Experiments with Multiple Abstraction Heuristics in Symbolic Verification

Kairong Qian, Albert Nymeyer, and Steven Susanto

School of Computer Science & Engineering,
The University of New South Wales, Sydney Australia
{kairongq, anymeyer, ssus290}@cse.unsw.edu.au

Abstract. In this work we investigate a symbolic heuristic search algorithm in a model checker. The symbolic search algorithm is built on a system that manipulates binary decision diagrams (BDDs). We study the performance of the search algorithm in terms of the number of BDD operations, size of the BDDs, number of nodes they contain and runtime. We study the heuristic distribution of the state space, we measure effort by computing the *mean heuristic value*, and we compare single and multiple heuristics. In the case of multiple heuristics, we consider admissible and non-admissible merge strategies. We experiment on problems from a variety of domains. We find that multiple heuristics can perform significantly worse than single heuristics in symbolic search in at least one domain. In general, the effect of the heuristics on the symbolic search in the different domains varies markedly, and we conjecture that the different behaviour is caused by intrinsic differences in the characteristics of the state space.

1 Introduction

Formal verification techniques such as model checking [3] have gained much attention in the past decade. From the time Binary Decision Diagrams (BDDs) were introduced [2], symbolic model checking [16] that uses BDDs have been very successful in handling designs that have extremely large state spaces. While BDDs can represent the state space compactly, symbolic model checking of course still suffers the problem of "state space explosion" as it still must enumerate the full state space. This enumeration is typically done using a 'blind' breadth-first or depth-first search strategy. The blindness of the search is an unnecessary handicap that results in many irrelevant states being visited.

Heuristic-search algorithms such as A* and IDA* have been employed in AI research to solve many hard state-space search problems [13]. The big advantage of using a heuristic search strategy is that only part of the state space needs to be searched. Many verification of system design techniques, for example model checking, involve searches for a defect in a model. Coupling symbolic model checking with heuristic search techniques yields a more efficient technique to detect defects.

J.-D. Zucker and L. Saitta (Eds.): SARA 2005, LNAI 3607, pp. 290–304, 2005.

In [7, 9, 12, 17] traditional explicit-based heuristic search algorithms have been modified to use BDDs to represent the state space. These methods can enhance the "bug-hunting" capabilities of symbolic model checkers because of the action of the heuristics in guiding the search. In [6], heuristics are classified as *property-specific*, *structural* and *abstraction heuristics*. Property-specific heuristics can usually be derived from analysing the property that is being verified. Structural heuristics guide the search algorithm by taking into account the structure of the state space. These two heuristics often work well in explicit state model checking. The third class of heuristics are developed from the abstractions of the model that is being verified. In this class of heuristics, the abstraction is a 'relaxation' of the system, and is generated by removing complicating detail from the concrete model. Abstractions here are called "patterns", and the resulting heuristics "pattern databases".

A pattern database [4] stores the distance of a pattern to some sub-goal state. Pattern databases were originally developed to solve many hard combinatorial puzzles in AI, e.g. n^2-1 puzzles. In [5, 19, 6] this notion is extended and combined with data abstraction, which is another technique to reduce the size of the state space. In earlier work [19, 20], we have seemlessly integrated symbolic pattern databases and symbolic model-checking algorithms.

Research in pattern databases has been well studied in AI. Holte and Hernádvölgyi [11] studied the trade-off between time and space for memory based heuristics. Korf and Felner [14] used so-called disjoint pattern databases, and solved very challenging search problems like Rubick's cube. Multiple pattern databases have been comprehensively investigated in [10], as well as the relationship between the distribution of heuristic values and the search effort. Felner et al in [8] studied the generation of admissible heuristics by partitioning the problem into disjoint sub-problems. They use both static and dynamic schemes of partitioning. All this work only considered explicit-state search algorithms of course, and were only concerned with admissible heuristics, and only experimented on classical AI problems.

In this paper, we conduct experiments in heuristic- and BDD-based symbolic search in model checking for models from various domains. We seek to understand the effect that symbolic state enumeration brings to heuristic search, particularly with respect to each of the domains. We do this for both single-heuristic and multiple-heuristic search strategies. In the next section we briefly describe the framework we use for generating heuristics for a model checker. In Section 3 we lay the formal groundwork for the work. In Sections 4 to 8 we describe a series of experiments. We do an overall evaluation and draw conclusions in Section 9.

