GAMER: Bridging Planning and General Game Playing with Symbolic Search*

Stefan Edelkamp and Peter Kissmann
Faculty of Computer Sciences
TU Dortmund, Germany
{stefan.edelkamp,peter.kissmann}@cs.uni-dortmund.de

Abstract

This paper presents the planner GAMER for IPC-6 that solves domain-independent action planning problems cost-optimally with BDDs. Based on experience in general game playing, we support multi-actions that make it easier to tackle conditional effects or temporally extended goals. While STRIPS problems can be linearly converted to multi-action PDDL, the reverse compilation may require a description that is exponentially larger. The algorithmic contributions are bidirectional weighted graph search based on perimeters and planning for optimal net-benefit.

Introduction

Optimal sequential planning for minimizing the sum of action costs is a natural search concept for many applications. Due to the unfolding of the planning graph, in large search depth parallel optimal planners (Blum and Furst 1995; Kautz and Selman 1996) appear to be less effective.

The state-of-the-art in cost-optimal sequential planning includes heuristic search planning with flexible abstraction heuristics (Helmert, Haslum, and Hoffmann 2007) and with explicit-state pattern database search (Haslum et al. 2007). These planners refer to a fully instantiated planning problem together with a minimized state encoding of atoms into multi-variate variables (Helmert 2004; Edelkamp and Helmert 1999) and compete well with other step-optimal heuristic search planners (Zhou and Hansen 2004; Grandcolas and Pain-Barre 2007; Hickmott et al. 2006).

Binary Decision Diagrams (BDDs) (Bryant 1985) are used for optimal sequential planning to reduce the memory requirements of the state sets as problem sizes increase (Cimatti, Roveri, and Traverso 1998; Jensen et al. 2006). State variables may be encoded binary and ordered along their causal graph dependencies. Sets of planning states are represented in form of Boolean functions, and actions are formalized as transition relations. This allows to compute the successor state set, which determines all states reached by applying one action to the states in the input set. Iterating the process (starting with the representation of the initial state) yields symbolic breadth-first search (BFS). The fact that BFS performs well wrt. heuristic search is an observation also encountered in explicit-state search (Helmert and Röger 2007). It is related to the exponential increase in the number of states below the optimum cost for many benchmark domains, even if only small constant errors in the heuristic are assumed.

*Thanks to DFG for support in ED 74/3 and 74/2.

Planning with BDDs

We consider optimal STRIPS with costs. For a given planning problem $\mathcal{P} = (\mathcal{S}, \mathcal{A}, \mathcal{I}, \mathcal{G}, \mathcal{C})$, the task is to find a sequence of actions $a_1, \ldots, a_k \in \mathcal{A}$ from $\mathcal{I}$ to $\mathcal{G}$ with minimal $\sum_{i=1}^{k} \mathcal{C}(a_i)$. We assume that the highest action cost is bounded by some constant C. Additionally, we consider a minimized planning state enconding that leads to SAS$^+$ planning (Helmert 2004).

While symbolic planning sometimes refers to analyzing planning graphs (Blum and Furst 1995) or to checking the satisfiability of formulas (Kautz and Selman 1996), we refer to the exploration in the context of using BDDs (Bryant 1985). The advantage of BDD-based compared to SAT-based planning is that the number of variables does not increase with the search depth.

Actions are formalized as relations, representing sets of tuples of predecessor and successor states. This allows to compute the image as a conjunction of the state set (formula) and the transition relation *Trans* (formula), existentially quantified over the set of predecessor state variables. This way, all states reached by applying one action to one state in the input set are determined. Iterating the process (starting with the representation of the initial state) yields a symbolic implementation of breadth-first search. Fortunately, by keeping sub-relations Trans_a attached to each action $a \in \mathcal{A}$ it is not required to build a monolithic transition relation. The image of a state set S then reads as $\bigvee_{a \in \mathcal{A}} (\exists x.\, (\mathit{Trans}_a(x, x') \wedge S(x)))$.

