cGamer: Constrained Gamer

Álvaro Torralba and **Vidal Alcázar**
{alvaro.torralba, vidal.alcazar}@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

Gamer is a symbolic planner that performs (in this case) bidirectional symbolic search. It already participated in 2008 and 2011, so in this paper we will focus on the improvements since then. The main improvements are: fixing some bugs and implementing basic improvements; a disjunctive partitioning of the transition relations; and the exploitation of state invariants both during the preprocessing phase and during search.

Motivation

Symbolic search can obtain exponential savings in both time and space compared to regular explicit-state search (McMillan 1993). This is achieved by using binary decision diagrams (BDDs) (Bryant 1986) to represent both sets of states and transition relations (TRs). Furthermore, the use of BDDs allows a seamless implementation of detection of subsumed states and collision of frontiers, which opens up the possibility of using regression and bidirectional search algorithms efficiently (Alcázar, Fernández, and Borrajo 2014).

Gamer (Kissmann and Edelkamp 2011), the vanilla version of cGamer, already participated in the International Planning Competitions (IPC) of 2008 and 2011. While it won the competition in 2008, in 2011 it solved 148 problems, which was kind of underwhelming given that the winner solved 185 tasks. As described in Section 6.5.5 of Peter Kissmann's PhD Thesis (Kissmann 2012), after fixing some bugs and implementing some basic improvements (such as using a more efficient parser for grounded PDDL) Gamer is able to solve 13 additional problems (own results). Also, on a *per domain* analysis we can see that Gamer, although still worse overall, outperforms the rest of the participants in quite a few domains. All this means that symbolic search cannot be ruled out and that Gamer, under the right circumstances, is still a strong contestant.

Driven by recent research, an important increase in performance in Gamer has been obtained. Two orthogonal works, one dealing with the handling of the TRs (Torralba, Edelkamp, and Kissmann 2013) and another dealing with the encoding of state invariants in symbolic search (Torralba and Alcázar 2013), reported significantly better coverage than the one obtained by Gamer. cGamer, the planner described here, implements some techniques proposed in these previous works.

Symbolic Search

Symbolic search was originally proposed in the area of model checking (McMillan 1993). The basic idea is to perform set-based search as opposed to the traditional search expanding one state at a time. Sets of states are represented with efficient data structures, like Binary Decision Diagrams (BDDs) (Bryant 1986). To perform the search, planning actions are represented with one or more Transition Relations (TRs). The successor generation in symbolic search is performed with the *image* and *pre-image* operations of a set of states with respect a transition relation. Given a set of states and a TR, the *image* operation computes the set of successor states that can be reached from any state in the set by applying any operator represented by the TR. Similarly, the *pre-image* operation computes the set of predecessor states in regression.

Our predecessor planner, Gamer, is a symbolic search planner that implements two algorithms: symbolic breadth-first search and symbolic A* with symbolic pattern database heuristics (Culberson and Schaeffer 1998; Edelkamp 2002). From what we observed both in the results of IPC-11 and our experimentation, using pattern databases (PDBs) is often worse than just using bidirectional blind search. This is the case for most domains without non-unit action costs, and even in unit-cost domains selecting patterns and precomputing the PDBs is often not better than just searching in both directions with no heuristics.

The main advantage of the bidirectional search is that it can interleave the search in forward and backward directions. Gamer estimates which direction is easier to expand by taking into account the time spent in the last steps. Thus, the new version of Gamer features a symbolic bidirectional Dijkstra search, that is able to cope with non-unit cost domains (Edelkamp, Kissmann, and Torralba 2012).

Preprocessing

Apart from using Gamer's parser, we employ Fast Downward's translator and preprocessor (Helmert 2006; 2009) to generate the SAS$^+$ encoding of the task.[1] There are

[1] Using both Gamer and Fast Downward in the preprocessing phase is redundant. We did so for convenience, as not all the necessary techniques were implemented in both preprocessors.

three noteworthy considerations regarding the preprocessing phase in cGamer:

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

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),*$\langle$*none of those*$\rangle\}$ 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 $\langle$*none of those*$\rangle$ fluent. With this policy the number of resulting variables and fluents is usually lower. This may be counterproductive if techniques that depend on the causal graph are used, but this only affects us when choosing the ordering of the variables.

