SymBA*: A Symbolic Bidirectional A* Planner

Álvaro Torralba and **Vidal Alcázar** and **Daniel Borrajo**
{alvaro.torralba, vidal.alcazar, daniel.borrajo}@uc3m.es
Universidad Carlos III de Madrid
Madrid, Spain

Peter Kissmann
kissmann@cs.uni-saarland.de
Saarland University
Saarbrücken, Germany

Stefan Edelkamp
edelkamp@tzi.de
University of Bremen
Bremen, Germany

Abstract

Lately, several important advancements have been obtained in symbolic search. First, bidirectional blind search has obtained good results on many domains. Second, perimeter-based abstraction heuristics have been proposed as an important improvement over regular abstraction heuristics. Motivated by the synergy between bidirectional search and perimeter-based abstraction heuristics, here we present SymBA*, which performs bidirectional A* using the frontiers of the opposite search to infer informed perimeter-based abstraction heuristics.

Motivation

Most cost-optimal planners are based on A* guided with an admissible heuristic. Bidirectional search has not been explored so extensively, due to the inherent difficulties of regression in planning and the computational cost of detecting collision between frontiers (Alcázar, Fernández, and Borrajo 2014). However, symbolic search (McMillan 1993) reasons using sets of states, avoiding the otherwise complex problem of subsumption of states and substantially reducing the cost of detecting the collision of frontiers. Moreover, recent advances in symbolic search planning have helped to overcome some of the problems of performing regression in planning (Torralba and Alcázar 2013). Thus, recent results have shown that symbolic bidirectional blind search has become one of the best alternatives for cost-optimal planning, outperforming not only A*-based planners but also BDDA*, the symbolic search variant of A*.

These recent improvements have risen the question of whether it is possible to use heuristics in combination with symbolic bidirectional search. Bidirectional heuristic search has a long history (Pohl 1969; de Champeaux 1983), but the hardness of proving optimality reduces considerably the advantages it has over regular A* (Kaindl and Kainz 1997). Because of this, bidirectional heuristic search has fallen out of flavor, with the majority of the search and planning community working mostly with regular A*.

Abstraction heuristics, like Pattern Databases (PDBs) (Culberson and Schaeffer 1998) and Merge-and-Shrink (M&S) (Helmert et al. 2014), are commonly used admissible heuristics. These heuristics can be enhanced with a perimeter (Felner and Ofek 2007; Eyerich and Helmert 2013; Torralba, Linares López, and Borrajo 2013), which leads to a strictly more informed heuristic than both the abstraction heuristic and the perimeter alone. Now, abstraction heuristics do not require a perimeter of a fixed radius to obtain better estimates – any frontier in the original space can be used as a seed to improve the heuristic as long as the g value of the expanded states is optimal. Because of this, we propose the use of the frontier in one direction in a bidirectional search algorithm to enhance an abstraction heuristic used by the search in the opposite direction.

The aim of the SymBA* planner is to exploit the synergy between bidirectional search and perimeter-based abstraction heuristics. SymBA* performs bidirectional searches on different state spaces. It starts in the original search space and, when the search becomes too hard, it derives an abstraction heuristic enhanced by the frontier of the opposite direction. The planner decides at any point whether to advance the search in the original state space, enlarging the perimeter, or search in an abstract state space to provide better heuristic estimations for the unabstracted search.[1]

This paper describes the main algorithm used by the planner and technical and implementation details. A more elaborate theoretical discussion will be presented in a future paper.

Symbolic Bidirectional A*

SymBA* performs several symbolic bidirectional A* searches on different state spaces. We denote a bidirectional search on a state space, Θ_i, as S^{Θ_i}. A bidirectional search is composed of two unidirectional searches in opposite directions: a forward search, $S^{\Theta_i}_{fw}$, and a backward search, $S^{\Theta_i}_{bw}$. We will use $S^{\Theta_i}_{u}$ to denote a unidirectional search in an unspecified direction. $f(S^{\Theta_i}_{u})$ denotes the value of the f-layer with minimum f.

First, SymBA* starts a bidirectional search in the original state space. Since no abstraction heuristic has been derived yet, it behaves like symbolic bidirectional blind search.