2 Abstraction-Guided Symbolic Model Checking

The technique we use is called the abstraction-guided symbolic model checking framework. In our previous work [19, 18, 20], we have used a single abstraction to guide the BDD-based symbolic search. In this work we extend this approach and consider multiple abstractions. We briefly describe the general approach here and

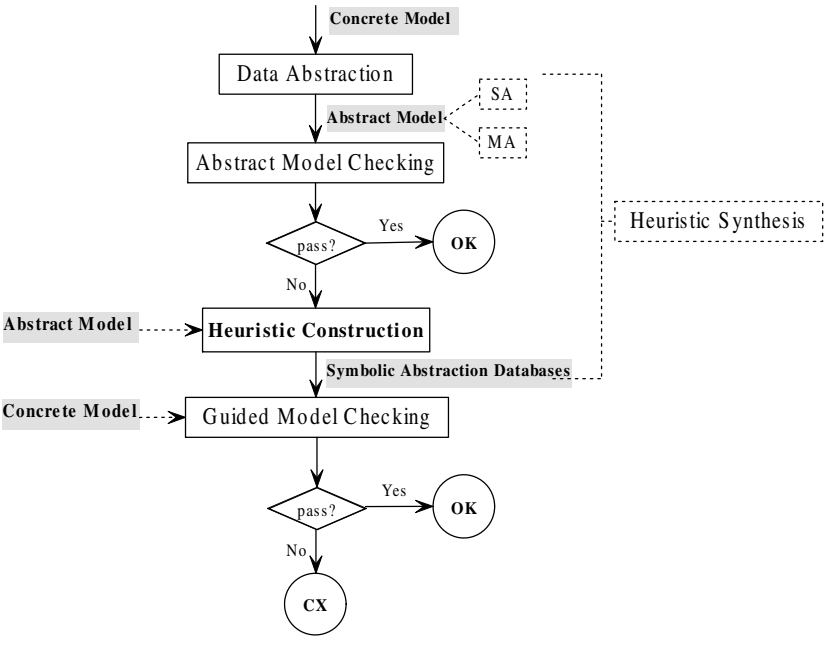

Fig. 1. The abstraction-guided model-checking framework

interested readers may refer to [19] or [20] for details. The abstraction-guided framework is depicted in Figure 1.

The process starts with the design, which we refer to as the concrete model. In the first step we generate a data abstraction of the concrete model. Note that in this step, we can generate more than one abstraction for the concrete model. We refer to a single abstraction as SA and multiple abstractions as MA in Figure 1. The abstract model(s) are taken as input by a symbolic model checker. If the model checker verifies the abstract model(s), we terminate as the data abstraction guarantees the soundness of the properties we are interested in. If the abstract model(s) fail the verification, we construct abstraction heuristic(s) using the abstract model(s). The guided model-checking algorithm is then invoked to check the concrete system using this heuristic as guide. The outcome of the heuristic model checker is either that the concrete model is verified, or a counter-example (*CX* in the figure) that will reveal the defect in the design (assuming the algorithm terminates of course).

Note that unlike other research in model checking, we use the same abstraction to (1) reduce the size of the model and (2) to guide the heuristic search algorithm. We have implemented this approach in a tool, called Golfer. This tool is built on top of the well-known symbolic model checker NuSMV[1].

[1] http://nusmv.irst.itc.it/

The heuristics that we construct from abstractions extend the notion of "pattern databases" developed in [4]. As our representation of the problem is based on BDDs, following [5, 6], we refer to these heuristics as *symbolic abstraction databases* (SADBs).

3 Symbolic Abstraction Databases

The terminology used in heuristic search in AI and in verification is quite different. In this section we define the notation we use, and explain our approach. We model AI search problems and the verification of safety properties using a finite-state model as follows.

Definition 1 (Finite Transition System). *A finite state transition system is a 4-tuple $M = (S, S_0, R, G)$, where*

- *S is a finite set of states*
- *$S_0 \subseteq S$ is a set of initial states*
- *$R \subseteq S \times S$ is a transition relation (or operator) that determines a set of successors for a given state $s \in S$*
- *$G \subseteq S$ is the set of goal states*

Definition 2 (Solution Path). *A path in a finite transition system (S, S_0, R, G), denoted by π, is a sequence of states $s_0, s_1, \ldots, s_n$ where $s_n \in G$ and for all $0 \leq i < n$, $s_i \in S \wedge (s_i, s_{i+1}) \in R$. If a path is a solution path then $s_0 \in S_0$. The length of π, written $|\pi|$, is just the number of states in the path.*

In verification, a solution path is called a *counter-example* as it demonstrates why the property that is being verified is not true.

Since we are only interested in symbolic heuristic search in this work, we encode M using Boolean expressions. Given a transition system $M = (S, S_0, R, G)$, we use a set of Boolean variables $X = \{x_1, x_2, \ldots, x_k\}$ to model the state space of M. A state can be represented by a truth assignment vector of X and all possible truth assignment vectors comprise the state space S. The Boolean functions $\mathcal{S}_0(x_1, x_2, \ldots, x_k)$ and $\mathcal{G}(x_1, x_2, \ldots, x_k)$ are characteristic functions that represent the states in S_0 and G (resp.). To encode R we need another set of Boolean variables $X' = (x'_1, x'_2, \ldots, x'_k)$ to represent the next state of a state s. Likewise, $\mathcal{R}(x_1, x_2, \ldots, x_k, x'_1, x'_2, \ldots, x'_k)$ is the characteristic function for R. In the discussion henceforth, we use $\mathcal{M} = (\mathcal{S}, \mathcal{S}_0, \mathcal{R}, \mathcal{G})$ to refer to the Boolean encoding of a transition system M.

