9-1-2001

Using Combinatorial Optimization Methods for Quantification Scheduling

P Chauhan
Carnegie Mellon University

Edmund M. Clarke
Carnegie Mellon University, emc@cs.cmu.edu

S Jha
University of Wisconsin - Madison

J Kukula
Synopsys, Inc

H Veith
TU Vienna

See next page for additional authors

Using Combinatorial Optimization Methods for Quantification Scheduling*

P. Chauhan[1] E. Clarke[1] S. Jha[2] J. Kukula[3] H. Veith[4] D. Wang[1]

[1] Carnegie Mellon University [2] University of Wisconsin - Madison
[3] Synopsys Inc. [4] TU Vienna, Austria

Abstract. Model checking is the process of verifying whether a model of a concurrent system satisfies a specified temporal property. Symbolic algorithms based on Binary Decision Diagrams (BDDs) have significantly increased the size of the models that can be verified. The main problem in symbolic model checking is the *image computation problem*, i.e., efficiently computing the successors or predecessors of a set of states. This paper is an in-depth study of the image computation problem. We analyze and evaluate several new heuristics, metrics, and algorithms for this problem. The algorithms use combinatorial optimization techniques such as *hill climbing*, *simulated annealing*, and *ordering by recursive partitioning* to obtain better results than was previously the case. Theoretical analysis and systematic experimentation are used to evaluate the algorithms.

1 Introduction

Model Checking and State Explosion. In model checking [CGP00], the system to be verified is represented as a finite *Kripke structure* or *labelled transition system.* A Kripke structure over a set of atomic propositions AP is a tuple $K = (S, R, L, I)$ where S is the set of states, $R \subseteq S \times S$ is the set of transitions, $I \subseteq S$ is the non-empty set of initial states, and $L : S \rightarrow 2^{AP}$ labels each state by a set of atomic propositions.

Given a Kripke structure $K = (S, R, I, L)$ and a specification ϕ in a temporal logic such as CTL, the *model checking problem* is the problem of finding all states s such that $K, s \models \phi$ and checking if the initial states are among these. Model checking algorithms usually exploit the fact that temporal operators can be characterized as $\mu-$calculus terms. For example, the set of states where the CTL formula $\mathbf{EF}\phi$ holds, is given by

$$\mathbf{EF}\phi \equiv \mu S.\phi \vee \mathbf{EX}S.$$

Recall that $\mathbf{EF}\phi$ expresses *reachability*, i.e., the existence of a path where ϕ eventually holds. Such fixpoint translations directly correspond to iterative algorithms.

* This research is sponsored by the Semiconductor Research Corporation (SRC), the Gigascale Research Center (GSRC), the National Science Foundation (NSF) under Grant No. CCR-9505472, and the Max Kade Foundation. One of the authors is also supported by Austrian Science Fund Project N Z29-INF. Any opinions, findings and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of GSRC, NSF, or the United States Government.

Symbolic Verification. In practice, the systems to be verified are described by programs in finite state languages such as SMV or VERILOG. These programs are then compiled into equivalent Kripke structures. The main practical problem in model checking is the *state explosion problem*: the size of the *state space* of the system is *exponential* in the size of its description. Therefore, even for systems of relatively modest size, it is often impossible to compute their state space explicitly.

In *symbolic verification*, the transition relation of the Kripke structure is not explicitly constructed, but instead a Boolean function representing the transition relation is computed. Sets of states are also represented as Boolean functions. Then, the fixpoint algorithms are applied to the formulas rather than to the Kripke structure. Since the Boolean formula is usually exponentially smaller than an explicit representation, symbolic verification is often able to alleviate the state explosion problem in these situations. Binary Decision Diagrams (BDDs) have been a particularly useful data structure for representing Boolean functions; in addition to their relative succinctness they provide a *canonical* representation for Boolean functions. As a result, equality of two Boolean functions can be easily checked in this representation.

At the core of all symbolic algorithms is *image computation*,[1] i.e., the task of computing the set of successors $\mathbf{Img}(S)$ of a set of states S, where

$$\mathbf{Img}(S) := \{s' : \exists s. R(s, s') \land s \in S\}.$$

Image computation is one of the major bottlenecks in verification. Often it is impossible to construct a single BDD for the transition relation R. Instead, R is represented as a *partitioned transition relation*, i.e., as the conjunction of several BDDs, each representing a piece of R. The problem is how to compute $\mathbf{Img}(S)$ without actually computing R.

As the above definition of **Img** indicates, the process of image computation involves quantifying state variables. In the BDD representation, this amounts to quantifying over several Boolean state variables. *Early quantification* [BCL91b, TSL$^+$90] is a technique which attempts to reorder the conjuncts so that the scope of each quantifier is minimized. The effect of early quantification is that the evaluation of single quantifiers can be done over *relatively small intermediate BDDs.* An exact definition of early quantification will be given in Section 2. The success of early quantification hinges heavily upon the derivation and ordering of sub-relations. Significant effort has been directed over the last decade to this problem. Since the problem is known to be NP-hard, various heuristics have been proposed for the problem.

In this paper, we propose and analyze several new techniques for efficient image computation using the partitioned representation. The main contributions of the paper are the following:

• We extend and analyze image computation techniques previously developed by Moon *et al.* [MKRS00]. These techniques are based on the *dependence matrix* of the partitioned transition relation. We explore various *lifetime metrics* related to this representation and argue their importance in predicting costs of image computation. Moreover, we provide effective heuristic techniques to optimize these metrics.

