Symbolic Search in Planning and General Game Playing

Peter Kissmann
TZI Universität Bremen, Germany
kissmann@tzi.de

Abstract

This short paper gives a brief overview over my PhD thesis and the work that I have already done. It is concerned with optimally solving planning problems (sequential as well as net-benefit) and optimally solving general games. For all these, symbolic search using BDDs is performed.

Introduction

The PhD thesis will consist of two main parts. The first one is concerned with planning (Fikes and Nilsson 1971), while the second one is concerned with general game playing (Genesereth, Love, and Pell 2005).

In planning we are interested in finding optimal plans. That is, for classical sequential planning we want to find a sequence of actions that transforms the initial state to one of the goal states with as few actions as possible (in case of no action costs) or the total cost of all these actions being minimal (if action costs are present). For net-benefit planning we want to find a sequence of actions that transforms the initial state to one of the goal states, similar to classical planning, with the addition that there are soft goals, which provide rewards for achieving them. Thus, we want to maximize the plan's net-benefit, which is the total reward for achieving soft goals minus the total cost of the actions within the plan.

In general game playing we are mainly concerned with finding optimal strategies, i. e., strongly solving the games, as well, and not so much with playing the games. A strong solution provides us with the optimal outcome (in case of perfect play of all players) for any possibly reachable state. So far, we were able to implement solutions for single- as well as two-player turn-taking games.

All our solving algorithms (for planning as well as for general game playing) perform symbolic search using binary decision diagams (BDDs) (Bryant 1986). These are utilized for saving memory, as they represent sets of states and are automatically minimized, although the algorithms must be designed to work well with this set-based approach.

The structure of this short paper (as well as the PhD thesis) will be as follows. First, we give a brief introduction to symbolic search. Next, we give some ideas on how we use symbolic search to find optimal plans. Finally, we introduce to our approach for solving general single-player and two-player turn-taking games.

Symbolic Search

When we speak of symbolic search we mean state space search using binary decision diagrams (BDDs) (Bryant 1986). With these, we can perform a set-based search, i. e., we do not expand single states but sets of states.

BDDs typically have a fixed variable ordering and are reduced using two rules so that only a minimal number of internal BDD nodes is needed to represent a given formula / set of states.[1] The resulting representation is unique and no duplicates are present in any BDD.

BDDs enable us to completely search some state spaces we could not exhaustively search in the explicit case. E. g., in the game Connect Four $4{,}531{,}985{,}219{,}092$ states are reachable. We use 85 bits to encode each state (two bits for each cell and an additional one to denote the active player), so that in case of explicit search we would need about 43.5 TB to store all of them, while with BDDs 16 GB are sufficient. If we store only the current BFS layer and flush the previous one to a hard disk, the largest one even fits into 12 GB.

Unfortunately, most planning problems as well as general games contain variables, so that we do not know the exact size of a state, but this information is mandatory for BDDs. If we perform some instantiation (e. g., (Helmert 2008; Kissmann and Edelkamp 2009a)), we come up with a variable-free format.

To decrease the number of BDD variables, we try to find groups of mutually exclusive predicates. For planning we use the algorithm described by Edelkamp and Helmert (1999), while for general game playing we perform a simulation-based approach similar to Kuhlmann, Dresner, and Stone (2006) and Schiffel and Thielscher (2007) who identify the input and output parameters of each predicate. Often, input parameters denote the positions on a game board while the output parameters specify its content. Predicates sharing the same name and the same input but differ-

[1] These are reduced ordered binary decision diagrams (ROBDDs)). Whenever we speak of BDDs it is really ROBDDs we have in mind.

ent output parameters can never be true at the same time and thus are mutually exclusive. If we find a group of n mutually exclusive predicates, we need only $\lceil \log n \rceil$ BDD variables to encode these.