If a system is modelled using a set of Boolean variables $X = \{x_1, x_2, \ldots, x_k\}$, we call $X_p \subset X$ a *pattern set*, and call those variables that are not in the pattern set $X_{\overline{p}}$. Given a pattern set and Boolean encoding of a transition system $\mathcal{M} = (\mathcal{S}, \mathcal{S}_0, \mathcal{R}, \mathcal{G})$, we can abstract the transition system as follows.

Definition 3 (Abstraction). *The abstraction of $\mathcal{M} = (\mathcal{S}, \mathcal{S}_0, \mathcal{R}, \mathcal{G})$ w.r.t a pattern set X_p is also a transition system $\hat{\mathcal{M}} = (\hat{\mathcal{S}}, \hat{\mathcal{S}}_0, \hat{\mathcal{R}}, \hat{\mathcal{G}})$ represented by its Boolean encodings, where*

- $\hat{\mathcal{S}}$ *is a disjunction of all minterms of variables in* X_p
- $\hat{\mathcal{S}}_0 \equiv \exists X_{\overline{p}} \mathcal{S}_0(x_1, x_2, \ldots, x_k)$
- $\hat{\mathcal{R}} \equiv \exists X_{\overline{p}} X'_{\overline{p}} \mathcal{R}(x_1, x_2, \ldots, x_k, x'_1, x'_2, \ldots, x'_k)$
- $\hat{\mathcal{G}} \equiv \exists X_{\overline{p}} \mathcal{G}(x_1, x_2, \ldots, x_k)$

Definition 4 (Symbolic Abstraction Databases). *Given a Boolean encoding* $\hat{\mathcal{M}} = (\hat{\mathcal{S}}, \hat{\mathcal{S}}_0, \hat{\mathcal{R}}, \hat{\mathcal{G}})$ *of a transition system, we call a set* $\sigma = \{(B_0, 0), (B_1, 1), \ldots, (B_n, n)\}$ *symbolic abstraction database such that:*

- (B_i, i)*, where* B_i *is a Boolean characteristic function and* $i \geq 0$
- $B_0 \equiv \hat{\mathcal{G}}$ *and* $B_i \equiv \exists X'_p (B_{i-1}[X_p / X'_p] \wedge \hat{\mathcal{R}})$ *for all* $0 < i \leq n$
- $B_n \wedge \hat{\mathcal{S}}_0 \not\equiv False$ *and* $B_i \wedge \hat{\mathcal{S}}_0 \equiv False$ *for all* $0 \leq i < n$
- $B_i \wedge B_l = \emptyset$ *for all* $i \neq j$

The length of a symbolic abstraction database is $|\sigma|$.

It is proved in [5, 19] if there is a solution path π in a concrete system M, there must exist a corresponding abstract solution path $\hat{\pi}$ in the abstraction $\hat{M}$ and $|\hat{\pi}| \leq |\pi|$. Note that the existence of $\hat{\pi}$ corresponding to π provides the theoretical justification for using $\hat{\pi}$ to guide the search. The lower-bound characteristic of the abstract solution path allows the symbolic abstraction database to be used as an admissible heuristic to estimate the actual number of transitions (distance) between the current and goal states in the concrete system.

Definition 5 (Disjoint SADBs). *Two symbolic abstraction databases* σ_1 *and* σ_2 *are disjoint if their corresponding pattern sets* X_{p1} *and* X_{p2} *are disjoint, i.e.* $X_{p1} \cap X_{p2} = \phi$.

Given a state s and characteristic function $\mathcal{F}_s$ in the concrete model and a symbolic abstraction database $\sigma = \{(B_0, 0), (B_1, 1), \ldots, (B_n, n)\}$, if $\mathcal{F}_s \wedge B_j \neq False$ for some pair (B_j, j), then we call the value j an *estimator* and denote it as $\sigma(s)$. Given a symbolic abstraction database, this estimator is unique as all B_i are disjoint.

Theorem 1 (Additiveness). *Let* $|\pi_s|$ *be the path length of* s *in* M*, and* σ_1 *and* σ_2 *be two disjoint symbolic abstraction databases. Let* $\sigma_1(s)$ *and* $\sigma_2(s)$ *be the estimator of* s *in each of them. Then* $\sigma_1(s) + \sigma_2(s) \leq |\pi_s|$.

This theorem guarantees that disjoint SADBs can be "added" together and the result will still be an admissible heuristic. In this work we in fact also consider pattern sets that are not disjoint. In verification, admissibility is less of an issue as we are more interested in finding just a good path to a defect in the system, not necessarily the shortest path. Our formal discussion about the disjoint SADBs actually shares many aspects with the disjoint pattern database heuristics in AI research [5, 14, 8].