• We show that the problem of minimizing the lifetime metric of [MKRS00] is NP

[1] The techniques in this paper also apply to preimage computation. For ease of exposition, we restrict ourselves to image computation.

complete. More importantly, the reduction used to prove this result explains the close connection between efficient image computation and the well studied problem of computing the *optimal linear arrangement* for an undirected graph.
• We model the interaction between various sub-relations in the partitioned transition relation as a weighted graph, and introduce a new class of heuristics called *ordering by recursive partitioning.*
• We have performed extensive experiments which indicate the effectiveness of our techniques. By implementing these techniques, we have also contributed to the code base of the symbolic model checker NuSMV [CCGR99].

The main conclusion to be drawn from our analysis is the following: For complicated industrial designs, the effort initially spent on ordering algorithms is clearly amortized during image computation. In other words, the benefits of good orderings outweigh the cost of slow combinatorial optimization algorithms.

The remainder of this paper is organized as follows: in Section 2, we introduce notations and definitions used throughout the paper. Section 3 reviews the state of the art for this problem. Section 4 discusses various algorithms that facilitate early quantification. Section 5 describes experimental results. Finally, we conclude in Section 6 with some directions for future research.

2 Preliminaries

Notation: Every state is represented as a vector $b_1 \dots b_n \in \{0,1\}^n$ of Boolean values. The transition relation R is represented by a Boolean function $T(x_1, \dots, x_n, x'_1, \dots, x'_n)$. Variables $X = x_1, x_2, \dots, x_n$ and $X' = x'_1, x'_2, \dots, x'_n$ are called *current state* and *next state* variables respectively. $T(X, X')$ is an abbreviation for $T(x_1, \dots, x_n, x'_1, \dots, x'_n)$. Similarly, functions of the form $S(X) = S(x_1, \dots, x_n)$ describe sets of states. We will occasionally refer to S as the *set*, and to T as the *transition relation.* For simplicity we will use X to denote both the set $\{x_1, \dots, x_n\}$ and the vector $\langle x_1, \dots, x_n \rangle$. Then the set of variables on which f depends is denoted by $Supp(f)$.

Example 1. **[3 bit counter. (Running Example)]** Consider a 3-bit counter with bits x_1, x_2 and x_3. x_1 is the least significant and x_3 the most significant bit. The state variables are $X = x_1, x_2, x_3$, $X' = x'_1, x'_2, x'_3$. The transition relation of the counter can be expressed as

$$T(X, X') = (x'_1 \leftrightarrow \neg x_1) \wedge (x'_2 \leftrightarrow x_1 \oplus x_2) \wedge (x'_3 \leftrightarrow (x_1 \wedge x_2) \oplus x_3).$$

In later examples, we will compute the image $\mathbf{Img}(S)$ of the set $S(X) = \neg x_1$. Note that $S(X)$ contains those states where the counter is even.

Partitioned BDDs: For most realistic designs it is impossible to build a single BDD for the entire transition relation. Therefore, it is common to represent the transition relation as a conjunction of smaller BDDs $T_1(X, X')$, $T_2(X, X')$, $\dots$, $T_l(X, X')$, i.e.,

$$T(X, X') = \bigwedge_{1 \le i \le l} T_i(X, X'),$$

where each T_i is represented as a BDD. The sequence $T_1, \dots, T_l$ is called a *partitioned transition relation.* Note that T is *not actually computed*, but only the T_i's are kept in memory.

Example 2. **[3 bit counter, ctd.]** For the 3 bit counter, a very simple partitioned transition relation is given by the functions $T_1 = (x'_1 \leftrightarrow \neg x_1), T_2 = (x'_2 \leftrightarrow x_1 \oplus x_2)$ and $T_3 = (x'_3 \leftrightarrow (x_1 \wedge x_2) \oplus x_3)$.

Partitioned transition relations appear naturally in hardware circuits where each latch (i.e., state variable) has a separate transition function. However, a partitioned transition relation of this form typically leads to a very large number of conjuncts. A large partitioned transition relation is similar to a CNF representation. So as the number of conjuncts increases, the advantages of BDDs are gradually lost. Therefore, starting with a very fine partition $T_1, \ldots, T_l$ obtained from the bit relations, the conjuncts T_i are grouped together into *clusters* $C_1, \ldots, C_r$, $r < l$ such that each C_i is a BDD representing the conjunction of several T_i's. The image $\mathbf{Img}(S)$ of S is given by the following expression.

$$\begin{aligned}
\mathbf{Img}(S(X)) &= \exists X \cdot (T(X, X') \wedge S(X)) && (1)\\
&= \exists X \cdot (\bigwedge_{1 \le i \le l} T_i(X, X') \wedge S(X)) && (2)\\
&= \exists X \cdot (\bigwedge_{1 \le i \le r} C_i(X, X') \wedge S(X)) && (3)
\end{aligned}$$

Note that in general $\exists x(\alpha \wedge \beta)$ is not equivalent to $(\exists x \alpha) \wedge (\exists x \beta)$. Consequently, to compute $\mathbf{Img}(S(X))$, formula 3 instructs us to compute first a BDD for $\bigwedge_{1 \le i \le r} C_i(X, X') \wedge S(X)$. As argued above, partitioned transition relations have been introduced to *avoid* computing this potentially large BDD.

Early Quantification: Under certain circumstances, existential quantification can be distributed over conjunction using *early quantification* [BCL91b, TSL$^+$90]. Early quantification is based on the following observation: if we know that α *does not contain* x, then $\exists x(\alpha \wedge \beta)$ is equivalent to $\alpha \wedge (\exists x \beta)$. In general, we have l conjuncts and n variables to be quantified. Since loosely speaking, clusters correspond to semantic entities of the design to be verified, it is expected that not all variables appear in all clusters. Therefore, some of the quantifications may be shifted over several C_i's. For a given sequence $C_1, \ldots, C_r$ of clusters, we obtain

$$\begin{aligned}
\mathbf{Img}(S(X)) = \exists X_1 \cdot (C_1(X, X') \wedge \exists X_2 \cdot (C_2(X, X') \ldots \\
\exists X_r \cdot (C_r(X, X') \wedge S(X)))) \qquad (4)
\end{aligned}$$

where X_i is the set of variables which do not appear in $Supp(C_1) \cup \ldots \cup Supp(C_{i-1})$ and each X_i is disjoint from each other. Existentially quantifying out a variable from a formula f reduces $|\,Supp(f)|$ which usually corresponds to a reduced BDD size. The success of early quantification strongly depends on the order of the conjuncts $C_1, \ldots, C_r$.