[1]A connection can be made with hierarchical heuristic search (Holte, Grajkowski, and Tanner 2005), in particular with Switchback (Larsen et al. 2010) and its improved version Short-Circuit (Leighton, Ruml, and Holte 2011), as both traverse the abstract state lazily to avoid searching parts that are irrelevant for the problem at hand.

This search continues until the next step in both directions is deemed as not searchable, because SymBA* estimates that it will take too much time or memory. Only then, a new bidirectional search is initialized in an abstract state space. Both the forward and backward searches are initialized with the frontiers of the current original search. The abstract searches provide heuristic estimations for the original search, increasing the f-value of states in the search frontier. Eventually, the search in the original state space will be simplified (as the number of states with minimum f-value will be smaller)[2]and SymBA* will continue expanding states in the original search space.

A distinction must be made between bidirectional search in the original state space and the ones performed in abstract state spaces. The first type of search aims to find a plan, so techniques like nipping and pruning (Kwa 1989) should remain activated to prune both search frontiers whenever they meet. On the other hand, searches on abstract state spaces are used to derive heuristic estimates for the original search. The additional pruning is deactivated in order to guarantee that the derived heuristics are admissible.

Algorithm 1: SymBA*

```
1  $S^{\Theta}_{fw} \leftarrow \langle \mathcal{I}, \Theta \rangle$
2  $S^{\Theta}_{bw} \leftarrow \langle \mathcal{G}, \Theta \rangle$
3  $SearchPool \leftarrow \{S^{\Theta}_{fw}, S^{\Theta}_{bw}\}$
4  $\pi \leftarrow None$
5  while $\max(f(S^{\Theta}_{fw}), f(S^{\Theta}_{bw})) < Cost(\pi)$ do
6      if $\exists S \in SearchPool \mid$ IsCandidate(S) then
7          $S^{\Theta_i}_{u} \leftarrow$ SelectSearch($SearchPool$)
8          $\pi' \leftarrow$ Expand-frontier($S^{\Theta_i}_{u}$)
9          if $\Theta_i = \Theta \wedge \pi' \neq \varnothing \wedge Cost(\pi') < Cost(\pi)$ then
10             $\pi \leftarrow \pi'$
11         Notify-h($S^{\Theta_i}, S^{\Theta}$)
12     else
13         $\alpha \leftarrow$Select-abstraction($S^{\Theta}_{fw}, S^{\Theta}_{bw}$)
14         $\left\langle S^{\Theta_\alpha}_{fw}, S^{\Theta_\alpha}_{bw} \right\rangle \leftarrow$Apply($\alpha, S^{\Theta}_{fw}, S^{\Theta}_{bw}$)
15         $SearchPool \leftarrow SearchPool \cup \{S^{\Theta_\alpha}_{fw}, S^{\Theta_\alpha}_{bw}\}$
16 return $\pi$
```

Algorithm 1 shows the main algorithm of SymBA*, which decides whether to advance the search in the original state space or in one of the abstract state spaces. SymBA* maintains a pool of all the current active searches. The pool is initialized with a bidirectional search on the original state space. The algorithm proceeds while the current best solution so far has not been proven optimal (line 5). At each iteration, the algorithm filters the searches that are valid candidates from the pool and selects the most promising ones.

A search is a valid candidate if and only if it is both *useful* and *searchable*. The search in the original search space is always useful. A search in an abstract search space is useful if and only if there are still states from the opposite frontier that do not correspond to a state already expanded in the abstract space. The main intuition behind this is that non-useful searches cannot possibly simplify the next step in the original search space. A search is *searchable* if the estimated time needed to perform the next step does not surpass the bounds imposed by our parameters. Among all the searches that are *valid candidates*, we select those that have a greater minimum f-value, because they are closer to proving that the current solution is optimal. If more than one search has the same minimum f-value, we select the one whose next step is estimated to take less time.

Once a search has been selected, the procedure `ExpandFrontier` expands the set of states that have a minimum g-value among those that have a minimum f-value, like in the standard implementation of BDDA*. If this was in the original state space, a new plan may be found (if the new plan has a lower cost, it is stored). If this was in an abstract state space, we update the heuristic value of the other searches in the opposite direction in the pool, both abstract and original. In order to easily re-evaluate the heuristic, we use the Symbolic List A* implementation proposed in (Edelkamp, Kissmann, and Torralba 2012).

