Constrained Symbolic Search: On Mutexes, BDD Minimization and More

Álvaro Torralba, Vidal Alcázar
{atorralb,valcazar}@inf.uc3m.es
Planning and Learning Group
Universidad Carlos III de Madrid
Leganés (Madrid), Spain

Abstract

Symbolic search allows saving large amounts of memory compared to regular explicit-state search algorithms. This is crucial in optimal settings, in which common search algorithms often exhaust the available memory. So far, the most successful uses of symbolic search have been bidirectional blind search and the generation of abstraction heuristics like Pattern Databases. Despite its usefulness, several common techniques in explicit-state search have not been employed in symbolic search. In particular, mutexes and other constraining invariants, techniques that have been proven essential when doing regression, are yet to be exploited in conjunction with BDDs. In this paper we analyze the use of such constraints in symbolic search and its combination with minimization techniques common in BDD manipulation. Experimental results show a significant increase in performance, considerably above the current state of the art in optimal planning.

Introduction

Automated Planning consists on finding a sequence of actions, commonly called a plan, that achieves a set of goals from a given initial state. In optimal planning, in which a plan of least cost must be found, the most popular approach is using A* (Hart, Nilsson, and Raphael 1968) combined with an admissible heuristic. The main shortcoming of A* is its memory requirements. When using heuristics in optimal search, all the nodes whose *f-value* is less than the cost of the optimal solution must be expanded (unless they are pruned by an optimality-preserving pruning technique). Hence, if the heuristic is not accurate enough, the number of generated nodes may exceed what the main memory can store.

Several alternatives to A* have been proposed, like symbolic search, which uses Binary Decision Diagrams (BDDs) (Bryant 1986) to represent sets of states instead of storing them individually. When using BDDs, a potentially exponential saving in memory may be obtained. This means that symbolic versions of common search algorithms, like symbolic blind search and BDD-A* (Edelkamp and Reffel 1998), are often able to solve problems that the explicit-state versions are unable to solve due to memory problems.

The main uses of symbolic search have been so far regular bidirectional search and the generation of abstraction heuristics (Edelkamp and Reffel 1998; Torralba, Linares López, and Borrajo 2013) for their use in both symbolic and explicit search algorithms. These methods require performing regression on the goals of the problem. Regression in planning is known to be less robust than progression due to the existence of multiple goal states, the presence of partially-defined states and the impact that spurious states (states that are not reachable from the initial state) have on the search (Bonet and Geffner 2001). To alleviate this, constraints obtained from invariants of the problem such as binary static mutexes (Blum and Furst 1997) and invariant groups (Helmert 2006) have been thoroughly employed (Bonet and Geffner 2001; Haslum et al. 2007). Surprisingly enough, the use of these constraints has not been extrapolated to symbolic search except for the monotonicity analysis required to transform the planning task into SAS$^+$ (Kissmann and Edelkamp 2011). Although the size of a BDD does not have to be proportional to the number of states it contains, there may exist a correlation, in which case constraints may help.

Also, the use of constraints is commonplace in BDD manipulation. Some minimizing operations have been specifically designed to reduce the size of a BDD when subject to a given constraint in the form of another BDD. The use of these minimization operations may translate not only into smaller (and thus more memory-efficient) BDDs but also in faster BDD manipulation, as the time consumed by the logical operations performed over BDDs depends on their size. Taking all this into account, in this paper we study the impact of constraints such as mutexes and invariant groups in symbolic search and symbolic abstractions and how minimization of BDDs affects the performance of these techniques.

Background

In this section some background regarding automated planning and BDDs is presented. In particular, the SAS$^+$ formalization of a planning task, the invariants used throughout this paper and a description of BDDs will be given.

SAS$^+$

A planning task in SAS$^+$ (Bäckström and Nebel 1995) is defined as a tuple $\Pi = (\mathcal{V}, s_0, s_\star, \mathcal{O})$. $\mathcal{V}$ is a set of state

variables, and every variable $v \in \mathcal{V}$ has an associated extended domain $D_v^+ = D_v \cup \{\mathbf{u}\}$ composed by the regular domain of each variable, D_v, and the undefined value $\mathbf{u}$. The total state space is defined as $S_v^+ = D_{v_1}^+ \times \ldots \times D_{v_n}^+$ and the value of a variable $v \in \mathcal{V}$ in a given state s, also known as a variable-value pair or *fluent*, is defined as $s[v]$. Partial states are states in which at least one fluent $s[v_i] = \mathbf{u}$. s_0 is the initial state, defined over $\mathcal{V}$ such that $s_0[v_i] \neq \mathbf{u}\ \forall v_i \in \mathcal{V}$. $s_\star$ is the partial state that defines the goals. $\mathcal{O}$ is a set of operators (actions), where each operator is a tuple $o = (pre(o), post(o), prev(o))$, where $pre(o), post(o), prev(o) \in S_v^+$ represent the *pre-*, *post-* and *prevail-conditions* respectively.

An operator o in a state s is applicable *in progression* if $\forall v_i \in \mathcal{V} : (prev(o)[v_i] = \mathbf{u} \vee prev(o)[v_i] = s[v_i]) \wedge (pre(o)[v_i] = \mathbf{u} \vee pre(o)[v_i] = s[v_i])$. The resulting state s' from the application of o in s is equal to s except that $\forall v_i \in \mathcal{V} : post(o)[v_i] \neq \mathbf{u}$ then $s'[v_i] = post(o)[v_i]$. An operator $o \in \mathcal{O}$ is applicable in a partial state s *in regression* if $\forall v_i \in V : s[v_i] = \mathbf{u} \vee s[v_i] = post(o)[v_i] \vee s[v_i] = prev(o)[v_i]$ and $\exists v_i \in V : s[v_i] = post(o)[v_i] \wedge s[v_i] \neq \mathbf{u}$. The resulting state s' obtained from applying o in s in regression is equal to s except that $\forall v_i \in \mathcal{V} : prev(o)[v_i] = \mathbf{u}$ then $s'[v_i] = pre(o)[v_i]$ and $\forall v_i \in \mathcal{V} : prev(o)[v_i] \neq \mathbf{u}$ then $s'[v_i] = prev(o)[v_i]$.

