Symbolic Search

Stefan Edelkamp
University of Dortmund, Germany

INTRODUCTION

Symbolic search solves state space problems consisting of an initial state, a set of goal states, and a set of actions using a succinct representation for state sets. The approach lessens the costs associated with the exponential memory requirements for the state sets involved as problem sizes get bigger.

Symbolic search has been associated with the term *planning via model checking* (Giunchiglia and Traverso 1999). While initially applied to *model check* hardware verification problems (McMillan 1993), symbolic search features many modern *action planning* systems (Ghallab et al. 2000).

Symbolic search algorithms explore the underlying problem graph by using functional expressions to represent sets of states and actions. Compared with the space requirements induced by standard explicit-state search algorithms, symbolic representations additionally save space by sharing parts of the state vector. Algorithm designs change, as not all search algorithms adapt to the exploration of state sets.

BACKGROUND

Binary decision diagrams or BDDs are one option for a space-efficient representation for state sets.

A BDD (Bryant 1992; see Figure 1), is a data structure to manipulate Boolean functions efficiently. BDDs are finite state machines over the alphabet {0,1} with a 1-sink that operates as an accepting state. Each internal node is labelled with the variable (index) for selecting the outgoing transition (either 1 or 0, see figure) for a given variable assignment. For evaluating a BDD, a path is traced from the root to the sinks (all paths obey the same variable ordering). What distinguishes BDDs from decision trees is the use of reduction rules, detecting unnecessary variable tests and repeating subgraphs. This leads to a unique representation, polynomial in the number of input variables for many interesting functions. The reduced and ordered BDD representation is unique; a clear benefit to the satisfiability test for Boolean formulas, which by the virtue of Cook's Theorem (1971) is an NP-hard problem

In symbolic search, BDDs accept the state vector representation. According functions are satisfied, if the state vector for the input assignment is a member of the represented set. The characteristic function can be identified with the state set it represents.

The transition relation Trans represents the actions (see Figure 2). It refers to current state variables x and next state variables x' and is satisfied, if there is an action that transforms a state vector into one of its successors. The transition relation for the entire prob-

Figure 1.

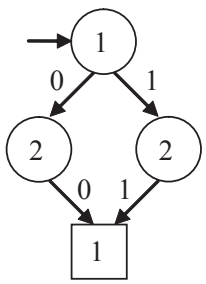
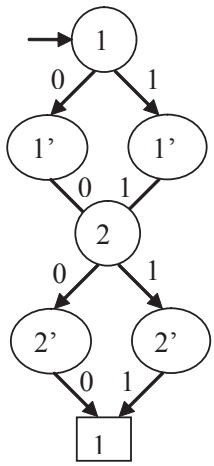


Figure 2.



lem decomposes in the disjunction of the transition relations for singleton actions. The order of variables in the state vector is crucially influencing the size of the BDD. Unfortunately, the problem of finding the ordering that minimizes the BDD size is NP-hard (Wegener 2000). The interleaved representation for the Trans(x,x') that alternates between x and x' variables often leads to small BDDs.

The image of a state set States wrt. the transition relation Trans is computed as Image(x') := $\exists$x (Trans(x,x') $\wedge$ States(x)), where x and x' are vectors of Boolean state variables. The result of this image operation is a characteristic function of all states reachable from States in one step. In order to repeat the process, x with x' have to be substituted for the next iteration by computing the *relational product* States(x) := $\exists$x' ((x=x') $\wedge$ Image(x')). In an interleaved variable ordering with alternating indices for x and x', this operation reduces to a mere textual replacement of node labels.

.

SYMBOLIC SEARCH ALGORITHMS

State space problems numbers of finite domain can be encoded via atomic propositions. A binary encoding is more efficient than a unary one such that most BDD libraries include finite domain variable support. For basic calculus, relations are pre-computed. For example, the binary relation Inc(a,b) for a+1=b is the disjunction of all possible value assignments of a to j and all possible value assignments of b to j+1 for all j counted from 1 to the domain size minus 1. For constructing the ternary relation Add(a,b,c), denoting a+b=c, the enumeration of all possible assignments for a, b and c is less efficient than computing the term Add(a,b,c) := (b=0 $\wedge$ a=c) $\vee$ $\exists$b',c' (Inc(b',b) $\wedge$ Inc(c',c) $\wedge$ Add(a,b',c')) recursively. Starting with the first clause the second clause is applied until convergence.

Symbolic Breadth-First Search

In iteration i of the symbolic variant of breadth-first-search the set of states States[i] reachable from the initial state s in i steps is computed. The search is initialized with States[0] set to the initial state set. In order to terminate the search the algorithm checks, whether or not a state is represented in the intersection of the set States[i] with the set of goal states. Since States[0],…,States[i-1] have been computed without success, given a non-empty intersection, i is the optimal solution length. To avoid an infinite search behaviour in case of the absence of a solution, Reach = States[0] $\vee$…$\vee$ States[i-1] is omitted from States[i] by setting States[i] := States[i] $\wedge\neg$Reach before updating Reach := Reach $\vee$ S[i]. For some problem classes (like undirected or acyclic graphs) the *duplicate elimination scope* {0,…,i-1} can be reduced to a limited number of breadth-first search levels.

By keeping the intermediate BDDs contained in the memory, a legal sequence of states linking the initial state to any goal state g in States[i] $\cap$ G is a successful solution. The state on an optimal path to a goal g in layer i must be located in the second last breadth-first search layer i-1. All states that are contained in the intersection of the predecessors of the goal g are and States[i-1] are reachable in an optimal number of steps and reach the goal in one step. Any of these states can be chosen to continue *solution reconstruction*. Eventually the initial state is found. If layers have been eliminated to recover main memory, divide-and-conquer solution reconstruction methods are required (Jensen et al. 2006). Variants of symbolic breadth-first search compute cost-optimal solutions subject to general cost functions (Edelkamp 2006).

Backward breadth-first search exploits the relational representation for the actions to compute the *preimage* according to the formula Preimage(x) := $\exists$x' (States(x') $\wedge$ Trans(x,x')). Consequently, the search starts with the goal state set and iterates until it hits the start state. Bidirectional symbolic breadth-first search executes concurrent iterations of forward and backward breadth-first search until the two search frontiers meet.

Symbolic Dijkstra's Single Source Shortest Paths Algorithm

Action costs are a natural search concept. In many applications, costs can only be bounded integers. Examples for such discrete cost actions are *macros* as exploited in the macro-problem solver by Korf (1985).

Let the *weighted transition relation* Trans(c,x,x') evaluate to 1, if the step from x to x' has cost c $\in$ {1,…,C}, encoded in binary. The symbolic version of Dijkstra's single-source shortest paths algorithm (1959) then works as follows. The priority relation Queue(f,x) is initialized with the representation of the start state and f-value 0. Until a goal state is reached, in each iteration, the algorithms determines the minimum f-value