

External Symbolic Heuristic Search with Pattern Databases

Stefan Edelkamp

Computer Science Department
University of Dortmund, Germany
stefan.edelkamp@cs.uni-dortmund.de

Abstract

In this paper we propose refinements for optimal search with symbolic pattern databases in deterministic state-space planning. As main memory is limited, external heuristic search is combined with the power of symbolic representation. We start with an external version of symbolic breadth-first search. Then an alternative and external implementation for BDDA* to include different heuristic evaluation functions into the symbolic search process is presented.

We evaluate the approach in benchmarks taken from the 4th international planning competition.

Introduction

Symbolic state space search acts on sets of states rather than on singular ones. An appropriate data structure to represent these state sets are binary decision diagrams, BDDs for short (Bryant 1986). The effective usage of BDDs for symbolic exploration has been suggested by (McMillan 1993) in the context of *symbolic model checking*. Successful model checkers based on BDDs are e.g. nuSMV (Cimatti *et al.* 1997) and μ cke (Biere 1997).

Alternatives for the exploration with BDDs are *bounded model checkers* as introduced by (Biere *et al.* 1999), which base on the SATPLAN exploration scheme (Kautz and Selman 1996). The key idea of symbolic exploration with binary decision diagrams is to avoid (or at least lessen) the costs associated with the exponential blowup of the Boolean formulae involved as problem sizes get bigger.

BDD technology has first been applied in AI planning by (Cimatti *et al.* 1997). Subsequently, symbolic model checkers have been extensively exploited to solve especially non-deterministic planning tasks (Cimatti *et al.* 1998). Deterministic symbolic planners rely on an efficient state encoding (Edelkamp and Helmert 2001).

The rise of heuristic search procedures for symbolic model checking, as in (Reffel and Edelkamp 1999), (Qian and Nymeyer 2004) and (Santone 2003) indicates that directed state space exploration techniques enrich formal methods for validation and verification.

Copyright © 2005, American Association for Artificial Intelligence (www.aaai.org). All rights reserved.

Many symbolic heuristic search algorithms extend the BDD version of the A* algorithm, BDDA* for short (Edelkamp and Reffel 1998); initially proposed in the context of AI puzzle solving. ADDA*, developed by (Hansen *et al.* 2002), is an alternative implementation of BDDA* with so-called *algebraic decision diagrams* (ADDs). SetA* (Jensen *et al.* 2002) refines the partitioning in the BDDA* algorithm. Additionally, it works on a matrix representation of g and h values. (Qian and Nymeyer 2003) observed that for some AI benchmarks less informed and simpler heuristics turn out to be more efficient than complex and better informed ones.

Another option to cope with the limits of main memory is external exploration (Sanders *et al.* 2002). We refer to single disk model as invented by (Aggarwal and Vitter 1988). One of the first applications of external search in model checking is due to (Stern and Dill 1998).

External search has been brought to AI in form of *delayed duplicate detection* by (Korf 2003) in the context of complete solutions to rectangular ($n \times m$)-Puzzles. (Zhou and Hansen 2004b) incorporated so-called *structured duplicate detection* to external-memory graph search, which exploits the regular structure of state spaces according to different projection function. (Korf 2004) successfully extended delayed duplicate detection to best-first search and also considered the omission of the visited list as proposed in *frontier search*. In his proposal, it turned out that any 2 of the 3 options are compatible, yielding the following portfolio of algorithms: *breadth-first frontier search with delayed duplicate detection*, *best-first frontier search*, and *best-first search with external non-reduced closed list*. In the last case, a buffered traversal in a bucket-based queue is needed.

With External A* (Edelkamp *et al.* 2004) we showed how to integrate all three aspects *delayed duplicate detection*, *best-first search*, and *frontier search* in one algorithm. The algorithm was originally applied to solve hard $(n^2 - 1)$ -Puzzle instances. Recently, we have extended External A* to explicit-state model checking (Jabbar and Edelkamp 2005), studying general heuristics and graph structures.

This paper combines external search with symbolic search. It is structured as follows. We first review the basics of Symbolic A* and External A* search. Next, we

introduce pattern databases based on an SAS⁺ encoding of the planning problem, and show how such problems are inferred automatically from standard STRIPS. Subsequently, we propose the new algorithms External BDD-BFS and External BDDA*. We then discuss different refinements to the algorithms, especially the efficient integration of multiple pattern databases. In the experimental section, we present some results to IPC-4 planning benchmarks. Finally, we draw conclusions.

Symbolic A*

The use of BDDs for symbolic exploration has been suggested by (McMillan 1993) in the context of *symbolic model checking*. Different to Boolean formulae, the BDD representation is unique. (Minato *et al.* 1990) showed how to store several BDDs in a joint structure. Improved implementation issues are to be found in (Yang *et al.* 1998a). For a survey on the theory and the applications of BDDs, cf. (Wegener 2000).

Given a fixed-length binary code for the state space of a search problem, BDDs can be used to represent the *characteristic function* of a set of states (which evaluates to true for a given bit string if and only if it is a member of that set). The characteristic function can be identified with the set itself. We assume states to be represented as binary strings. Transitions are formalized as relations, i.e. as sets of tuples of predecessor and successor states, or alternatively as the characteristic function of such sets. By conjoining this formula with any formula describing a set of states and querying the BDD engine for the possible instantiations of the predecessor variable set, we can calculate all states that can be reached in some state from the input set. This is the *relational product operator*

$$Image(x) = \exists x' T(x, x') \wedge States(x)[x \setminus x']$$

that is used to calculate the *image* from a set of predecessor states *States* and a transition relation *T*. The term $[x \setminus x']$ denotes the replacement of variables to allow iterated image computation. Based on the fact that disjunction and existential quantification commute, for the operator transition subrelations T_O of *T* with $T = \bigvee_{O \in \mathcal{O}} T_O$ and \mathcal{O} being the set of operators we have

$$Image(x) = \bigvee_{O \in \mathcal{O}} (\exists x' T_O(x, x') \wedge States(x)[x \setminus x']).$$

In *informed search* with every state in the search space we associate an estimate *h*. The rank of a state is the combined value $f = g + h$ of generating path length *g* and estimate *h*. For symbolic search, the estimator BDD *H* can be seen as a relation of tuples (*value*, *state*), which is *true* if and only if $h(state) = value$. Equivalently, *H* can be represented as an array of BDDs $H[i]$ with variables that refer to states.

