Using Edge-Valued Decision Diagrams for Symbolic Generation of Shortest Paths*

Gianfranco Ciardo Radu Siminiceanu

College of William and Mary,
Williamsburg, Virginia 23187
{ciardo,radu}@cs.wm.edu

Abstract. We present a new method for the symbolic construction of shortest paths in reachability graphs. Our algorithm relies on a variant of edge–valued decision diagrams that supports efficient fixed–point iterations for the joint computation of both the reachable states and their distance from the initial states. Once the distance function is known, a shortest path from an initial state to a state satisfying a given condition can be easily obtained. Using a few representative examples, we show how our algorithm is vastly superior, in terms of both memory and space, to alternative approaches that compute the same information, such as ordinary or algebraic decision diagrams.

1 Introduction

Model checking [13] is an exhaustive, fully automated approach to formal verification. Its ability to provide counterexamples or witnesses for the properties that are checked makes it increasingly popular. In many cases, however, this feature is the most time– and space–consuming stage of the entire verification process. For example, [15] shows how to construct traces for queries expressed in the temporal logic CTL [11] under fairness constraints. Another direction is taken in SAT–based model checking, where satisfiabilty checkers are used to find shortest–length counterexamples (as is the case of the bounded model checking technique [4]), conduct the entire reachability analysis [1], or combine the state–space exploration method with SAT solvers [24].

Since a trace is usually meant to be examined by a human, it is particularly desirable for a model–checking tool to compute a minimal–length trace. Unfortunately, finding such trace is an NP-complete problem [17], thus a sub–optimal trace is sought in most cases. For some operators, finding minimal–length witnesses is instead easy in principle. An example is the EF operator, which is closely related to the (backward) reachability relation: a state satisfies EFp if there is an execution path from it to a state where property p holds. Even using symbolic encodings [7], though, the generation and storage of the sets of states required to generate an EF witness can be a major limitation in practice.

* Work supported in part by the National Aeronautics and Space Administration under NASA Grants NAG-1-2168 and NAG-1-02095.

Our goal is then to adapt a very fast and memory–efficient state–space generation algorithm we recently developed [10] and endow the symbolic data structure with information that captures the minimum distance of each state from any of the initial states. Knowledge of this distance significantly simplifies the generation of shortest–length EF witnesses. To encode this information, we employ a variant of the edge–valued decision diagrams [21], appropriately generalized so that it is applicable to our fast state–space generation strategy. We show that the new variant we define is still canonical, and emphasize the importance of using edge–values, which give us increased flexibility when performing guided fixed–point iterations.

The paper is organized as follows. Section 2 defines basic concepts in discrete–state systems, ordinary and edge–valued decision diagrams, state–space generation, and traces, and formulates the one–to–many shortest path problem. Section 3 introduces our extensions to edge–valued decision diagrams, including a different type of canonical form, EV$^+$MDDs. Section 4 discusses the efficient manipulation of EV$^+$MDDs and our algorithm for constructing the distance function. Section 5 evaluates the performance of the new data structure and algorithm by comparing them with existing technologies: regular and algebraic decision diagrams. Section 6 concludes with final remarks and future research directions.

2 State spaces, decision diagrams, and distances

A discrete–state model is a triple $(\widehat{\mathcal{S}}, \mathcal{X}_{init}, \mathcal{N})$, where the discrete set $\widehat{\mathcal{S}}$ is the *potential state space* of the model; the set $\mathcal{X}_{init} \subseteq \widehat{\mathcal{S}}$ contains the *initial states*; and $\mathcal{N} : \widehat{\mathcal{S}} \to 2^{\widehat{\mathcal{S}}}$ is the *transition function* specifying which states can be reached from a given state in one step, which we extend to sets: $\mathcal{N}(\mathcal{X}) = \bigcup_{i \in \mathcal{X}} \mathcal{N}(i)$. We consider structured systems modeled as a collection of K *submodels*. A (global) system state i is then a K-tuple $(i_K, \ldots, i_1)$, where i_k is the *local state* for submodel k, for $K \geq k \geq 1$, and $\widehat{\mathcal{S}}$ is given by $\mathcal{S}_K \times \cdots \times \mathcal{S}_1$, the cross–product of K local state spaces $\mathcal{S}_k$, which we identify with $\{0, \ldots, n_k - 1\}$ since we assume that $\widehat{\mathcal{S}}$ is finite. The *(reachable) state space* $\mathcal{S} \subseteq \widehat{\mathcal{S}}$ is the smallest set containing $\mathcal{X}_{init}$ and closed with respect to $\mathcal{N}$, i.e.:

$$\mathcal{S} = \mathcal{X}_{init} \cup \mathcal{N}(\mathcal{X}_{init}) \cup \mathcal{N}(\mathcal{N}(\mathcal{X}_{init}) \cup \cdots = \mathcal{N}^*(\mathcal{X}_{init}).$$

Thus, $\mathcal{S}$ is the fixed point of equation $\mathcal{S} = \mathcal{N}(\mathcal{S})$ when $\mathcal{S}$ is initialized to $\mathcal{X}_{init}$.