Spurious States, Mutexes and Invariant Groups

A spurious state is a state not reachable from s_0, as defined by Bonet and Geffner (2001)[1]. Such states may be generated when doing regression and are *dead ends* in backward search. A set of fluents $M = \{f_1, \ldots, f_m\}$ is a set of mutually exclusive fluents of size m (mutex of size m) if there is no reachable (non-spurious) state $s \subseteq S$ such that all elements in M are true in s. Computing mutexes is exponential on size *m*, so in most cases finding mutexes of size $m > 2$ is not practical. The most common method to compute mutexes of size two is h^m (Bonet and Geffner 2001) with $m = 2$.

Planners that use SAS$^+$ transform a planning task expressed in the Planning Domain Definition Language (PDDL) into SAS$^+$ by computing invariant groups. An invariant group is a set of fluents θ such that every fluent $f_i \in \theta$ is mutex with every other fluent $f_j \in \theta$ if $f_j \neq f_i$ and such that exactly one fluent $f_i \in \theta$ must be true in every non-spurious complete state. Every variable $v_i \in \mathcal{V}$ is an invariant group, although not every invariant group corresponds to a variable. Invariant groups are computed doing a monotonicity analysis (Helmert 2006). Note that per definition binary mutexes can also be computed doing a monotonicity analysis, although the set of mutexes obtained is a subset of the mutexes obtained with h^2.

Binary Decision Diagrams and Symbolic Search

Decision Diagrams are data structures inspired by the graphical representation of a logical function. A Binary Decision Diagrams (BDD) is a directed acyclic graph with two terminal nodes (called *sinks*) labeled with *true* and *false*. All the internal nodes are labeled with a binary variable $v \in \mathcal{V}$ and have two outgoing edges that correspond to the cases in which v is *true* and *false* respectively. When representing a multi-valued variable $v \in \mathcal{V}$, $\lceil \log_2 |D_v| \rceil$ binary variables are used instead. For any assignment of the variables on a path from the root to a sink, the represented function will be evaluated to the value labeling the sink. A more general type of decision diagram is an Algebraic Decision Diagram (ADD) (Bahar et al. 1997), which can have an arbitrary number of different sink nodes. This allows evaluating the represented function to values different from *true* and *false*.

Variables in a BDD are ordered. This has two advantages: first, the operations performed between BDDs with the same variable ordering are quadratic in the worst case; second, for a given set of states an ordered BDD guarantees uniqueness. This is achieved thanks to the reducing operations called the *deletion rule*, in which nodes with outgoing edges that lead to the same successor are removed, and the *merging rule*, in which nodes labeled with the same variable are merged if their respective successors are the same.

The set of operators $\mathcal{O}$ is represented using one or more *Transition Relations* (TRs). A TR is a function defined over two sets of variables, one set x representing the *from*-set and another set x' representing the *target*-set. Any given TR represents one or more operators $o \in \mathcal{O}$ with the same cost. To compute the successors of a set of states S_g, the *image* operation is used. The definition of *image* is as follows: $image(S_g, TR_i) = \exists x\ .\ S_g(x) \wedge TR_i(x, x')[x' \leftrightarrow x]$. Thus, *image* is carried out in three steps:

1. The conjunction with $TR_i(x, x')$ filters preconditions on x and applies effects on x'.
2. The existential quantification of the predecessor variables x removes their values relative to the predecessor states.
3. $[x' \leftrightarrow x]$ denotes the swap of the two sets of variables, setting the value of the successor states in x.

Similarly, regression uses the *pre-image* operator: $pre\text{-}image(S_g, TR_i) = \exists x'\ .\ S_g(x') \wedge TR_i(x, x')[x \leftrightarrow x']$.

Encoding Mutexes as a BDD

Ever since the first application of heuristic backward search in domain-independent planning, binary static mutexes have been considered essential (Bonet and Geffner 2001). Binary mutexes allow pruning spurious states that otherwise would be considered for expansion during search. Expanding such states may lead to an exponential decrease in performance, as none of the successors of a spurious state may lead to the initial state by doing regression. Their use in explicit-state search is straightforward: simply prune every state s such that fluents $f_i, f_j \in s$ are mutex.

Despite the impact that the use of mutexes in regression has, surprisingly this technique has not been employed in symbolic search. Although it is obvious that a *per state* application of mutexes in symbolic search is impossible, there are alternatives. In particular, we propose *creating a BDD that represents in a succinct way all the states that would be pruned if mutexes were used*. This BDD can be used to

[1] An alternative definition is given by Zilles and Holte (2010).

discard all the states that have been generated using a TR in a similar way as it is done with the closed list.

The *mutex BDD* ($mBDD$) is created in a very simple way. Every binary mutex is a conjunction of fluents f_i, f_j such that, if $f_i, f_j \in s$ in state s, then s is spurious. Hence, the set of states that can be pruned using mutexes are those in which at least one such conjunction of fluents is true. This way, the logical expression represented by $mBDD$ is the disjunction of all the mutexes found with h^2 (represented by the conjunction of both fluents). Formally, if M is the set of binary mutexes found by h^2:

$$mBDD = \bigvee_{<f_i,f_j>\in M} f_i \wedge f_j$$

Spurious states determined by mutexes are pruned by computing the difference $S_g \setminus mBDD$ of a newly generated set of states S_g with the mutex BDD $mBDD$. In terms of BDD manipulation this is done by computing the logical conjunction of S_g with the negation of $mBDD$: $S_g \wedge \neg mBDD$. This operation is the same as the one done in symbolic search with the BDD that represents the set of closed states, used to prune duplicates.

Pruning with Invariant Groups

Invariant groups are groups of fluents such that exactly one fluent must be true in every reachable complete state. This invariant of the problem is always respected when doing progression, but it may be violated in regression (Alcázar et al. 2013). Figure 1 shows an example of such an invariant violated in the *floortile* domain.

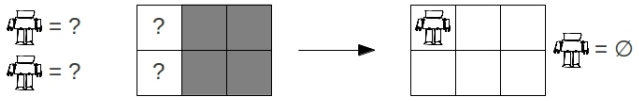

Figure 1: Spurious state in *floortile* that violates a constraint induced by an invariant group. The second robot in the state at the right has no valid location because all cells are either clear or occupied.