Bidirectional search algorithms are distributed in the sense that two search frontiers are searched concurrently. Since symbolic predecessors are the inverse of symbolic successors, symbolic bidirectional BFS algorithms start from the partial goal assignment. It comes at a low price as it does not rely on any heuristic evaluation function.

For positive action costs, the first plan reported by the single-source shortest-paths search algorithm of Dijkstra's (1959) is optimal. For implicit graphs, we need two data

Algorithm 1 Symbolic-Shortest-Paths.

Input: $\mathcal{P} = (\mathcal{S}, \mathcal{A}, \mathcal{I}, \mathcal{G}, \mathcal{C})$ in symbolic form with $\mathcal{I}(x)$, $\mathcal{G}(x)$, and $\mathit{Trans}_a(x, x')$
Output: Optimal solution path

$\mathit{Open}[0](x) \leftarrow \mathcal{I}(x)$
for all $f = 0, 1, 2, \ldots$
  for all $l = 0, \ldots, f-1$
    $\mathit{Open}[f] \leftarrow \mathit{Open}[f] \setminus \mathit{Open}[f-l]$
  if $(\mathit{Open}[f-C] \vee \ldots \vee \mathit{Open}[f] = \bot)$
    return "Exploration completed, no plan found"
  $\mathit{Min}(x) \leftarrow \mathit{Open}[f](x)$
  if $(\mathit{Min}(x) \wedge \mathcal{G}(x) \neq \bot)$
    return *ConstructSolutionPath*$(\mathit{Min}(x) \wedge \mathcal{G}(x))$
  for all $i = 1, \ldots, C$
    $\mathit{Succ}_i(x') \leftarrow \bigvee_{a \in \mathcal{A}, \mathcal{C}(a)=i} (\exists x. \mathit{Min}(x) \wedge \mathit{Trans}_a(x, x'))$
    $\mathit{Succ}_i(x) \leftarrow \exists x' (\mathit{Succ}_i(x') \wedge x = x')$
    $\mathit{Open}[f+i](x) \leftarrow \mathit{Open}[f+i](x) \vee \mathit{Succ}_i(x)$

structures, one to access nodes in the search frontier and one to detect duplicates.

The symbolic-shortest-paths search procedure is implemented in Algorithm 1. The BDD *Open*[0] is initialized to the representation of the initial state. Unless the goal is reached, in one iteration we first choose the next f-value together with the BDD *Min* of all states in the priority queue having this value. Then for each $a \in \mathcal{A}$ with $\mathcal{C}(a) = i$ the transition relation $\mathit{Trans}_a(x, x')$ is applied to determine the BDD for the subset of all successor states that can be reached with cost i. In order to attach new f-values to this set, we insert the result into bucket $f + i$.

Action cost values zero are possible but require a little more effort to implement. We have to perform an additional BFS to compute the closure for each bucket: once a zero-cost image is encountered for a bucket to be expanded, a fixpoint is computed. This results in the representation of all states that are reachable by applying one action with non-zero cost followed by a sequence of zero-cost actions.

Partial Symbolic Pattern Databases

Perimeter search (Dillenburg and Nelson 1994) tries to reap the benefits of front-to-front evaluations in bidirectional search while avoiding the computational efforts involved in re-targeting the heuristics towards a continuously changing search frontier. It conducts a cost-bounded best-first search starting from the goal nodes; the nodes on the final search frontier, called the perimeter, are stored in a dictionary. Then a forward search, starting from $\mathcal{I}$, employs front-to-front evaluation with respect to these nodes. Alternatively, in front-to-goal perimeter search all nodes outside the perimeter are assigned to the maximum of the minimum of the goal distances of all expanded nodes in the perimeter and an additionally available heuristic estimate. Although larger perimeters provide better heuristics, they take increasingly longer to compute. The memory requirements for storing the perimeter are considerable and, more crucially, the effort for multiple heuristic computations can become large.