2.1 Decision diagrams

It is well known that the state spaces of realistic models are enormous, and that decision diagrams are an effective way to cope with this *state–space explosion* problem. Their boolean incarnation, binary decision diagrams (BDDs) [5], can compactly encode boolean functions of K variables, hence subsets of $\{0, 1\}^K$, which can then be manipulated very efficiently. BDDs have been successfully employed to verify digital circuits and other types of synchronous and asynchronous systems. In the last decade, their application has expanded to areas of computer science beyond computer–aided verification. A comprehensive overview of decision diagrams is presented in [14].

We consider exclusively *ordered* decision diagrams (the variables labelling nodes along any path from the root must follow the order $i_K, \ldots, i_1$) that are either *reduced* (no duplicate nodes and no node with all edges pointing to the same node, but edges possibly spanning multiple levels) or *quasi–reduced* (no duplicate nodes, and all edges spanning exactly one level), either form being *canonical*. We adopt the extension of BDDs to integer variables, i.e., multi–valued decision diagrams (MDDs) [19], an example of which is in Figure 1. MDDs are often more naturally suited than BDDs to represent the state space of arbitrary discrete systems, since no binary encoding must be used to represent the local states for level k when $n_k > 2$. An even more important reason to use MDDs in our work, as it will be apparent, is that they better allow us to exploit the *event locality* present in systems exhibiting a globally–asynchronous locally–synchronous behavior. When combined with the Kronecker representation of the transition relation inspired by [2] and applied in [9, 22], MDDs accommodate different fixed–point iteration strategies that result in remarkable efficiency improvements [10].

To discuss locality in a structured model, we require a *disjunctively–partitioned* transition function [18], i.e., $\mathcal{N}$ must be a union of (asynchronous) transition functions: $\mathcal{N}(i_K, \ldots, i_1) = \bigcup_{e \in \mathcal{E}} \mathcal{N}_e(i_K, \ldots, i_1)$, where $\mathcal{E}$ is a finite set of *events* and $\mathcal{N}_e$ is the transition function associated with event e. Furthermore, we must be able to express each transition function $\mathcal{N}_e$ as the cross–product of K local transition functions: $\mathcal{N}_e(i_K, \ldots, i_1) = \mathcal{N}_{e,K}(i_K) \times \cdots \times \mathcal{N}_{e,1}(i_1)$. This is a surprisingly natural requirement: for example, it is satisfied by any Petri net [23], regardless of how it is decomposed into K subnets (by partitioning its places into K sets). Moreover, if a given model does not exhibit this behavior, we can always coarsen K or refine $\mathcal{E}$ so that it does. If we identify $\mathcal{N}_{e,k}$ with a boolean matrix of size $n_k \times n_k$, where entry (i_k, j_k) is 1 iff $j_k \in \mathcal{N}_{e,k}(i_k)$, the overall transition relation is encoded by the boolean *Kronecker* expression $\sum_{e \in \mathcal{E}} \bigotimes_{K \geq k \geq 1} \mathcal{N}_{e,k}$. We say that event e *affects* level k if $\mathcal{N}_{e,k}$ is not the identity, we denote the top and bottom levels affected by e with $Top(e)$ and $Bot(e)$, respectively, and we let $\mathcal{E}_k = \{e \in \mathcal{E} : Top(e) = k\}$.

2.2 Symbolic state–space generation: breadth–first vs. saturation

The traditional approach to generating the reachable states of a system is based on a breadth–first traversal, as derived from classical fixed–point theory, and applies a monolithic $\mathcal{N}$ (even when encoded as $\bigcup_{e \in \mathcal{E}} \mathcal{N}_e$): after d iterations, the currently–known state space contains exactly all states whose distance from any state in $\mathcal{X}_{init}$ is at most d. However, recent advances have shown that non–BFS, guided, or chaotic [16], exploration can result in a better iteration strategy.

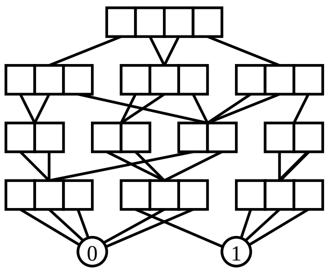
An example is the *saturation* algorithm introduced in [10], which exhaustively *fires* (explores) all events of $\mathcal{E}_k$ in an MDD node at level k, thereby bringing it to its final "saturated" form. We only briefly summarize the main characteristics of

$\mathcal{S}_4 = \{0,1,2,3\}$

$\mathcal{S}_3 = \{0,1,2\}$

$\mathcal{S}_2 = \{0,1\}$

$\mathcal{S}_1 = \{0,1,2\}$



$\mathcal{S} = \{0210, 1000, 1010, 1100, 1110, 1210, 2000, 2010, 2100, 2110, 2210, 3010, 3110, 3200, 3201, 3202, 3210, 3211, 3212\}$

Fig. 1. A 4-level MDD on $\{0{,}1{,}2{,}3\}\times\{0{,}1{,}2\}\times\{0{,}1\}\times\{0{,}1{,}2\}$ and the encoded set $\mathcal{S}$.

saturation in this section, since the algorithm we present in Section 4.1 follows the same idea, except it is applied to a richer data structure.