After instantiation, we know the precise number of moves of all the players. In case of general game playing, where we can have simultaneous moves, we can also generate possible combinations of moves of all players, which results in $\mathcal{M}$ (for planning, $\mathcal{M}$ simply corresponds to the set of all possible actions). Each move $m \in \mathcal{M}$ can be represented by a BDD trans_m, so that the complete transition relation trans is the disjunction of all these: $\mathit{trans} := \bigvee_{m \in \mathcal{M}} \mathit{trans}_m$.

To perform symbolic search, we need two sets of variables: one set, S, for the current states, the other one, S', for the successor states. To calculate the successors of a state set *from*, in symbolic search we use the *image* operator:

$$\mathit{image}\,(\mathit{from}) := \exists S.\,(\mathit{trans}\,(S, S') \wedge \mathit{from}\,(S))\,.$$

As these successors are represented using only S', we need to swap them back to S.[2] This way, if we start at the initial state, each call of the image results in an entire breadth-first search (BFS) layer. So, BFS is simply the iteration of the image, until a fix-point is reached.

As the transition relation *trans* is the disjunction of a number of moves, it is equivalent to generate the successors using one move after the other and afterwards calculate the disjunction of all these states:

$$\mathit{image}\,(\mathit{from}) := \bigvee_{m \in \mathcal{M}} \exists S.\,(\mathit{trans}_m\,(S, S') \wedge \mathit{from}\,(S))\,.$$

This way, we do not need to calculate a monolithic transition relation, which takes time and often results in a BDD too large to fit into RAM.

The inverse operation of the image is also possible. The *pre-image* results in a BDD representing all the states that are predecessors of the given set of states *from*:

$$\mathit{pre\text{-}image}\,(\mathit{from}) := \exists S'.\,(\mathit{trans}\,(S, S') \wedge \mathit{from}\,(S'))\,.$$

With this, we can perform a BFS in backward direction as well.

Action Planning

In this section, we give a brief overview of the algorithms we use for finding optimal plans for classical sequential planning as well as net-benefit planning.

Classical Planning

A classical planning problem is a tuple $\mathcal{P} = \langle \mathcal{S}, \mathcal{M}, \mathcal{I}, \mathcal{T}, c \rangle$ with $\mathcal{S} \subseteq 2^{\mathcal{V}}$ representing a set of states, $2^{\mathcal{V}}$ the powerset of variables $\mathcal{V}$, $\mathcal{M} : \mathcal{S} \mapsto \mathcal{S}$ a set of actions transforming states into states, $\mathcal{I} \in \mathcal{S}$ an initial state, $\mathcal{T} \subseteq \mathcal{S}$ a set of terminal states, and $c : \mathcal{M} \mapsto \mathbb{N}_0^+$ a cost function specifying a certain cost for each action in $\mathcal{M}$. The goal is to find a sequence of actions that transforms $\mathcal{I}$ to $\mathcal{T}$.

[2]We omit the explicit mention of this in the pseudo-codes to enhance readability. Whenever we write of an image (or pre-image), we assume such a swapping to be performed immediately after the image (or pre-image) itself.

We are interested in a plan with minimal total cost, i. e., the sum of the costs of all actions of the plan must be minimal.[3] For this we perform symbolic A* search. Some algorithms, such as BDDA* (Edelkamp and Reffel 1998) and SetA* (Jensen, Bryant, and Veloso 2002), have been proposed for this. We perform BDDA*, which uses a two-dimensional matrix of BDDs. The element at position (g, h) represents all the states with g being the distance from $\mathcal{I}$ and h an estimate on the distance to $\mathcal{T}$.

Starting at $\mathcal{I}$, we proceed through this matrix along f-diagonals, with $f = g + h$. For each diagonal we start with the BDD with smallest g value, expand it using the image function and go on until all states of this diagonal have been expanded and we switch over to the next larger f diagonal. Once a goal state is reached, we are done and can reconstruct the plan (if we can assume the heuristic estimate to be at least admissible).