Problems in the *floortile* domain consist of two or more robots that have to paint the cells of a grid. The initial state contains the locations where the robots are and the goal contains the painted cells. In regression it is possible to find a plan in which a single robot traverses and paints the whole grid. This means that there may be a partial state in which all the cells are either painted, clear or occupied by the first robot while the position of the other robots remains unknown. If we take into account the variables (which are in fact invariant groups) that represent the position of the other robots, we can see that there are no legal values for them, as a robot cannot be at a painted cell, a clear cell or a cell occupied by another robot. Such a state is spurious and thus can be safely pruned.

Given an invariant group $\theta = f_1, \ldots, f_n$, two constraints may be deduced: first, the set of all the mutexes of the form $\neg(f_i \wedge f_j)$ if $f_i \neq f_j$ (which enforces that at most one fluent can be true at the same time); second, the fact that at least one fluent $f_i \in \theta$ must be true in every reachable complete state. The first constraint overlaps with the mutexes computed with h^2, so it does not make sense to include it as a constraint of the problem if a *mutex BDD* is used. The latter however can be encoded as an additional constraint of the form $f_1 \vee f_2 \vee \ldots \vee f_n$ for every invariant group. Such a constraint can be included in the *mutex BDD*, allowing to prune spurious states like the one previously presented with no further modification.

Pruning Spurious Operators

The use of constraints deduced from invariant groups is not limited to states generated in regression. Any set of fluents that does not correspond to a complete state can be detected as unreachable by disambiguating it (Alcázar et al. 2013). Disambiguating consists on solving a CSP using the invariant groups for which there is no known fluent yet as variables and the mutexes as constraints. If there is no valid assignment for one or more invariant groups, then there is no state in which all the fluents belonging to the set are true and so the set of fluents is unreachable.

When grounding a planning instance, spurious operators may be instantiated if done naively. Spurious operators do not generate spurious states in progression because they are never applicable, but they may have a negative impact when combined with other techniques, like regression, abstractions and delete-relaxation heuristics. Most planners prune instantiated operators with unreachable fluents in their preconditions, but no additional method is used to detect spurious operators. In this work we disambiguate the preconditions of all the instantiated operators $o \in \mathcal{O}$ and prune those operators with spurious sets of preconditions during the preprocessing phase.

Encoding Constraints in the TRs

Mutexes allow pruning spurious states after they are generated. However, more efficient alternatives that avoid the generation of the state exist. In particular, the use of *e-deletion* (Vidal and Geffner 2005), another invariant of the problem, avoids the generation of spurious states in explicit-state search by modifying the definition of applicability in regression (Alcázar et al. 2013). The definition of *e-deletion* is as follows:

Definition 1. *An operator o e-deletes a fluent f if f must be false in every state resulting from the execution of an ordered set of operators whose last operator is o.*

There are three cases in which an operator *e-deletes* a fluent f: it deletes f; it has a set of preconditions mutex with f and does not add f; or it adds a set of fluents mutex with f. For example, in *Blocksworld* the operator *(stack b c) e-deletes (on a b)* because it adds *(clear b)*, which is mutex with *(on a b)*. In multi-valued representations, deleting a fluent means changing the value of the variable it corresponds to, which is equivalent to adding a fluent mutex with f. Hence, the first case is a particular instance of the third case in multi-valued representations.

To avoid the generation of spurious states from a given state s in regression, one must make sure not to use an op-

erator that *e-deletes* some fluent $f \in s$ to generate a successor state. Formally, if $e\text{-}del(o)$ is the set of fluents *e-deleted* by an operator $o \in \mathcal{O}$, o is not applicable in regression in a partial state s if $e\text{-}del(o) \cap s \neq \emptyset$. An intuitive way of understanding the concept of *e-deletion* in regression is to consider the fluents $f_i \in e\text{-}del(o)$ as negative preconditions of a in regression.

In explicit-state search a partial state with one or more undefined variables may implicitly represent spurious states. In symbolic search, however, some considerations are necessary: first, there is no $\mathbf{u}$ value; second, the result of the application of several operators is aggregated. This means that it does not suffice to encode $e\text{-}del(o)$ as negative effects of an operator in a TR, as the union of the successor states may generate a BDD that covers spurious states if the predecessor sets covered spurious states as well.

A possibility could be replicating all the constraints in the TRs, although this may lead to a great degree of redundancy. Hence, a more succinct alternative is preferable. The first step is pruning the spurious states in the BDD that represents $s_\star$. This is done by intersecting with the *mutex BDD* as described before. Afterwards, the relevant constraints are added as prevail conditions of the operators. This serves to encode both *e-deletion* (fluents that must remain false "during" the execution of an operator) and *disambiguation* of operators (Alcázar et al. 2013), as the constraints may enforce the value of some variables when the *pre-image* is computed.

Definition 2. *Let a constraint c be a logical function over a set of fluents $m(c)$ such that $\forall f \in m(c)$, f appears in c. Let $o \in \mathcal{O}$ be an operator and $\mathcal{V}_u(o) \subseteq \mathcal{V} = \{v_i : post(o)[v_i] \neq \mathbf{u} \wedge pre(o)[v_i] = \mathbf{u}\}$. Let $M_o = \{m(c_1), \ldots, m(c_n)\}$ be the set of fluent sets such that $\forall m(c_i) \in M_o : (m(c_i) \cap pre(o) \neq \emptyset) \vee (\exists v_i \in \mathcal{V}_u(o) : m(c_i) \cap D_{v_i} \neq \emptyset)$.*

Then, the e-deleting operator o^e becomes o with prevail conditions $prev(o^e) = prev(o) \wedge \bigwedge_{\forall m(c_i) \in M_o} c_i$.

This suffices to guarantee that the successor set does not cover spurious states.

Theorem 1. *Let $S \subseteq \neg mBDD$ be a state set that does not contain states detected as spurious and o^e the e-deleting version of an operator o. Let S' be the resulting state set from applying o^e in regression to S. Then, $S' \subseteq \neg mBDD$ does not contain states that can be detected as spurious.*

Proof. If state $s' \in S'$ is generated using o^e in regression from state $s \in S$, then by definition $s'[v] = s[v]\ \forall v \notin pre(o) \cup \mathcal{V}_u$. As $s \in \neg mBDD$, s' automatically satisfies all the constraints not related to fluents in $pre(o)$ or variables in $\mathcal{V}_u$. As prevail conditions of the operator must be true both in s and s', s' satisfies all the constraints related to fluents in M_o, that is, related to fluents in $pre(o)$ or variables in $\mathcal{V}_u(o)$. As TRs apply the semantics of the operators they encode over sets of states, the result above can be extended to sets of states in symbolic search. □

After pruning the spurious states from $s_\star$ the *mutex BDD* is not needed anymore, so no intersections with it are necessary and so the *mutex BDD* can be discarded.