Saturation considers the nodes in a bottom–up fashion, i.e., when a node is processed, all its descendants are already known to be saturated. There are major advantages in working with saturated nodes. A saturated node at level k encodes a fixed point with respect to events in $\mathcal{E}_k \cup \ldots \cup \mathcal{E}_1$, thus it need not be visited again when considering such events. By contrast, traditional symbolic algorithms manipulate and store a large number of non–saturated nodes; these nodes cannot be present in the encoding of the final state space, thus will necessarily be deleted before reaching the fixed–point and replaced by (saturated) nodes encoding a larger subspace. Similar advantages apply to the manipulation of the auxiliary data structures used in any symbolic state–space generation algorithm, the *unique table* and the *operation cache*: only saturated nodes are inserted in them, resulting in substantial memory savings. Exploring a node exhaustively once, instead of once per iteration, also facilitates the idea of *in–place–updates*: while traditional algorithms frequently create updated versions of a node, to avoid using stale unique table and cache entries, saturation only checks–in a node when all possible updates on it have been performed.

Experimental studies [10] show that our saturation strategy performs orders of magnitude faster than previous algorithms. Even more important, its peak memory requirements are often very close to the final requirements, unlike traditional approaches where the memory consumption grows rapidly until midway through the exploration, only to drop sharply in the last phases. Our next challenge for saturation is then applying it to other types of symbolic computation, such as the one discussed in this paper: the generation of shortest–length traces, where the use of chaotic iteration strategies would not seem applicable at first.

2.3 The distance function

The *distance* of a reachable state $i \in \mathcal{S}$ from the set of initial states $\mathcal{X}_{init}$ is defined as $\delta(i) = \min\{d : i \in \mathcal{N}^d(\mathcal{X}_{init})\}$. We can naturally extend $\delta : \mathcal{S} \to \mathbb{N}$ to all states in $\widehat{\mathcal{S}}$ by letting $\delta(i) = \infty$ for any non–reachable state $i \in \widehat{\mathcal{S}} \setminus \mathcal{S}$. Alternatively, given such a function $\delta : \widehat{\mathcal{S}} \to \mathbb{N} \cup \{\infty\}$, we can identify $\mathcal{S}$ as the subset of the domain where the function is finite: $\mathcal{S} = \{i \in \widehat{\mathcal{S}} : \delta(i) < \infty\}$.

The formulation of our problem is then: Given a description of a structured discrete–state system $(\widehat{\mathcal{S}}, \mathcal{X}_{init}, \mathcal{N})$, determine the distance to all reachable states, i.e., compute and store $\delta : \widehat{\mathcal{S}} \to \mathbb{N} \cup \{\infty\}$ (note that the reachable state space $\mathcal{S}$ is not an input, rather, it is implicitly an output). This can be viewed as a least fixed–point computation for the functional $\Phi : \mathcal{D} \to \mathcal{D}$, where $\mathcal{D}$ is the set of functions mapping $\widehat{\mathcal{S}}$ onto $\mathbb{N} \cup \{\infty\}$. In other words, Φ refines an approximation of the distance function from the initial $\delta^{[0]} \in \mathcal{D}$, defined as $\delta^{[0]}(i) = 0$, if $s \in \mathcal{X}_{init}$, $\delta^{[0]}(i) = \infty$ otherwise, via the iteration

$$\delta^{[m+1]}(i) = \Phi(\delta^{[m]})(i) = \min\left(\delta^{[m]}(i), \min\left\{1 + \delta^{[m]}(i') \;\middle|\; i \in \mathcal{N}(i')\right\}\right).$$

Note that the state–space construction is itself a fixed–point computation, so we seek now to efficiently combine the two fixed–point operations into one. Before showing our algorithm to accomplish this, in Section 3, we first describe a few approaches to compute distance information based on existing decision diagrams technology.

2.4 Explicit encoding of state distances

Algebraic decision diagrams (ADDs) [3] are an extension of BDDs where multiple terminals are allowed (thus, they are also called MTBDDs [12]). ADDs can encode arithmetic functions from $\widehat{\mathcal{S}}$ to $\mathbb{R} \cup \{\infty\}$. The value of the function on a specific input (representing a state in our case) is the value of the terminal node reached by following the path encoding the input. While ADDs are traditionally associated to boolean argument variables, extending the arguments to finite integer sets is straightforward.