Net-Benefit Planning

A net-benefit planning problem $\mathcal{P}_{NB}$ additionally contains some soft goals $\mathcal{T}_S \in \mathcal{V}$ along with a utility function $u : \mathcal{T}_S \mapsto \mathbb{N}_0^+$ specifying the reward for achieving these goals. The goal again is to find a sequence of actions that transforms $\mathcal{I}$ to $\mathcal{T}$.

We are interested in a plan with maximal net-benefit, which is the total utility for achieving soft goals minus the total action cost. We slightly transform this problem into a minimization problem by assuming that we try to achieve all soft goals and need to pay a cost (according to u) for violating one (i. e., failing to achieve one). Thus, we need to minimize the total violation cost plus the total action cost.

While it is possible to calculate an upper bound on the total violation cost (which is the sum for violating all of them), no such bound can be found for the total action cost. Thus, for the first one we can come up with a BDD representing the bound while for the latter we need to rely on the data structures used. We use an open list that is sorted according to the distance to $\mathcal{I}$, which equals the total action cost up to this point. So, for each state (including the terminal ones) we already know the precise total action cost.

To calculate a plan achieving optimal net-benefit we perform an algorithm (see (Edelkamp and Kissmann 2009)) that has some similarities to symbolic branch-and-bound planning (Jensen et al. 2006). We iteratively increase a lower bound on the net-benefit (this bound is the total action cost so far; as we add the total action cost and the total violation cost, both of which are assumed to be at least 0, the optimal net-benefit is always at least as high as the current total action cost). At the same time, we decrease upper bounds on the net-benefit as well as the total violation cost.

Whenever we reach a terminal state we check its total violation cost. If this is smaller than the current upper bound on it we update this bound. If the net-benefit for this state is smaller than the best one so far, we update that as well and calculate a plan for reaching this terminal state.

[3]In case of no action costs being present we need to find a plan with a minimal number of actions. Thus, it is equvalent to assume all actions having equal costs, e. g., 1.

Once we reach the situation where either all states are explored or the total violation cost reaches 0 (i.,e., all soft goals are satisfied) or the total action cost is at least as large as the best net-benefit found so far, we are done and can return the last calulated plan.

General Game Playing

In this section, we first of all give a motivation as to why general game playing might be interesting for the planning community as well. Afterwards, we present our approaches for solving general single- and two-player turn-taking games.

General Game Playing and Action Planning

A general game is a tuple $\mathcal{G} = \langle \mathcal{S}, p, \mathcal{M}, \mathcal{I}, \mathcal{T}, \mathcal{R} \rangle$ with $\mathcal{S}$ being the set of reachable states, p the number of players, $\mathcal{M} : \mathcal{S} \mapsto \mathcal{S}$ the set of possible moves for each state, $\mathcal{I} \in \mathcal{S}$ the initial state, $\mathcal{T} \subseteq \mathcal{S}$ the set of terminal states, and $\mathcal{R} : \mathcal{T} \mapsto \{0, \dots, 100\}^p$ the reward for each player in all terminal states.

General game playing can be seen as a generalization of classical planning. While in planning typically only one agent is asked to find a plan, in general game playing one or more players try to find a good next move in parallel.

In particular, in a competition setting (Genesereth, Love, and Pell 2005) all agents are provided with a description of a game in the logic-based game description language GDL (Love, Hinrichs, and Genesereth 2006), along with a startup and a move time. The startup time is relevant at the beginning of a game and can be used to perform precomputations, while afterwards the move time is used to determine the time the players have to calculate their next move. This move they send to a general game playing server that administers the process in keeping the game state up-to-date and informing all agents of the chosen moves.

The agents aim not only at achieving some goal state, but they are also awarded with certain rewards in the range from 0 to 100 (higher scores are to be prefered).

In the default setting, all players perform their moves simultaneously. If some predicate denoting the current player is set and updated after each move accordingly, it is possible to model turn-taking games as well. For these, all the players that are not active can only perform a noop, i. e., a move that does not change anything about the game's state.