We not only treat the system symbolically in this work, we also encode the guided heuristic search symbolically. In [19, 20] we describe how SADBs are

used by a symbolic model checker to detect safety violations. We further generalise the algorithm to use multiple SADBs. The search algorithm still works as in [19]; the only difference being the way SADBs are queried. The essence of the guided symbolic model checking algorithm is that each frontier BDD is split into several smaller, so-called sub-BDDs (representing *sub-frontiers*), where each of these sub-BDDs corresponds to a different heuristic value. Note that this splitting in necessary in the A* algorithm as the heuristics help drive the search. Edelkamp [7], who also studied the symbolic A* algorithm, used a different method however. In his method, the heuristic values are encoded into the BDD directly. It is not clear whether this method has any advantages over our method however. The heuristics in our research therefore fulfil two tasks: as a search guide and as a mechanism to split BDDs. The splitting is carried by the *restrict* operation (denoted as $\downarrow$) on BDDs [2].

Let $\mathcal{D}$ be a BDD representing a set of states of M and $\Phi = \{\sigma_1, \ldots, \sigma_m\}$ be a set of SADBs. The algorithm below splits the BDD representing $\mathcal{D}$ and assigns each sub-BDD an estimator according to the merge strategy of the SADBs.

Procedure Splitting $(\mathcal{D}, \Phi, m\_strategy)$
1 $result \leftarrow \{(\mathcal{D}, 0)\}$
2 **for** i in $1..m$ **do**
3 $temp \leftarrow result$
4 $result \leftarrow \{\}$
5 **for** each $(d, h) \in temp$ **do**
6 **for** each $(B_j, j) \in \sigma_i$ **do**
7 $I \leftarrow d \downarrow B_j$
8 $d \leftarrow d \wedge \overline{I}$
9 **if** $(I \neq \phi$ & $m\_strategy = add)$
10 $result \leftarrow result \cup \{(I, j + h)\}$
11 **if** $(I \neq \phi$ & $m\_strategy = max)$
12 $result \leftarrow result \cup \{(I, max(j, h))\}$
13 **if** $(d \neq \phi)$
14 $result \leftarrow result \cup \{(d, |\Phi|)\}$
15 **return** $result$

The input *m_strategy* is the merge strategy used on the SADBs in the symbolic heuristic search. The only possible values are *add* and *max*. Note that for disjoint SADBs, both are admissible. If Φ is not disjoint, only *max* will be admissible.

While abstraction-based heuristics can reduce the search space in both the AI and verification domains, the method used to derive abstractions in these domains is very different. For example, puzzles and planning problems in the AI domain can usually be physically modelled, so the abstraction of the problem often involves the detection of physical patterns. In the n^2-1 puzzle, for example, we have *corner* and *fringe* patterns [4]. In verification, however, problems are generally modelled by a large number of variables and the physical relationship between the variables is neither physical nor obvious, often due to the high level of concurrency of the model. Finding patterns in verification can be very difficult indeed.

In [20] we presented a procedure that automatically finds abstraction patterns by using a data dependency analysis. The idea is based on the notion that variables that are only indirectly related have a weaker influence on each other than variables that are directly related. We build a variable dependency tree rooted by the variables that occur in the goal state. The weakest variables, which appear furtherest away from the root in the variable dependency graph, are ignored. Note that this method does not always work well, particularly in AI problems, where typically all variables are directly dependent on each other.

4 Experiment Set-Up

In explicit-state heuristic search, the number of states (or nodes) that are generated by the search algorithm can be used to evaluate the effectiveness of the heuristic. In BDD-based heuristic search, however, we cannot use the number of states that are generated by the algorithm as states are symbolically represented by Boolean functions. We note that the number of nodes in a BDD is not related to the number of states it represents. In fact, the effort that a symbolic search algorithm must make to solve a problem is largely determined by the internal operations of the BDD engine, not the number of states in the system.

The following attributes will be used to capture the search effort in our experiments.

IM. The number of BDD image computations. These computations determine the successor states of the search. It is called the relational product and involves quantifier elimination, which is an expensive computation in symbolic model checking. (IM is related to the size of the *closed* set in the explicit-state A* algorithm.)

SP. The number of splitting operations. The splitting operation involves the 'restrict' operation on BDDs. It is also expensive. (SP is related to the size of the *open* set in explicit-state A* algorithm.)

ND. The total number of BDD nodes allocated. While the number of BDD nodes is not directly related to the number of states, it still reflects the memory usage of the algorithm and hence is the major memory measurement.

AS. The average size of all BDDs. Reducing the size of BDDs is important because some BDD operations have exponential time complexity in terms of the BDD size.

TM. The CPU time consumed by the search algorithm. In general the CPU time is strongly related to the values of IM and SP.

We have used 5 models in our experiments: two of them are puzzles from the AI heuristic search domain, and the other three are real-world design models of concurrent systems. Note that all models have at least one goal state that is reachable from the initial state set.