Quantification Scheduling. The size of the intermediate BDDs in image computation can be reduced by addressing the following two questions:

Clustering: How to derive the clusters $C_1, \ldots, C_r$ from the bit-relations $T_1, \ldots, T_l$?

Ordering: How to order the clusters so as to minimize the size of the intermediate BDDs?

These two questions are not independent. In particular, a bad clustering results in a bad ordering. Moon and Somenzi [MS00] refer to this combined problem as the *quantification scheduling* problem. The ordering of clusters is known as the *conjunction schedule*.

Our algorithms are based on the concepts of *dependence matrices* (introduced in [MKRS00, MS00]) and *sharing graphs*.

Definition 1 (Moon et al). *The* ***dependence matrix*** *of an ordered set of functions* $\{f_1, f_2, \ldots, f_m\}$ *depending on variables* $x_1, \ldots, x_n$ *is a matrix* D *with* m *rows and* n *columns such that* $d_{ij} = 1$ *if function* f_i *depends on variable* x_j*, and* $d_{ij} = 0$ *otherwise.*

Thus, each row corresponds to a formula, and each column to a variable. For image computation, we will associate the rows with the conjuncts of the partitioned transition relation, and the columns with the state variables. For example, $f_m = S(X), f_{m-1} = C_r, \ldots$. Thus, different choices for $f_i, 1 \leq i \leq m$ correspond to different orderings.

We will assume that the conjunction is taken in the order $f_m, f_{m-1}, \ldots, f_2, f_1$, i.e., we consider an expression of the form $\exists X\ (f_1 \wedge (f_2 \wedge \ldots \wedge (f_{m-1} \wedge f_m)))$. If a variable occurs *only* in f_m, we can quantify it early by pushing it to the right just before f_m.

Example 3. **[3 bit counter, ctd.]** For $f_4 = S(X)$, $f_3 = T_3$, $f_2 = T_2$, $f_1 = T_1$ the dependency matrix for our running example looks as follows:

	v_1	v_2	v_3	v_1'	v_2'	v_3'
$f_1 = T_1$	1	0	0	1	0	0
$f_2 = T_2$	1	1	0	0	1	0
$f_3 = T_3$	1	1	1	0	0	1
$f_4 = S(X)$	1	0	0	0	0	0

In general, for a variable x_j, let l_j denote the smallest index i in column j such that $d_{ij} = 1$. Analogously, h_j denotes the largest index. We can quantify away the variable x_j as soon as the conjunct corresponding to the row l_j has been considered. The variable does not appear in any conjuncts after h_j. Hence, $h_j - l_j$ can be viewed as the *lifetime* of a variable. Moon, Kukula, Ravi and Somenzi [MKRS00] define the following metric and use it extensively in their algorithms.

Definition 2 (Moon, Kukula, Ravi, Somenzi). *The* ***normalized average lifetime*** *of the variables in a dependence matrix* $D_{m \times n}$ *is given by*

$$\lambda = \frac{\sum_{1 \leq j \leq n} (h_j - l_j + 1)}{m \cdot n}$$

Note that the definition of λ assumes that $S(X)$ is given. Therefore, since λ *depends* on $S(X)$, the ordering has to be recomputed in each step of the fixpoint computation. We are considering static ordering techniques here, which are computed independently of any particular $S(X)$, so it is necessary to make assumptions about the structure of $S(X)$. We obtain two lifetime metrics λ_U and λ_L depending on whether we assume

$Supp(S) = X$ or $Supp(S) = \emptyset$. It is easy to see that $\lambda_L \leq \lambda \leq \lambda_U$. The terms *average active lifetime* and *total active lifetime* are also used to denote λ_L and λ_U respectively. Moon and Somenzi argue in favour of using λ_L. We will evaluate the effectiveness of each of these metrics to predict image computation costs.

3 Related Work

The importance of the clustering and ordering problem was first recognized by Burch *et al.* [BCL91a] and Touati *et al.* [TSL$^+$90]. Geist and Beer [GB94] proposed a simple heuristic algorithm, in which they ordered conjuncts in the increasing order of the number of support variables. All these techniques are static techniques. Subsequently, the same clusters and ordering are used for all the image computations during symbolic analysis. Since the clustering and ordering problems are not independent, these techniques typically begin by first ordering the conjuncts and then clustering them and finally ordering the clusters again using the same heuristics. The first successful heuristic (commonly known as IWLS95) for this problem is due to Ranjan *et al.* [RAP$^+$95]. They have an elaborate heuristic procedure for ordering the initial conjuncts and the clusters. The ordering procedure maintains a set Q of conjuncts that are already ordered and a set R of conjuncts that are yet to be ordered. Note that we have used the word conjunct here to mean both the conjuncts before clustering and the clusters in the final ordering phase. The next conjunct in the order is chosen from R using a heuristic score. The score is computed by using four factors: the maximum BDD index of a variable that can be quantified, the number of next state variables that would be introduced, the number of support variables, and the number of variables that will be quantified away. After the ordering phase, the clusters are derived by repeatedly conjoining the conjuncts until the BDD of the cluster grows larger than some partition size limit, at which point a new cluster is started. Bwolen Yang proposed a similar technique in his thesis [Yan99]. However, he introduces a pre-merging phase where conjuncts are initially merged pairwise based on the sharing of support variables and the maximum BDD size constraint. His ordering heuristic is based on six factors which are similar to those used by Ranjan *et al.* [RAP$^+$95]. However, he also takes into account the relative growth in BDD sizes. The clustering algorithm Yang uses is the same as the one used in IWLS95. A recent paper by Moon and Somenzi [MS00] presents an ordering algorithm (henceforth referred to as FMCAD00) based on computing the *Bordered Block Triangular* form of the dependence matrix. Their clustering algorithm is based on the sharing of support variables (affinity). They report large performance gains with respect to the IWLS95 technique.