Of course, single-player games are possible as well. Thus, classical action planning might be seen as the special case of a single-player game where the agent tries to reach a goal state where it achieves a reward of 100 (although in general game playing, the number of steps taken is irrelevant, if it is not part of the reward specification).

Net-benefit planning might be seen as a better fit for this. If we assume all actions to have a cost of zero, net-benefit planning would also find a plan that leads to a terminal state achieving the maximal reward of 100. But the main difference is that action planning is offline, i. e., the plan is calculated before it is executed, while general game playing typically is online.

Non-deterministic planning is sometimes refered to as planning in an oily world, i. e., the agent can choose actions but cannot be certain of their outcome, which is determined by the non-determinism. This can be seen as a two-player turn-taking game with the planner being the player and the environment deciding about the non-determinism its opponent (e. g., (Jensen, Veloso, and Bowling 2001; Bercher and Mattmüller 2008; Kissmann and Edelkamp 2009b)), so that even there we can see a connection between planning and general game playing. But as general game playing supports any number of players and also simultaneous moves, it is still more general than this.

Solving General Games

What we are interested in here is not playing general games, but solving them strongly, i. e., we want to find the game theoretic value for each reachable state. Solving complex games has long been a major challenge in AI. One of the last outstanding results is due to Schaeffer et al. (2007), who were finally able to solve the game of American Checkers.

In the context of general game playing, such a solving is even more involved, as the designers of the algorithms do not know what games their programs might be presented with. Thus, a more general approach is mandatory, so it does not seem promising to assume that in the near future similar successes in such complex games will be achieved. Nonetheless, having a tool that potentially can solve any game (given enough time and memory) is still a compelling idea.

In the following, we will present some ideas on how we use symbolic search to solve general single-player as well as two-player turn-taking games.

Solving General Single-Player Games To solve general single-player games we first of all calculate all reachable states by performing symbolic BFS. Afterwards, we perform several backward searches starting at the game's terminal states $\mathcal{T}$. The first search starts at those states giving a reward of 100, the next one at those giving 99 and so on.

For each of these backward searches we use the pre-image to find all the preceding states. As we are not confronted with an opponent, we can be sure that we can achieve a reward equaling that of our starting point for these states as well. Next, we remove these states from the set of all reachable states and iterate the pre-image calculation, until this results in no new information.

Once we are done with all terminal states, we have the precise information as to what reward we can achieve from any of the reachable states. More details on this algorithm can be found in Edelkamp and Kissmann (2007).

Solving General Two-Player Turn-Taking Games For general two-player turn-taking games, things are a bit more involved. As we are concerned with an opponent, we need to know what it would do (in case of optimal play). Two reasonable assumptions are that it either tries to maximize its own reward or that it tries to maximze the difference of its reward to our reward. The latter case appears to be a reasonable assumption in a competition scenario where only two players play against each other: If one player can achieve more points than its opponent, it might be more beneficial than achieving a higher number of points (with the opponent potentially achieving even more).

To come up with a solution (for more details see Kissmann and Edelkamp (2010)), we start by calculating all reachable states but do not perform any duplicate elimination, so that some states might appear in several BFS layers.

The solving process is again performed in backward direction. We start at the final layer and calculate the rewards for all terminal states (by calculating the conjunction with the corresponding reward BDDs). Afterwards, we iterate through the layers l in decreasing order and first of all calculate the rewards for the terminal states within these. For each of the BDDs representing solved states of the successor layer $l + 1$ (which we do in the order specified by the opponent model), we calculate the pre-image, followed by the conjunction with the non-terminal states of layer l, to come up with the states where the players (in case of optimal play) achieve the same rewards. Before going over to the next solved states, we remove the newly solved ones from layer l, so that none get solved multiple times.

