núñez, D. Borrajo, C. Linares López, Automatic construction of optimal static sequential portfolios for AI planning and beyond, Artificial Intelligence Journal 226 (2015) 75–101.
[23] D. Tolpin, T. Beja, S. E. Shimony, A. Felner, E. Karpas, Toward rational deployment of multiple heuristics in A*, in: F. Rossi (Ed.), Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), 2013, pp. 674–680.
[24] M. W. Barley, S. Franco, P. J. Riddle, Overcoming the utility problem in heuristic generation: Why time matters, in: S. Chien, M. B. Do, A. Fern, W. Ruml (Eds.), Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS), 2014, pp. 38–46.
[25] C. Domshlak, M. Katz, A. Shleyfman, Enhanced symmetry breaking in cost-optimal planning as forward search, in: L. McCluskey, B. Williams, J. R. Silva, B. Bonet (Eds.), Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS), 2012, pp. 343–347.
[26] D. Hall, A. Cohen, D. Burkett, D. Klein, Faster optimal planning with partial-order pruning, in: D. Borrajo, S. Kambhampati, A. Oddi, S. Fratini (Eds.), Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS), 2013, pp. 100–108.
[27] J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, L. J. Hwang, Symbolic model checking: 10ˆ20 states and beyond, Information and Computation 98 (1992) 142–170.
[28] K. L. McMillan, Symbolic model checking, Kluwer Academic publishers, 1993.
[29] 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.
[30] G. Cabodi, P. Camurati, Symbolic FSM traversals based on the transition relation, IEEE Trans. on CAD of Integrated Circuits and Systems 16 (1997) 448–457.
[31] S. Edelkamp, M. Helmert, MIPS: The model-checking integrated planning system, AI Magazine 22 (2001) 67–72.
[32] 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*, 1997, pp. 130–142.
[33] P. Bertoli, A. Cimatti, M. Roveri, P. Traverso, Planning in nondeterministic domains under partial observability via symbolic model checking, in: B. Nebel (Ed.), Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), 2001, pp. 473–478.
[34] R. I. Bahar, E. A. Frohm, C. M. Gaona, G. D. Hachtel, E. Macii, A. Pardo, F. Somenzi, Algebraic decision diagrams and their applications, Formal Methods in System Design 10 (1997) 171–206.
[35] M. Helmert, P. Haslum, J. Hoffmann, Flexible abstraction heuristics for optimal sequential planning, in: M. S. Boddy, M. Fox, S. Thiébaux (Eds.), Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS), 2007, pp. 176–183.
[36] R. C. Holte, J. Newton, A. Felner, R. Meshulam, D. Furcy, Multiple pattern databases, in: S. Zilberstein, J. Koehler, S. Koenig (Eds.), Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS), 2004, pp. 122–131.
[37] A. Felner, R. E. Korf, S. Hanan, Additive pattern database heuristics, Journal of Artificial Intelligence Research (JAIR) 22 (2004) 279–318.
[38] M. Katz, C. Domshlak, Optimal admissible composition of abstraction heuristics, Artificial Intelligence Journal 174 (2010) 767–798.
[39] 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), 2007, pp. 1007–1012.
[40] S. Edelkamp, Automated creation of pattern database search heuristics, in: S. Edelkamp, A. Lomuscio (Eds.), Proceedings of the Workshop on Model Checking and Artificial Intelligence (MoChArt), volume 4428 of *Lecture Notes in Computer Science*, 2006, pp. 35–50.
[41] S. Franco, Á. Torralba, L. H. S. Lelis, M. Barley, On creating complementary pattern databases, in: Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17, 2017, pp. 4302–4309.
[42] 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*, 1998, pp. 81–92.
[43] 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), 2011, pp. 992–997.
[44] K. Dräger, B. Finkbeiner, A. Podelski, Directed model checking with distance-preserving abstractions, in: A. Valmari (Ed.), Proceedings of the Workshop on Model Checking Software (SPIN), volume 3925 of *Lecture Notes in Computer Science*, 2006, pp. 19–34.
[45] R. Nissim, J. Hoffmann, M. Helmert, Computing perfect heuristics in polynomial time: On bisimulation and merge-and-shrink abstraction in optimal planning, in: T. Walsh (Ed.), Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), 2011, pp. 1983–1990.

[46] M. Katz, J. Hoffmann, M. Helmert, How to relax a bisimulation?, in: L. McCluskey, B. Williams, J. R. Silva, B. Bonet (Eds.), Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS), 2012, pp. 101–109.
[47] S. Sievers, M. Wehrle, M. Helmert, Generalized label reduction for merge-and-shrink heuristics, in: C. E. Brodley, P. Stone (Eds.), Proceedings of the AAAI Conference on Artificial Intelligence (AAAI), 2014, pp. 2358–2366.
[48] G. Fan, M. Müller, R. Holte, Non-linear merging strategies for merge-and-shrink based on variable interactions, in: S. Edelkamp, R. Barták (Eds.), Proceedings of the Symposium on Combinatorial Search (SoCS), 2014, pp. 53–61.
[49] S. Sievers, M. Wehrle, M. Helmert, An analysis of merge strategies for merge-and-shrink heuristics, in: A. J. Coles, A. Coles, S. Edelkamp, D. Magazzeni, S. Sanner (Eds.), Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS), 2016, pp. 294–298.
[50] G. Manzini, Bida: An improved perimeter search algorithm, Artificial Intelligence Journal 75 (1995) 347–360.
