Exact BDD Minimization for Path-Related Objective Functions

Rüdiger Ebendt and Rolf Drechsler

Institute of Computer Science, University of Bremen, 28359 Bremen, Germany
{ebendt,drechsle}@informatik.uni-bremen.de

Abstract. In this paper we investigate the exact optimization of BDDs with respect to path-related objective functions. We aim at a deeper understanding of the computational effort of exact methods targeting the new objective functions. This is achieved by an approach based on *Dynamic Programming* which generalizes the framework of Friedman and Supowit. A prime reason for the computational complexity can be identified using this framework.
For the first time, experimental results give the minimal expected path length of BDDs for benchmark functions. They have been obtained by an exact *Branch&Bound* method which can be derived from the general framework. The exact solutions are used to evaluate a heuristic approach. Apart from a few exceptions, the results prove the high quality of the heuristic solutions.

1 Introduction

Reduced ordered *Binary Decision Diagrams* (BDDs) were introduced in [6] and are well-known from logic synthesis and hardware verification.

Run time and space requirement of BDD-based algorithms depend on the size of the BDD. However, this size is very sensitive to a chosen variable ordering [6]. In general, determining an optimal variable ordering is a difficult problem. It has been shown that it is NP-complete to decide whether the number of nodes of a given BDD can be improved by variable reordering [4]. Therefore, heuristic methods have been proposed, based on structural information or on dynamic reconstruction [23]. Evaluation of heuristic solutions showed that they are often far away from the best known solution. Consequently, for applications like logic synthesis using multiplexor-based BDD circuits exact methods are also required: here a reduction in the number of BDD nodes directly transfers to a smaller chip area. Moreover, exact methods can provide the basis for the evaluation of heuristics.

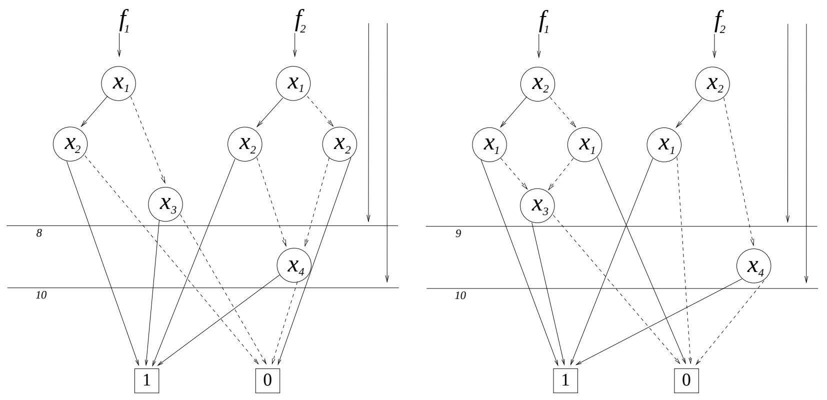
Similar questions arise for *alternative, path-related* objective functions. The optimization with respect to the *number of paths* in a BDD has been studied in [14]: the number of paths in a circuit derived from a BDD corresponds to the number of paths in the BDD. It is proportional to the number of faults under the path delay fault model. Hence minimizing the number of paths can significantly reduce the time for testing BDD circuits [10]. It also can be used for minimizing *Disjoint-Sum-Of-Products* (DSOPs) which are used in the calculation of spectra of Boolean functions or as starting point for the minimization of *Exclusive-Sum-Of-Products* (ESOPs): in a BDD for a Boolean function f, each path to the 1-terminal corresponds to a (partial) assignment to the variables, i.e. to a product of the literals of f. The products derived from different paths are disjoint. Collecting them in a sum yields a DSOP. Another field of application is Boolean satisfiability (SAT): the number of paths in BDDs is related to the number of backtracks of a SAT-solving procedure [22]. Optimization can support concepts to integrate SAT and BDDs. The optimization with respect to the *Expected Path Length* (EPL) has e.g. been studied in [20, 12]. It is motivated by the reduction of the time needed to evaluate many test vectors with a BDD in functional simulation, e.g. [19, 18]. Minimization of EPL as well as of the *Maximal Path Length* (MPL) in BDDs is also motivated by logic synthesis: first, every variable missing in a path of the BDD corresponds to a don't care. Thus shortening the EPL can help providing don't care values for minimization. Second, the longest path in the BDD corresponds to the critical path in a derived circuit. Hence minimization with respect to MPL/EPL is expected to support synthesis approaches targeting the delay of the resulting circuits. The minimization of MPL has been studied in [12, 21].

To evaluate the quality of heuristic results, again a comparison with exact solutions is of great help. In this paper a new exact EPL minimization algorithm is given and the computational hardness of the remaining exact optimization problems is analyzed. For that purpose a known approach to sequencing optimization problems [2, 3, 16] based on *Dynamic Programming* (DP) is generalized. This is done by replacing the previously used sufficient condition by a *weaker sufficient and necessary* condition. In this sense, a least restrictive framework is obtained. Next, this framework is used as a formal tool to analyze the given problems. The problems of exact BDD node minimization as well as of EPL-minimization can be solved with DP-based approaches for *Branch&Bound* (B&B) derived by this framework. However, the problems of minimizing the number of paths in BDDs and of MPL-minimization can not be solved even with the new conditions. A prime reason for this can be identified, the violation of *Bellmann's principle* [1].

Experiments show that, apart from a few exceptions, the results of a recent heuristic approach to minimize the EPL in BDDs [12] are of the same quality as exact solutions.

2 Preliminaries

In this section, basic notations and definitions are given.



(a) An α-minimal ordering (see Sec. 2.3) for $\{x_1, x_2, x_3\}$.

(b) A suboptimal ordering for $\{x_1, x_2, x_3\}$.

Fig. 1. Two BDDs for $f_1 = x_1 \cdot x_2 + \overline{x}_1 \cdot x_3$ and $f_2 = x_1 \cdot x_2 + \overline{x}_2 \cdot x_4$.