BDD Minimization

The main motivation for using a *mutex BDD* is to prune states so the BDDs that represent sets of states are smaller. However, computing the difference with the *mutex BDD* does not guarantee that the resulting BDD will be smaller. Imagine the following case: a planning task has a single fluent as goal, which means that at the layer 0 in regression we have a BDD with a single inner node. If the *mutex BDD* is used to prune unreachable states, the BDD resulting from the difference of the original BDD with the *mutex BDD* will be considerably bigger, as it will include additional information. It will represent fewer states, as the states that contain the goal fluent and violate some mutex will be effectively pruned, but it will also increase in size, which may be detrimental to the search.

In symbolic search, the performance of the search algorithm is often heavily linked to the size of the BDDs it works with. Both memory and time (in terms of BDD manipulation) benefit from working with BDDs that succinctly represent a given boolean function. In the literature, mainly in works published by the Model Checking community, several minimization algorithms have been proposed (Coudert and Madre 1990; McMillan 1996; Hong and Beerel 1997). These algorithms share in common that they work with a *function BDD* f and they receive an additional *constraint BDD* c (also called *restrict* or *care BDD*). The minimizing algorithms aim to find a BDD g that represents f's function in an incompletely specified way and is smaller than the conjunction (a logical *and*) of f and c.

It is easy to see that in regression f corresponds to the BDDs that represent sets of states, whereas c can be any BDD that imposes some kind of restriction over c. In our case, both the *closed list BDD* and the *mutex BDD* correspond to the definition of c. Hence, using minimization algorithms instead of the conjunction is straightforward. These operations are more expensive to compute than the conjunction, but if g is smaller the computation of *image* and *pre-image* (which are the most expensive operations in most planning instances) may require less time. The following are the minimization algorithms considered in this work:

- *restrict* (Coudert and Madre 1990): it performs sibling-substitution recursively to prune the parents of such siblings. If g is larger than f, f is returned instead.
- *constrain* (McMillan 1996): also known as *generalized-cofactor*, it is the same as *restrict* but uses support variables to find additional matches between sibling nodes.
- *don't care minimization* (Hong et al. 2000): also known as *leaf-identifying compaction*, it assigns *don't cares* to binary values to avoid sibling-substitutions that cause the growth of g. It ensures that g is smaller than f, as opposed to *restrict* and *constrain*.
- *non-polluting-and*: same as *restrict*, but the variables in c that do not appear in f are ignored.

Constrained Symbolic Abstraction Heuristics

The use of regression is not limited to backward search. For instance, Pattern Databases (Culberson and Schaef-

fer 1998) perform regression over the goals in an abstraction of the original problem to create a lookup table that is used as the distance estimation in the original problem. Pattern Databases (PDBs) in explicit-search that make use of mutexes are known as Constrained PDBs (Haslum et al. 2007). Constrained PDBs prune transitions that go through abstracted states that violate the constrains, which may strengthen the derived heuristic. A symbolic version of PDBs for their use in symbolic search has also been proposed (Edelkamp 2002), so in this work we propose a constrained version of symbolic PDBs.

A more recent development, Symbolic Merge and Shrink (SM&S) (Torralba, Linares López, and Borrajo 2013), proposed deriving a heuristic from a backward symbolic search that uses abstractions to reduce the size of the g-layers once the BDDs that represented them surpassed a given size. These abstractions are automatically derived using the M&S algorithm (Helmert, Haslum, and Hoffmann 2007), originally defined in Model Checking (Dräger, Finkbeiner, and Podelski 2009). M&S incrementally builds an abstraction, using a shrinking policy such as bisimulation (Nissim, Hoffmann, and Helmert 2011) to keep its size bounded. As in symbolic backward search, the use of a *mutex BDD* is also possible, creating a constrained version of SM&S.

Experimentation

In this section we analyze the impact of using constraints and BDD minimization in different settings, ranging from symbolic blind search to the generation of symbolic abstraction heuristics. Our motivation is to check whether h^2 mutexes and invariant groups improve over the constraints inherent to the SAS$^+$ formulation of the problem and to see if more complex BDD manipulation pays off.

h^2 was implemented on top of FAST DOWNWARD. For symbolic blind search and symbolic A* (BDD-A*) with PDBs we use the GAMER planner (Kissmann and Edelkamp 2011). As each planner uses its own SAS$^+$ variables, mutex fluents are extrapolated to GAMER's SAS$^+$ encoding. All the experiments with GAMER use the same variable ordering, which is optimized prior to the search. For the SM&S abstractions with explicit-state A* we use the implementation in FAST DOWNWARD (Helmert 2006).

Results with different methods to prune spurious states are reported. Unless otherwise stated, the *mutex BDD* contains constraints from both h^2 mutexes and invariant groups. $\mathcal{M}_\emptyset$ is the baseline version which does not use the *mutex BDD*. $\mathcal{M}^{\&}$ computes the conjunction with the negation of the *mutex BDD*. The four aforementioned BDD-minimization algorithms are used: *don't care minimization*($\mathcal{M}^{dcm}$), *restrict*($\mathcal{M}^{res}$), *constrain*($\mathcal{M}^{con}$) and *non-polluting-and*($\mathcal{M}^{np\&}$). Finally *e-del* is the version in which the TRs take into account *e-deletion* and invariant group constraints instead of using a *mutex BDD*. Results are reported with the regular set of operators ($\mathcal{O}$) and with the set of operators after pruning spurious operators found by disambiguating their preconditions ($\mathcal{O}^-$).

We run experiments on the benchmarks of the optimal track of the International Planning Competition 2011[2]. All our experiments were run and validated with the IPC-2011 software on a single core of an Intel(R) Xeon(R) X3470 processor at 2.93GHz. The experimental setting is the same as in IPC-2011: 1800 seconds per problem and 6GB of available memory. Time score and coverage follow the same rules as in IPC-2011 too. BDD operations are implemented using Fabio Somenzi's CUDD[3] 2.5.0 library. For the *image* and *pre-image* computation we used a disjunctive partition of the TRs, merging the TR of each operator up to a maximum size of 100,000 nodes. This method is simple and has proved to be more efficient than other approaches (Torralba, Edelkamp, and Kissmann 2013). The *mutex BDD* is created by merging the conjunction of the individual pairs of fluents that are mutex and the constraints from the invariant groups. Similarly, if a single *mutex BDD* surpasses 100,000 nodes, it is also encoded as several smaller ones.