In essence, a partial pattern database (Anderson, Holte, and Schaeffer 2007) is a perimeter in the abstract space (with the interior stored). Any node in the original space has a heuristic estimate to the goal: if it is in the partial pattern database, the recorded goal distance value is returned; if not, the radius d of the perimeter is returned. This heuristic is both admissible and consistent. Building a partial pattern database starts from the abstract goal and stores heuristic values. When a memory- or time-limit is reached it terminates and the heuristic values are used for the forward search. The value d is the minimum cost of all abstract nodes outside the partial pattern database.

On one extreme, a partial pattern database with no abstraction reverts to exactly a perimeter. On the other extreme, a partial pattern database with a coarse-grained abstraction will cover the entire space, and performs exactly like an ordinary pattern database. Bidirectional BFS is interleaved partial pattern database search without abstraction.

An ordinary pattern database represents the entire space; every state visited during the forward search has a corresponding heuristic value in the database. The pattern database abstraction level is determined by the amount of available memory. Fine-grained abstraction levels are not possible, because the memory requirements increase exponentially with finer abstractions. Explicit-state partial pattern databases generally do not cover the entire space.

A partial symbolic pattern database is the outcome of a partial symbolic backward shortest paths exploration in abstract space. It consists of a set of abstract states $\mathcal{S}'_{<d}$ and their goal distance, where $\mathcal{S}'_{<d}$ contains all states in $\mathcal{S}'$ with cost-to-goal less than d, and where d is a lower bound on the cost of any abstract state not contained in $\mathcal{S}'_{<d}$. The state sets are kept as BDDs $H[i]$ representing $\mathcal{S}'_{<i+1} \setminus \mathcal{S}'_{<i}$, $i \in \{0, \ldots, d-1\}$. The BDD $H[d]$ represents the state $\mathcal{S}' \setminus \mathcal{S}'_{<d}$ such that a partial pattern database partitions the abstract search space. The construction works similar to Algorithm 1, but backwards. For forward search, symbolic heuristic A* search (Edelkamp and Reffel 1998) is used.

General Game Playing

General Game Playing (GGP) is concerned with playing or solving general games, i. e. games that are not known beforehand. Similar to AI planning, for writing an algorithm to play or solve general games, no specialized knowledge about them can be used. We work with games given in the Game Description Language (GDL) (Love, Hinrichs, and Genesereth 2006), which was invented by the logic group of Stanford University. Since 2005, an annual GGP Competition (Genesereth, Love, and Pell 2005) takes place, which was last won 2007 by Yngvi Björnsson's and Hilmar Finnsson's CADIAPLAYER. GDL is designed for the description of general complete information games satisfying the restrictions to be finite, discrete, and deterministic.

When a game ends, all players receive a certain reward. This is an integer value within $\{0, \ldots, 100\}$ with 0 being the worst and 100 the optimal reward. Thus, each player will try to get a reward as high as possible (and maybe at the same time keep the opponent's reward as low as possible). The description is based on the Knowledge

Interchange Format (KIF) (Genesereth and Fikes 1992), which is a logic based language. The most successful players (Clune 2007; Kuhlmann, Dresner, and Stone 2006; Schiffel and Thielscher 2007) are mainly interested in constructing competitive players, whereas we aim at classifying (solving) them (Edelkamp and Kissmann 2007; 2008). That is, we want to get the rewards for all players in case of optimal play (optimal rewards) for each of the reachable states. Thus, when the classification is done, we can exploit the information to obtain a perfect player.