4 Algorithms for Ordering Clusters

The algorithms we propose also follow the order-cluster-order strategy. The ordering algorithms that we present in this section are used before and after clustering. Our clustering strategy is as in IWLS95. For the sake of clarity of notation, let us assume that the clusters $C_1, C_2, \ldots, C_r$ have been constructed and we are ordering them. But the discussion applies equally well to ordering the initial conjuncts $T_1, \ldots, T_n$.

We present two classes of algorithms. The first one is based on dependence matrix and the other one on *sharing graphs*.

In Section 2 we defined a dependence matrix D corresponding to the set of clusters $C_1, \cdots, C_r$. As already pointed out, the number of support variables provides a good estimate of the size of a BDD. Therefore, we seek a schedule in which the lifetime of variables is low. Moon and Somenzi [MS00] provide a method to convert a dependence matrix into bordered block triangular form with the goal of reducing λ_L.

4.1 Minimizing $\boldsymbol{\lambda}$ is NP-complete

The main result of this subsection (Theorem 1) motivates the use of various combinatorial optimization methods.

Let λ-OPT be the following decision problem: given a dependence matrix D and a number r, does there exist a permutation σ of the rows of D such that $\lambda < r$? The following theorem shows that $\lambda - OPT$ is NP-complete. The reduction is from the *optimal linear arrangement problem (OLA)* [GJ79, page 200]. Due to space limitations the proof is given in the appendix.

Theorem 1. *λ-OPT is NP-complete.*

The complexity of this problem was not explored by Moon and Somenzi [MS00]. There exists a variety of heuristics for solving the optimal linear arrangement problem and related problems in combinatorial optimization. Some of these heuristics are based on hill climbing and simulated annealing. There are two important characteristics of this class of algorithms. First of all, they all try to minimize an underlying cost function. Second, these heuristics use a finite set of *primitive transformations*, which allows them to move from one solution to another. In our case, the set of swaps of the rows of the dependence matrix constitutes the set of moves and the cost function can be chosen to be either λ_L or λ_U. Our experimental results (Section 5) confirm that λ_L correlates with image computation costs much better than λ_U does, in accordance with the claim of [MS00]. Simulated annealing is a more general and flexible strategy than hill climbing.

4.2 Hill Climbing

Hill climbing is the simplest greedy strategy in which at each point, the solution is improved by choosing two rows to be swapped in such a manner as to achieve best improvement in the cost function. This process is repeated until no further move improves the solution. Since the best move is chosen at each point, this strategy is also called *steepest descent hill climbing*. However, this algorithm can easily get stuck in local optima. Randomization is used to alleviate this problem as follows: The best move that improves the solution is accepted only with some probability p, and with probability $1-p$, a random move is accepted. This allows the algorithms to get out of local optima. Note that with $p = 1.0$, we get the steepest descent hill climbing. The algorithm can be run multiple number of times, each time beginning with a random permutation, and the best solution that is achieved is accepted.

Figure 1 describes the algorithm in exact terms. The hill climbing procedure is repeated $NumStarts$ times. In the algorithm, σ denotes a permutation of the rows of the dependency matrix. Hill climbing is performed until no further improvement in λ is possible.

```
HillClimbOrder($D$)
$\lambda_{best} = 2$ // any number greater than 1 will do, since $\lambda$ is always less than 1
for $i = 1$ to $NumStarts$
    let $\sigma'$ be a random permutation of conjuncts.
    while there exists a swap in $\sigma'$ to reduce $\lambda$
        make the best swap with probability $p$,
        or make a random swap with probability $1 - p$ to update $\sigma'$.
    if $\lambda' < \lambda_{best}$
        $\lambda_{best} = \lambda'$
        $\sigma_{best} = \sigma$
    endif
endfor
```

Fig. 1. Hill climbing algorithm for minimizing λ

4.3 Simulated Annealing

The physical process of annealing involves heating a piece of metal and letting it cool down slowly to relieve stresses in the metal. The simulated annealing algorithm (introduced by Metropolis *et al.* [MRR$^+$53]) mimicks this process to solve large combinatorial optimization problems [KJV83]. Drawing analogy from the physical process of annealing, the algorithm begins at a high "temperature", where the set of moves is essentially random. This allows larger jumps from local to global optima. Gradually, the temperature is decreased and the moves become less random favoring greedy moves over random moves for achieving a global optimum. Finally, the algorithm terminates at "freezing" temperatures where no further moves are possible. At each stage, the temperature is kept constant until "thermal quasi-equilibrium" is reached. While random moves help in the beginning, when the algorithm has a greater tendency to get stuck in local optima, the greedy moves help to achieve a global optimum once the solution is in the proximity of one. In practice, simulated annealing has been successfully used to solve optimization problems from several domains.