In ELEVATORS and TRANSPORT neither additional h^2 mutexes nor constraints from invariant groups were found. In FLOORTILE, NOMYSTERY, OPENSTACKS, PARC-PRINTER, PEG-SOLITAIRE and WOODWORKING the *mutex BDD* had fewer than 10,000 nodes in all the problems. More than one individual *mutex BDD* were needed in some instances of the following domains (number of BDDs in the worst case between parentheses): BARMAN (2), PARKING (25), SCANALYZER (10), SOKOBAN (13), TIDYBOT (8) and VISITALL (6). Overall neither the size of the *mutex BDDs* nor the time spent pruning unreachable states was significant. Note that the size of the *mutex BDDs* also depends on the order of the variables: an order more suitable to their representation might reduce their size.

When encoding the constraints in the TRs (*e-del* configuration), the sizes of the TRs vary. In BARMAN, TIDYBOT and VISITALL the TRs actually become smaller in most instances. In OPENSTACKS, PEG-SOLITAIRE and WOODWORKING the size is roughly the same ($\pm 10\%$). In FLOORTILE the TRs grow by more than 50%, in NOMYSTERY and PARC-PRINTER they become several times bigger and in SOKOBAN and specially in SCANALYZER the number of individual TRs needed to encode the operators grows by a significant amount. The worst case is PARKING, in which the size of the TRs of the individual operators blows up when the constraints are added and exceed the available memory even before beginning the search. This is due to the high number of mutexes that are found in this domain, as all variables in PARKING interact heavily with each other.

Using constraints to prune spurious operators reduces the number of operators in six domains: TIDYBOT, WOODWORKING, NOMYSTERY, SCANALYZER, BARMAN and PARKING. The geometric mean of the percentage of pruned operators is 85%, 46%, 37%, 36%, 28% and 10% respectively. Additionally, in NOMYSTERY a few extra mutexes are found if h^2 is recomputed without the pruned operators and in TIDYBOT some fluents are found to be unreachable.

[2] http://www.plg.inf.uc3m.es/ipc2011-deterministic

[3] http://vlsi.colorado.edu/ fabio/CUDD

	BACKWARD									FORWARD	
	$\mathcal{M}_\emptyset$		$\mathcal{M}^{\&}$		$\mathcal{M}^{dcm}$	$\mathcal{M}^{res}$	$\mathcal{M}^{con}$	$\mathcal{M}^{np\&}$	*e-del*	$\mathcal{M}_\emptyset$	
	$\mathcal{O}$	$\mathcal{O}^-$	$\mathcal{O}$	$\mathcal{O}^-$	$\mathcal{O}^-$	$\mathcal{O}^-$	$\mathcal{O}^-$	$\mathcal{O}^-$	$\mathcal{O}^-$	$\mathcal{O}$	$\mathcal{O}^-$
BARMAN	0.00	0.00	5.69	5.98	1.13	1.58	1.64	5.91	**9.29**	7.35	7.47
ELEVATORS	2.48	2.49	2.48	2.48	2.48	2.49	2.49	2.49	2.49	**15.88**	15.82
FLOORTILE	5.06	5.18	13.25	**13.60**	10.02	11.89	12.60	13.41	13.39	0.71	0.73
NOMYSTERY	10.46	10.96	10.21	10.95	10.48	10.61	10.41	10.94	**11.04**	9.63	9.33
OPENSTACKS	15.30	14.68	16.19	15.97	7.59	10.75	11.47	15.85	15.95	**19.77**	18.22
PARC-PRINTER	4.23	4.18	15.14	15.27	12.04	13.54	13.60	14.69	**15.48**	5.31	5.55
PARKING	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**	**0.00**
PEG-SOLITAIRE	0.00	0.00	2.07	1.92	0.85	0.85	0.85	1.92	2.20	**16.74**	16.38
SCANALYZER	8.72	8.48	8.38	8.43	3.08	3.65	3.62	8.45	**9.00**	8.49	8.20
SOKOBAN	0.27	0.27	16.48	**16.68**	11.15	12.35	14.50	16.12	16.64	15.87	14.96
TIDYBOT	0.00	1.00	2.71	6.65	0.87	0.93	0.87	4.74	6.80	10.05	**13.92**
TRANSPORT	1.79	1.79	1.79	1.79	1.79	1.79	1.79	1.79	1.79	**6.00**	5.85
VISITALL	8.51	8.51	8.52	8.52	8.43	8.52	**8.77**	8.52	8.57	7.62	7.60
WOODWORKING	10.76	9.83	17.43	17.69	8.47	8.95	9.03	17.13	**18.68**	5.31	4.97
TOTAL	67.58	67.37	120.34	125.92	78.40	87.89	91.64	121.96	**131.32**	128.73	128.99
COVERAGE	92	94	144	148	113	121	122	145	**150**	149	149

Table 1: Time score and total coverage of unidirectional blind search.

Symbolic Unidirectional Blind Search

First we start with the simplest case, backward blind search. We compare the performance of symbolic backward search with and without constraints against symbolic forward search. Table 1 shows the time score comparison of different configurations of forward and backward blind search.

The impact of pruning spurious operators ($\mathcal{O}^-$) is small except in TIDYBOT and sometimes does not compensate the time spent computing h^2 if this is only done to prune such operators (forward and backward $\mathcal{M}_\emptyset$). As expected, it benefits backward search more than forward search, as spurious operators may be applicable in regression. Additionally, in versions that already compute h^2 for other purposes, the score with $\mathcal{O}^-$ stays roughly the same or improves, so operator pruning is always recommended in these cases.

Using constraints to prune spurious states improves the results by a very significant margin, almost doubling the time score and solving 50% more problems overall. In BARMAN, SOKOBAN, TIDYBOT and PEG-SOLITAIRE almost no search was accomplished by backward search without mutexes, while pruning spurious states allows backward search to solve some problems in those domains. For example, the maximum g-layer expanded backwards without mutexes in the first problem of BARMAN is 5, whereas with mutexes the optimal solution, whose cost is 90, is found. Also, although computing the *mutex BDD* requires some time and memory, it does not harm in any domain.