Name	Description	Type
puz	$n^2 - 1$ sliding tile puzzle ($N = 8$)	puzzle
perm	N-pancake puzzle ($N = 10$)	puzzle
dme	distributed mutual exclusive ring	circuit
ns	Needham-Schroeder public key protocol	protocol
peter	Peterson's mutual exclusion algorithm	protocol

To construct SADBs we need to define the pattern set that is to be used for the abstraction. For each puzzle model we use 4 different pattern sets that are commonly used in AI heuristic search literature. For each of the verification models, we use our data dependency analysis to also generate 4 pattern sets. The pattern sets for each model are not necessarily disjoint as the optimality of the solution path is not a primary concern in our work.

We ran the model checker for each of the (single) SADBs generated by the pattern sets. We also used multiple SADBs that were constructed by merging 3 of the 4 SADBs for each model. We in fact constructed multiple SADBs by all of the $\mathbf{C}_4^3$ combinations of SADBs and reported the best performance. As well, both the *add* and *max* merge strategies were studied.

Caveat: At this point, we should point out that symbolic search algorithms are not really suitable for solving the puzzle-like problems that often occur in AI heuristic search, where good heuristics are known. The advantage of the BDDs in manipulating sets of states in single operations is often outweighed by the computational complexity of these BDD operations [17]. In essence, only when the BDDs represent large sets of states does their use pay off. In general, for these types of problems, explicit-state searches are often faster and require less memory. We use these models in this work, however, for comparison purposes.

5 Heuristic Distribution Experiment

The aim of this experiment was to study the distribution of the heuristic over the state space. In Figure 2, we show the number of states in the SADB for different heuristic values. In our symbolic approach, all states with the same heuristic value is represented by a single BDD. To compute these results, we needed to calculate how many (abstract) states a BDD can represent (which is of course different to the number of nodes in the BDD). Note the logarithm scale on the axis for the number of states. The diagram on the left is taken from randomly chosen SADBs for the 8-puzzle model, and on the right, for the DME circuit model.

In the figure we observe that different abstraction show similar behaviour for each model. Comparing AI and verification, however, we observe that the number of states *increases* exponentially as the heuristic value increases in the case of the 8-puzzle, but *decreases* exponentially for the DME model.

We conjecture that this phenomenon is caused by fundamental differences in the nature of the state spaces in these domains. Puzzles, for example, often have few goal states, whereas safety properties in verification can be violated

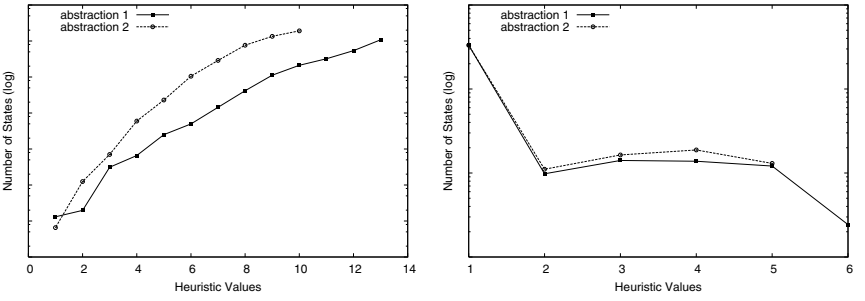

Fig. 2. 8-puzzle (left) and DME (right)

in many states, and hence there are many goal states. Having many goal states means that the value of the heuristic tends to remain small. When we construct SADBs, we use a backward breath-first traversal of the abstract model's state space. In the case of puzzles, the resulting search tree grows exponentially. In verification, the size of the search tree decreases as many states share the same predecessor. A conclusion one could draw from the behaviour we observe is that heuristic search using SADBs has less to gain in verification models than in AI models (in other words, the improvement over blind search will be less in verification).

6 Mean Heuristic Value Experiment

In this experiment we wished to study whether or not the *mean heuristic value* [15, 5], is a good predictor of the effort needed by the symbolic heuristic search. In explicit-state heuristic search, it is well known to be a good predictor of the search effort. The intuition is that it may not be in a symbolic setting because it does not factor in the added computational overhead that BDDs have.

The mean heuristic value (MHV) $\overline{h}$ of a SADB σ determines the overall distribution of heuristic values and is defined as follows.

$$\overline{h} = \sum_{i=1}^{|\sigma|} i \times (|\{s \in \sigma | \sigma(s) = i\}|/|\{s | s \in \sigma\}|) \tag{1}$$

Equation 1 actually computes the weighted mean heuristic value of σ. A high value for $\overline{h}$ indicates that a larger proportion of states have large heuristic values than have low values, and therefore the level of "informedness" is high. Ideally, the value of $\overline{h}$ for a SADB σ should be as close as possible to $|\sigma|$, and in that case it is said to be "well-informed".

We chose three models, and for each model we use 4 different SADBs. The results are shown below.