In the original implementation of BDDA* (Edelkamp and Reffel 1998) a 1-level bucket-based queue (Dial 1969) is simulated, with a bucket for each state set of common *f*-values. The states that fall into a common

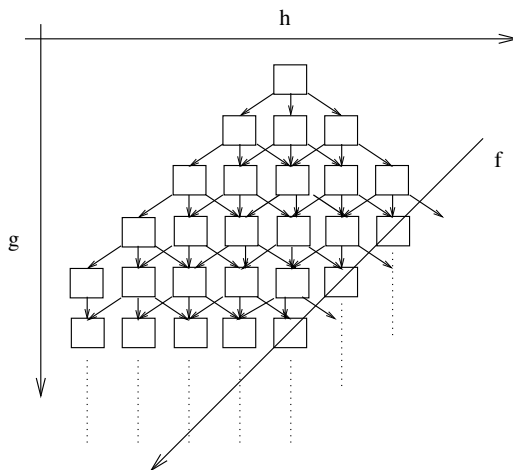


Figure 1: 2D Bucket implementation of BDDA*.

bucket are represented in form of a BDD. The search tree is traversed in best-first manner. In each expansion step, an entire bucket is expanded. Successor states are generated by applying the transition relation and are re-inserted into the queue. The according buckets are addressed using BDD arithmetics. Optimality and completeness are inherited from the fact that given an optimistic heuristic, A* will find an optimal solution.

Viewed differently, BDDA* extends a symbolic version of Dijkstra's single source shortest path algorithm, where a BDD for the horizon *Open* is initially set to the BDD representation of the start state with associated merit. Until we encounter a goal state in each iteration this algorithm extracts the BDD representation of *all* states with minimal *f*-value f_{min} . If no goal state is found, the (weighted) transition relation is applied in a relational product to determine the BDD for the set of successor states. Then the new *f*-values for this set are computed. Finally, the BDD for *Open* in the next iteration is obtained by the disjunction of the successor set with the remaining queue. The extensions of BDDA* with respect to the symbolic version of Dijkstra's algorithm is to include *H* in computing the *f*-values.

In the alternative implementation SetA* of BDDA* proposed by (Jensen *et al.* 2002) a 2D bucket layout as shown in Figure 1 is used. One advantage is that the state sets to be expanded next are generally smaller, and the hope is that the BDD representation is as well. The other advantage is that, given the bucket a state set is addressed by, each set already has both the *g* and the *h* value attached to it, and the effort to compute the *f*-values for the successors is trivial. As we will see, this approach has a tight connection to the bucket representation that is chosen for *External A** search.

For the analysis of the efficiency of BDDA*, we assume a *consistent* heuristic, i.e. for every node *u* and every successor *v* of *u* we expect the equation $h(u) \leq h(v) + 1$ to be true. It is easy to see (cf. Figure 1) that in case of a consistent heuristic, the number

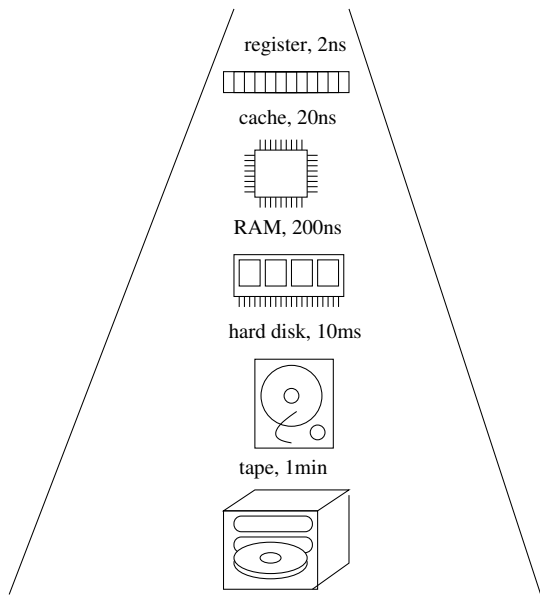


Figure 2: The memory hierarchy.

of iterations in BDDA* is bounded by $O((f^*)^2)$ iterations, where f^* is the optimal solution length. For an optimal heuristic, i.e., a heuristic that estimates the shortest path distance, we have at most $h(s) = f^*$ iterations in BDDA*. On the other hand, if the heuristic is equivalent to the zero function (breadth-first search), we need f^* iterations, too. In general there are at most $f^* + 1$ different h -values and at most $f^* + 1$ different g -values that are encountered during the search process. Consequently, for each period between two successive f -values, we have at most f^* iterations.

As the f -value of the successor states v of u is either unchanged or incremented by $h(v) - h(u) + 1$, one might want to encode the heuristic function incrementally for u and v in common (Jensen *et al.* 2002). For this case, the heuristic function partitions the transition relation according to the range of $h(v) - h(u)$. For some cases, like the *Manhattan Distance* heuristic in the $(n^2 - 1)$ -Puzzle and the *Max-Atom* heuristic in STRIPS action planning (Bonet and Geffner 2001), this appears to be a perfect choice, since the incremental representation is either simpler or not much different compared to the ordinary one. However, for pattern database search, an incremental estimator representation is involved.

External A*

Most modern operating systems hide secondary memory accesses from the programmer, but offer one consistent address space of *virtual memory* that can be larger than internal memory. When the program is executed, virtual addresses are translated into physical addresses. Only those portions of the program currently needed for the execution are copied into main memory. Application programs may exhibit *locality* in their pattern of memory accesses: i.e., data residing in a few pages are

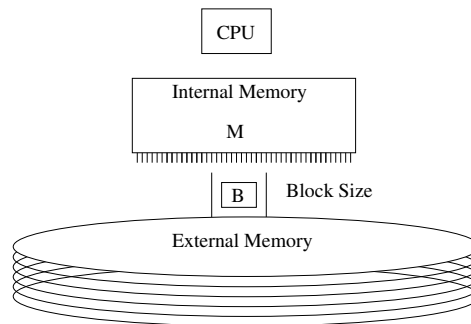


Figure 3: The external memory model.

repeatedly referenced for a while, before the program shifts attention to another working set. Caching and pre-fetching heuristics have been developed to reduce the number of page faults (the referenced page does not reside in the cache and has to be loaded from a higher memory level). By their nature, however, these methods are general-purpose and can not always take full advantage of locality inherent in algorithms. Algorithms that explicitly manage the memory hierarchy (cf. Figure 2) can lead to substantial speedups, since they are more informed to predict and adjust future memory access.