The reported results include the invariant group constraints in all the configurations that use constraints. When disabling the use of these constraints the same coverage is obtained, although the time score worsens perceptibly in SOKOBAN and slightly in WOODWORKING and OPENSTACKS, losing 6 points overall with the $\mathcal{M}^{\&}$ configuration.

BDD minimization is not useful in this setting. This is because reducing the BDD that represents the set of predecessor states by including some spurious states often means that the set of successor states is bigger. This has no impact in terms of memory, as the set of successor states can be minimized afterwards in the same way, but it affects negatively the time required to compute the *pre-image* (apart from requiring more time than a regular conjunction with the *mutex BDD*). Overall and although in some cases a memory reduction of up to a third for some BDDs is obtained, the extra time does not pay off. Thus, $\mathcal{M}^{\&}$ dominates all the BDD minimization configurations in all domains.

Finally, encoding *e-deletion* (and invariant group constraints) in the TRs instead of using a *mutex BDD* is the most efficient version. This is because *e-del* directly generates BDDs that do not contain spurious states, instead of generating a potentially much bigger BDD with spurious states and intersecting it with the *mutex BDD* afterwards. Surprisingly, the size of the TRs does not affect the performance, which means that the performance of *pre-image* depends on the size of the resulting BDD rather than on the size of the predecessor BDD and the TRs.

On a per domain comparison we can observe that the directionality of the domains has a huge impact on the performance of the planner (Massey 1999). This hints that a bidirectional approach is probably more efficient than unidirectional search, which is explored in the following subsection.

Symbolic Bidirectional Blind Search

After assessing the viability of constraints in isolation, we now use them in a state-of-the-art symbolic bidirectional blind version of GAMER. Spurious operator pruning is enabled in all the configurations. The only other modification with respect to the version of GAMER used in (Torralba, Edelkamp, and Kissmann 2013) is that we dynamically check the time and memory consumed per step. During a step, if it uses more than twice the memory or time than the last step in the opposite direction, we interrupt it and switch the direction of the search. This makes $\mathcal{M}_\emptyset$ solve four fewer problems in TIDYBOT (7 instead of 11), three of which are recovered thanks to spurious operator pruning. We exclude

	$\mathcal{M}_\emptyset$		$\mathcal{M}^\&$		*e-del*	
BARMAN	5.46	8	9.78	11	**12.00**	**12**
ELEVATORS	18.13	**19**	**18.17**	**19**	17.92	**19**
FLOORTILE	5.89	10	**13.56**	**14**	13.46	**14**
NOMYSTERY	**16.00**	**16**	15.06	**16**	15.11	**16**
OPENSTACKS	17.68	**20**	18.93	**20**	**19.16**	**20**
PARC-PRINTER	5.79	8	14.09	15	**15.37**	**16**
PARKING	**0.00**	**0**	**0.00**	**0**	**0.00**	**0**
PEG-SOLITAIRE	14.65	17	**18.42**	**19**	18.38	**19**
SCANALYZER	8.52	**9**	8.28	**9**	**9.00**	**9**
SOKOBAN	13.63	**19**	**18.49**	**19**	18.21	**19**
TIDYBOT	8.98	10	**15.53**	**16**	15.33	**16**
TRANSPORT	8.67	**9**	**8.83**	**9**	8.67	**9**
VISITALL	**10.98**	**11**	10.95	**11**	10.84	**11**
WOODWORKING	10.81	16	17.74	**19**	**18.47**	**19**
TOTAL	145.17	172	187.83	197	**191.94**	**199**

Table 2: Time score and coverage of bidirectional blind search.

	$\mathcal{M}_\emptyset$ ($\mathcal{O}$)	$\mathcal{M}_\emptyset$ ($\mathcal{O}^-$)	$\mathcal{M}^\&$	*e-del*
BARMAN	6	7	**8**	**8**
ELEVATORS	**19**	**19**	**19**	**19**
FLOORTILE	12	12	**14**	**14**
NOMYSTERY	**14**	**14**	**14**	**14**
OPENSTACKS	**20**	**20**	**20**	**20**
PARC-PRINTER	**9**	**9**	**9**	**9**
PARKING	**1**	**1**	**1**	**1**
PEG-SOLITAIRE	**17**	**17**	**17**	**17**
SCANALYZER	**9**	**9**	**9**	**9**
SOKOBAN	**20**	**20**	**20**	**20**
TIDYBOT	14	**17**	**17**	**17**
TRANSPORT	**6**	**6**	**6**	**6**
VISITALL	**11**	**11**	**11**	**11**
WOODWORKING	**19**	**19**	**19**	**19**
TOTAL	177	181	**184**	**184**

Table 3: Coverage of Symbolic PDBs + BDD-A*.

BDD-minimization methods because, as shown in the unidirectional case, they are not useful in the symbolic blind search setting. Thus, we only compare the baseline with the version that uses the *mutex BDD* ($\mathcal{M}^\&$) and the one that uses *e-deletion* (*e-del*), both of them also with invariant group constraints.

Table 2 shows that symbolic bidirectional blind search is able to improve over both forward and backward search. As in the unidirectional version, a consistent improvement is obtained when using constraints, increasing the coverage from 172 problems to 199 with *e-del*. NOMYSTERY is the only domain where applying mutexes actually makes the search slightly slower, although this does not affect coverage. $\mathcal{M}^\&$ and *e-del* yield similar results in most domains (a difference in score smaller than 0.5 points), although an increase in performance is obtained by *e-del* in BARMAN, PARC-PRINTER, SCANALYZER and WOODWORKING.

Overall the results of the bidirectional blind search version of GAMER with *e-deletion* are remarkable, solving 14 more problems than the winner of IPC 2011, FAST DOWNWARD STONE SOUP (Helmert, Röger, and Karpas 2011).

BDD-A^* with PDBs

After observing the positive impact of constraints on symbolic blind search, we analyze the case of symbolic abstraction heuristics, in which symbolic backward search is performed in an abstracted state space. We performed experiments with BDD-A* guided by symbolic PDBs computed as described by Kissmann and Edelkamp (2011). Table 3 shows the results. First, pruning spurious operators allows solving four additional problems, one in BARMAN and 3 in TIDYBOT. When using mutexes, the coverage goes up to 184 problems with both $\mathcal{M}^\&$ and *e-del*. The use of BDD-minimization operators performed slightly worse than $\mathcal{M}^\&$ and *e-del*, so we did not include them for conciseness. The time score was also left out because BDD-A* spends half of the available time computing the PDBs, which skews the time score and makes it not as representative.