SADBs	MHV	IM	SP	ND	AS	TM
puz-1	**11.90**	**144**	**305**	**268421**	**361**	**1.050**
puz-2	12.30	679	1581	547046	404	3.390
puz-3	9.03	495	1043	453495	455	2.970
puz-4	8.30	1353	2769	400189	429	6.900
dme-1	2.73	616	1121	193004	2921	74.110
dme-2	**3.72**	**26**	**26**	**280683**	**3232**	**3.220**
dme-3	3.68	26	26	289808	6268	11.090
dme-4	3.57	26	26	524103	12394	45.540
ns-1	**8.13**	**28**	**106**	**92126**	**815**	**6.840**
ns-2	5.37	421	1693	371040	964	122.420
ns-3	6.16	1110	6361	629380	1843	389.670
ns-4	5.34	475	2073	519868	1874	228.980

The statistics that concern the BDDs come from the BDD engine of the model checker. Other statistics are generated from profilers. For each of the three models we give the results for each of the 4 SADBs. The value of MHV is calculated using equation 1. For each model, we bold the row that has the shortest run-time (TM).

For the puzzle model, the bolded row has the best performance in terms of every attribute. While the MHV for *puz-1* is not the highest one, the MHV is nevertheless a good predictor of performance.

For the two verification models, the SADB with the shortest run-time has the highest value of MHV. It is hence a very good predictor in the symbolic setting, contrary to our intuition.

Unrelated to the MHV, we observe that *dme-2*, *dme-3* and *dme-4* use exactly the same number of image computations and splitting operations (IM and SP). However, in spite of the fact that *dme-2* uses more BDD nodes (ND) than *dme-1*, and will hence use more memory, it is still the best performer.

7 Multiple SADB Experiment

This experiment concerns the main focus of this work, and that is compare the performance of single and multiple SADBs. Holte et al. [10] found that heuristic search that is based on multiple SADBs out-performs search based on single SADBs (for the same amount of memory). But Holte et al's work is based on an explicit-state search. So does it apply to a symbolic heuristic search as well?

We run each model with each of the 4 single SADBs. From these 4 SADBs, we created 10 multiple SADBs ($\mathbf{C}_4^2$ from combinations of 2 SADBs plus $\mathbf{C}_4^3$ from 3 SADBs). We ran the symbolic search on each model with each of these 10 heuristics. We show the results for just two of the models in the following table.

SADB(s)	IM	SP	ND	AD	TM
puz-sgl	144	305	268421	361	1.050
puz-mpl-1	133	284	353075	341	1.360
puz-mpl-2	203	440	409425	343	1.650
puz-mpl-3	160	349	299479	361	1.200
peter-sgl	40	97	109795	1332	4.414
peter-mpl-1	230	591	379365	816	14.273
peter-mpl-2	40	105	134202	822	16.442
peter-mpl-3	40	100	126754	716	16.357

In the table, the results for the single SADB (denoted by the *.sgl* suffix) are the ones that had the best performance. For the 10 multiple SADBs, we show just the best performing 3, and these are indicated by the *.mpl* suffix. We used both the *add* and *max* strategies to merge the SADBs.

Contrary to Holte et al's findings, the results in the table above show that multiple heuristics perform worse than single heuristics. The first two rows in the table, *puz-sgl* and *puz-mpl-1*, are particularly interesting as they reveal that even when the multiple SADB search uses less image computations and splitting operations (which we noted earlier are the primary determinants of the computational complexity), it performs worse than the single SADB search. We note the multiple SADB models use more BDDs nodes, and the average size of the BDDs is smaller, in both models.

These results are quite surprising. Multiple pattern databases in explicit-state heuristic search are effective because they improve the overall heuristic distribution and hence result in smaller search trees. In symbolic heuristic search, the heuristic is (also) used to split the frontier BDDs, and one conjectures, it is this computation that causes the problem. Thus, much of the effort of symbolic heuristic search is spent on splitting the BDDs, offsetting any gains that may be had from the higher-quality, multiple heuristic.

8 Merge Strategy Experiment

In our final experiment we compared the performance of the *add* and *max* merge strategies for multiple SADBs. We carried out this experiment by using both strategies to merge both 3 and 4 single SADBs into a multiple SADB. We note that the *add* strategy is not admissible so it can generate non-optimal paths.

Unlike our earlier experiments, this time we present tables for each of the problem domains separately. The domains are AI puzzles, electronic circuits and communication/security protocols. We do this because we found that the choice of merge strategy effected the performance in a different way for each of these domains. The only change in the table format to our earlier experiments is the addition of the attribute LE, which indicates the solution length returned by the search. You can see from this column when a search generated a non-optimal path.

AI Puzzles. In the table below we see the results for the AI puzzles.

SADBs	IM	SP	ND	TM	LE
puz-3-add	133	284	368552	1.450	24
puz-3-max	203	440	409425	1.650	24
puz-4-add	168	357	353075	1.360	24
puz-4-max	169	391	433784	1.880	24
perm-3-add	27	106	93224	5.720	13
perm-3-max	769	3446	1199309	294.340	11
perm-4-add	289	1833	237584	39.510	15
perm-4-max	2619	14427	859840	511.550	11

We observe that, while the *add* merge strategy can result in a non-optimal path, the resulting search is faster than that produced by the (optimal) *max* merge strategy. In fact, in the case of *perm*, it is one or two orders of magnitude faster. Note that there is almost the same difference between the *max* and *add* strategies in the number of image computations and partition operations, so the result is not surprising. There is a trade-off here: speed comes at the cost of optimality.