Multi-Actions

Our implementation for the classification of general games originally worked with a variation of PDDL that we called GDDL (Edelkamp and Kissmann 2007). This we extended further to use what we call *multi-actions* for the moves. These multi-actions consist of a global precondition and several precondition/effect pairs where the preconditions can be arbitrary formulas while the effects are single (Boolean) variables. Such a multi-action pre_1, eff_1, ..., pre_n, eff_n might also be written as $\mathit{global} \wedge (\mathit{pre}_1 \leftrightarrow \mathit{eff}_1) \wedge \ldots \wedge (\mathit{pre}_n \leftrightarrow \mathit{eff}_n)$. Intuitively, we interpret it as follows. If *global* holds, we can perform this action. It will create the current state's successor by applying the precondition/effect pairs The effect has to hold in the successor state iff the corresponding precondition holds in the current state. All variables not in any of the effects will be set to *false*. Thus, we need to set the frame explicitly.

Our planner allows both STRIPS actions and multi-actions. The latter concept is clearly more expressive. From a STRIPS representation of an action $a = (P, A, D)$ with precondition list P, add list A and delete list D, the corresponding multi-action can be derived by setting *global* to P, and for each $p \in A$ we include $\mathit{true} \Leftrightarrow p$. Adding the frame, i.e. the variables that do not change to a STRIPS action is simple and can be automated. We can omit the variables in D as all variables not appearing in any effect will be set to *false* and thus are deleted from the state. The backward translation, however, is involved.

The advantages of multi-actions are two-fold. First, conditional effects are easy to integrate, and secondly, temporally extended goals/state trajectory constraints have a better fit. Conditional effects are included in the ADL fragment of PDDL. Their representation in PDDL requires a construct `when`, which can be expressed naturally, when operating with multi-actions. The point we stress is that the compilation into a BDD is simpler using the multi-action representation than when considering conditional effects as integrated actions that have to be linked. The exponential gap for the conditional effects is well-known (Nebel 2000).

It has been shown earlier (Edelkamp 2006b) that state trajectory constraints can be compiled to PDDL 2 using an automata-based approach. For each constraint, an automaton is built that runs concurrent to the search and monitors the violatedness of it. The automata are then translated back into STRIPS actions. For this case, the transition relation of each constraint is synchronously composed with the state space. Synchronicity has been simulated by extra propositions connecting the state spaces.

Net-Benefit

Net-benefit planning at IPC-6 concerns trading utility received by achieving soft goals for total cost of the actions used to achieve them. Planners competing in the optimal track are expected to find best plans in terms of a linear objective function. We consider optimal planning with action costs, goal utilities and no metric quantities. For preference constraints of type $(\mathit{preference}\ p\ \phi_p)$ we associate a Boolean variable $\mathit{violated}_p$ (denoting the violation of p). Moreover, maximization problems can be easily transformed into minimization problems, and adding a constant offsets does not change the solution set. If we have an indicator relation $\Phi_p(v, x) = (\mathit{violated}_p \Leftrightarrow \neg\phi_p)$ the conjunct $\bigwedge_p \Phi_p(v, x)$ evaluates all preferences together. As most BDD packages already support variables of finite domain, we abstract from the binary representation of the number for v.

In planning with preferences (and no action costs) (Edelkamp 2006a), we no longer have increasing total costs to be minimized. Hence, we cannot neglect states with a cost evaluation larger than the current one. Essentially, we are forced to look at all states. The branch-and-bound algorithm incrementally improves an upper bound U on the solution length. When it comes to analyzing a layer in which more than one goal is contained a goal g with the minimum value $m(g)$ is selected for solution reconstruction. Based on the range of the variables in the domain metric we first compute the minimum and maximum value ($\min_m$ and $\max_m$) that m can take. For indicator variables $\mathit{violated}_i \in \{0, 1\}$, $i \in \{1, \ldots, k\}$, we have $\mathit{min}_{\mathit{violated}_i} = 0$ and $\mathit{max}_{\mathit{violated}_i} = 1$. If m is a linear function $\alpha_1\mathit{violated}_1 + \ldots + \alpha_k\mathit{violated}_k$ with $\alpha_i \geq 0$, $i \in \{1, \ldots, k\}$ we obtain

$$\mathit{min}_m = \sum_{i=1}^{k} \alpha_i \cdot \mathit{min}_{\mathit{violated}_i} = 0$$

$$\mathit{max}_m = \sum_{i=1}^{k} \alpha_i \cdot \mathit{max}_{\mathit{violated}_i} = \sum_{i=1}^{k} \alpha_i$$