2.1 BDDs

Reduced ordered *Binary Decision Diagrams* (BDDs) are directed acyclic graphs (DAGs) where a Shannon decomposition

$$f = x_i f_{x_i} + \overline{x}_i f_{\overline{x}_i} \quad (1 \leq i \leq n)$$

is carried out with each node. Nodes v are labeled with variables in $X_n = \{x_1, \ldots, x_n\}$ (denoted by $\text{var}(v)$), edges are 1- or 0-edges, leading to one of the two child nodes denoted $\text{then}(v)$ and $\text{else}(v)$. The variables are bound to values in $\mathbf{B} := \{0, 1\}$. They are encountered at most once and in the same order, the "variable ordering" denoted π, on every path from the root to one of the two terminal nodes $\mathbf{1}$ and $\mathbf{0}$. For this reason the nodes can be partitioned into n *levels*, each of which contains the nodes labeled with one particular variable. If this is the first variable in the ordering, the level is called the first level, etc. For $1 \leq k \leq n$, the level is called the kth level if the variable is $\pi[k]$. Formally, variable orderings map level numbers to variables. The set of all orderings is denoted Π. For a BDD F, a prefix π, i.e. πF, expresses that F respects the ordering π. The term $\text{nodes}(F, x_i)$ denotes the set of nodes labeled with x_i (the "x_i-level" of F) and $\text{label}(F, x_i)$ abbreviates $|\text{nodes}(F, x_i)|$.

Note that reduced diagrams are considered, derived by removing redundant nodes and merging isomorphic subgraphs. In the following we assume shared BDDs with *Complement Edges* (CEs) [5] without mentioning it further (and without using CEs in the illustrations). Note that all results reported here directly transfer to BDDs without CEs. For examples of shared BDDs, see Fig. 1 (for now, the additional annotations can be disregarded, they will become important in Section 7), for more details see [6].

For a BDD F over X_n representing a Boolean function F, let $c(F, k)$ denote the set of nodes in levels below the kth level of F (including the terminal nodes) that are either externally referenced (i.e. they represent user functions) or referenced directly from the nodes in levels $1, \ldots, k$ of F. The set $c(F, 0)$ is equal to the set of externally referenced nodes (output nodes) in F. We will also need the notation $k(F, k) = c(F, k) \setminus \{\mathbf{1}, \mathbf{0}\}$. By the definition of c, every path starting at an output node and ending at a terminal node must traverse a node in $c(F, k)$. This property can be used to derive formulas that describe path related objective functions, as is seen later in Section 5.

The nodes in $c(F, k)$ represent the cofactors of f in the first k variables of the ordering F respects. To denote sets of cofactors of f with respect to a set of variables $X \subseteq X_n$, we use the notation $\mathrm{cof}(f, X)$.

2.2 Path-Related Objective Functions

Paths in a BDD start at a root node and end at a terminal node. The length of a path is the number of inner nodes on the path. Next, path-related objective functions are defined: the EPL of a BDD expresses the expected number of variable tests needed to evaluate an input assignment along a path from an output node to a terminal node. This number is determined as the average path length under all such input assignments. For a BDD F it is denoted $\epsilon(F)$. For a BDD node v, $\epsilon(v)$ is the EPL of the sub-BDD rooted at v. In case of a single-rooted BDD F, the EPL is simply the ϵ-value of the root node, otherwise it is the average of the weighted[1] ϵ-values for all output nodes. In [7], the term *Average Path Length* (APL) of a BDD is used for the unweighted sum of the EPLs of the single-rooted component BDDs forming the multi-output BDD. Further, let $\omega_\epsilon(v)$ denote the probability that an evaluation of input assignments which starts at an output node traverses v. Other path-related objective functions for BDDs are the number of paths and the maximal path length: let $\alpha(v)$ denote the number of paths from v to a terminal node, and let $\alpha(F)$ denote the number of paths from an output node to a terminal node, respectively. Let $\mu(v)$ denote the maximal length of a path from v to a terminal node, and let $\mu(F)$ denote the maximal length of a path from an output node to a terminal node, respectively. For a node v, let $\omega_\alpha(v)$ denote the number of paths from an output node to v and let $\omega_\mu(v)$ denote the maximal length of a path from an output node to v, respectively. Further, $\mu\_\mathrm{via}(v)$ denotes the maximal length of a path via v.

2.3 Miscellaneous

For the sake of completeness, the classical objective function BDD size will also be denoted by a Greek letter, namely ν. Sequences s are denoted using brackets, e.g. $s = \langle e_1, \ldots, e_k \rangle$. By $s \circ e$ we denote the concatenation of s with e to $\langle e_1, \ldots, e_k, e \rangle$. Further, let last: $\mathbb{R}^n \rightarrow \mathbb{R}$; $\mathrm{last}(x_1, \ldots, x_n) = x_n$ for all $x_1, \ldots, x_n \in \mathbb{R}$.

[1] The weight equals the number of external references to the output node.

We also make use of the following notations: let $I \subseteq X_n$. Throughout the paper, $\Pi(I)$ denotes the set of all orderings whose first $|I|$ positions constitute I. Let cost: $\{F \mid F \text{ is a BDD}\} \times 2^{X_n} \rightarrow \mathbb{R}$ be a cost function on BDDs, e.g. for BDD size, $\text{cost}(F, I)$ denotes the number of nodes in F labeled with a variable in I and it is $\text{cost}(F, X_n) = |F|$. If $\text{cost}(F, X_n) = \kappa(F)$ for an objective function κ, we have a *cost function for* κ. Then

$$\mathit{min\_cost}_I = \min_{\pi \in \Pi(I)} \text{cost}(\pi F, I)$$

denotes the minimal cost under all orderings in $\Pi(I)$. In the case of a cost function for κ, we call the ordering π leading to this minimum a κ-minimal ordering for I. We write Π_I for the set of all κ-minimal orderings for I. Note that $\mathit{min\_cost}_{X_n} = \min_{\pi \in \Pi} \kappa(\pi F)$.