The probability of making a move that *increases* the cost function is related to the temperature t_i at the i-th iteration, and is given by $e^{-\Delta\lambda/t_i}$. Thus at higher temperatures, the probability of accepting random moves is high. The gradual decrease of temperature is called the *cooling schedule.* If the temperature is decreased by a fraction r in each stage, we get an exponential cooling schedule. Thus beginning with an initial temperature of t_0, the temperature in the i-th iteration is $t_0 r^i$. It has been shown that a logarithmic cooling schedule is guaranteed to achieve an optimal solution with high probability [B'e92, Haj85]. However, this is an extremely slow cooling schedule and simple cooling schedules like exponential schedules perform well for many problems. Figure 2 describes our algorithm. The parameter $NumStarts$ controls the number of times the temperature is decreased. The parameter $NumStarts2$ controls the number of iterations at a fixed temperature t_i.

4.4 Sharing Graphs and Separators

We build *sharing graphs* as defined below to model interaction between clusters.

```
   SimAnnealOrder($D$)
   for $i = 1$ to $NumStarts$
1      $t_i \leftarrow t_0 r^i$
2      for $j = 1$ to $NumStarts2$
3          permute two random rows of $D$ to get $D_i$
4          if $(\lambda_i < \lambda)$    // greedy move
5              $\lambda \leftarrow \lambda_i; D \leftarrow D_i$
6          else    // random move
7              with probability $e^{\frac{-(\lambda_i - \lambda)}{t_i}}$, set $\lambda \leftarrow \lambda_i; D \leftarrow D_i$
8          endif
9      endfor
10 endfor
```

Fig. 2. Simulated annealing algorithm to minimize λ

Definition 3. *A* ***sharing graph*** *corresponding to a set of Boolean functions* $\{f_1, f_2, \ldots, f_m\}$ *is a weighted graph* $G(V, E, w_e)$*, where* $V = \{f_1, f_2, \ldots, f_m\}$*,* $E = V \times V$ *and* $w_e : E \rightarrow \Re$ *is a real-valued weight function.*

We shall use heuristic weight functions to express interaction between clusters. Intuitively, the stronger the interaction between two clusters, the closer they should be in the ordering. IWLS95 and Bwolen Yang's heuristics order the conjuncts based on this type of interaction between conjuncts. We propose to use graph algorithms on sharing graphs to order the conjuncts. We define the weight $w(T_i, T_j)$ of an edge (T_i, T_j) in the sharing graph as

$$w(T_i, T_j) = W_1 \cdot \frac{Supp(T_i) \cap Supp(T_j)}{|\,Supp(T_i)| + |\,Supp(T_j)|} + W_2 \cdot \frac{BddSize(T_i \wedge T_j)}{BddSize(T_i) + BddSize(T_j)}$$

The first factor ($W_1 \geq 0$) denotes the relative weight of sharing of support between two conjuncts, while the second factor ($W_2 \leq 0$) denotes the weight of the relative growth in the sizes of BDDs if these two conjuncts are conjoined. Therefore, a higher edge weight between two conjuncts indicates a higher degree of interaction and consequently these conjuncts should appear "close" in the ordering.

A separator partitions the vertices of a weighted undirected graph into two sets such that the total weight of the edges between two partitions is "small". Formally, an *edge separator* is defined as follows:

Definition 4. *Given a weighted undirected graph* $G(V, E)$ *with two weight functions* $w_e : E \rightarrow \Re$ *and* $w_v : V \rightarrow \Re$*, and a positive constant* $\gamma < 0.5$*, an* edge separator *is a collection of edges* E_s *such that removing* E_s *from* G *partitions* G *into two disconnected subgraphs* V_1 *and* V_2*, and* $\frac{|\sum_{v \in V1} w_v(v) - \sum_{v \in V2} w_v(v)|}{\sum_{v \in V} w_v(v)} < \gamma$*.*

Usually, γ is chosen very close to zero so that the size of the two sets is approximately the same. The weight of the edge separator E_s is simply the sum of the weight of the edges in E_s. It has been shown that finding an edge separator of minimum weight is NP-complete [GJ79, pp. 209], in fact finding an approximation is NP-hard, too [BJ92]. The problem of finding a good separator occurs in many different contexts and a wide range of application areas. A large number of heuristics have been proposed for the problem. One of the most important heuristics is due to Kernighan and Lin [KL70]. Variations of this heuristic [FM82] have been found to work very well in practice.

By finding a good edge separator of the sharing graph, we obtain two sets of vertices with a low level of interaction between them. Thus the vertices of these two sets can be put apart in the ordering. A complete ordering is achieved by recursively invoking the algorithm on the two halves. Since this ordering respects the interaction strengths between conjuncts, we expect to achieve smaller BDD sizes.

We use the Kernighan-Lin algorithm for finding a good edge separator E_s. This produces two sets of vertices L and R. A vertex $v \in L$ that has an edge of non-zero weight to a vertex in R is called an *interface vertex.* L_I denotes the set of interface vertices in L. Similarly, R_I denotes the set of interface vertices in R. We invoke the algorithm to recursively order $L \setminus L_I$, L_I, R_I, and $R \setminus R_I$. Finally, the order on the vertices is given by the order on $L \setminus L_I$ followed by the order on L_I, followed by the order on R_I, and followed by the order on $R \setminus R_I$. Figure 3 describes the complete algorithm.

```
   KLinOrder($G(V,E),W$)
1  Find a separator $E_s$ using
   Kernighan-Lin heuristic
2  Let $L$ and $R$ be two partitions of
   vertices induced by $E_s$.
3  $L_i \leftarrow Interface(L)$.
4  $R_i \leftarrow Interface(R)$.
5  Recursively call the procedure on
   the subgraphs induced by $L \setminus L_i$, $L_i$,
   $R_i$ and $R \setminus R_i$.
6  Order the vertices as
   $KLinOrder(L \setminus L_i) \prec KLinOrder(L_i) \prec$
   $KLinOrder(R_i) \prec KLinOrder(R \setminus R_i)$.
```