The standard model for comparing the performance of external algorithms consists of a single processor, a small internal memory that can hold up to M data items, and an unlimited secondary memory. The size of the input problem (in terms of the number of records) is abbreviated by N . Moreover, the *block size* B governs the bandwidth of memory transfers. It is often convenient to refer to these parameters in terms of blocks, so we define $m = M/B$ and $n = N/B$. It is usually assumed that at the beginning of the algorithm, the input data is stored in contiguous block on external memory, and the same must hold for the output. Only the number of block read and writes are counted, computations in internal memory do not incur any cost (cf. Figure 3). The single disk model for external algorithms has been invented by (Aggarwal and Vitter 1988). An extension of the model considers D disks that can be accessed simultaneously. When using disks in parallel, the technique of *disk striping* can be employed to essentially increase the block size by a factor of D . Successive blocks are distributed across different disks.

It is convenient to express the complexity of external-memory algorithms using a number of frequently occurring primitive operations. The simplest operation is *scanning*, which means reading a stream of records stored consecutively on secondary memory. In this case, it is trivial to exploit disk- and block-parallelism. The number of I/Os is $\Theta(\frac{N}{DB}) = \Theta(\frac{n}{D})$. Another important operation is external *sorting*. The proposed algorithms fall into two categories: those based on the *merging* paradigm, and those based on the *distribution* paradigm. The algorithms' complexity is

$$\Theta\left(\frac{N}{DB} \log_{M/B} \frac{N}{B}\right) = \Theta\left(\frac{n}{D} \log_m n\right).$$

*External A** (Edelkamp *et al.* 2004) maintains the search space on disk. The priority queue data structure is represented as a list of buckets. In the course of the algorithm, each bucket $Open[i, j]$ will contain all states u with path length $g(u) = i$ and heuristic estimate $h(u) = j$. As same states have same heuristic estimates, it is easy to restrict duplicate detection to buckets of the same h -value. By an assumed undirected state space problem graph structure, we can restrict aspirants for duplicate detection furthermore. If all duplicates of a state with g -value i are removed with respect to the levels $i, i - 1$ and $i - 2$, then there no duplicate state will remain for the entire search process. For breadth-first-search in explicit graphs, this is in fact the algorithm of (Munagala and Ranade 2001). We consider each bucket for the *Open* list as a different file that has an individual internal buffer. A bucket is *active* if some of its states are currently expanded or generated. If a buffer becomes full, then it is flushed to disk. The algorithm maintains the two values g_{\min} and f_{\min} to address the correct buckets. The buckets of f_{\min} are traversed for increasing g_{\min} -value unless the g_{\min} exceeds f_{\min} . Due to the increase of the g_{\min} -value in the f_{\min} bucket, an active bucket is *closed* when all its successors have been generated. Given f_{\min} and g_{\min} , the corresponding h -value is determined by $h_{\max} = f_{\min} - g_{\min}$. According to their different h -values, successors are arranged into different horizon lists. Duplicate elimination is delayed.

There can be two cases that can give rise to duplicate nodes (cf. Figure 4): two different nodes of the *same* predecessor bucket generate matching nodes, and two nodes belonging to *different* predecessor buckets generating matching nodes. These two cases can be dealt with by merging all the pre-sorted buffers corresponding to the same bucket, resulting in one sorted file. This file can then be scanned to remove duplicates from it. In fact, both the merging and duplicates removal can be done simultaneously. The resulting output file is processed to eliminate already expanded duplicate nodes from it. It is clear that identical nodes have the same heuristic estimate, so duplicates have to appear in some buckets on the same vertical line. It suffices to perform duplicate removal only for the bucket that is to be expanded next.

One may presort elements within main memory capacity with any internal sorting algorithm at hand. Suggested by Korf’s work (Korf 2003), one may also use external sorting based on hash partitions, i.e., an external variant of key-dependent sorting. This idea corresponds to the well-known *bucket sort* algorithm, where numbers $a \in [0, 1)$ are thrown into n different buckets $b_i = [i/n, (i + 1)/n)$, according to some arbitrary hash function. All the lists that are contained in one bucket can be sorted independently by some other sorting algorithm. The sorted lists for b_i are concatenated to a fully sorted list. Then the lists have to be merged. Even considerable work to sort the subproblems does not affect the gain of a good partitioning.

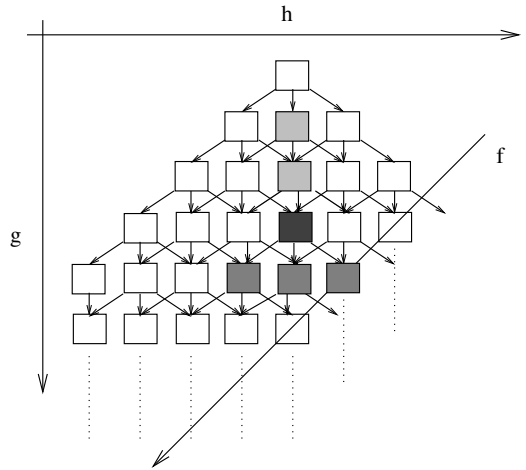


Figure 4: External A* with consistent heuristic in a uniform undirected graph.

Since External A* simulates A* and changes only the order of elements to be expanded that have the same f -value, completeness and optimality are inherited from the properties of A*. The I/O complexity for External A* in an implicit unweighted and undirected graph with a consistent estimates is bounded by $O(\text{sort}(|E|) + \text{scan}(|V|))$, where $|V|$ and $|E|$ are the number of nodes and edges in the explored subgraph of the state space problem graph, and $\text{scan}(n)$ ($\text{sort}(n)$) are the number of I/O needed to scan (sort) n elements. Since each state is expanded at most once, all sortings can be done in time $O(\text{sort}(|E|))$ I/Os. Filtering, evaluating, and merging are all available in scanning time of the buckets in consideration. The I/O complexity for predecessor elimination depends on the number of buckets that are referred to during file subtraction/reduction. As we assumed undirected graphs, this value is bounded by 2, so that each reduced bucket is scanned at most a constant number of times.