[51] J. F. Dillenburg, P. C. Nelson, Perimeter search, Artificial Intelligence Journal 65 (1994) 165–178.
[52] R. E. Korf, Depth-first iterative-deepening: An optimal admissible tree search, Artificial Intelligence Journal 27 (1985) 97–109.
[53] C. Linares López, A. Junghanns, Perimeter search performance, in: J. Schaeffer, M. Müller, Y. Björnsson (Eds.), Proceedings of the Conference on Computers and Games, volume 2883 of *Lecture Notes in Computer Science*, 2002, pp. 345–359.
[54] H. Kaindl, G. Kainz, Bidirectional heuristic search reconsidered, Journal of Artificial Intelligence Research (JAIR) 7 (1997) 283–317.
[55] A. Felner, N. Ofek, Combining perimeter search and pattern database abstractions, in: I. Miguel, W. Ruml (Eds.), Proceedings of the Symposium on Abstraction, Reformulation and Approximation (SARA), volume 4612 of *Lecture Notes in Computer Science*, 2007, pp. 155–168.
[56] C. Linares López, Multi-valued pattern databases, in: M. Ghallab, C. D. Spyropoulos, N. Fakotakis, N. M. Avouris (Eds.), Proceedings of the European Conference on Artificial Intelligence (ECAI), volume 178 of *Frontiers in Artificial Intelligence and Applications*, 2008, pp. 540–544.
[57] Á. Torralba, Symbolic Search and Abstraction Heuristics for Cost-Optimal Planning, Ph.D. thesis, Universidad Carlos III de Madrid, 2015.
[58] 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*, 2007, pp. 20–34.
[59] S. Edelkamp, P. Kissmann, Á. Torralba, Symbolic A^* search with pattern databases and the merge-and-shrink abstraction, in: L. D. 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*, 2012, pp. 306–311.
[60] M. Helmert, G. Röger, S. Sievers, On the expressive power of non-linear merge-and-shrink representations, in: R. I. Brafman, C. Domshlak, P. Haslum, S. Zilberstein (Eds.), Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS), 2015, pp. 106–114.
[61] F. Somenzi, Cudd: Cu decision diagram package release 2.5.0, 2012. URL: `http://vlsi.colorado.edu/ fabio/CUDD/`.
[62] M. Helmert, The Fast Downward planning system, Journal of Artificial Intelligence Research (JAIR) 26 (2006) 191–246.
[63] 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), 2015, pp. 2–6.
[64] 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), 2005, pp. 1163–1168.
[65] 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), 2013, pp. 175–183.
[66] 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), 2013, pp. 206–214.
[67] C. A. Knoblock, Automatically generating abstractions for planning, Artificial Intelligence Journal 68 (1994) 243–302.
[68] C. Bäckström, P. Jonsson, Planning with abstraction hierarchies can be exponentially less efficient, in: Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI), 1995, pp. 1599–1605.
[69] R. I. Brafman, C. Domshlak, Structure and complexity in planning with unary operators, Journal of Artificial Intelligence Research (JAIR) 18 (2003) 315–349.
[70] M. Helmert, A planning heuristic based on causal graph analysis, in: S. Zilberstein, J. Koehler, S. Koenig (Eds.), Proceedings of the AAAI Conference on Artificial Intelligence (AAAI), AAAI, 2004, pp. 161–170.
[71] Á. Torralba, P. Kissmann, Focusing on what really matters: Irrelevance pruning in merge-and-shrink, in: L. Lelis, R. Stern (Eds.), Proceedings of the Symposium on Combinatorial Search (SoCS), 2015, pp. 122–130.
[72] J. Hoffmann, P. Kissmann, Á. Torralba, "distance"? who cares? tailoring merge-and-shrink heuristics to detect unsolvability, in: T. Schaub, G. Friedrich, B. O'Sullivan (Eds.), Proceedings of the European Conference on Artificial Intelligence (ECAI), volume 263 of *Frontiers in Artificial Intelligence and Applications*, 2014, pp. 441–446.
[73] S. Sievers, M. Wehrle, M. Helmert, A. Shleyfman, M. Katz, Factored symmetries for merge-and-shrink abstractions, in: Proceedings of the AAAI Conference on Artificial Intelligence (AAAI), 2015, pp. 3378–3385.
[74] 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), 2008, pp. 1452–1453.
[75] 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), 2011, pp. 18–23.
[76] S. Eriksson, G. Röger, M. Helmert, Unsolvability certificates for classical planning, in: Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS), 2017, pp. 88–97.
[77] T. Ball, A. Podelski, S. K. Rajamani, Boolean and cartesian abstraction for model checking c programs, in: International Conference on Tools and Algorithms for the Construction and Analysis of Systems, 2001, pp. 268–283.
[78] J. Seipp, M. Helmert, Counterexample-guided cartesian abstraction refinement, in: Proceedings of the International Conference on Automated Planning and Scheduling (ICAPS), 2013, pp. 347–351.

[79] K. Ravi, K. L. McMillan, T. R. Shiple, F. Somenzi, Approximation and decomposition of binary decision diagrams, in: Proceedings of the Design Automation Conference (DAC), 1998, pp. 445–450.
[80] Á. Torralba, Sympa: Symbolic perimeter abstractions for proving unsolvability, in: UIPC 2016 Planner Abstracts, 2016, pp. 8–11.