The compactness of the ADD representation is related to the merging of nodes, exploited to a certain degree in all decision diagrams. In this case, there is a unique root, but having many terminal values can greatly reduce the degree of node merging, especially at the lower levels, with respect to the *support decision diagram*, i.e., the MDD that encodes $\mathcal{S} \subseteq \widehat{\mathcal{S}}$. In other words, the number of terminal nodes for the ADD that encodes $\delta : \widehat{\mathcal{S}} \to \mathbb{N} \cup \{\infty\}$ equals the number of distinct values for δ (hence the "explicit" in the title of this section); if we merged all finite–valued terminals into one, thus encoding just $\mathcal{S}$ but not the state distances, many ADD nodes may be merged into one MDD node.

An alternative explicit encoding of state distances can be achieved by simply using a forest of MDDs. This approach is derived from the traditional ROBDD method, by extending it to multi–valued variables. Each of the distance sets $\mathcal{N}^d(\mathcal{X}_{init}) = \{i \in \mathcal{S} \mid \delta(i) = d\}$ (or $\{i \in \mathcal{S} \mid \delta(i) \leq d\}$, which may require fewer nodes in some cases) can be encoded using a separate MDD. Informally, this reverses the region where most sharing of nodes occurs compared to ADDs: the roots are distinct, but they may be likely to share nodes downstream.

The cardinality of the range of the function is critical to the compactness of either representation: the wider the range, the less likely it is that nodes are merged. Figure 2 (a) and (b) show an example of the same distance function represented as an ADD or as a forest of MDDs, respectively.

i_3	0	0	0	0	1	1	1	1
i_2	0	0	1	1	0	0	1	1
i_1	0	1	0	1	0	1	0	1
f	0	2	3	2	2	4	1	0

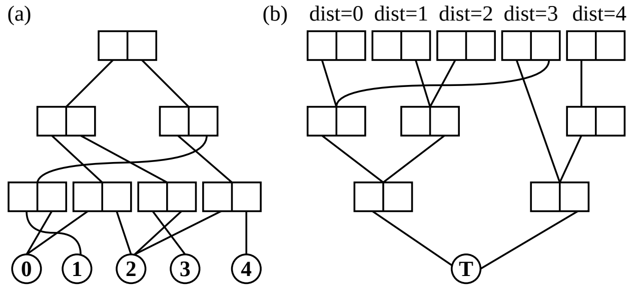


Fig. 2. Storing the distance function: an ADD (a) vs. a forest of MDDs (b).

2.5 Symbolic encoding of state distances

The idea of associating numerical values to the edges of regular BDDs was proposed in [20, 21], resulting in a new type of decision diagrams, edge–valued BDDs (EVBDDs)[1]. In the following definition of EVBDDs, instead of using the original terminology and notation, we use the terminology and notation needed to introduce the new data structure presented in the next section, so that differences and similarities will be more apparent.

Definition 1. An EVBDD is a directed acyclic graph that encodes a total function $f : \{0,1\}^K \to \mathbb{Z}$ as follows:

1. There is a single terminal node, at level 0, with label 0, denoted by $\langle 0|0\rangle$.
2. A non–terminal node at level k, $K \geq k \geq 1$, is denoted by $\langle k|p\rangle$, where p is a unique identifier within level k, and has two children, $\langle k|p\rangle[0].child$ and $\langle k|p\rangle[1].child$ (corresponding to the two possible values of i_k) which are nodes at some (not necessarily the same) level $l, k > l \geq 0$.
3. The 1-edge is labelled with an integer value $\langle k|p\rangle[1].val \in \mathbb{Z}$, while **the label of $\langle k|p\rangle[0].val$ is always (implicitly) 0**.
4. There is a single *root* node $\langle k_r|r\rangle$, for some $K \geq k_r \geq 0$, with no incoming edges, except for a "dangling" edge labelled with an integer value $\rho \in \mathbb{Z}$.
5. Canonicity restrictions analogous to those of reduced ordered BDDs apply:
   uniqueness: if $\langle k|p\rangle[0].child = \langle k|q\rangle[0].child$, $\langle k|p\rangle[1].child = \langle k|q\rangle[1].child$, and $\langle k|p\rangle[1].val = \langle k|q\rangle[1].val$, then $p = q$;
   reducedness: there is no *redundant* node $\langle k|p\rangle$ satisfying $\langle k|p\rangle[0].child = \langle k|p\rangle[1].child$ and $\langle k|p\rangle[1].val = 0$.