If we have $|E| = O(|V|)$, the complexity reduces to $O(\text{sort}(|V|))$. For small integer weights in $\{1, \dots, C\}$ the successors of the nodes in the active bucket are no longer spread across three, but over $3 + 5 + \dots + 2C + 1 = C \cdot (C + 2)$ buckets (cf. Figure 5). For duplicate reduction, we have to subtract the $2C$ buckets prior to its nodes’ expansion. Consequently, the I/O complexity for External A* in an implicit unweighted and undirected graph, is bounded by $O(\text{sort}(|E|) + C \cdot \text{scan}(|V|))$.

We establish the solution path by backward chaining starting with the target state. There are two main options. Either we store predecessor information with each state on disk or, more elegantly, for a state in depth g we intersect the set of possible predecessors with the buckets of depth $g - 1$. Any state that is in the intersection is reachable on an optimal solution path, so that we can recur. The time complexity is bounded by the scanning time of all buckets in consideration and surely in $O(\text{scan}(|V|))$. It has been shown (Edelkamp *et al.*

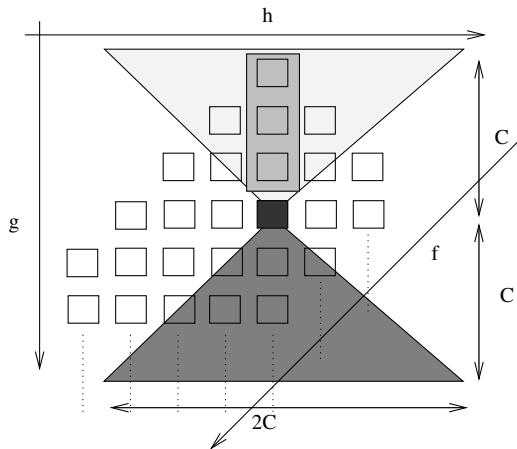


Figure 5: External A* with a consistent estimate in a nonuniform undirected graph.

2004) that the lower bound for the delayed duplicate detection is $\Omega(\text{sort}(|V|))$ I/Os.

SAS⁺ Encoding of Planning Domains

Action Planning refers to a world description in predicate logic. A number of atomic propositions describe what can be true or false in each state of the world. By applying operators in a world, we arrive at another world where different atoms might be true or false.

Propositional STRIPS planning problems (Fikes and Nilsson 1971) may efficiently be encoded with numbers as follows. An *SAS⁺-planning problem* (Helmert 2004) is a quadruple $\mathcal{P} = (\mathcal{V}, \mathcal{O}, \mathcal{I}, \mathcal{G})$, with $\mathcal{V} = \{v_1, \dots, v_n\}$ being a set of state variables with finite domain \mathcal{D}_v . A *partial assignment* for \mathcal{V} is a function s over \mathcal{V} , such that $s(v) \in \mathcal{D}_v$, if $s(v)$ is defined; Moreover, \mathcal{O} is a set of operators given by a pair of preconditions and effects, and \mathcal{I} and \mathcal{G} being the initial and goal state in the form of a partial assignment.

The process of finding a suitable SAS⁺ encoding is illustrated with a simple planning problem, where a truck T is supposed to deliver a package P from A to B. The initial state is defined by the atoms (AT P A), and (AT T A). Goal states have to satisfy the condition (AT P B). The domain provides three actions named LOAD to load a truck with a package at a certain location, the inverse operation UNLOAD, and DRIVE to move a truck from one location to another. The first pre-processing step will detect that only the AT and IN predicates are fluents and thus need to be encoded. In a next step, some mutual exclusion constraints are discovered. In our case, we will detect that a given object will always be at or in at most one other object, so propositions such as (AT P A) and (IN P T) are mutually exclusive. This result is complemented by *fact space exploration*. Ignoring delete effects of operators, we exhaustively enumerate all propositions that can be satisfied by any legal sequence of actions applied to the initial state, thus ruling out illegal propositions such as

(IN A P), (AT P P) or (IN T B). We discover that three Boolean variables are sufficient to encode any reachable state. The first one is required for encoding the current city of the truck. It is set if (AT T B) is true, and cleared otherwise, i.e., if (AT T A) is true. The other two variables encode the status of the package.

Inferring an optimized SAS⁺ encoding by fact space exploration refers to (Edelkamp and Helmert 1999), where it has been used to minimize the binary state description length, however, without using the term SAS⁺. With respect to the IPC-4 competition domains, the inference of the SAS⁺ encoding had to be slightly refined. Most of the problems were due to the automatic translation of the ADL description to a STRIPS description. The first problem were operators with delete effects that are not queried in the precondition list. The precompiler could be adjusted without any burden. The second problem were variables that were either true or false, like BLOCKED and NON-BLOCKED. The inference engine determined that the two predicates could in principle be encoded together, but since we have different atoms as instances for these predicates that are true in the initial state, the precompiler conservatively divided larger variable domains into Boolean flags, one for each atom. We improved this splitting to allow contradictory atom pairs to be encoded together.

Symbolic Pattern Databases

Abstraction transformations are a way to create admissible heuristics automatically. Such a transformation ϕ maps a state in the original problem space into an abstract state. If the shortest path for all states u, v in the original space is larger than the shortest path between $\phi(u)$ and $\phi(v)$ in the abstract state space, so that this distance can be used as an admissible heuristic for the search in the original search space.

The main requirement for a proper abstractions is that the mapping ϕ is a *state space homomorphism*, i.e. that for all edges (u, v) in the original state space graph, there must also be an edge $(\phi(u), \phi(v))$ in the abstract state space graph. A very general abstraction technique is to merge nodes in the state space graph, possibly eliminating self-loops and multiple edges.

To define abstract STRIPS planning problems, we may omit propositions and restrict to propositions that are in a given set R . This defines the domain abstraction function ϕ_R . The interpretation of ϕ_R is that all Boolean variables for propositional predicates not in R are mapped to a *don't care symbol*. For SAS⁺ encoded problem domains, we can provide state space abstractions that neglect variables or reduce their domains. The automated choice of the optimal abstraction ϕ is a hard combinatorial task, and is approximated in planning practice.