3 Previous Work

To keep the paper self-contained, we briefly review previous work related to our studies. Our analysis is founded on results from two fields of research: the first field is sequencing optimization by DP, the second is BDD optimization. This paper presents research in the intersection of both fields.

3.1 Sequencing Optimization

Aiming at exact optimization with reasonable run times, it is mandatory to keep the size of the search space within sane limits: an exhaustive search essentially would compare every single input datum to every other input datum to find the solution. Hence, an exhaustive search requires $n!$ operations on the data. More mature methods manage to reduce the size of the search space to one of only 2^n states. Moreover, this space can often be pruned by B&B. Following this general outline, the framework for exact BDD minimization [15] was based on a more general approach to solve *sequencing optimization problems* [3, 16]. It makes use of *Bellmann's principle* [1]:

> *If the (total) sequence $e_1, \ldots, e_k, \ldots, e_n$ via e_k is optimal then the sub-sequence $e_1, \ldots, e_k$ must be optimal. Moreover, optimality of the overall sequence is preserved if the optimal sub-sequence is replaced by another optimal sub-sequence over $e_1, \ldots, e_k$.* (1)

Sometimes it is useful to define the optimality of a sequence over $\{e_1, \ldots, e_k\}$ as the cost minimality under all sequences over $\{e_1, \ldots, e_k\}$ that respect some condition, e.g. the condition of ending with the last element e_k. E.g., it is clear that when computing the shortest path between two nodes in a finite DAG, optimal sub-paths ending at some intermediate node must be part of a shortest path via the intermediate node.

This principle makes it possible to base the computation of optimal solutions on that of optimal partial solutions. Once partial solutions have been calculated, they may be reused several times during the algorithm, i.e. *memoization* can be used. A programming paradigm that is based on Bellmann's principle and memoization is *Dynamic Programming* [1]. In [3, 16], n-element sequencing problems were solved with DP-approaches that make use of recurrent equations for the partial solution costs. These are derived by repeatedly applying (1) to m-element starting sequences ($1 \leq m \leq n$) with a fixed last element (an example will be given at the end of the section).

The tackled problems all respected the following condition:

For all $k = 1, \ldots, n$:

- *Let* $\mathrm{cost}(e_1, \ldots, e_k) = \sum_{i=1}^{k} \mathrm{cost}(e_i)$.
- *Let* $\mathrm{cost}(e_k)$ *depend only on what elements are preceding* e_k *(i.e. be independent of their order).* (2)

This is a sufficient condition for the validity of (1): $\mathrm{cost}(e_k)$ is invariant under all orderings for $e_1, \ldots, e_k$. Hence, $\mathrm{cost}(e_1, \ldots, e_{k-1})$ must be minimal iff $\mathrm{cost}(e_1, \ldots, e_k) = \mathrm{cost}(e_1, \ldots, e_{k-1}) + \mathrm{cost}(e_k)$ is minimal. Hence, Bellmann's principle holds, and it is not necessary to construct all of the $n!$ orders for the n elements of the sequence.

In the following, this framework will be referred to as the framework of Bellmann/Held/Karp. Next, as an illustrating example, it is described how this idea has been used by Friedman and Supowit for exact node minimization [15]. In brief, the optimal variable ordering is computed iteratively by computing for increasing k's $\mathit{min\_cost}_I$ for each k-element subset I of X_n, until $k = n$: then, the BDD has a variable ordering yielding a BDD size of $\mathit{min\_cost}_{X_n}$. This is an optimal variable ordering.

This is done by a gradual schema of continuous minimum updates.

Let F be a BDD. Before the first step of the schema, $I = \emptyset$. Considering step k, let $I \subseteq X_n$ be a state which has been generated in the previous (i.e. $(k-1)$th) step. I' is a successor state of I, generated in the kth step by transitions $I \longrightarrow I \cup \{x_i\} =: I'$ ($x_i \in X_n \setminus I$).[2] The minimal cost and the best sequence for I' is computed using the following reccurrent equation [15].

$$\mathit{min\_cost}_{I'} = \min_{x_i \in I'} \left[\mathit{min\_cost}_{I' \setminus \{x_i\}} + \mathrm{label}(\pi_i F, x_i) \right] \tag{3}$$

where π_i is a variable ordering contained in $\Pi(I' \setminus \{x_i\})$ such that $\pi_i(|I'|) = x_i$. The starting value is $\mathit{min\_cost}_\emptyset = 0$.

This recurrence is based on the principle expressed in (1). The optimal order for an $|I'|$-element sub-sequence of variables is determined by minimizing over all possible last variables x_i. By (1), for every such variable the optimal sub-sequence of the first $(|I'|-1)$ variables must be part of the optimal sub-sequence

[2] The notation $\ldots =: I'$ is used for convenience. It has the same semantics as $I' := \ldots$ which is that of a defining assignment.

for all $|I'|$ elements ending with x_i (since "ending with x_i" is just a special case of "via" as stated in (1)). In essence, (1) holds as a direct consequence of the following: the term $\text{label}(\pi_i F, x_i)$ only depends on which variables occur before x_i in the ordering. This has been shown in [15] and is a sufficient condition following (2).

The state space considered here is 2^{X_n} which is of a size growing much slower with n than $n!$. By the use of B&B with lower and upper bounds on BDD size, it can be further reduced [9, 11]. But also recent approaches like the A^*-based approach in [13] still depend on the use of such a smart state encoding.

3.2 BDD Optimization

Section 1 already gave an overview of work in this field. Our approach in part is founded on the following previous results [12], [17].