Comparing the results of Tables 1 and 3 we can see that constraints have a smaller impact when using abstractions. PDB heuristics are only useful in PARKING, SOKOBAN and TIDYBOT, where an additional problem per domain is solved. Constraints helped more in cases in which backward blind search was not feasible at all, as the number of problems in which a single step backward is not possible is reduced when using them. Overall, bidirectional blind search performs almost as good or better than BDD-A* with symbolic PDBs, as it can exploit the directionality of the planning instances better.

We can conclude that the pruning power of constraints is reduced considerably in abstracted spaces: the constraints in which abstracted variables appear are of no use and the abstracted space is less constrained than the original one, so there is less margin for improvement. Nevertheless the use of constraints never hurts and is in all cases equal or better than the configuration with no mutexes, so there is no reason why constraints should not be used.

Symbolic Merge-and-Shrink

Now we analyze a different type of abstraction heuristic: Symbolic Merge-and-Shrink (SM&S) (Torralba, Linares López, and Borrajo 2013) with explicit-state A*. SM&S does symbolic backward search until the size of the BDD surpasses a given threshold. The BDD is then reduced using shrinking strategies until its size is again below the threshold so the search can be continued. SM&S does not discard variables, so most constraints are useful during the whole generation of the heuristic, and initially performs search in the original space, which means that the full potential of constraints is exploited until shrinking is necessary. Table 4 shows the experimentation done with SM&S. All the configurations have spurious operator pruning enabled and use a threshold of 10,000 abstract states when shrinking the M&S abstractions. Both FAST DOWNWARD and GAMER variable orderings have been used. For each setting *bisimulation* (bop) and *greedy bisimulation* were tested. Among the BDD-minimizing operations $\mathcal{M}^{np\&}$ performed well, so its results are also included. Time score is obviated for the same reasons as for BDD-A^*.

	FAST DOWNWARD						GAMER					
	$\mathcal{M}_\emptyset$		$\mathcal{M}^{\&}$		$\mathcal{M}^{np\&}$		$\mathcal{M}_\emptyset$		$\mathcal{M}^{\&}$		$\mathcal{M}^{np\&}$	
	bop	gop	bop	gop	bop	gop	bop	gop	bop	gop	bop	gop
BARMAN	4	4	**8**	**8**	**8**	**8**	4	4	**8**	**8**	**8**	**8**
ELEVATORS	18	18	18	18	18	18	**19**	**19**	**19**	**19**	**19**	**19**
FLOORTILE	12	12	**14**	**14**	**14**	**14**	12	12	**14**	**14**	**14**	**14**
NOMYSTERY	**18**	16	**18**	16	**18**	16	16	14	16	14	16	14
OPENSTACKS	15	15	**16**	**16**	**16**	**16**	**16**	**16**	**16**	**16**	**16**	**16**
PARC-PRINTER	**12**	**12**	**12**	**12**	**12**	**12**	**12**	**12**	**12**	**12**	**12**	**12**
PARKING	6	**7**	3	4	3	5	6	1	6	1	6	1
PEG-SOLITAIRE	**20**	**20**	**20**	**20**	**20**	**20**	19	19	19	19	19	19
SCANALYZER	9	**10**	9	**10**	9	**10**	9	9	9	9	9	9
SOKOBAN	19	19	19	19	19	19	**20**	**20**	**20**	**20**	**20**	**20**
TIDYBOT	**13**	**13**	12	**13**	**13**	12	**13**	12	9	9	11	10
TRANSPORT	7	7	7	7	7	7	**8**	**8**	**8**	**8**	**8**	**8**
VISITALL	12	12	12	12	12	12	**13**	**13**	**13**	**13**	12	12
WOODWORKING	7	8	7	8	7	8	12	12	**16**	**16**	**16**	**16**
TOTAL	172	173	175	177	176	177	179	171	185	178	**186**	178

Table 4: Coverage of Explicit-A* with the SM&S heuristic.

In all the configurations the use of constraints improves the overall coverage, although in some domains some problems are lost. This loss of performance is due to the extra information that constraints add, which may cause the BDD to grow beyond the threshold earlier, forcing shrinking before it is really necessary. This is further increased when the variable order is not suitable for representing the set of spurious states, explaining the poor performance of constraints in TIDYBOT with the GAMER ordering. The most successful configuration, $\mathcal{M}^{np\&}$ with *bisimulation* and the GAMER ordering, solves 186 problems, 7 more than the most successful configuration without constraints. As expected, the benefit of using constraints with SM&S lies between regular search and symbolic PDBs. An important remark is that, unlike the regular (tabular) Merge-and-Shrink in explicit-state search, in which taking into account mutexes is not trivial, the use of constraints in SM&S is straightforward.

Discussion

In this work we showed the relative pruning power of h^2 mutexes in symbolic search, proving that the constraints encoded in the SAS^+ formulation do not suffice to detect a significant amount of spurious states in many domains. Additional constraining techniques were successfully employed, and the impact of BDD-minimization operations that work with constraining BDDs was tested. Note that h^2 mutexes were implicitly used before (Jensen et al. 2006), although the reported results did not show a significant increase in performance.

The results seem to contradict the assumption in planning that progression is more robust than regression, at least in optimal symbolic search. Previous results on both symbolic and explicit-state search suggested that forward search outperformed backward search in the IPC benchmarks. For example, in (Torralba, Edelkamp, and Kissmann 2013) it is reported that the percentage of forward search performed by GAMER with bidirectional blind search is greater than 90% for 6 out of 14 domains and 75% overall. Only in FLOORTILE and WOODWORKING backward search was superior to forward search. Another example is Patrik Haslum's forward version of HSPr in the IPC 2008[4], which was as good or better than the original backward version in all the domains except SCANALYZER. However, the results shown in Table 1 change this picture: when using h^2 mutexes, the results of backward and forward symbolic search are close, with a relatively high degree of variability between different domains.