Re-computing the heuristic estimate for each state this option cannot possibly speed-up search. Of course, this assumes that the heuristic is computed once for a single problem instance; if it were stored and reused

over multiple instances, its calculation could be amortized. In Hierarchical A* (Holte *et al.* 1996), estimate values are computed on demand, but cached.

An alternative approach is to completely evaluate the entire abstract search space prior to the base level search. For a fixed goal state t and any abstracted space, a *pattern database* is a lookup table indexed by the abstract state $\phi(u)$, containing the shortest path length from $\phi(u)$ to the abstract goal state $\phi(t)$. The size of a pattern database is the number of states in the abstract state space. Pattern databases have been introduced by (Culberson and Schaeffer 1998). They have shown to be very effective in solving random instances of the Rubik’s Cube (Korf 1997) and the Twenty-Four-Puzzle (Korf and Felner 2002) optimally.

The easiest way to to create a pattern database is by conducting a shortest path search in backward direction, starting at $\phi(t)$. This procedure is sometimes termed *retrograde analysis*. A precondition is that operators are *invertible*, i.e., the set of legal reachable states that can be transformed into a target state must be efficiently computable. By constructing the inverse graph of the reachable abstract state set on-the-fly, however, it is possible to generate pattern databases for directed non-invertible problem spaces, too.

In explicit pattern database construction, the pattern databases are often represented as *perfect hash tables*, with entries referring to the distance value only. In this case, the size for the pattern database is the binary encoding length for the distance values times the minimum perfect hash table size.

Symbolic pattern databases (Edelkamp 2002) are pattern databases that have been constructed with BDD exploration for latter use either in symbolic or explicit heuristic search. Each shortest path layer is represented by one BDD. Better scaling seems to favor symbolic pattern database exploration. The results we obtained indicate that by far more new states are encountered than BDD nodes were necessary to represent them. The measured effect of symbolic representation corresponded to memory gains in the orders of magnitude. Note that in theory there is no such relation or limit. It depends on how regular the structure is of the set of states one aims to represent. In principle, the leaf BDD 1 can represent a state-set of any size.

External Symbolic BFS Search

As said, transition relations are Boolean expressions for operator application. They encode all valid (state, successor state) pairs utilizing twice the number of Boolean state variables. In our implementation, the transition relations are computed for all grounded operators, as Boolean expressions of their precondition, add and delete lists. In other words, the transition function is kept partitioned so that the relational product for computing the image splits into several ones. We use a balanced binary tree for combining the disjuncts.

Symbolic BFS search is performed while maintaining each BDD level on disk. Figure 6 gives a pseudo-

Procedure External BDD-BFS

```

Open[0] ← I
gmin ← 0
while (G ∧ Open = 0)
  Reduce(Open[gmin])
  Open[gmin + 1](x') ←
    √O ∈ O (∃x TO(x, x') ∧ Open[gmin](x)[x \ x'])
  gmin ← gmin + 1

```

Figure 6: External Symbolic Breadth-First Search.

code implementation of the algorithm. The description does not differ from an internal implementation, except that BDDs are stored on disk and duplicate elimination is done by performing a boolean subtraction operation between two encoded state sets residing on disk. The inputs of the algorithm are the characteristic functions of the initial and goal state, and the operators transition relation. The optional *reduction of Open* in *Reduce* takes the actual bucket and subtracts as many previous layers as there are in the *duplicate elimination scope*.

Since BDDs are also large graphs, improving memory locality has been studied e.g., in the breadth-first synthesis of BDDs that constructs a diagram levelwise (Hu and Dill 1993). There is a trade-off between memory overhead and memory access locality so that hybrid approaches based on context switches have been explored (Yang *et al.* 1998b). Efficiency analysis show that BDD reduction of a decision diagram G can be achieved in $\mathcal{O}(\text{sort}(|G|))$ I/Os while Boolean operator application to combine two BDDs G_1 and G_2 can be performed in $\mathcal{O}(\text{sort}(|G_1| + |G_2|))$ I/Os (Arge 1995).

External Symbolic A* Search

External BDDA* is designed in the spirit of External A*. For the ease of describing the algorithm, we consider each bucket for the *Open* list as a different file. This accumulates to at most $\mathcal{O}((f^*)^2)$ files. Figure 7 depicts the pseudo-code of the *External BDDA* Search* algorithm for consistent estimates and uniform graphs. The additional input is a pattern database heuristic in form of BFS layer BDDs $H[i]$, $i \in \{1, \dots, \max\}$. Besides optional file handling, the main differences to earlier versions of BDDA* is that there is no need for performing BDD arithmetics. Compared to SetA*, the implementation applies to pattern database heuristics, that are non-incremental.

The algorithm maintains two variables g_{\min} and f_{\min} to address the current bucket. According to their different h -values, successors are arranged into different horizon lists. The implementation of algorithm is not restricted to consistent or admissible heuristics. Duplicate elimination with respect to the set of already expanded states is incorporated in the procedure *Reduce*. As with External A*, in undirected problem graphs we restrict the duplicate scope in *Reduce* to the two previ-

Procedure External BDDA*

```

 $Open[0, h(\mathcal{I})] \leftarrow \mathcal{I}$ 
 $f_{\min} \leftarrow \min\{i + j > f_{\min} \mid Open[i, j] \neq 0\}$ 
while ( $f_{\min} \neq \infty$  or  $\mathcal{G} \wedge Open = 0$ )
   $g_{\min} \leftarrow \min\{i \mid Open[i, f_{\min} - i] \neq 0\}$ 
   $h_{\max} \leftarrow f_{\min} - g_{\min}$ 
   $Reduce(Open[g_{\min}, h_{\max}])$ 
   $A(x') \leftarrow \bigvee_{O \in \mathcal{O}} (\exists x. T(x, x') \wedge Open[g_{\min}, h_{\max}](x)[x \setminus x'])$ 
  for each  $i \in \{0, \dots, \max\}$ 
     $A_i \leftarrow H[i] \wedge A$ 
     $Open[g_{\min} + 1, i] \leftarrow Open[g_{\min} + 1, i] \vee A_i$ 
   $f_{\min} \leftarrow \min\{i + j \mid Open[i, j] \neq 0\} \cup \{\infty\}$ 

```

Figure 7: External Symbolic A* Search.

ous layers. By performing duplicate removal we reduce the state, but non-necessarily the BDD node count.