In planning with preferences and total action cost, for the sake of brevity, we assume no scaling of the total cost value. Let $\mathit{benefit}(\pi) = \sum_i \alpha_i \mathit{violated}_i(s_n)$. Then the metric we consider is $m(\pi) = \mathit{benefit}(\pi) + \mathit{total\text{-}cost}(\pi)$. Integrating *total-cost* into the cost metric using BDD arithmetic is difficult as the value *total-cost* is not bounded from above.

Fortunately, with the bucket to be expanded we already have the current f-value for evaluating *total-cost* at hand. This allows to use a different upper bound in the branch-and-bound algorithm. The pseudo-code is presented in Algorithm 2. With V we denote the current best solution obtained according to evaluating $m(\pi)$, which improves over time. With V' we denote the old value of V. As the f-value increases monotonically, we can also adapt V to improve over time. The pseudo-code also adapts a small refinement, by observing that the upper bound U for $\mathit{benefit}(\pi)$ is bounded V, which is effective if the impact *total-cost* is small.

As with the other branch-and-bound algorithm the net-benefit procedure look at all goal states that are not dominated. For action costs $\mathcal{C}(a) \in \{1, \ldots, C\}$, $a \in \mathcal{A}$, the symbolic net-benefit algorithm finds an optimal solution.

Algorithm 2 Net-Benefit Planning Algorithm.

Input: Problem $\mathcal{P}$ with action cost $\mathcal{C}(a)$, $a \in \mathcal{A}$
Cost function $\mathcal{M}$ to be minimized, preferences Φ_p
Output: Cost-optimal plan from $\mathcal{I}$ to state satisfying $\mathcal{G}$

$U \leftarrow \max_m$; $f = \min_m + 1$
$V \leftarrow V' \leftarrow \infty$
Bound$(v) \leftarrow \bigvee_{i=\min_m}^{U-1}(v = i)$
Open$[f] \leftarrow \mathcal{I}$
loop
  Open$[f](x) \leftarrow$ *Open*$[f](x) \wedge \neg \bigvee_{j=0}^{f-1}$ *Open*$[f](x)$
  if $(\bigvee_{i=f-C}^{f}$ *Open*$[i] = \bot)$ **or** $(U = \min_m)$
    return *RetrieveStoredPlan*()
  Intersection$(x) \leftarrow$ *Open*$(x) \wedge \mathcal{G}(x)$
  Eval$(v, x) \leftarrow$ *Intersection*$(x) \wedge \bigwedge_p \Phi_p(v, x)$
  Metric$(v, x) \leftarrow Eval(v, x) \wedge$ *Bound*(v)
  if $($*Metric*$(v, x) \neq \bot)$
    $U' \leftarrow \min_m$
    while $($*Eval*$(v, x) \wedge (v = U') = \bot)$ **and** $U' + f < V$
      $U' \leftarrow U' + 1$
    if $(U' + f < V)$
      $V \leftarrow U' + f$
      if $(V - f < U)$ $U \leftarrow V - f$
      if $(V < V')$
        $V' \leftarrow V$
        ConstructAndStorePlan$($*Eval*$(v, x) \wedge (v = U'))$
        Bound$(v) \leftarrow \bigvee_{i=\min_m}^{U-1}(v = i)$
  for all $i = 1, \ldots, C$
    Succ$_i(x') \leftarrow \bigvee_{a \in \mathcal{A}, \mathcal{C}(a)=i} \exists x.$ *Open*$[f](x) \wedge$ *Trans*$_a(x, x')$
    Succ$_i(x) \leftarrow \exists x'.$ *Succ*$_i(x') \wedge (x = x')$
    Open$[f + i](x) \leftarrow$ *Open*$[f + i](x) \vee$ *Succ*$_i(x)$

References

Anderson, K.; Holte, R.; and Schaeffer, J. 2007. Partial pattern databases. In *SARA*, 20–34.