Fig. 3. An ordering algorithm based on graph separators

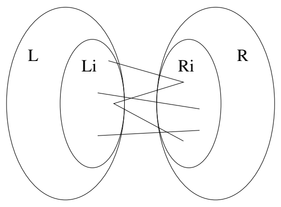


Fig. 4. Kernighan-Lin partition

5 Experimental Results

In order to evaluate the effectiveness of our algorithms, we ran reachability and model checking experiments on circuits obtained from the public domain and industry. The "S" series of circuits are ISCAS'93 benchmarks, and the "IU" series of circuits are various abstractions of an interface control circuit from Synopsys. For a fair comparison, we implemented all the techniques in the NuSMV model checker. All experiments were done on a 200MHz quad Pentium Pro processor machine running the Linux operating system with 1GB of main memory. We restricted the memory usage to 900MB, but did not set a time limit. The two performance metrics we measured are *running time* and *peak number of live BDD nodes*. We provided a prior ordering to the model checker and turned off the dynamic variable reordering option. This was done so that the effects of BDD variable reordering do not "pollute" the result. We also recorded the fraction of time spent in the clustering and ordering phases. The cost of these phases is amortized over several image computations performed during model checking and reachability analysis.

In the techniques that we have described, several parameters have to be chosen. For example, the cooling schedule in the case of simulated annealing needs to be determined. We ran extensive "tuning experiments" to find the best value for these parameters. Due to space constraints, we do not describe all those experiments. However, the choice of lifetime metric to optimize is a crucial one and hence in our first set of experiments, we evaluate the effectiveness of these metrics for predicting image computation costs.

Our algorithms for combinatorial optimization of lifetime metrics can choose to work with either upper or lower approximations of lifetimes. We ran the following experiment to estimate the correlation between the performance, and λ_L and λ_L respectively. We generate various conjunction schedules for a number of benchmarks by different ordering methods and by varying various parameters of the optimization methods. Each schedule gives us different values for lifetime metrics. We measure the running time and the peak number of live BDD nodes used for the model checking or reachability phase. For each circuit, this gives us four scatter plots for running time vs lifetime metric and space vs lifetime metric. A statistical correlation coefficient between runtime/space and lifetime metric indicates the effectiveness of a metric for predicting the runtime/space requirement. The following Table 1 concisely summarizes the correlation results.

Circuit	Runtime		Space	
	λ_L	λ_U	λ_L	λ_U
IU40	0.560	0.303	0.610	0.227
IU70	0.603	0.336	0.644	0.263
TCAS	0.587	0.366	0.628	0.240
S1269	0.536	0.402	0.559	0.345
S3271	0.572	0.350	0.602	0.297

Table 1. Correlation between various lifetime metrics and runtime/space for a representative sample of benchmarks.

It is clear from this data that the active lifetime (λ_L) is a much more accurate predictor of image computation costs than total lifetime (λ_U). Hence, simulated annealing and hill climbing techniques optimize λ_L.

In the following set of experiments (Table 2), we compare our techniques against the FMCAD00 strategy [MS00]. The first column indicates the total running time of the benchmark (including ordering/clustering and model checking/reachability phases), the second column indicates the peak number of live BDD nodes in thousands during the whole computation, the third column indicates time used by ordering phase, the next two columns indicate λ_L and λ_U achieved. From hill climbing and simulated annealing, we only report the results of simulated annealing, as both of them belong to the same class of algorithms. Moreover, we found out that in general, simulated annealing achieves better performance than hill climbing.

The algorithm KLin based on edge separators achieves lower peak live node count for several circuits than FMCAD00. For the 15 large benchmarks for which FMCAD'00 takes more than 100 secs to finish, KLin wins 10 cases in terms of Peak live BDD nodes, and 7 cases in terms of running time. In some cases, the savings in space is 40%.

The result for the simulated annealing algorithm that minimizes λ is shown in Table 2. Again, in comparison to FMCAD00, for the 15 non-trivial benchmarks, simulated annealing wins 14 cases and ties for the other in space, and wins 11 cases in time. In some cases, the savings in space is 55%. The simulated annealing algorithm can also complete 16 reachability steps for the S1423 circuit, which to our knowledge has not be achieved by other techniques. Comparing KLin and simulated annealing, simulated annealing achieves the better results for all the nontrivial benchmarks.

The improvements in execution times are less than the improvements in space, especially for smaller circuits. This is because separator based algorithms spend more time in the ordering phase itself. However, for larger circuits, this cost gets amortized by the smaller BDDs achieved during analysis. An important observation that can be made is that in general, our algorithms spend more time in the initial ordering phase as compared to FMCAD00. This is to be expected since both KLin and simulated annealing are optimization methods.

The last two columns in Table 2 indeed demonstrate that our algorithms improve various λs with respect to FMCAD'00. The main objective of our algorithms was to improve λ_L, though we can see that they also result in better λ_Us in general.

6 Conclusions and Future Work

We have given convincing evidence that variable lifetimes have a crucial impact on the performance of image computation algorithms. We have also presented new algorithms for conjunction scheduling based on hill climbing, simulated annealing, and graph separators and shown the effectiveness of them. The performance of these algorithms was comparable to that of the current best known methods. Our experiments clearly demonstrate that for large circuits, we can achieve savings in memory in the range of 50-60%. Since fine-tuned image computation algorithms are obviously most important for large circuits, we believe that our results are a significant contribution to model checking and reachability analysis. On the implementation side, we have contributed several new image computation algorithms to the NuSMV model checker, and believe that it will be a valuable research tool.