Note, that in principle this can also be adapted to multi-player games. If we assume an opponent model to be given, we can perfom just the same algorithm. But the result is rather unstable, as it highly depends on the opponent model. Unfortunately, we are not aware of any reasonable assumptions (as is the case in two-player games), so it is not yet clear how to determine a correct one. Thus, the result we would get for multi-player games would be similar to the result of the Max^n algorithm (Luckhardt and Irani 1986) that resolves ties by simply choosing the first possible option. But if the players do not play accordingly, the result is worthless. Thus, some more effort needs to be invested in this area, before we are able to come up with good results.

Conclusion

In this short paper, we presented some brief ideas for how we find optimal plans for classical sequential or net-benefit planning as well as how we come up with optimal solutions for general single- and two-player turn-taking games.

References

Bercher, P., and Mattmüller, R. 2008. A planning graph heuristic for forward-chaining adversarial planning. In *18th European Conference on Artificial Intelligence (ECAI)*, volume 178, 921–922. IOS Press.

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

Edelkamp, S., and Helmert, M. 1999. Exhibiting knowledge in planning problems to minimize state encoding length. In *5th European Conference on Planning (ECP)*, volume 1809 of *LNCS*, 135–147. Springer.

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. 2009. Optimal symbolic planning with action costs and preferences. In *21st International Joint Conference on Artificial Intelligence (IJCAI)*, 1690–1695.

Edelkamp, S., and Reffel, F. 1998. OBDDs in heuristic search. In *22nd Annual German Conference on Artificial Intelligence (KI)*, volume 1504 of *LNCS/LNAI*, 81–92. Springer-Verlag.

Fikes, R. E., and Nilsson, N. J. 1971. STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence* 2(3–4):189–208.

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

Helmert, M. 2008. *Understanding Planning Tasks: Domain Complexity and Heuristic Decomposition*, volume 4929 of *LNCS*. Springer.

Jensen, R. M.; Hansen, E. A.; Richards, S.; and Zhou, R. 2006. Memory-efficient symbolic heuristic search. In *16th International Conference on Automated Planning and Scheduling ICAPS*, 304–313. AAAI.

Jensen, R. M.; Bryant, R. E.; and Veloso, M. M. 2002. SetA*: An efficient BDD-based heuristic search algorithm. In *18th National Conference on Artificial Intelligence (AAAI)*, 668–673. AAAI Press.

Jensen, R. M.; Veloso, M. M.; and Bowling, M. H. 2001. OBDD-based optimistic and strong cyclic adversarial planning. In *6th European Conference on Planning (ECP)*, 265–276.

Kissmann, P., and Edelkamp, S. 2009a. Instantiating general games. In *IJCAI-Workshop on General Game Playing*, 43–50.

Kissmann, P., and Edelkamp, S. 2009b. Solving fully-observable non-deterministic planning problems via translation into a general game. In *32nd Annual German Conference on Artificial Intelligence (KI)*, volume 5803 of *LNCS*, 1–8. Springer.

Kissmann, P., and Edelkamp, S. 2010. Layer-abstraction for symbolically solving general two-player games. In *ICAPS-Workshop on Planning in Games*. To appear.

Kuhlmann, G.; Dresner, K.; and Stone, P. 2006. Automatic heuristic construction in a complete general game player. In *21st National Conference on Artificial Intelligence (AAAI)*, 1457–1462. AAAI Press.

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.

Luckhardt, C. A., and Irani, K. B. 1986. An algorithmic solution of N-person games. In *5th National Conference on Artificial Intelligence (AAAI)*, 158–162. Morgan Kaufmann.

Schaeffer, J.; Burch, N.; Björnsson, Y.; Kishimoto, A.; Müller, M.; Lake, R.; Lu, P.; and Sutphen, S. 2007. Checkers is solved. *Science* 317(5844):1518–1522.

Schiffel, S., and Thielscher, M. 2007. Fluxplayer: A successful general game player. In *22nd AAAI Conference on Artificial Intelligence (AAAI)*, 1191–1196. AAAI Press.