The function encoded by an EVBDD node $\langle k|p\rangle$ is recursively defined by

$$f_{\langle k|p\rangle}(i_k,\ldots,i_1) = \begin{cases} f_{\langle k|p\rangle[0].child}(i_l,\ldots,i_1) & \text{if } i_k = 0 \\ f_{\langle k|p\rangle[1].child}(i_r,\ldots,i_1) + \langle k|p\rangle[1].val & \text{if } i_k = 1 \end{cases}$$

[1] We observe that also binary moment diagrams (BMDs), independently introduced in [6], associate values to edges. For BMDs however, evaluating the function on a particular argument requires the traversal of multiple paths, as opposed to a unique path for EVBDDs. Thus, while very effective for verifying circuits such as a multiplier, BMDs are not as suited for our approach.

i_3	0	0	0	0	1	1	1	1
i_2	0	0	1	1	0	0	1	1
i_1	0	1	0	1	0	1	0	1
f	0	2	3	2	2	4	1	0

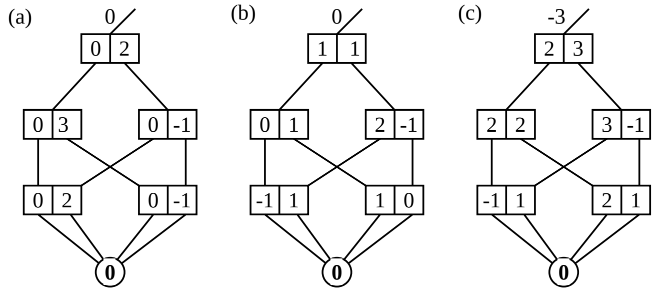


Fig. 3. Canonical (a) and non–canonical (b),(c) EVBDDs for the same function f.

where l and r are the levels of $\langle k|p\rangle[0].child$ and $\langle k|p\rangle[1].child$, respectively, and $f_{\langle 0|0\rangle} = 0$. The function encoded by an EVBDD *edge*, that is, a (*value*,*node*) pair is then simply obtained by adding the constant *value* to the function encoded by the *node*. In particular, the function encoded by the EVBDD is $f = \rho + f_{\langle k_r|r\rangle}$.

Note that the nodes are *normalized* to enforce canonicity: the value of the 0-edge is always 0. If this requirement were relaxed, there would be an infinite number of EVBDDs representing the same function, obtained by rearranging the edge values. An example of multiple ways to encode the function of Figure 2 with non–canonical EVBDDs is shown in Figure 3, where, for better readability, we show the edge value in the box from where the edge departs, except for the top dangling arc. Only the EVBDD in Figure 3(a) is normalized. This node normalization implies that $\rho = f(0, \ldots, 0)$ and may require the use of both negative and positive edge values even when the encoded function is non–negative, as is the case for Figure 3(a). More importantly, if we want to represent functions such our distance $\delta : \widehat{\mathcal{S}} \to \mathbb{N} \cup \{\infty\}$, we can allow edge values to be ∞; however, if $\delta(0, \ldots, 0) = \infty$, i.e., state $(0, \ldots, 0)$ is not reachable, we cannot enforce the required normalization, since this implies that ρ is ∞, and f is identically ∞ as well. This prompted us to introduce a more general normalization rule, which we present next.

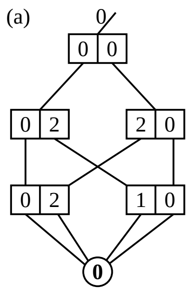
3 A new approach

We use quasi–reduced, ordered, non–negative edge–valued, multi–valued decision diagrams. To the best of our knowledge, this is the first attempt to use edge–valued decision diagrams of any type in fixed–point computations or in the generation of traces.

3.1 Definition of EV$^+$MDDs

We extend EVBDDs in several ways. The first extension is straightforward: from binary to multi–valued variables. Then, we change the normalization of nodes to a slightly more general one needed for our task. Finally, we allow the value of

i_3	0	0	0	0	1	1	1	1
i_2	0	0	1	1	0	0	1	1
i_1	0	1	0	1	0	1	0	1
f	0	2	3	2	2	4	1	0



i_1	0	0	0	0	1	1	1	1
i_2	0	0	1	1	0	0	1	1
i_3	0	1	0	1	0	1	0	1
f	0	2	3	∞	∞	4	1	0

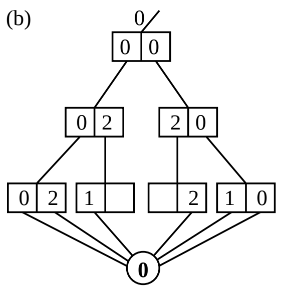


Fig. 4. Storing total (a) and partial (b) arithmetic functions with EV$^+$MDDs.

an edge to be ∞, since this is required to describe our distance functions. Note that the choice to use quasi–reduced instead of reduced decision diagrams is not dictated by limitations in the descriptive power of EVBDDs, but by efficiency considerations in the saturation–based algorithm we present in Section 4.