There are some interesting research directions to pursue. First of all, we need to understand the behavior of our algorithms on broader class of systems. The examples were mostly chosen from the circuit domain, and we would like to see the effectiveness of these algorithms on other class of circuits. Secondly, techniques which switch between different conjunction schedules depending on intermediate state sets in the

Circuit	#FF	#inp.	log_2 of	Total Time			Peak space			Ordering time			λ_L			λ_U		
			#reach	I	II	III	I	II	III	I	II	III	I	II	III	I	II	III
IDLE	73	0	14.63	159	161	182	289	276	223	2	20	29	0.329	0.293	0.200	0.421	0.515	0.487
GUID	91	0	47.58	14	20	24	137	106	138	4	15	19	0.346	0.220	0.165	0.394	0.452	0.294
S953	29	16	8.98	1	2	3	15	13	15	1	1	3	0.290	0.290	0.271	0.507	0.485	0.410
IU30	30	138	18.07	28	104	63	290	563	290	3	24	34	0.360	0.368	0.324	0.459	0.522	0.634
IU35	35	183	22.49	13	29	11	257	366	202	4	24	6	0.364	0.373	0.304	0.573	0.360	0.308
IU40	40	159	25.85	13	37	14	353	384	232	5	21	5	0.326	0.336	0.302	0.508	0.326	0.334
IU45	45	183	29.82	**MOut**	11256	165	**MOut**	3952	483	10	32	39	0.360	0.353	0.300	0.465	0.663	0.569
IU50	50	615	31.57	476	522	540	1627	1599	1602	16	52	77	0.319	0.418	0.133	0.459	0.654	0.403
IU55	55	625	33.94	982	891	870	4683	3358	3298	14	90	84	0.384	0.386	0.324	0.583	0.432	0.515
IU65	65	632	39.32	**MOut**	1260	1083	**MOut**	7048	6793	18	81	100	0.389	0.353	0.353	0.659	0.448	0.423
IU70	70	635	42.07	5398	3033	2855	17355	9099	9964	38	95	129	0.303	0.296	0.286	0.424	0.393	0.486
IU75	75	322	46.59	5367	4218	3822	16538	12193	9404	45	115	140	0.398	0.371	0.349	0.731	0.692	0.526
IU80	80	350	49.80	**MOut**	6586	4824	**MOut**	22234	17993	49	127	136	0.372	0.335	0.322	0.570	0.628	0.345
IU85	85	362	52.14	**MOut**	**MOut**	6933	**MOut**	**MOut**	25661	59	141	154	0.332	0.303	0.287	0.623	0.597	0.591
TCAS	139	0	106.87	5058	5285	4598	11931	12376	9140	27	173	165	0.173	0.182	0.227	0.299	0.306	0.261
S1269	37	18	30.07	2109	2466	1875	1440	1736	893	10	19	24	0.584	0.622	0.449	0.659	0.929	0.589
S1512	57	29	40.59	799	1794	651	159	190	135	15	24	30	0.412	0.394	0.386	0.521	0.619	0.714
S5378	179	35	57.71*	18036	**MOut**	10168	1632	**MOut**	1279	42	49	67	0.124	0.114	0.099	0.219	0.164	0.152
S4863	104	49	72.35	3565	3109	3013	1124	947	910	38	45	56	0.102	0.103	0.086	0.251	0.109	0.179
S3271	116	26	79.83	4234	3286	3399	8635	6240	6203	33	30	30	0.224	0.185	0.184	0.366	0.306	0.226
S3330	132	40	86.64	23659	19533	24563	12837	9866	11381	69	123	150	0.214	0.217	0.227	0.299	0.335	0.378
SFE†	293	69	218.77	863	916	762	147	146	130	14	84	76	0.383	0.354	0.344	0.554	0.624	0.531
S1423	74	17	37.41**	23325	19265$^\#$	35876	65215	27653$^\#$	48366	10	17	35	0.486	0.501	0.301	0.622	0.622	0.460

Table 2. Comparing FMCAD00(I), Kernighan-Lin separator (II) and Simulated annealing (III) algorithms. The times are reported in seconds. The peak space is reported by the peak number of live BDD nodes in thousands. (**MOut**)–Out of memory, (†)–SFEISTEL, (*)–8 reachability steps, (**)–14 reachability steps, ($^\#$)–13 reachability steps. The lifetimes reported are after the final ordering phase.

fixpoint computation seem to be promising. We also plan to study in greater depth the effect of various parameters of our methods and automatic ways to tune them.

References

[BCL91a] J. R. Burch, E. M. Clarke, and D. E. Long. Representing circuits more efficiently in Symbolic Model Checking. In *28th ACM/IEEE Design Automation Conference*, 1991.

[BCL91b] J. R. Burch, E. M. Clarke, and D. E. Long. Symbolic Model Checking with partitioned transition relations. In A. Halaas and P. B. Denyer, editors, *Proceedings of the International Conference on Very Large Scale Int egration*, Edinburgh, Scotland, August 1991.

[B'e92] C. J. P. B'elisle. Convergence theorems for a class of simulated annealing algorithms. *Journal of Applied Probability*, 29:885–892, 1992.

[BJ92] T. N. Bui and C. Jones. Finding good approximate vertex and edge partitions is NP-hard. *Information Processing Letters*, 42:153–159, 1992.

[CCGR99] A. Cimatti, E. M. Clarke, F. Giunchiglia, and M. Roveri. NuSMV: A new Symbolic Model Verifier. In N. Halbwachs and D. Peled, editors, *Proceedings of the International Conference on Computer-Aided Verification (CAV'99)*, number 1633 in Lecture Notes in Computer Science, pages 495–499. Springer, July 1999.

[CGP00] E. M. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 2000.

[FM82] C.M. Fiduccia and R.M. Mattheyses. A linear time heuristic for improving network partitions. In *19th ACM/IEEE Design Automation Conference*, pages 175–181, 1982.