One important feature is that external sorting as with explicit state storage is no longer required. In symbolic search, merging and duplicate removal are performed simultaneously by applying Boolean disjunction. File subtraction as required in procedure *Reduce* is another file-based BDD operation. Omitting or approximating this step will not affect completeness or optimality.

We have not shown the implementation of multiple symbolic pattern databases H_j , $j \in \{1, \dots, k\}$, which for our case required an accumulated addressing of the *Open* buckets. Another option is to use a multi-dimensional *Open*-list.

Refinements

We have implemented all possible image operations in a bi-directional fashion. Subsequently, pattern databases can be constructed for regression search starting with a BFS for the initial state. As the initial state contains all state information a backward pattern database heuristic for forward search is more informed than in a design for forward search. For symbolic search it turns out that backward exploration is computationally expensive, so traversing the abstract space for generating the pattern databases backwards was the better choice in most cases. On the other hand for regression type planners, like most variants of the planner HSP, forward pattern databases can be exploited.

During symbolic breadth-first enumeration, we allow backward exploration to be mixed with forward exploration, without affecting optimality. *Target enlargement*, sometimes referred to as a variant of perimeter search, is available. Different to bi-directional External BDD-BFS, in the form implemented, bi-directional External BDDA* is not necessarily admissible.

Multiple, in particular disjoint pattern databases are very important to derive well-informed, especially admissible estimates. For external search, pattern databases and the buckets for the distributed storage

of the *Open* list may reside on disk. For two non-empty pattern databases representing the heuristic estimates h_1 and h_2 , we proceed as follows. We take the successor set and compute first the partition according to the first pattern database and then distribute each of the results according to the second pattern database. This yields a two-dimensional state set table. Next we combine the state sets with common $h = h_1 + h_2$ value via disjunction to fill the *Open* list entry at column h .

Experiments

We have successfully applied the implementation to several problem instances from IPC-1 to IPC-4. Here we present experiments on selected benchmark problems from the 2004 planning competition, since that was the only competition, where the optimal planners have been judged separately to the non-optimal ones. The experiments were run on a 2.66 GHz Pentium IV with 512 MB, slightly less than the resources (3 GHz, 1 GB) available on the competition machine. As in the competition, the CPU time bound was set to 1/2 hour.

All heuristic search results are based on unidirectional search, i.e. target enlargement is switched off. CPU time is total, including parsing, pattern database construction, and generating the operator BDDs. The individual estimated pattern database sizes vary in between 2^{10} and 2^{50} , so that explicit storage would clearly exceed main memory capacity.

As we are optimizing the number of actions in the plan, the competitor to relate to is BFHSP. All other optimal planners that have participated in IPC-4 optimize the parallel plan length (makespan). This is a very different optimization criterion and the sequential number of steps in parallel plans produced by these planners is far from the optimum. Most parallel planners can be parameterized to do sequential optimization, but are by far weaker in coverage. BFHSP is a planner based on the principle of breadth-first heuristic A* search (Zhou and Hansen 2004a). The state-of-the art explicit-state search engine uses the *Max-Triple* heuristic (Haslum and Geffner 2000) to focus the search.

The results of the Airport domain are shown in Table 1. In almost all cases the algorithm works on the optimal diagonal ($g + h = f^*$) only, by means that the heuristic estimator is perfect. This allows to illustrate the growth of the BDD sizes for External BDD-BFS and External BDDA* during the exploration in Figure 8 (Problem 9). For BDD-BFS we took the breadth-first layer as iteration number, while for BDD-BDDA* we took the number of nodes on the f^* diagonal. For this problem the accumulated total number of BDD nodes for BDD-BFS exceeds main memory capacity, so that external exploration was necessary. On the other hand, the number of states that were represented in the BDDs in External BDDA* was surprisingly much smaller (cf. Figure 9). On a second glance, this anomaly is no longer surprising, as the savings in the number of states by applying heuristics are high, and very few represented states often call for somewhat larger BDDs.

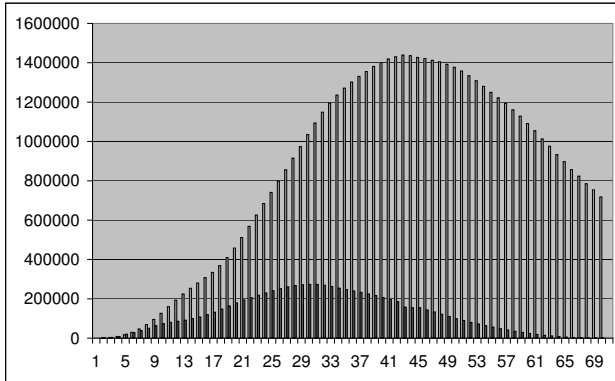


Figure 8: Number of BDD nodes with respect to the iteration number of External BDD-BFS (top curve) and External BDDA* (bottom curve).

Compared to BFHSP, the solution qualities match. Until it comes to the very large instances, the total time performance, however, is clearly inferior. The additional overhead to build the BDDs and the databases supports the observation that the symbolic representation does not pay-off during search. On the other hand, symbolic exploration does contribute during pattern database construction, where the picture is the inverse; the number of represented states exceeds the number of BDD nodes by magnitudes.

As we have not shown statistics for constructing the pattern databases, we select Problem 16 for illustration. Parsing was available in 0.33 seconds, 7 databases were generated, for which 5 did not yield any exploration result. The diameter of the first pattern database is 108 and it took 132 seconds to build it. The average heuristic value is 46.75. The second pattern database has a diameter of 56, was built in 1.5 seconds for an average heuristic value of 28. As the setup time was 171 seconds in total, most of the remaining 40 seconds were spent in computing the BDD representation for the 498 operators. The External BDDA* search phase itself took less than 10 seconds to complete.