Definition 2. Given a function $f : \widehat{\mathcal{S}} \to \mathbb{Z} \cup \{\infty\}$, an EV$^+$MDD for $f \neq \infty$ is a directed acyclic graph with labelled edges that satisfies the following properties:

1. There is a single terminal node, at level 0, with label 0, denoted by $\langle 0|0\rangle$.
2. A non–terminal node at level k, $K \geq k \geq 1$, is denoted by $\langle k|p\rangle$, where p is a unique identifier within the level, and has $n_k \geq 2$ edges to children, $\langle k|p\rangle[i_k].child$, labelled with values $\langle k|p\rangle[i_k].val \in \mathbb{N} \cup \{\infty\}$, for $0 \leq i_k < n_k$.
3. If $\langle k|p\rangle[i_k].val = \infty$, the value of $\langle k|p\rangle[i_k].child$ is irrelevant, so we simply require it to be 0 for canonicity; otherwise, $\langle k|p\rangle[i_k].child$ is the index of a node at level $k-1$.
4. There is a single root node, $\langle K|r\rangle$, with no incoming edges, except for a "dangling" incoming edge labelled with an integer value $\rho \in \mathbb{Z}$.
5. **Each non–terminal node has at least one outgoing edge labelled with 0**.
6. All nodes are unique, i.e., if $\forall i_k, 0 \leq i_k < n_k, \langle k|p\rangle[i_k].child = \langle k|q\rangle[i_k].child$, $\langle k|p\rangle[i_k].val = \langle k|q\rangle[i_k].val$, then $p = q$.

Figure 4 shows two EV$^+$MDDs storing a total and a partial[2] function, respectively (the total function encoded is that of Figures 2 and 3). Note that, unlike the normalization for EVBDDs, our normalization requires that the labels on (non–dangling) edges be non–negative, and at least one per node be zero, but not in a pre–determined location; compare the EVBDD of Figure 3(a) with the equivalent EV$^+$MDD of Figure 4(a). The function encoded by the EV$^+$MDD node $\langle k|p\rangle$ is

$$f_{\langle k|p\rangle}(i_k, \ldots, i_1) = \langle k|p\rangle[i_k].val + f_{\langle k-1|\langle k|p\rangle[i_k].child\rangle}(i_{k-1}, \ldots, i_1)$$

[2] By "partial, we mean that some of its values can be ∞; whenever this is the case, we omit the corresponding value and edge from the graphical representation.

and we let $f_{\langle 0|0\rangle} = 0$. As for EVBDDs, the function encoded by the EV$^+$MDD $(\rho, \langle K|r\rangle)$ is $f = \rho + f_{\langle K|r\rangle}$. However, now, $\rho = \min\{f(i) : i \in \mathcal{S}_k \times \cdots \times \mathcal{S}_1\}$. In our application, we will encode distances, which are non–negative, thus $\rho = 0$. If we wanted to cope with the degenerate case $\mathcal{X}_{init} = \emptyset$, so that f is identically ∞, we could allow a special EV$^+$MDD with $\rho = \infty$ and root $\langle 0|0\rangle$.

3.2 Canonicity of EV$^+$MDDs

Lemma 1. From every non–terminal EV$^+$MDD node, there is an outgoing path with all edges labelled 0 reaching $\langle 0|0\rangle$.

Corollary 1. The function $f_{\langle k|p\rangle}$ encoded by a node $\langle k|p\rangle$ is non–negative and $\min(f_{\langle k|p\rangle}) = 0$.

Definition 3. The graphs rooted at two EV$^+$MDD nodes $\langle k|p\rangle$ and $\langle k|q\rangle$ are isomorphic if there is a bijection b from the nodes of the first graph to the nodes of the second graph such that, for each node $\langle l|s\rangle$ of the first graph and each $i_l \in \mathcal{S}_l$ (with $k \geq l \geq 1$):

$$b(\langle l|s\rangle)[i_l].child = b(\langle l|s\rangle[i_l].child) \quad \text{and} \quad b(\langle l|s\rangle)[i_l].val = \langle l|s\rangle[i_l].val.$$

Theorem 1. (*Canonicity*) If two EV$^+$MDDs $(\rho_1, \langle K|r_1\rangle)$ and $(\rho_2, \langle K|r_2\rangle)$ encode the same function $f : \widehat{\mathcal{S}} \to \mathbb{N} \cup \{\infty\}$, then $\rho_1 = \rho_2$ and the two labelled graphs rooted at $\langle K|r_1\rangle$ and $\langle K|r_2\rangle$ are isomorphic.