Computing h^2 Invariants and Pruning Spurious Operators

We have implemented the computation of the h^2 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 Effects

Conditional effects are compiled away by using *adl2strips* (Hoffmann et al. 2006), a tool initially developed by Jörg Hoffmann and modified later by Sergio Núñez that converts more expressive planning instances into STRIPS. *adl2strips* implements two different methods for compiling away conditional effects (Gazen and Knoblock 1997; Nebel 2011). Both compilations have their pros and cons. Gazen and Knoblock's compilation may generate an exponential number of STRIPS actions on the size of the input task. On the other hand, Nebel's compilation guarantees that the number of STRIPS actions is polynomial on the size of the input task, but increases the plan length. Therefore, we use Gazen and Knoblock's compilation for tasks with at most three conditional effects in the same PDDL action and Nebel's compilation otherwise.

Disjunctive Transition Relations

Torralba, Edelkamp, and Kissmann (2013) identified the *image* and *pre-image* operations and the subsequent union of successor sets as a bottleneck. In other words, the encoding of planning operators in the Transition Relations (TRs) may have a large impact on the overall performance. In Gamer, a TR was used to represent each operator. Several alternatives were proposed; among them, computing a disjunction of the TRs of operators with the same cost stood out for its simplicity and results.

Since the disjunction of all the TRs of the problem is sometimes not tractable to compute, we set a limit MAX_TR_SIZE on the maximum number of nodes that any TR may have. If this limit is reached during the computation of the disjunction of TRs, several disjunctive TRs smaller than MAX_TR_SIZE will be used instead of a single TR. When multiple disjunctive TRs must be used, the choice of which TRs must be merged is critical, as the number and size of the resulting disjunctive TRs depends on it. This is done using a balanced merging approach based on the disjunction tree used in the original Gamer to compute the disjunction of successor sets. For the competition, cGamer uses a limit of MAX_TR_SIZE=100k nodes.

Encoding State Invariants in Symbolic Search

Regression is common in symbolic search, both as a part of the main search algorithm and as a way to derive admissible heuristics. Although constraints derived from state invariants are commonly used in explicit-state regression, in symbolic regression this had not been done. Hence, Torralba and Alcázar (2013) proposed several ways of encoding these constraints in symbolic search. According with the experimental results, the most efficient method to apply constraints in symbolic search is to encode them in the TRs.

In cGamer we encode binary mutexes derived from the computation of h^2 and invariant groups derived from Fast Downward's monotonicity analysis. Before computing the disjunction of the TRs described in the previous section, the TR corresponding to each operator is enriched with all the constraints that may be violated after applying the operator. Thus, no state violating the constraints is generated in the search.

An important difference with respect to the version presented in (Torralba and Alcázar 2013) is that we also derive mutexes from the backward computation of h^2. These backward mutexes are used to prune the forward search in a similar manner to how forward h^2 mutexes are used in regression.

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. We'd like to thank both Jörg Hoffmann and Sergio Núñez for *adl2strips*.

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.

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

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

Edelkamp, S.; Kissmann, P.; and Torralba, Á. 2012. Symbolic A^* search with pattern databases and the merge-and-shrink abstraction. In *European Conference on Artificial Intelligence (ECAI)*, 306–311.

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

Gazen, B. C., and Knoblock, C. A. 1997. Combining the expressivity of ucpop with the efficiency of graphplan. In *ECP*, 221–233.

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

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

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

Hoffmann, J.; Edelkamp, S.; Thíebaux, S.; Englert, R.; Liporace, F.; and Trüg, S. 2006. Engineering benchmarks for planning: the domains used in the deterministic part of IPC-4. *Journal of Artificial Intelligence Research (JAIR)* 26:453–541.

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

Kissmann, P. 2012. *Symbolic Search in Planning and General Game Playing*. Ph.D. Dissertation, Universität Bremen, Germany.

McMillan, K. L. 1993. *Symbolic Model Checking*.

Nebel, B. 2011. On the compilability and expressive power of propositional planning formalisms. *CoRR* abs/1106.0247.

Torralba, Á., and Alcázar, V. 2013. Constrained symbolic search: On mutexes, BDD minimization and more. In *Symposium on Combinatorial Search (SoCS)*, 175–183.

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