For the *Pipesworld* domain the results are shown in Table 2. Comparing the results of our external symbolic search engines and in difference to the Airport domain the plan lengths reported by BFHSP were not optimal. There is a large gap in run time from Problem 10 to Problem 11. This is mainly due to the larger exploration effort for constructing the pattern databases that are traversed backwards. Although the BDD sizes are small, backward image computation consumed considerably much time. This observation, that backward search is often more complex than expected, can be explained by the fact that the running time of the re-

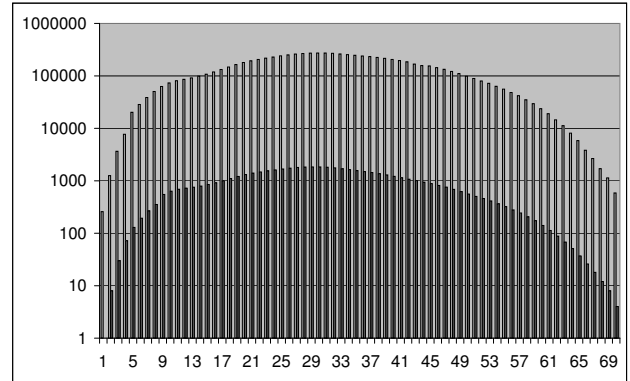


Figure 9: Anomaly in Airport: number of BDD nodes in External BDDA* (on a logarithmic scale) is much larger than the number of represented states.

lational product to compute the backward image is not merely dependent on the size of the BDD but also on the number of subproblem disjunctions and with some respect to the number of states represented. Even if the effect is difficult to quantify and formalize, we see it as another argument that pruning state sets by heuristic symbolix search can ease exploration.

In *PSR* (small) we applied (bi-directional) External BDD-BFS and unidirectional External BDDA*. The result are shown in Table 3. Compared to BFHSP we could also solve Problem 48 and Problem 49, and thereby the entire problem suite. Only one suboptimal (Fast Downward) and one optimal planner (SATPLAN) were able to find solutions in this domain.

Finally, in Table 4 we present the results we obtained in the *Satellite* domain. The state sets increase rapidly, such that a symbolic representation pays off. External BDD-BFS performs almost equivalent to BFHSP, while External BDDA* is the clear-cut winner.

Conclusion

Symbolic heuristic search with BDDs has seen a number of applications in Model Checking and AI. Given the existing limitations in main memory, with External BDD-BFS and External BDDA* we have devised and implemented alternative approaches to the *state explosion* problem for optimal implicit graph search.

The paper contributes the first study of combining external and symbolic heuristic search. As pattern databases are state-of-the-art in current optimal AI problem solving, we started with symbolic pattern databases to improve the traversal of planning space abstractions. We have shown limitation to existing symbolic exploration techniques and presented some new strategies. E.g., we studied different aspects of inte-

	EX-BDD-BFS		EX-BDDA*		BFHSP	
	time	sol	time	sol	time	sol
1	0.25	8	0.30	8	0.05	8
2	0.26	9	0.33	9	0.04	9
3	0.41	17	0.73	17	0.25	17
4	0.43	20	0.53	20	0.23	20
5	0.65	21	0.77	21	0.64	21
6	3.64	41	4.60	41	0.06	41
7	3.70	41	4.57	41	0.06	41
8	38.34	62	19.83	62	0.93	62
9	1,093	71	554	71	9.42	71
10	0.52	18	0.75	18	0.03	18
11	0.77	21	1.04	21	0.04	21
12	5.50	39	7.44	39	0.07	39
13	5.49	37	7.99	37	0.06	37
14	59.07	60	32.62	60	1.15	60
15	58.64	58	34.40	58	1.14	58
16	-	78	180	78	44.13	78
17	-	88	1,358	88	800	88

Table 1: Results in the *Airport* domain.

	EX-BDD-BFS		EX-BDDA*		BFHSP	
	time	sol	time	sol	time	sol
1	0.28	5	0.35	5	0.05	5
2	0.42	12	0.94	12	0.10	12
3	0.74	8	0.45	8	0.15	11
4	1.54	11	0.76	11	0.20	14
5	1.90	8	0.66	8	0.31	9
6	4.00	10	0.93	10	0.58	11
7	5.30	8	1.11	8	1.21	10
8	16.24	10	1.20	10	1.86	11
9	307	13	1.95	13	102	16
10	512	18	3.03	18	1468	19
11	-	-	535	20	147	22
12	-	-	1,032	24	-	-
13	-	-	199	16	-	-

Table 2: Results in the *Pipesworld* domain.

gration for singleton and multiple pattern databases. The chosen application area is STRIPS action planning and the results for finding optimal plans in challenging STRIPS problems are encouraging.

In some cases the symbolic representation for the state set during the overall search process can be an additional burden if the state sets that are considered are small. Consequently, our future plans include the integration of symbolic pattern databases in explicit external search. An additional research avenue is to transfer the results into model checking practice.

Acknowledgements The author would like DFG for support in the projects ED 74/2 and ED 74/3.

References

A. Aggarwal and J. S. Vitter. The input/output complexity of sorting and related problems. *Journal of the*

	EX-BDD-BFS		EX-BDDA*		BFHSP	
	time	sol	time	sol	time	sol
⋮	⋮	⋮	⋮	⋮	⋮	⋮
45	15.65	34	1.48	34	0.12	34
46	432	34	346	34	764	34
47	130	27	7.32	27	1.04	27
48	113	37	117	37	-	-
49	-	≥ 22	1,436	47	-	-
50	88.37	23	47.30	23	1.03	23

Table 3: Results in the *PSR* domain.

	EX-BDD-BFS		EX-BDDA*		BFHSP	
	time	sol	time	sol	time	sol
1	0.24	9	0.35	9	0.05	9
2	0.42	13	0.94	13	0.08	13
3	1.84	11	0.45	11	0.16	11
4	31.44	17	0.76	17	3.08	17
5	223	15	6.96	15	119	15
6	-	-	12.70	20	-	-
7	-	-	174	21	-	-
8	-	-	293	26	-	-

Table 4: Results in the *Satellite* domain.

ACM, 31(9):1116–1127, 1988.

L. Arge. The I/O - complexity of ordered binary decision diagram manipulation. In *International Symposium on Algorithms and Computation (ISAAC)*, pages 82–91, 1995.

A. Biere, A. Cimatti, E. Clarke, and Y. Zhu. Symbolic model checking without BDDs. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, pages 193–207, 1999.

A. Biere. μ cke - efficient μ -calculus model checking. In *Computer-Aided Verification (CAV)*, pages 468–471, 1997.

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

R. E. Bryant. Graph based algorithms for boolean function manipulation. *IEEE Transaction on Computing*, C35(8):677–691, 1986.