Theorem 1. *Let F be a BDD representing a Boolean function f and let v be a node in F. Fixed probabilities are assumed for the variable assignments to values in $\mathbf{B}$. The term $\omega_\epsilon(v)$ is* invariant *with respect to variable ordering iff a) the function represented by v and b) the number of the v-level are preserved.*

Theorem 2. *Let F be a BDD with the underlying DAG (V, E). Then*

$$\epsilon(F) = \sum_{v \in V \setminus \{\mathbf{1}, \mathbf{0}\}} \omega_\epsilon(v). \tag{4}$$

4 Generalized Cost Function for Path-Related Objective Functions

Let a function acc map series with at most n elements to $\mathbb{R}$ and let it respect the following condition:

$$\text{acc}(c_1, \ldots, c_k) = \text{acc}(\text{acc}(c_1, \ldots, c_{k-1}), c_k) \ (1 \leq k \leq n)$$

Then, for $I \subseteq X_n$, a general form of a cost function that is appropriate for a recursion schema is:

$$\text{cost}(\pi F, I) = \text{acc}(c_1, \ldots, c_{|I|}) \text{ where}$$
$$c_k = \bigodot_{v \in \textsc{Cut}(\pi F, k)} C(v) \quad (1 \leq k \leq |I|)$$

Since a cost function can be uniquely determined by the choices of acc, $\odot$, $\textsc{Cut}$, and C, it is convenient to give cost functions by tuples $(\text{acc}, \odot, \textsc{Cut}, C)$, e.g. $\text{cost\_size} = (\sum, \sum, \text{nodes}, 1)$. For all nodes v, the contribution is $1(v) = 1$. By this, in the kth summand of acc, only the nodes in the kth level are counted, respectively. Depending on the choice of acc and $\odot$, more complex cost functions can be expressed.

5 Sufficient Condition for DP-based Exact Minimization

All path-related BDD optimization problems are special sequencing problems. This raises the question whether DP-based B&B optimization methods using the framework of Bellmann/Held/Karp outlined in Section 3 can be found. Assuming this framework could be used, an approach following the framework would be promising since a B&B method for node minimization already is known (see Section 3). For this reason it is investigated whether the sufficient condition (2) holds for the remaining path-related objective functions ϵ, α, and μ. In the course of the analysis, a new exact method for exact minimization of the EPL in BDDs is derived from this framework.

Expected Path Length. First, the objective function ϵ is considered. By Theorem 1 the following result can be deduced straightforwardly.

Lemma 1. *Let F be a BDD representing f, $I \subseteq X_n$, $k = |I|$, and $x_i \in I$. Then there exists a constant c such that $\sum_{v \in \mathrm{nodes}(\pi F, x_i)} \omega_\epsilon(v) = c$ for each $\pi \in \Pi(I)$ with $\pi(k) = x_i$.*

Consequently, (2) is respected and (1) holds. Let F be a BDD. Analogously to (3) we can derive the recurrence

$$min\_cost_{I'} = \min_{x_i \in I'} \left[min\_cost_{I' \setminus \{x_i\}} + \sum_{v \in \mathrm{nodes}(\pi_i F, x_i)} \omega_\epsilon(v) \right] \qquad (5)$$

where π_i is a variable ordering contained in $\Pi(I' \setminus \{x_i\})$ such that $\pi_i(|I'|) = x_i$. The starting value again is $min\_cost_\emptyset = 0$. By (4), $min\_cost_{X_n} = \min_{\pi \in \Pi} \epsilon(\pi F)$. Using (5), for increasing k's, a DP-approach can compute min_cost_I for each k-element subset I of X_n, until $k = n$. This yields a BDD of minimal ϵ-value. In Section 7, pseudo-code for the derived DP-approach will be given and it will be discussed in more detail.

Other Path-Related Objective Functions. Next, the use of the framework of Bellmann/Held/Karp is discussed for the other path-related objective functions. It is clarified that the sufficient condition (2) is *not* respected by the objective functions $\kappa \in \{\alpha, \mu\}$, regardless of which of the cost functions for κ known today are used.

Let F be a BDD with an underlying DAG $G = (V, E)$. Cost functions are based on equations describing the contribution of a single node v to $\alpha(F)$ or $\mu(F)$. We give the following equations describing this interrelation: let $0 \le k \le n$. For α, it is

$$\alpha(F) = \sum_{v \in c(F,k)} \alpha(v) \cdot \omega_\alpha(v), \qquad (6)$$

$$\alpha(F) = \sum_{v \in c(F,n)} \omega_\alpha(v). \qquad (7)$$

For μ, it is

$$\mu(F) = \max_{v \in V} \mu\text{\_via}(v), \text{ or, more specific,} \tag{8}$$

$$\mu(F) = \max_{v \in c(F,k)} \mu\text{\_via}(v), \tag{9}$$

and

$$\mu(F) = \max_{v \in c(F,n)} \omega_\mu(v). \tag{10}$$

For $1 \leq k \leq n$ every path from an output node to a terminal node must traverse a node in $c(F,k)$. Hence, e.g. in (6) the number of paths in F can be calculated by summing up the number of paths via a node for nodes in $c(F,k)$. For every such node v, this number is the product of ingoing paths multiplied with the number of outgoing paths. Altogether we have $\text{cost}(v) = C(v) = \alpha(v) \cdot \omega_\alpha(v)$ for $v \in c(F,k)$ and $C(v)$ is zero for $v \notin c(F,k)$.

By analogous arguments it is straightforward to see that (7)-(10) hold. The more general equations are (9) and (6). At present, no other equations describing node contributions for the considered objective functions are known.

Theorem 3. *The sufficient condition of the DP-approach of Bellmann, Held, and Karp does not hold for any of the known cost functions for α (number of paths in a BDD) and μ (maximal path length in a BDD). Hence, this approach to exact minimization can not be applied here, regardless of which of the known cost functions is used.*

However, this alone does not give strong evidence that sound DP-approaches would not exist in general: condition (2) is a *sufficient* but *not* a *necessary* condition for the validity of Bellmann's principle. Other sufficient conditions might exist which guarantee that Bellmann's principle is respected. In the next section, a sufficient and *necessary*, i.e. least restrictive condition and the resulting generalized framework is introduced.

6 Generalized Dynamic Programming Framework

In this section the following question is addressed: regarding (feasible) approaches based on DP and Bellmann's principle, can the two problems of minimizing $\kappa \in \{\alpha, \mu\}$ be solved? To ease the analysis, the framework of Bellmann/Held/Karp is generalized in this section. The sufficient condition of the previous framework is replaced by a *sufficient and necessary* condition for the validity of Bellmann's principle. In this sense, the presented approach is least restrictive. The new condition is operational in that it can be used to check whether a DP procedure can be easily derived for a given minimization problem. In the next section, this generalized framework will be used to show that Bellmann's principle is violated for the two problems, regardless of which of the known cost functions for the objectives are used. This means that even with the new condition no feasible exact algorithm can be derived for α and μ.