If there are no valid search candidates (line 12), a new bidirectional search is added to the pool (which amounts to two new searches). First, we select a new abstraction strategy (line 13). Using the strategy, we relax the current frontiers of the original state space search, until the frontier size is small enough to continue the search and there is no previous equivalent search (line 14). Finally, the new search is included in the pool to be selected in subsequent iterations.

Abstraction Hierarchies

The SymBA* planner can be used with different abstraction hierarchies. We use the Symbolic M&S abstractions described in (Torralba, Linares López, and Borrajo 2013) and Symbolic PDB abstractions.

For the pattern selection, instead of choosing a particular fixed pattern for the abstraction like previous domain-independent methods (Haslum et al. 2007; Kissmann and Edelkamp 2011; Edelkamp 2006), we seek a complete hierarchy. This hierarchy consists of an ordering on the variables such that the variables at the beginning are abstracted first until a small enough abstraction is obtained. This ordering is similar to the one used by linear merge strategies of M&S, although in this case it is reversed, as the merge strategies consider important variables first whereas we abstract away important variables last.

These orderings are computed by adding goal variables to the ordering (Goal first strategies) or variables causally connected to variables already in the ordering (CG first strategies). Tie-breaking is performed either randomly (Random) or by taking Gamer's ordering (Gamer) or its reverse (Reverse Gamer). The SymBA* planner makes use of three different strategies, derived from the previously described criteria:

- CG Goal Random: Adds variables causally connected to

[2]This is not entirely true in the symbolic case, as having fewer states does not mean that the BDD that represents them is smaller, but in most cases there is a positive correlation.

variables already in the ordering. If there is none, a goal variable is added. Ties are broken randomly.

- Goal CG Gamer: Goal variables are added first. Then, variables causally connected to variables already in the ordering are added. Ties are broken by adding to the ordering variables lower in Gamer's ordering.
- Reverse Gamer: The ordering is the same as Gamer's, but reversed. This aims to obtain abstractions that are efficiently computable, as the variables are abstracted away from the top of the BDD. Goals and causal links are explicitly ignored, although they are taken into account in Gamer's ordering. Also, at some point the abstraction may not contain goal variables, although this is fine as long as we work with a perimeter.

The SymBA* Planner Configuration

The SymBA* planner is implemented on top of the Fast Downward planning system (Helmert 2006). We have presented two different configurations of SymBA* to the IPC-2014 competition: SymBA*-1 and SymBA*-2. They differ on the abstraction hierarchies used. They both use PDB abstractions, and SymBA*-2, additionally, uses M&S abstractions.

Each call to `Select-abstraction` returns a different abstraction, choosing an abstraction from each of the following hierarchies in a round robin schema:

1. (only in SymBA*-2) M&S using bisimulation shrinking, with a maximum number of 10000 abstract states
2. PDBs with CG Goal Random
3. PDBs with Goal CG Gamer
4. PDBs with Reverse Gamer

The strategies are always used in the same order. We bound the time available for selecting the abstraction to 500 seconds. Moreover, when the planner has spent 1500 seconds or 3GB, we consider that searching more abstractions to generate better heuristics will not pay off. In that case, the search continues only in the original state space, with the heuristics that have been generated so far.

All symbolic searches use the latest improvements on image computation (Torralba, Edelkamp, and Kissmann 2013) and h^2 constraints for symbolic search (Torralba and Alcázar 2013). In particular, we use a disjunctive partitioning of TRs with a maximum TR size of 100k nodes and a timeout of 60 seconds to generate the TRs. Constraints from the state invariants are encoded directly in the TRs.

Searches are considered to be searchable whenever their frontier has fewer than 10 million nodes and the next step is estimated to take at most 30 seconds. If the time bound is surpassed, the bound on the number of frontier nodes is updated to half of the current frontier. To guarantee that new abstract searches are searchable, the size of the frontier is reduced by abstracting away variables from the abstraction hierarchy until the relaxed frontier has fewer nodes than 80% of the bound on the number of nodes.

Preprocessing

There are two noteworthy considerations regarding the preprocessing phase in SymBA*:

- How the SAS$^+$ variables are selected.
- How to compute h^2 (Bonet and Geffner 2001) and prune spurious operators.