A. Cimatti, E. Giunchiglia, F. Giunchiglia, and P. Traverso. Planning via model checking: A decision procedure for AR. In *European Conference on Planning (ECP)*, pages 130–142, 1997.

A. Cimatti, M. Roveri, and P. Traverso. Automatic OBDD-based generation of universal plans in non-deterministic domains. In *National Conference on Artificial Intelligence (AAAI)*, pages 875–881, 1998.

J. C. Culberson and J. Schaeffer. Pattern databases. *Computational Intelligence*, 14(4):318–334, 1998.

R. B. Dial. Shortest-path forest with topological ordering. *Communication of the ACM*, 12(11):632–633, 1969.

- S. Edelkamp and M. Helmert. Exhibiting knowledge in planning problems to minimize state encoding length. In *European Conference on Planning (ECP)*, pages 135–147, 1999.
- S. Edelkamp and M. Helmert. The model checking integrated planning system MIPS. *AI-Magazine*, pages 67–71, 2001.
- S. Edelkamp and F. Reffel. OBDDs in heuristic search. In *German Conference on Artificial Intelligence (KI)*, pages 81–92, 1998.
- S. Edelkamp, S. Jabbar, and S. Schroedl. External A*. In *German Conference on Artificial Intelligence (KI)*, pages 226–240, 2004.
- S. Edelkamp. Symbolic pattern databases in heuristic search planning. In *Artificial Intelligence Planning and Scheduling (AIPS)*, pages 274–283, 2002.
- R. Fikes and N. Nilsson. Strips: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2:189–208, 1971.
- E. A. Hansen, R. Zhou, and Z. Feng. Symbolic heuristic search using decision diagrams. In *Symposium on Abstraction, Reformulation and Approximation (SARA)*, pages 83–98, 2002.
- P. Haslum and H. Geffner. Admissible heuristics for optimal planning. In *Artificial Intelligence Planning and Scheduling (AIPS)*, pages 140–149, 2000.
- M. Helmert. A planning heuristic based on causal graph analysis. In *International Conference on Automated Planning and Scheduling (ICAPS)*, pages 161–170, 2004.
- R. C. Holte, M. B. Perez, R. M. Zimmer, and A. J. Donald. Hierarchical A*: Searching abstraction hierarchies. In *National Conference on Artificial Intelligence (AAAI)*, pages 530–535, 1996.
- A. J. Hu and D. L. Dill. Reducing BDD size by exploiting functional dependencies. In *Design Automation*, pages 266–271, 1993.
- S. Jabbar and S. Edelkamp. I/O efficient directed model checking. In *Conference on Verification, Model Checking and Abstract Interpretation (VMCAI)*, 2005. 313–329.
- R. M. Jensen, R. E. Bryant, and M. M. Veloso. SetA*: An efficient BDD-based heuristic search algorithm. In *National Conference on Artificial Intelligence (AAAI)*, pages 668–673, 2002.
- H. Kautz and B. Selman. Pushing the envelope: Planning propositional logic, and stochastic search. In *National Conference on Artificial Intelligence (AAAI)*, pages 1194–1201, 1996.
- R. E. Korf and A. Felner. Disjoint pattern database heuristics. In J. Schaeffer and H. J. van den Herik, editors, *Chips Challenging Champions*, pages 13–26. Elsevier Science, 2002.
- R. E. Korf. Finding optimal solutions to Rubik’s Cube using pattern databases. In *National Conference on Artificial Intelligence (AAAI)*, pages 700–705, 1997.
- R. E. Korf. Breadth-first frontier search with delayed duplicate detection. In *Workshop on Model Checking and Artificial Intelligence (MoChArt)*, pages 87–92, 2003.
- R. Korf. Best-first frontier search with delayed duplicate detection. In *National Conference on Artificial Intelligence (AAAI)*, 2004. 650–657.
- K. L. McMillan. *Symbolic Model Checking*. Kluwer Academic Press, 1993.
- S. Minato, N. Ishiura, and S. Yajima. Shared binary decision diagram with attributed edges for efficient boolean function manipulation. In *Design Automation Conference (DAC)*, pages 52–57, 1990.
- K. Munagala and A. Ranade. I/O-complexity of graph algorithms. In *Symposium on Discrete Algorithms (SODA)*, pages 87–88, 2001.
- K. Qian and A. Nymeyer. Heuristic search algorithms based on symbolic data structures. In *Australian Conference on Artificial Intelligence (ACAI)*, pages 966–979, 2003.
- K. Qian and A. Nymeyer. Guided invariant model checking based on abstraction and symbolic pattern databases. In *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, pages 497–511, 2004.
- F. Reffel and S. Edelkamp. Error detection with directed symbolic model checking. In *World Congress on Formal Methods (FM)*, pages 195–211, 1999.
- P. Sanders, U. Meyer, and J. F. Sibeyn. *Algorithms for Memory Hierarchies*. Springer, 2002.
- A. Santone. Heuristic search + local model checking in selective μ -calculus. *IEEE Transaction on Software Engineering*, 29(7), 2003.
- U. Stern and D. Dill. Using magnetic disk instead of main memory in the murphi verifier. In *International Conference on Computer Aided Verification (CAV)*, pages 172–183, 1998.
- I. Wegener. *Branching programs and binary decision diagrams - theory and applications*. SIAM, 2000.
- B. Yang, R. E. Bryant, D. R. O’Hallaron, A. Biere, O. Coudert, G. Janssen, R. K. Ranjan, and F. Somenzi. A performance study of BDD based model checking. In *Formal methods in Computer-Aided Design (FMCAD)*, pages 255–289, 1998.
- B. Yang, Y.-A. Chen, R. E. Bryant, and D. R. Hallaron. Space- and time-efficient BDD construction via working set control. In *Asia and South Pacific Design Automation (ASP-DAC)*, pages 423–432, 1998.
- R. Zhou and E. Hansen. Breadth-first heuristic search. In *International Conference on Automated Planning and Scheduling (ICAPS)*, pages 92–100, 2004.
- R. Zhou and E. Hansen. Structured duplicate detection in external-memory graph search. In *National Conference on Artificial Intelligence (AAAI)*, 2004. 683–689.