Proof. It is easy to see that, since the value on the dangling edges of the two EV$^+$MDDs equals the minimum value ρ the encoded function f can assume, we must have $\rho_1 = \rho_2 = \rho$, and the two nodes $\langle K|r_1\rangle$ and $\langle K|r_2\rangle$ must encode the same function $f - \rho$. We then need to prove by induction that, if two generic EV$^+$MDD nodes $\langle k|p\rangle$ and $\langle k|q\rangle$ encode the same function, the labelled graphs rooted at them are isomorphic.
Basis ($k = 1$): if $\langle 1|p\rangle$ and $\langle 1|q\rangle$ encode the same function $f : \mathcal{S}_1 \to \mathbb{N} \cup \{\infty\}$, $\langle 1|p\rangle[i_1].child = \langle 1|q\rangle[i_1].child = 0$ and $\langle 1|p\rangle[i_1].val = \langle 1|q\rangle[i_1].val = f(i_1)$ for all $i_1 \in \mathcal{S}_1$, thus the two labelled graphs rooted at $\langle 1|p\rangle$ and $\langle 1|q\rangle$ are isomorphic.
Inductive step (assume claim true for $k-1$): if $\langle k|p\rangle$ and $\langle k|q\rangle$ encode the same function $f : \mathcal{S}_k \times \cdot \times \mathcal{S}_1 \to \mathbb{N} \cup \{\infty\}$, consider the function obtained when we fix i_k to a particular value t, i.e., $f_{i_k=t}$. Let g and h be the functions encoded by $\langle k|p\rangle[t].child$ and $\langle k|q\rangle[t].child$, respectively; also, let $\langle k|p\rangle[t].val = \alpha$ and $\langle k|q\rangle[t].val = \beta$, and observe that the functions $\alpha{+}g$ and $\beta{+}h$ must coincide with $f_{i_k=t}$. However, because of Corollary 1, we know that both the g and h evaluate to 0, their minimum possible value, for at least one choice of the arguments $(i_{k-1}, \ldots, i_1)$. Thus, the minimum of values $\alpha{+}g$ and $\beta{+}h$ can have are α and β, respectively; since $\alpha{+}g$ and $\beta{+}h$ are the same function, they must have the same minimum, hence $\alpha = \beta$. This implies that $g = h$ and, by inductive hypothesis, that $\langle k|p\rangle[t].child$ and $\langle k|q\rangle[t].child$ are isomorphic. Since this argument applies to a generic child t, the two nodes $\langle k|p\rangle$ and $\langle k|q\rangle$ are then themselves isomorphic, completing the proof. □

```
UnionMin(k : level, (α, p) : edge, (β, q) : edge) : edge
 1 if $\alpha = \infty$ then return $(\beta, q)$;
 2 if $\beta = \infty$ then return $(\alpha, p)$;
 3 if $k = 0$ then return $(\min(\alpha, \beta), 0)$;       • the only node at level $k = 0$ has index $0$
 4 if $UCacheFind(k, p, q, \alpha-\beta, (\gamma, u))$ then     • match $(k, p, q, \alpha-\beta)$, return $(\gamma, u)$
 5   return $(\gamma + \min(\alpha, \beta), u)$;
 6 $u \leftarrow NewNode(k)$;                   • create new node at level $k$ with edges set to $(\infty, 0)$
 7 $\mu \leftarrow \min(\alpha, \beta)$;
 8 for $i_k = 0$ to $n_k - 1$ do
 9   $p' \leftarrow \langle k|p\rangle.child[i_k]$; $\alpha' \leftarrow \alpha - \mu + \langle k|p\rangle.val[i_k]$;
10   $q' \leftarrow \langle k|q\rangle.child[i_k]$; $\beta' \leftarrow \beta - \mu + \langle k|q\rangle.val[i_k]$;
11   $\langle k|u\rangle[i_k] \leftarrow UnionMin(k-1, (\alpha', p'), (\beta', q'))$;         • continue downstream
12 $CheckInUniqueTable(k, u)$;
13 $UCacheInsert(k, p, q, \alpha - \beta, (0, u))$; Value was $\mu$ in the published version, not $0$.
   There is actually no need to store this value in the cache, as it is always $0$.
14 return $(\mu, u)$;
```

Fig. 5. The *UnionMin* algorithm for EV$^+$MDDs.

4 Operations with EV$^+$MDDs

We are now ready to discuss manipulation algorithms for EV$^+$MDDs. We do so in the context of our state–space and distance generation problem, although, of course, the *UnionMin* function we introduce in Figure 5 has general applicability. The types and variables used in the pseudo–code of Figures 5 and 7 are *event* (model event, e), *level* (EV$^+$MDD level, k), *index* (node index within a level, p, q, p', q', s, u, f), *value* (edge value, α, β, α', β', μ, γ, ϕ), *local* (local state index i_k, j_k), and *localset* (set of local states for one level, $\mathcal{L}$). In addition, we let *edge* denote the pair $(value, index)$, i.e., the type of $\langle k|p\rangle[i]$; note that only *index* is needed to identify a child, since the level itself is known: $k-1$.

The *UnionMin* algorithm computes the minimum of two partial functions. This acts like a dual operator by performing the union on the support sets of states of the two operands (which must be defined over the same potential state space $\widehat{\mathcal{S}}$), and by finding the minimum value for the common elements. The algorithm starts at the roots of the two operand EV$^+$MDDs, and recursively descends along matching edges. If at some point one of the edges has value ∞, the recursion stops and returns the other edge (since ∞ is the neutral value with respect to the minimum); if the other edge has value ∞ as well, the returned value is $(\infty, 0)$, i.e., no states are added to the union; otherwise, if the other edge has finite value, we have just found states reachable in one set but not in the other. If the recursion reaches instead all the way to the terminal node $\langle 0|0\rangle$, the returned value is the minimum of the two input values α and β.