SAS$^+$ Variable Selection

Switching from Gamer's SAS$^+$ encoding to the Fast Downward version (Helmert 2009), we observed a decrease of performance in some benchmark domains. We changed the selection of which invariant groups are used as SAS$^+$ variables in order to avoid that degradation in performance.

The Fast Downward planner chooses invariant groups with the highest cardinality as SAS$^+$ variables, until all the fluents of the problem have been considered in a variable. Aiming to further reduce the number of SAS$^+$ variables selected, we prefer to select invariant groups that contain fluents that do not appear in other invariant groups. We base our criterion on the observation that, since all the fluents of the problem have to be included in a SAS$^+$ variable, invariant groups that have a fluent which does not appear in other invariant groups will always be selected anyway.

As an example, think of the following case, based on a simplified version of the IPC-2011 *floortile* domain: we have two robots on a grid such that the robots cannot be at the same cell at the same time. Two types of invariant groups are detected:

1. Each robot is at exactly one single cell:
   (at robot1 cell1),(at robot1 cell2),...
2. Each cell either is clear or has a robot at it:
   (clear cell1),(at robot1 cell1),(at robot2 cell1)

Invariants of the first type have larger cardinality, so Fast Downward would encode this problem with a variable per robot that represents the location of the robot ({*(at robot1 cell1),(at robot1 cell2),...*}) and a variable of the kind {*(clear cell1),⟨none of those⟩*} per cell. In our case, we prefer to select invariant groups of the second type first because each fluent *(clear cell1)* only takes part on a single invariant group. Thus, we would only have a variable per cell of the kind {(clear cell1),(at robot1 cell1),(at robot2 cell1)}, which amounts to fewer variables and fluents.

This leads to the use of *"exactly-one"* invariant groups as variables in most cases, avoiding the use of *"at-most-one"* invariant groups if possible – which require an additional ⟨*none of those*⟩ fluent. With this policy the number of resulting variables and fluents is usually lower. This may be counterproductive for techniques that depend on the causal graph, like the abstraction strategies that we use.

Computing h^2 Invariants and Pruning Spurious Operators

We have implemented the computation of the h^2 heuristic in Fast Downward's preprocessor. We also implemented a backward version of h^2 (Haslum 2008), which identifies pairs of propositions that cannot be reached from goal states in regression.

We use the mutexes obtained from h^2 and the *"exactly-one"* invariant groups from Fast Downward's monotonicity analysis to disambiguate the preconditions and the effects of the operators of the problem (Alcázar et al. 2013). We discard operators whose preconditions or effects are spurious sets of fluents, that is, contradict the previously inferred state invariants. We do this because the number of ground operators is significantly reduced in many planning domains with respect to the standard preprocessor of Fast Downward.

The discovery and use of the state invariants during this phase is interleaved: whenever new mutexes or spurious operators are discovered in this process, we repeat the computation of h^2 in both directions and the operator disambiguation until no more constraints are inferred. We set a limit of 300 seconds for this phase.

Conditional Effect Support

The Fast Downward planning system partially supports conditional effects. They are correctly handled by the A^* search, but the current public version of M&S does not support them. We chose to disable M&S relaxations in these cases, so that only Symbolic Perimeter PDBs were used in domains with conditional effects.

To handle conditional effects in symbolic search, we encode them in the TRs. According to the semantics of conditional effects, they are applied in order. If more than one effect is applied over the same variable, the last one overwrites all the others. Hence, to generate the TR with conditional effects we group all the effects by the variable they modify. Each conditional effect is encoded as the conjunction of its condition, its effect and the negation of the conditions of previous effects over the same variable.

For the h^2 computation, conditional effects are not compiled away, but rather we ignore conditional preconditions and all the delete effects that are conditional or that delete a conditional effect.

Technical Details

To perform BDD operations, we used version 2.5 of Fabio Somenzi's CUDD library. The planner is compiled in 32-bit (-m32), using the compiler optimization (-O3) and with support of c++-11 features (–std=c++11).

Acknowledgements

This work has been partially supported by a FPI grant from the Spanish government associated to the MICINN project TIN2008-06701-C03-03, and it has also been supported by the project TIN2011-27652-C03-02.