Next, a *necessary and sufficient* condition is formulated which in fact is equivalent to the principle of Bellmann (1) itself. In the new condition, the assumptions of (2) that

– the cost of the sequences is accumulated by summation
– $\mathrm{cost}(e_k)$ be fixed with respect to the ordering of the sub-sequence $e_1, \ldots, e_{k-1}$

are dropped and hence the condition is less restrictive. The resulting generalized framework is least restrictive in the sense that it is directly based on this principle itself. This contrasts to the framework of Bellmann/Held/Karp which can only be applied if a condition which is more restrictive than Bellmann's principle holds for the considered optimization problem. An advantage of the following new condition in comparison to (1) is the increased *operationality*, i.e. it is easier to detect whether a given sequencing problem respects the condition or not.

Theorem 4.

Let s_1, s_2 be two sequences (orders) of the elements in $\{e_1, \ldots, e_{k-1}\}$ and let s_1 be an optimal sequence. Let $\mathrm{cost}(s)$ denote the cost of a sequence s. Iff both

$$\mathrm{cost}(s_1) = \mathrm{cost}(s_2) \Rightarrow \mathrm{cost}(s_1 \circ e_k) = \mathrm{cost}(s_2 \circ e_k) \tag{11}$$
$$\mathrm{cost}(s_1) < \mathrm{cost}(s_2) \Rightarrow \mathrm{cost}(s_1 \circ e_k) < \mathrm{cost}(s_2 \circ e_k). \tag{12}$$

hold, Bellmann's principle as stated in (1) is respected.

Next, the recursive schema of the generalized framework for the exact BDD minimization with respect to path-related objective functions is given, together with sufficient and necessary conditions following (11) and (12). Thereby, we focus on the problem of BDD optimization, giving the schema for BDDs right away. However, note that it is straightforward to transfer the idea to (any) other sequencing problem. For a better understanding of the next theorem notice that the general flow of the schema is similar to the one already given in (3). Condition 1) of the following theorem states that the node contributions must not depend on the order of variables which are situated at levels $k > |I'|$. This is because otherwise the recurrence of the schema would not be well-defined since it depended on future values. Although it might look a bit over-formal, Condition 2) is just a straightforward "translation" of (11) and (12) into the BDD context. When collecting the node contributions, the schema can choose between two forms of a cut through the BDD as the general function SET is used. As before, the correctness of the schema follows from Bellmann's principle.

Theorem 5. *Let κ be an objective function for BDDs and let F be a BDD. Let $x_i \in I' \subseteq X_n$. Let $\mathrm{cost} = (\mathrm{acc}, \odot, \textsc{Set}, C)$ be a cost function for κ, where* SET *is a function identifier in $\{\mathrm{nodes}, c\}$. Further, let $\pi_i^* \in \Pi_{I' \setminus \{x_i\}}$ such that $\pi_i^*(|I'|) = x_i$.*

Assume that the following conditions are respected:

1) For $v \in \textsc{Set}(\pi_i^* F, |I'|)$, $C(v)$ does not depend on the last $n - |I'|$ positions in π_i^*.

2) Let $I_1, I_2 \subseteq X_n$, $x_j \notin I_1$, $I_2 = I_1 \cup \{x_j\}$, $\pi_1, \pi_2 \in \Pi(I_1)$ where $\pi_1(|I_2|) = \pi_2(|I_2|) = x_j$, and let π_1 be κ-minimal for $|I_1|$.
For shorter notation,

$$\mathrm{coll}_1(\pi_1 F, |I_2|) := \bigodot_{v \in \textsc{Set}(\pi_1 F, |I_2|)} C(v) \text{ and}$$
$$\mathrm{coll}_2(\pi_2 F, |I_2|) := \bigodot_{v \in \textsc{Set}(\pi_2 F, |I_2|)} C(v).$$

It must be

$$\begin{aligned}
&\mathrm{cost}(\pi_1 F, I_1) = \mathrm{cost}(\pi_2 F, I_1) \\
&\quad \Rightarrow \mathrm{acc}(\mathrm{cost}(\pi_1 F, I_1), \mathrm{coll}_1(\pi_1 F, |I_2|)) = \mathrm{acc}(\mathrm{cost}(\pi_2 F, I_1), \mathrm{coll}_2(\pi_2 F, |I_2|)), \\
&\mathrm{cost}(\pi_1 F, I_1) < \mathrm{cost}(\pi_2 F, I_1) \\
&\quad \Rightarrow \mathrm{acc}(\mathrm{cost}(\pi_1 F, I_1), \mathrm{coll}_1(\pi_1 F, |I_2|)) < \mathrm{acc}(\mathrm{cost}(\pi_2 F, I_1), \mathrm{coll}_2(\pi_2 F, |I_2|)).
\end{aligned}$$

Further, let $\mathit{min\_cost}_\emptyset = \mathrm{cost}(F, \emptyset)$. Then the following recurrent equation for min_cost

$$\mathit{min\_cost}_{I'} = \min_{x_i \in I'} \left[\mathrm{acc}(\mathit{min\_cost}_{I' \setminus \{x_i\}}, \bigodot_{v \in \textsc{Set}(\pi_i^* F, |I'|)} C(v)) \right]$$

holds and we have

$$\mathit{min\_cost}_{X_n} = \min_{\pi \in \Pi} \kappa(\pi F).$$

Further, a DP-method to compute $\mathit{min\_cost}_{X_n}$ exists. It is operating on the state space 2^{X_n}.