Electronic Circuit. This model has been constructed from a real electronic circuit design and has been a widely used benchmark for symbolic model checking.

SADBs	IM	SP	ND	TM	LE
dme-3-add	169	232	142574	10.520	37
dme-3-max	616	1122	186500	77.380	27
dme-4-add	169	232	171070	10.260	41
dme-4-max	326	546	198282	16.220	27

We observe that the *add* strategy clearly results in a faster model checker than *max* but at the cost of a much longer path to a goal state.

Communication/Security Protocols. The two communication protocol models generate quite different results.

SADBs	IM	SP	ND	TM	LE
ns-3-add	1452	11831	218287	187.460	19
ns-3-max	32	273	113118	7.040	14
ns-4-add	1954	14800	318377	228.680	19
ns-4-max	130	1054	130216	12.580	14
peter-3-add	230	591	379365	14.273	49
peter-3-max	40	100	130955	2.970	41
peter-4-add	79	199	151200	25.890	49
peter-4-max	40	105	134202	16.442	41

Quite the opposite of the previous results, the *add* strategy for these models results in a model checker that takes a lot longer to find a longer, non-optimal path to a goal state. Clearly an unsatisfactory heuristic for this class of model.

In summary, the inadmissibility of the *add* merge strategy may lead to (very) sub-optimal paths, and a substantial speed-up in the search in some models, but a worsening in others.

9 Evaluation and Conclusion

Predicting how and when BDD-based heuristic search algorithms will perform better than explicit-state algorithms is extremely difficult. It is well known, for example, that finding an optimal variable ordering for BDDs is an NP-hard problem [1]. We have not considered the variable ordering in this work yet (but have in earlier work [20]). BDDs can be 'exponentially' efficient in representing very large sets of states, and because of this, can be vastly superior to explicit-state search algorithms. However, when the sizes of the sets they represent are not large, the

computational overhead of manipulating BDDs can result in very poor performance indeed. The problem of predicting performance is compounded when you add heuristics, and compounded again when you allow multiple heuristics. So the problem we are addressing is indeed very difficult.

In AI, finding the shortest path to the goal state is paramount. In verification, finding a 'reasonably short' path is often sufficient. More important is the time it takes to find this path. The reason for this is that the model checker is being used as a debugger, and hence we need to know quickly whether there is an error in the specification or not. In verification therefore, we are often prepared to sacrifice optimality for speed.

While we have tried to be comprehensive in the experiments, we do of course:

- have only a small sample of models,
- have just a few abstractions (derived automatically for the verification models)
- have just 2 merge strategies: one admissible, one non-admissible.

On the positive side, we have attempted to bridge disparate fields, AI and verification, by understanding the behaviour of a technology, symbolic heuristic search, that is common to both. We can summarise the results of our experiments in the following way:

- The distribution of the heuristic over the state space is different for AI models than verification models. This difference could be caused by different characteristics of the state space: for example, there are typically more goal states in verification than in AI problems, and verification state spaces are less tree-like.
- The MHV still makes a good predictor of effort in symbolic heuristic search.
- Contrary to Holte et al. [10], we found that multiple symbolic heuristics performed worse than single symbolic heuristics. We conjecture that this is caused by the overhead of splitting the BDDs. Note that in some cases splitting a BDD results in larger BDDs than the original. This is an unfortunate side-effect of this method that cannot easily be avoided.
- If you have a naturally good heuristic distribution, as AI problems tend to have, then an 'aggressive', non-optimal merge strategy like *add* will result in multiple SADBs that perform much better than single SADBs; albeit at the possible cost of optimality.
- Verification problems that have poorly, or narrowly distributed heuristics should not use non-optimal merge strategies.

AI puzzles and electronic circuits typically have very dense state spaces, while protocol models have relatively sparse state spaces. Intuitively, dense state spaces will contain a larger number of solution paths than sparse state spaces. This could be the cause of the behaviour we observe in the merge-strategy experiment. A non-optimal strategy like *add* enables heuristic search algorithms like A* and IDA* to guide aggressively during the search because it increases the proportion of states that have larger heuristic values, and penetrates deeply into the state

space. Consequently, however, the search may miss shallow solutions and fruitlessly pursue dead-end paths. Note that AI puzzles and circuits have relatively fewer goal states than protocol models. We conjectured in Section 5 that this was the cause of the behaviour that we observed in the heuristic-distribution experiment. The topology of the state space is therefore potentially very important in determining the performance of the symbolic search.