References

Alcázar, V.; Borrajo, D.; Fernández, S.; and Fuentetaja, R. 2013. Revisiting regression in planning. In *International Joint Conference on Artificial Intelligence*, 2254–2260.

Alcázar, V.; Fernández, S.; and Borrajo, D. 2014. Analyzing the impact of partial states on duplicate detection and collision of frontiers. In *International Conference on Automated Planning and Scheduling*.

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

Borrajo, D.; Kambhampati, S.; Oddi, A.; and Fratini, S., eds. 2013. *Proceedings of the Twenty-Third International Conference on Automated Planning and Scheduling, ICAPS 2013, Rome, Italy, June 10-14, 2013*. AAAI Conference on Artificial Intelligence (AAAI).

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

de Champeaux, D. 1983. Bidirectional heuristic search again. *J. ACM* 30(1):22–32.

Edelkamp, S.; Kissmann, P.; and Torralba, Á. 2012. Symbolic A^* search with pattern databases and the merge-and-shrink abstraction. In Raedt, L. D.; Bessière, C.; Dubois, D.; Doherty, P.; Frasconi, P.; Heintz, F.; and Lucas, P. J. F., eds., *European Conference on Artificial Intelligence (ECAI)*, volume 242 of *Frontiers in Artificial Intelligence and Applications*, 306–311. IOS Press.

Edelkamp, S. 2006. Automated creation of pattern database search heuristics. In *MoChArt*, 35–50.

Eyerich, P., and Helmert, M. 2013. Stronger abstraction heuristics through perimeter search. In Borrajo et al. (2013).

Felner, A., and Ofek, N. 2007. Combining perimeter search and pattern database abstractions. In *Symposium on Abstraction, Reformulation and Approximation (SARA)*, 155–168.

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. AAAI Press.

Haslum, P. 2008. Additive and reversed relaxed reachability heuristics revisited. *Proceedings of the 6th International Planning Competition*.

Helmert, M.; Haslum, P.; Hoffmann, J.; and Nissim, R. 2014. Merge-and-shrink abstraction: A method for generating lower bounds in factored state spaces. *Journal of the ACM*.

Helmert, M. 2006. The Fast Downward planning system. *Journal of Artificial Intelligence Research (JAIR)* 26:191–246.

Helmert, M. 2009. Concise finite-domain representations for pddl planning tasks. *Artificial Intelligence* 173(5-6):503–535.

Holte, R. C.; Grajkowski, J.; and Tanner, B. 2005. Hierarchical heuristic search revisited. In *Symposium on Abstraction, Reformulation and Approximation (SARA)*, 121–133.

Kaindl, H., and Kainz, G. 1997. Bidirectional heuristic search reconsidered. *Journal of Artificial Intelligence Research (JAIR)* 7:283–317.

Kissmann, P., and Edelkamp, S. 2011. Improving cost-optimal domain-independent symbolic planning. In Burgard, W., and Roth, D., eds., *AAAI Conference on Artificial Intelligence (AAAI)*. AAAI Press.

Kwa, J. B. 1989. BS*: An admissible bidirectional staged heuristic search algorithm. *Artif. Intell.* 38(1):95–109.

Larsen, B. J.; Burns, E.; Ruml, W.; and Holte, R. 2010. Searching without a heuristic: Efficient use of abstraction. In *AAAI Conference on Artificial Intelligence (AAAI)*.

Leighton, M. J.; Ruml, W.; and Holte, R. C. 2011. Faster optimal and suboptimal hierarchical search. In *SOCS*.

McMillan, K. L. 1993. *Symbolic model checking*. Kluwer Academic publishers.

Pohl, I. S. 1969. *Bi-directional and Heuristic Search in Path Problems*. Ph.D. Dissertation, Stanford, CA, USA. AAI7001588.

Torralba, Á., and Alcázar, V. 2013. Constrained symbolic search: On mutexes, bdd minimization and more. In Helmert, M., and Röger, G., eds., *Symposium on Combinatorial Search (SoCS)*. AAAI Press.

Torralba, Á.; Edelkamp, S.; and Kissmann, P. 2013. Transition trees for cost-optimal symbolic planning. In Borrajo et al. (2013).

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