7 Hard and Feasible Instances of Path-Related Optimization

In the following, the DP schema derived in the previous section is applied to various problems of exact BDD minimization. First the two objective functions α and μ are considered and it is shown that, even with the least restrictive schema, no feasible exact algorithm can be derived for minimization of the number of paths and of MPL. This limits the hope to find a smarter encoding of the original (naive) search space of size $O(n!)$. However, such encodings are strongly desired since they break down the state space to one of a size of $O(2^n)$.

In the past state spaces of this size have been successfully handled for problem instances of moderate size by a number of intelligent pruning techniques, based on paradigms like DP, B&B, and A^* [9, 11, 13].

Then it is shown that feasible DP-based approaches can be derived from the framework for the remaining two problems, exact node minimization and minimization of the EPL in BDDs. Moreover, the DP-based schemas are extended to B&B approaches. This is the first time that a feasible exact method for minimization of the EPL in BDDs is presented.

Theorem 6. *The conditions of Theorem 5 do not hold for any of the known cost functions for α (number of paths in a BDD) and μ (maximal path length in a BDD). Hence, Bellmann's principle is violated and the approach of Theorem 5 can not be applied here, regardless of which of the known cost functions is used.*

Proof. See the Appendix.

As we concentrate on practical algorithms based on a DP formulation, e.g. B&B or A^*, this result does not strictly imply the inexistence of exponential time algorithms for Alpha and Mu. In the remainder of the section it is shown that the schema can be applied successfully to the objective functions ν and ϵ.

Theorem 7. *DP-methods to minimize the objective functions ν (number of nodes in a BDD) and ϵ (expected path length in a BDD) exist. They operate on the state space 2^{X_n} which can be further pruned by B&B.*

Proof. See the Appendix.

8 Experimental Results

In this section, experimental results are presented. All algorithms have been applied to circuits of the LGSynth93 benchmark set [8]. The tested methods target the two objective functions that allow a DP-based B&B-approach following the framework presented in this paper. This includes the exact B&B method for EPL minimization outlined in Section 7 after Theorem 7[3] (called ϵXACT) as well as the approach to EPL-sifting described in [12]. For a comparison in run time and since we were also interested in the EPL of BDDs which have been minimized with respect to the number of nodes, also the best B&B method for exact node minimization called JANUS [11] has been applied.

To put up a testing environment, all algorithms have been integrated into the CUDD package [24]. By this it is guaranteed that they run in the same

[3] Instead of (15) only min_cost_I has been used as a lower bound since otherwise the extra effort of computing the lower bound exceeded the gain in run time for all but the smallest benchmark functions.

system environment. A system with an Athlon processor running at 2.2 GHz, with a main memory of 512 MByte and a run time limit of 36,000 CPU seconds has been used for the experiments.

In a series of experiments, all methods have been applied to the benchmark functions given in Table 1. In the first column the name of the function is given. Column *in* (*out*) gives the number of inputs (outputs) of a function. The next two columns *time* and *space* give the run time in CPU seconds and the space requirement in MByte for the approach JANUS, respectively. The next column *opt. #* shows the minimal numbers of nodes for a BDD representing the respective function. Column ϵ gives the EPL for the respective BDD of minimum size. In the next two columns the same quantities run time and space requirement are given for the method ϵXACT, respectively. The next column *opt.* ϵ gives the optimum ϵ-value for a BDD representing the respective benchmark function. The next two columns show the run time and the space requirement for the approach to EPL-sifting. The last column $\hat{\epsilon}$ gives the heuristic ϵ-value as determined by EPL-sifting, respectively.

The results show that the run times of ϵXACT are generally larger than that of the exact node minimization method JANUS. There are two reasons for that: the BDDs created in intermediate steps during operation of ϵXACT can be significantly larger than those in the size-driven method JANUS. Moreover, ϵXACT needs to maintain an additional node attribute (the ω_ϵ-value) with time-consuming hash table accesses during variable swap operations.

Since the results of an exact approach to EPL-minimization are given, this allows for the evaluation of the previous heuristic approach called EPL-sifting which shows that it performs much faster (up to five orders of magnitude). Most of the time it achieves almost optimal results. However, it can also be observed that the results obtained by ϵXACT show an improvement in the ϵ-value of 9.6% on average. In some cases (see *comp*, *sct*, *cordic*, *t481*, and *vda*) the gain is significant and it can be more than 50% (see *comp*).

9 Conclusions

The exact optimization of BDDs with respect to path-related objective functions has been investigated. First, formal results have been given which show that these functions can be very sensitive to a chosen variable ordering. Second, a generalization of the framework of Bellmann/Held/Karp yielded deeper understanding of the reasons why it is hard to minimize BDDs with respect to the number of paths or to the maximum path length.

On the other hand we successfully derived a new exact algorithm for the expected path length in BDDs. It is a DP-based B&B method that can be obtained by the general framework.

Experimental results showed the feasibility of the exact approach. For the first time it became possible to evaluate a heuristic approach to EPL minimization.