The future work we are planning is the following:

- Take the BDD variable ordering into account.
- More work needs to be done to determine how to abstract the system automatically. This is of course an open research question. Currently our approach using a data dependency analysis is simplistic.
- To restrict the sizes of BDDs, we need to consider more effective mechanisms such as "high density" reachability analyses [21]. This is especially important for splitting the frontier BDDs,
- We need to understand which characteristics of the state space are important for the performance of the guided and symbolic approach. While we have tried to do this by considering models from different domains, more focussed experiments that shed light on this issue are needed. It would appear that you need to know what the topology of the state space is before deciding which search algorithm to apply.

References

1. B. Bollig and I. Wegener. Improving the variable ordering of obdds is np-complete. *IEEE Trans. Computers*, 45(9):993–1002, 1996.
2. R. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Transaction*, C-35(8):677–691, Aug 1986.
3. E. M. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999.
4. J. C. Culberson and J. Schaeffer. Searching with pattern databases. In *Proc. of the 11th Biennial Conf. of the Canadian Society for Computational Studies of Intelligence, Toronto, Ontario, Canada, May 21-24*, volume 1081 of *LNCS*, pages 402–416. Springer-Verlag, 1996.
5. S. Edelkamp. Symbolic pattern databases in heuristic search planning. In *Proc. of the Sixth Int. Conf. on Artificial Intelligence Planning Systems, April 23-27, Toulouse, France*, pages 274–283. AAAI, 2002.
6. S. Edelkamp and A. Lluch-Lafuente. Abstraction databases in theory and model checking practice. In *Proc. of Workshop on Connecting Planning Theory with Practice, Int. Conf. on Automated Planning and Scheduling, ICAPS, Whistler, Canada*, 2004.
7. S. Edelkamp and F. Reffel. OBDDs in heuristic search. In *KI-98: Advances in Artificial Intelligence, 22nd Annual German Conf. on Artificial Intelligence, Bremen, Germany, September 15-17, Proc.*, volume 1504 of *LNCS*, pages 81–92. Springer-Verlag, 1998.
8. A. Felner, R. E. Korf, and S. Hanan. Additive pattern database heuristics. *J. Artif. Intell. Res. (JAIR)*, 22:279–318, 2004.

9. E. Hansen, R. Zhou, and Z. Feng. Symbolic heuristic search using decision diagrams. In *Proc. of the Symp. on Abstraction, Reformulation and Approximation, Alberta, Canada*, volume 2371 of *LNCS*, pages 83–98. Springer-Verlag, 2002.
10. R. Holte, J. Newton, A. Felner, R. Meshulam, and D. Furcy. Multiple pattern databases. In *Proc. of 14th Intl. Conf. on Automated Planning & Scheduling (ICAPS)*, pages 122–131. AAAI, 2004.
11. R. C. Holte and I. T. Hernádvölgyi. A space-time tradeoff for memory-based heuristics. In *AAAI/IAAI*, pages 704–709, 1999.
12. R. M. Jensen, R. E. Bryant, and M. M. Veloso. Seta*: An efficient bdd-based heuristic search algorithm. In *Proc. of the Eighteenth National Conf. on Artificial Intelligence and 14th Conf. on Innovative Applications of Artificial Intelligence, Alberta, Canada*, pages 668–673. AAAI Press, 2002.
13. R. Korf. Finding optimal solutions to to Rubik's cube using pattern databases. In *Proc. of the 14th National Conf. on Artificial Intelligence and 9th Innovative Applications of Artificial Intelligence Conf., July 27-31, Providence, Rhode Island*, pages 700–705. AAAI Press/The MIT Press, 1997.
14. R. E. Korf and A. Felner. Disjoint pattern database heuristics. *Artif. Intell.*, 134(1-2):9–22, 2002.
15. R. E. Korf, M. Reid, and S. Edelkamp. Time complexity of iterative-deepening-a*. *Artif. Intell.*, 129(1-2):199–218, 2001.
16. K. McMillan. *Symbolic model checking.* Kluwer Academic Publishers, Boston, MA, 1993.
17. A. Nymeyer and K. Qian. Heuristic search algorithm based on symbolic data structures. In *Proc. of the 16th Australian Joint Conf. in Artificial Intelligence, Perth, Australia, 3-5 December*, volume 2903 of *LNAI*, pages 966–979. Springer-Verlag, 2003.
18. K. Qian and A. Nymeyer. Abstraction-based model checking using heuristical refinement. In *Proc. of the 2nd Int. Symp. on Automated Technology for Verification and Analysis, ATVA'04*, pages 165–178. Springer-Verlag, 2004.
19. K. Qian and A. Nymeyer. Guided invariant model checking based on abstraction and symbolic pattern databases. In *Proc. of the 10th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems, Barcelona, Spain,*, volume 2988 of *LNCS*, pages 497–511. Springer-Verlag, 2004.
20. K. Qian and A. Nymeyer. Abstraction-guided model checking using symbolic IDA* and heuristic synthesis. In *Submitted to FORTE'05 for publication*, 2005.
21. K. Ravi and F. Somenzi. High-density reachability analysis. In *ICCAD '95: Proc. of the 1995 IEEE/ACM international conference on Computer-aided design*, pages 154–158. IEEE Computer Society, 1995.