An important conclusion to be drawn is that symbolic search seems to work better in regression than explicit-state search. The underlying reason is subsumption of states in regression. This occurs when the set of fluents that compose a partial state is a subset of the set of fluents of a newly generated one. In this case the latter should be reported as a duplicate of the former, which is seamlessly detected when using a symbolic closed list but which is not trivially detected when using a closed list in Disjunctive Normal Form, as it happens in explicit-state search. Similarly, the detection of the collision of frontiers in bidirectional search is trivial if the backward search is symbolic (the forward search can be either symbolic or explicit-state), which tips the scales further in favor of symbolic search in regression.

Regarding the good performance of symbolic bidirectional blind search, the impact of the directionality of the domains (Massey 1999) explains why this configuration fares so well, as a simple alternating strategy allows choosing the direction in which the problem may be more easily solved. We leave the implementation of a bidirectional version of BDD-A* that uses constraints as future work. We also plan on investigating whether variable orders derived from the constraints may be more useful than the range of orderings tested in the literature (Kissmann and Hoffmann 2013). Such orderings may provide a more concise representation of the set of spurious states, which could overcome the drawbacks of using constraints in domains like PARKING, in which the *mutex BDD* becomes too large to be manageable.

[4] http://ipc.informatik.uni-freiburg.de/Results

Acknowledgments

This work has been partially supported by the Spanish government through MICINN project TIN2011-27652-C03-02. The first author has also been supported by a grant of Universidad Carlos III de Madrid. The second author has also been supported by a FPI grant associated to MICINN project TIN2008-06701-C03-03.

References

Alcázar, V.; Borrajo, D.; Fernández, S.; and Fuentetaja, R. 2013. Revisiting regression in planning. In *International Joint Conference on Artificial Intelligence (IJCAI)*.

Bäckström, C., and Nebel, B. 1995. Complexity results for SAS+ planning. *Computational Intelligence* 11:625–656.

Bahar, R. I.; Frohm, E. A.; Gaona, C. M.; Hachtel, G. D.; Macii, E.; Pardo, A.; and Somenzi, F. 1997. Algebraic decision diagrams and their applications. *Formal Methods in System Design* 10(2/3):171–206.

Blum, A., and Furst, M. L. 1997. Fast planning through planning graph analysis. *Artif. Intell.* 90(1-2):281–300.

Bonet, B., and Geffner, H. 2001. Planning as heuristic search. *Artif. Intell.* 129(1-2):5–33.

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

Coudert, O., and Madre, J. C. 1990. A unified framework for the formal verification of sequential circuits. In *International Conference on Computer-Aided Design (ICCAD)*, 126–129.

Culberson, J. C., and Schaeffer, J. 1998. Pattern databases. *Comput. Intell.* 14(3):318–334.

Dräger, K.; Finkbeiner, B.; and Podelski, A. 2009. Directed model checking with distance-preserving abstractions. *International Journal on Software Tools for Technology Transfer* 11(1):27–37.

Edelkamp, S., and Reffel, F. 1998. OBDDs in heuristic search. In *German Conference on Artificial Intelligence (KI)*, 81–92.

Edelkamp, S. 2002. Symbolic pattern databases in heuristic search planning. In *Conference on Artificial Intelligence Planning Systems (AIPS)*, 274–283.

Hart, P.; Nilsson, N.; and Raphael, B. 1968. A formal basis for the heuristic determination of minimum cost paths. *Systems Science and Cybernetics, IEEE Transactions on* 4(2):100–107.

Haslum, P.; Botea, A.; Helmert, M.; Bonet, B.; and Koenig, S. 2007. Domain-independent construction of pattern database heuristics for cost-optimal planning. In *AAAI Conference on Artificial Intelligence (AAAI)*, 1007–1012.

Helmert, M.; Haslum, P.; and Hoffmann, J. 2007. Flexible abstraction heuristics for optimal sequential planning. In *International Conference on Automated Planning and Scheduling (ICAPS)*, 176–183.

Helmert, M.; Röger, G.; and Karpas, E. 2011. Fast Downward Stone Soup: A baseline for building planner portfolios. In *ICAPS-Workshop on Planning and Learning (PAL)*, 28–35.

Helmert, M. 2006. The Fast Downward planning system. *J. Artif. Intell. Res. (JAIR)* 26:191–246.

Hong, Y., and Beerel, P. A. 1997. Safe BDD minimization using don't cares. In *In Design Automation Conf*, 208–213.

Hong, Y.; Beerel, P. A.; Burch, J. R.; and McMillan, K. L. 2000. Sibling-substitution-based BDD minimization using don't cares. *IEEE Trans. on CAD of Integrated Circuits and Systems* 19(1):44–55.

Jensen, R. M.; Hansen, E. A.; Richards, S.; and Zhou, R. 2006. Memory-efficient symbolic heuristic search. In *International Conference on Automated Planning and Scheduling (ICAPS)*, 304–313.

Kissmann, P., and Edelkamp, S. 2011. Improving cost-optimal domain-independent symbolic planning. In *AAAI Conference on Artificial Intelligence (AAAI)*, 992–997.

Kissmann, P., and Hoffmann, J. 2013. What's in it for my BDD? on causal graphs and variable orders in planning. In *International Conference on Automated Planning and Scheduling (ICAPS)*.

Massey, B. 1999. *Directions In Planning: Understanding The Flow Of Time In Planning*. Ph.D. Dissertation, Computational Intelligence Research Laboratory, University of Oregon.

McMillan, K. L. 1996. A conjunctively decomposed boolean representation for symbolic model checking. In *Computer Aided Verification (CAV)*, 13–25.

Nissim, R.; Hoffmann, J.; and Helmert, M. 2011. Computing perfect heuristics in polynomial time: On bisimulation and merge-and-shrink abstraction in optimal planning. In *International Joint Conference on Artificial Intelligence (IJCAI)*, 1983–1990.

Torralba, Á.; Edelkamp, S.; and Kissmann, P. 2013. Transition trees for cost-optimal symbolic planning. In *International Conference on Automated Planning and Scheduling (ICAPS)*.

Torralba, Á.; Linares López, C.; and Borrajo, D. 2013. Symbolic merge-and-shrink for cost-optimal planning. In *International Joint Conference on Artificial Intelligence (IJCAI)*.

Vidal, V., and Geffner, H. 2005. Solving simple planning problems with more inference and no search. In *Principles and Practice of Constraint Programming (CP)*, 682–696.

Zilles, S., and Holte, R. C. 2010. The computational complexity of avoiding spurious states in state space abstraction. *Artif. Intell.* 174(14):1072–1092.