Table 1. Results for expected path length

name	**in**	**out**	JANUS				ϵXACT			EPL-**sifting**		
			time	**space**	**opt. #**	ϵ	**time**	**space**	**opt.** ϵ	**time**	**space**	$\hat{\epsilon}$
cc	21	20	81s	36M	46	2.08	939s	50M	1.78	0.03s	$<$1M	1.78
cm150a	21	1	277s	37M	33	3.50	785s	23M	3.50	0.03s	$<$1M	3.50
cm163a	16	5	0.9s	$<$1M	26	2.34	4.5s	$<$1M	2.34	0.03s	$<$1M	2.34
cmb	16	4	0.3s	$<$1M	28	2.00	0.2s	$<$1M	2.00	0.03s	$<$1M	2.00
comp	32	3	3287s	130M	95	17.33	9419s	108M	4.00	0.13s	$<$1M	9.28
cordic	23	2	1.9s	$<$1M	42	8.92	50s	2M	4.73	0.03s	$<$1M	6.28
cps	24	102	2359s	61M	971	2.84	26335s	96M	2.31	0.10s	$<$1M	2.31
i1	25	16	20s	10M	36	1.76	232s	23M	1.72	0.03s	$<$1M	1.72
lal	26	19	450s	79M	67	2.73	10023s	310M	2.06	0.03s	$<$1M	2.08
mux	21	1	278s	36M	33	3.50	786s	22M	3.50	0.03s	$<$1M	3.50
pcle	19	9	5.2s	3M	42	3.00	169s	10M	2.50	0.03s	$<$1M	2.50
pm1	16	13	0.6s	$<$1M	40	2.16	1.6s	$<$1M	1.74	0.03s	$<$1M	1.75
s208.1	18	9	5.3s	2M	41	3.29	177s	10M	2.69	0.03s	$<$1M	2.69
s298	17	20	8.7s	3M	74	2.14	59s	5M	2.10	0.03s	$<$1M	2.10
s344	24	26	847s	111M	104	2.24	24872s	347M	2.22	0.03s	$<$1M	2.22
s349	24	26	851s	111M	104	2.24	24932s	347M	2.22	0.03s	$<$1M	2.22
s382	24	27	416s	75M	119	3.02	14831s	347M	2.15	0.04s	$<$1M	2.16
s400	24	27	413s	75M	119	3.02	14793s	347M	2.15	0.03s	$<$1M	2.16
s444	24	27	462s	82M	119	3.02	14637s	347M	2.15	0.04s	$<$1M	2.19
s526	24	27	833s	111M	113	2.41	16755s	347M	2.21	0.04s	$<$1M	2.21
s820	23	24	1080s	59M	220	2.60	9374s	93M	2.54	0.04s	$<$1M	2.54
s832	23	24	1127s	59M	220	2.60	9660s	93M	2.54	0.04s	$<$1M	2.55
sct	19	15	6s	3M	48	2.94	191s	10M	2.25	0.03s	$<$1M	2.36
t481	16	1	0.4s	$<$1M	21	9.00	4.5s	$<$1M	8.25	0.03s	$<$1M	9.00
tcon	17	16	0.6s	$<$1M	25	1.50	25s	5M	1.50	0.03s	$<$1M	1.50
ttt2	24	21	521s	82M	107	2.83	16189s	347M	2.55	0.03s	$<$1M	2.55
vda	17	39	30s	3M	478	4.51	512s	6M	4.39	0.05s	$<$1M	4.43

Appendix

Proof of Theorem 6.

Minimization of Number of Paths: The node contribution must be based on the cost function in (7), as all other equations define node contributions which depend on the lower part of the BDD (and thus this would violate Condition 1)). Consequently, the only choice for the cost function that respects Condition 1) is

$$\text{cost} = (\text{last}, \sum, c, \omega_\alpha)$$

First, clearly $\omega_\alpha(v)$ does not depend on the part of the ordering after the position of $\text{var}(v)$, thus Condition 1) is respected. Second, for a BDD πF, it is

$$\text{cost}(\pi F, X_n) = \text{last}(\dots, \sum_{v \in c(\pi F, n)} \omega_\alpha(v))$$

$$= \alpha(\pi F).$$

because of (7). We can choose an arbitrary value as the starting value of the recursion because the accumulation function is the function last. This yields the recurrence:

$$min\_cost_{I'} = \min_{x_i \in I'} \left[\sum_{v \in c(\pi_i^* F, |I'|)} \omega_\alpha(v) \right] \tag{13}$$

where $\pi_i^* \in \Pi_{I' \setminus \{x_i\}}$ such that $\pi_i^*(|I'|) = x_i$ is derived. Note that the equation is recurrent although no terms $min\_cost_{I' \setminus \{x_i\}}$ do occur since π_i^* results from previous steps. In particular notice that the first condition of the general recursion schema already forces these choices.

Next the validity of the second condition is disproven by giving a counter-example (see Fig. 1). It shows that Condition 2) may be violated.

In Fig. 1(a), the horizontal lines cut through the edges after the third and the fourth level. The nodes of the set $c(F, 3)$ are exactly the nodes with cut edges pointing to them. The depicted ordering $\pi_1 = x_1, x_2, x_3, x_4$ for a BDD $\pi_1 F$ is α-minimal for $I = \{x_1, x_2, x_3\}$. This can be seen by inspecting all $3! = 6$ possible permutations of I. We have $\mathrm{cost}_\alpha(\pi_1 F, I) = 8$. In Fig. 1(b), the ordering $\pi_2 = x_2, x_1, x_3, x_4$ for a BDD $\pi_2 F$ representing the same function causes a cost of 9 for I. Now let $I' = \{x_1, x_2, x_3, x_4\}$. It is $\mathrm{cost}_\alpha(\pi_1 F, I') = \mathrm{cost}_\alpha(\pi_2 F, I') = 10$, i.e. a suboptimal sub-ordering does not lead to higher “future” costs. This violates the second implication of Condition 2).

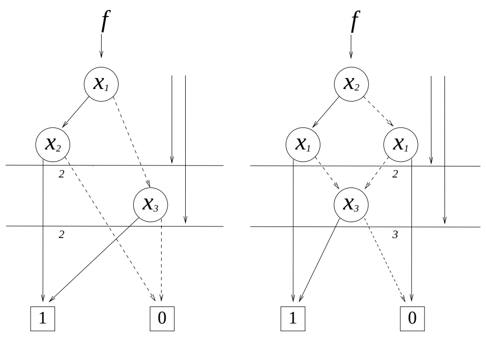
Minimization of Maximal Path Length: The consideration is analogous to that for the number of paths, essentially just $\sum$ is replaced by max and ω_α is replaced by ω_μ.

Again a counter-example shows that Condition 2) may be violated (the other condition again holds), see Fig. 2. In Fig. 2(a) the ordering $\pi_1 = x_1, x_2, x_3$ for a BDD $\pi_1 F$ is μ-minimal for $I = \{x_1, x_2\}$: since the function essentially depends on x_1, x_2, at least one path going through two nodes, one labeled x_1, the other x_2, must exist. This path is of minimal length 2. The ordering x_2, x_1, x_3 in Fig. 2(b) for a BDD $\pi_2 F$ representing the same function also causes a cost for I of 2. However, the cost for $I = \{x_1, x_2, x_3\}$ is 3, whereas it is only 2 in the BDD $\pi_1 F$. This violates the first implication of Condition 2). □

Proof of Theorem 7.