[GB94] D. Geist and I. Beer. Efficient Model Checking by automated ordering of transition relation partitions. In D. L. Dill, editor, *Sixth Conference on Computer Aided Verification (CAV'94)*, volume 818 of *LNCS*, pages 299–310, Stanford, CA, USA, 1994. Springer-Verlag.

[GJ79] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman & Co., 1979.

[Haj85] B. Hajek. A tutorial survey of theory and applications of simulated annealing. In *Proc. 24th IEEE Conf. Decision and Control*, pages 755–760, 1985.

[KJV83] S. Kirkpatrick, C. D. Gelatt Jr., and M. P. Vecchi. Optimization by simulated annealing. *Science*, 220:671–679, 1983.

[KL70] Brian Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs. *The Bell System Technical Journal*, pages 291–307, February 1970.

[MKRS00] In-Ho Moon, James H. Kukula, Kavita Ravi, and Fabio Somenzi. To split or to conjoin: The question in image computation. In *Proceedings of the 37th Design Automation Conference (DAC'00)*, pages 26–28, Los Angeles, June 2000.

[MRR$^+$53] N. Metropolis, A. W. Rosenbluth, M. N. Rosenbluth, A. H. Teller, and E. Teller. Equation of state calculations by fast computing machines. *Journal of Chemical Phyics*, 21(6):1087–1092, 1953.

[MS00] In-Ho Moon and Fabio Somenzi. Border-block triangular form and conjunction schedule in image computation. In Warren A. Hunt Jr. and Steven D. Johnson, editors, *Proceedings of the Formal Methods in Computer Aided Design (FMCAD'00)*, volume 1954 of *LNCS*, pages 73–90, November 2000.

[RAP$^+$95] R.K. Ranjan, A. Aziz, B. Plessier, C. Pixley, and R.K. Brayton. Efficient BDD algorithms for FSM synthesis and verification. In *IEEE/ACM International Workshop on Logic Synthesis*, Lake Tahoe, 1995. IEEE/ACM.

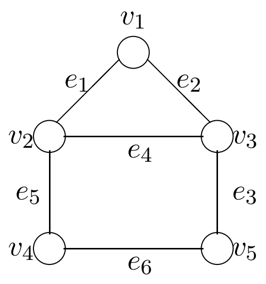
[TSL$^+$90] H. Touati, H. Savoj, B. Lin, R. K. Brayton, and A. Sangiovanni-Vincentelli. Implicit enumeration of finite state machines using BDDs. In *Proceedings of the IEEE international Conference on Computer Aided Design (ICCAD)*, pages 130–133, November 1990.

[Yan99] Bwolen Yang. *Optimizing Model Checking Based on BDD Characterization*. PhD thesis, Carnegie Mellon University, Computer Science Department, May 1999.

Appendix A

It is easy to see that for a given permutation σ of rows, we can compute λ in polynomial time ($O(n \cdot m)$) and check if $\lambda \leq r$.

To show that $\lambda - OPT$ is NP-hard, we reduce a known NP-complete problem called *optimal linear arrangement (OLA)* [GJ79, page 200] to $\lambda - OPT$. An instance of OLA consists of a graph $G(V, E)$ and a positive integer K. The question is whether there exists a permutation f of V such that $\sum_{(u,v)\in E} |f(u) - f(v)| \leq K$. The reduction consists of constructing a dependence matrix D and a number r such that $(V, E), K$ is a solution of OLA iff D, r is a solution to $\lambda - OPT$. An example of a reduction is given in figure 5.



$K = 9$

(a)

$$\begin{array}{c|cccccc} & 1 & 2 & 3 & 4 & 5 & 6 \\ \hline 1 & 1 & 1 & 0 & 0 & 0 & 0 \\ 2 & 1 & 0 & 0 & 1 & 1 & 0 \\ 3 & 0 & 1 & 1 & 1 & 0 & 0 \\ 4 & 0 & 0 & 1 & 0 & 0 & 1 \\ 5 & 0 & 0 & 0 & 0 & 1 & 1 \end{array}$$

$r = \frac{K+n}{n \cdot m} = 1/2$

(b)

Fig. 5. (a) An instance of Optimal Linear Arrangement, (b) its reduction to $\lambda - OPT$. The permutation v_1, v_2, v_3, v_5, v_4 is a solution to both.

Formally, D has $|V|$ rows corresponding to the vertices of $G(V, E)$, and $|E|$ columns corresponding to the edges of $G(V, E)$. For any edge $e_k = (v_i, v_j)$, set $d_{ik} = d_{jk} = 1$ and set all other d_{ij}'s to 0. Thus, in each column there are two occurences of the symbol 1. We set $r = \frac{K+n}{n \cdot m}$. Trivially we obtain the following equivalence:

$$\frac{\sum_{1\leq j\leq n}(h_j - l_j + 1)}{n \cdot m} \leq r$$
$$\Leftrightarrow \sum_{1\leq j\leq n} (h_j - l_j + 1) \leq r \cdot (n \cdot m)$$
$$\Leftrightarrow \sum_{1\leq j\leq n} (h_j - l_j + 1) \leq K + n$$

Let σ be a permutation of the vertices of V. Note that σ simultaneously is a permutation of the rows of D. We have to show that σ is a solution of $G(V, E), K$ iff σ is a solution of D, r.

The important observation is that because of the construction of D, the only non-zero entries in each column j correspond to the two vertices of the edge $e_j = (u, v)$. Therefore, we conlude that $h_j - l_j = |\sigma(u) - \sigma(v)|$. Continuing the above equivalence we obtain

$$\sum_{1\leq j\leq n} |\sigma(u) - \sigma(v)| + n \leq K + n$$
$$\Leftrightarrow \sum_{(u,v)\in E} |f(u) - f(v)| \leq K$$