Blum, A., and Furst, M. L. 1995. Fast planning through planning graph analysis. In *IJCAI*, 1636–1642.

Bryant, R. E. 1985. Symbolic manipulation of boolean functions using a graphical representation. In *DAC*, 688–694.

Cimatti, A.; Roveri, M.; and Traverso, P. 1998. Automatic OBDD-based generation of universal plans in non-deterministic domains. In *AAAI*, 875–881.

Clune, J. 2007. Heuristic evaluation functions for general game playing. In *AAAI*, 1134–1139.

Dijkstra, E. W. 1959. A note on two problems in connection with graphs. *Numerische Mathematik* 1:269–271.

Dillenburg, J. F., and Nelson, P. C. 1994. Perimeter search. *Artificial Intelligence* 65(1):165–178.

Edelkamp, S., and Helmert, M. 1999. Exhibiting knowledge in planning problems to minimize state encoding length. In *ECP*, 135–147.

Edelkamp, S., and Kissmann, P. 2007. Symbolic exploration for general game playing in PDDL. In *ICAPS-Workshop on Planning in Games*.

Edelkamp, S., and Kissmann, P. 2008. Symbolic classification of general multi-player games. In *ECAI*.

Edelkamp, S., and Reffel, F. 1998. OBDDs in heuristic search. In *KI*, 81–92.

Edelkamp, S. 2006a. Cost-optimal symbolic planning with state trajectory and preference constraints. In *ECAI*, 841–842.

Edelkamp, S. 2006b. On the compilation of plan constraints and preferences. In *ICAPS*, 374–377.

Genesereth, M. R., and Fikes, R. E. 1992. Knowledge interchange format, version 3.0 reference manual. Technical Report Logic-92-1, Stanford University.

Genesereth, M. R.; Love, N.; and Pell, B. 2005. General game playing: Overview of the AAAI competition. *AI Magazine* 26(2):62–72.

Grandcolas, S., and Pain-Barre, C. 2007. Filtering, decomposition and search-space reduction in optimal sequential planning. In *AAAI*.

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*, 1007–1012.

Helmert, M., and Röger, G. 2007. How good is almost perfect? In *ICAPS-Workshop on Heuristics for Domain-Independent Planning*.

Helmert, M.; Haslum, P.; and Hoffmann, J. 2007. Flexible abstraction heuristics for optimal sequential planning. In *ICAPS*, 176–183.

Helmert, M. 2004. A planning heuristic based on causal graph analysis. In *ICAPS*, 161–170.

Hickmott, S.; Rintanen, J.; Thiebaux, S.; and White, L. 2006. Planning via petri net unfolding. In *ECAI-Workshop on Model Checking and Artificial Intelligence*.

Jensen, R.; Hansen, E.; Richards, S.; and Zhou, R. 2006. Memory-efficient symbolic heuristic search. In *ICAPS*, 304–313.

Kautz, H., and Selman, B. 1996. Pushing the envelope: Planning propositional logic, and stochastic search. In *AAAI*, 1194–1201.

Kuhlmann, G.; Dresner, K.; and Stone, P. 2006. Automatic heuristic construction in a complete general game player. In *AAAI*, 1457–1462.

Love, N. C.; Hinrichs, T. L.; and Genesereth, M. R. 2006. General game playing: Game description language specification. Technical Report LG-2006-01, Stanford Logic Group.

Nebel, B. 2000. On the compilability and expressive power of propositional planning formalisms. *Journal of Artificial Intelligence Research* 12:271–315.

Schiffel, S., and Thielscher, M. 2007. Fluxplayer: A successful general game player. In *AAAI*, 1191–1196.

Zhou, R., and Hansen, E. 2004. Breadth-first heuristic search. In *ICAPS*, 92–100.