The cost function for the number of nodes is $\mathrm{cost} = (\sum, \sum, \mathrm{nodes}, 1)$ (see Section 4), for the expected path length it is $\mathrm{cost} = (\sum, \sum, \mathrm{nodes}, \omega_\epsilon)$. In both cases the term $min\_cost_\emptyset = 0$ is the starting value of the recursion and it is trivial to show that Conditions 1) and 2) are respected. Hence e.g. $min\_cost_{X_n} = \min_{\pi \in \Pi} |\pi F|$.

By that, essentially the same schemas as in (3) and (5) are obtained (with the minor specialization that π_i is chosen as π_i^*). Both DP-approaches can be turned into B&B methods by the use of lower bounds. In [9], the lower bound



(a) A μ-minimal ordering for $\{x_1, x_2, x_3\}$. (b) A suboptimal ordering for $\{x_1, x_2, x_3\}$.

Fig. 2. Two BDDs for $f = x_1 \cdot x_2 + \overline{x}_1 \cdot x_3$.

$$l\_b = min\_cost_I + \max\{|k(F, |I|)|\,, n - |I|\} + 1 \tag{14}$$

has been proposed. The idea of (14) also directly transfers to EPL-minimization. Here, it is possible to use the lower bound

$$l\_b = min\_cost_I + \sum_{v \in k(F,|I|)} \omega_\epsilon(v). \tag{15}$$

At the end of a step of the outlined DP-approach, all data for a state I for which the lower bound exceeds or equals the current upper bound (which is updated to the minimal BDD size seen so far with every intermediate BDD constructed), can safely be excluded from further consideration. This is because any ordering in $\Pi(I)$ must yield BDD sizes (or sums of ω_ϵ-values) larger than the smallest BDD (the smallest sum) encountered. $\square$

References

1. R. Bellman. *Dynamic Programming.* Princeton University Press, Princeton, New Jersey, 1957.
2. R. Bellmann. Combinatorial processes and dynamic programming. In *Proc. of Symp. in Applied Mathematics of the American Mathematical Society*, 1960.
3. R. Bellmann. Dynamic programming treatment of the traveling salesman problem. *J. Assoc. Comput. Mach.*, (9):61–63, 1962.
4. B. Bollig and I. Wegener. Improving the variable ordering of OBDDs in NP-complete. *IEEE Trans. on Comp.*, 45(9):993–1002, 1996.
5. K. Brace, R. Rudell, and R. Bryant. Efficient implementation of a BDD package. In *Design Automation Conf.*, pages 40–45, 1990.
6. R. E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Trans. on Comp.*, 35(8):677–691, 1986.

7. J. Butler, T. Sasao, and M. Matsuura. Average path length of binary decision diagrams. *IEEE Trans. on Comp.*, 54(9), September 2005.
8. Collaborative Benchmarking Laboratory. *1993 LGSynth Benchmarks*. North Carolina State University, Department of Computer Science, 1993.
9. R. Drechsler, N. Drechsler, and W. Günther. Fast exact minimization of BDDs. *IEEE Trans. on CAD*, 19(3):384–389, 2000.
10. R. Drechsler, J. Shi, and G. Fey. Synthesis of fully testable circuits from BDDs. *IEEE Trans. on CAD*, 23(3):440–443, 2004.
11. R. Ebendt, W. Günther, and R. Drechsler. An improved branch and bound algorithm for exact BDD minimization. *IEEE Trans. on CAD*, 22(12):1657–1663, 2003.
12. R. Ebendt, W. Günther, and R. Drechsler. Minimization of the expected path length in BDDs based on local changes. In *Asian and South Pacific Design Automation Conf.*, pages 866–871, 2004.
13. R. Ebendt, W. Günther, and R. Drechsler. Combining ordered-best first search with branch and bound for exact BDD minimization. *IEEE Trans. on CAD 2005*, 24(10):1515–1529, 2005.
14. G. Fey and R. Drechsler. Minimizing the number of paths in BDDs - theory and algorithm. *IEEE Trans. on CAD*, 25(1):4–11, 2006.
15. S. Friedman and K. Supowit. Finding the optimal variable ordering for binary decision diagrams. *IEEE Trans. on Comp.*, 39(5):710–713, 1990.
16. M. Held and R. Karp. A dynamic programming approach to sequencing problems. *J. Soc. Indust. Appl. Math.*, 10(1), 1962.
17. Y. Iguchi, T. Sasao, and M. Matsuura. Evaluation of multiple-output logic functions using decision diagrams. In *Asian South Pacific Design Automation Conf.*, pages 312–315, 2003.
18. Y. Jiang, S. Matic, and R. Brayton. Generalized cofactoring for logic function evaluation. In *Design Automation Conf.*, pages 155–158, 2003.
19. P. McGeer, K. McMillan, A. Saldanha, A. Sangiovanni-Vincentelli, and P. Scaglia. Fast discrete function evaluation using decision diagrams. In *Int'l Conf. on CAD*, pages 402–407, 1995.
20. S. Nagayama, A. Mishchenko, T. Sasao, and J. Butler. Minimization of average path length in BDDs by variable reordering. In *Proc. of International Workshop on Logic and Synthesis*, 2003.
21. S. Nagayama and T. Sasao. On the minimization of longest path length for decision diagrams. Proc. of International Workshop on Logic and Synthesis, 2004.
22. S. Reda, R. Drechsler, and A. Orailoglu. On the relation between SAT and BDDs for equivalence checking. In *Int'l Symp. on Quality of Electronic Design*, pages 394–399, 2002.
23. R. Rudell. Dynamic variable ordering for ordered binary decision diagrams. In *Int'l Conf. on CAD*, pages 42–47, 1993.
24. F. Somenzi. *CU Decision Diagram Package Release 2.4.0.* University of Colorado at Boulder, 2004.