If both α and β are finite and p and q are non–terminal, *UnionMin* "keeps" the minimum value on the incoming arcs to the operands, μ, and "pushes down" any residual value $\alpha - \mu$, if $\mu = \beta < \alpha$, or $\beta - \mu$, if $\mu = \alpha < \beta$, on the children of

i_3	0	0	0	0	1	1	1	1	2	2	2	2
i_2	0	0	1	1	0	0	1	1	0	0	1	1
i_1	0	1	0	1	0	1	0	1	0	1	0	1
f	0	∞	2	∞	2	∞	∞	1	3	∞	∞	2
g	0	2	∞	∞	2	4	∞	∞	1	3	∞	3
h	0	2	2	∞	2	4	∞	1	1	3	∞	2

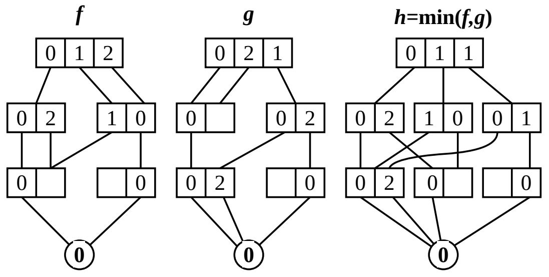


Fig. 6. An example of the *UnionMin* operator for EV$^+$MDDs.

p or q, respectively, in its recursive downstream calls. In this case, the returned edge (μ, u) is such that $\mu + f_{\langle k|u\rangle} = \min(\alpha + f_{\langle k|p\rangle}, \beta + f_{\langle k|q\rangle})$.

An example of the application of the *UnionMin* algorithm is illustrated in Figure 6. The potential state space is $\mathcal{S}_3 \times \mathcal{S}_2 \times \mathcal{S}_1 = \{0,1,2\} \times \{0,1\} \times \{0,1\}$. The functions encoded by the operands, f and g, are listed in the table to the left, along with the result function $h = \min(f, g)$.

Lemma 2. The call *UnionMin*$(k, (\alpha, p), (\beta, q))$ returns an edge (μ, u) such that $\mu = \min(\alpha, \beta)$ and $\langle k|u\rangle$ and its descendants satisfy property 5 of Definition 2, if $\langle k|p\rangle$ and $\langle k|q\rangle$ do.

Proof. It is immediate to see that $\mu = \min(\alpha, \beta)$. To prove that $\langle k|u\rangle$ satisfies property 5, we use induction: if $k = 0$, there is nothing to prove, since property 5 applies to non–terminal nodes only. Assume now that the lemma is true for all calls at level $k-1$ and consider an arbitrary call *UnionMin*$(k, (\alpha, p), (\beta, q))$, where the input nodes $\langle k|p\rangle$ and $\langle k|q\rangle$ satisfy property 5. If α or β is ∞, the returned node is one of the input nodes, so it satisfies property 5. Otherwise, since $\mu = \min(\alpha, \beta)$, at least one of $\alpha-\mu$ and $\beta-\mu$ is 0; say $\alpha-\mu = 0$. The values labelling the edges of $\langle k|u\rangle$ are computed in line 11 of *UnionMin*. Since $\langle k|p\rangle$ satisfies property 5, there exists $i_k \in \{0, \ldots, n_k-1\}$ such that $\langle k|p\rangle.val[i_k] = 0$. Then, for the corresponding iteration of the for–loop, α' is 0 and the edge returned by *UnionMin*$(k-1, (\alpha', p'), (\beta', q'))$ is $(\min(\alpha', \beta'), u') = (0, u')$, where $\langle k-1|u'\rangle$ satisfies property 5 by induction; thus, $\langle k|u\rangle[i_k].val$ is set to 0. $\Box$

We conclude the discussion of *UnionMin* by observing that the hash–key for the entries in our "union/min cache" is formed by the two nodes (passed as *level*, *index*, *index*, since the nodes are at the same level) plus the **difference** $\alpha - \beta$ of the values labelling two edges pointing to these nodes. This is better than using the key (k, p, q, α, β), which would unnecessarily clutter the cache with entries of the form $(k, p, q, \alpha + \tau, \beta + \tau, (\mu + \tau, u))$, for all the values of τ arising in a particular execution.

4.1 State–space and distance generation using EV$^+$MDDs

Our fixed–point algorithm to build and store the distance function δ, and implicitly the state space $\mathcal{S}$, is described by the pseudo–code for *BuildDistance*, *Saturate*, and *RecursiveFire*, shown in Figure 7. Given a model $(\widehat{\mathcal{S}}, \mathcal{X}_{init}, \mathcal{N})$ we follow these steps:

1. Encode $\mathcal{X}_{init}$ into an initial EV$^+$MDD node $\langle K|r\rangle$. This can be done by building the MDD for $\mathcal{X}_{init}$, then setting to 0 all edge values for edges going to *true* (called 1 in the MDD terminology of [10]), setting the remaining edge values to ∞, eliminating the terminal node *false*, and renaming the terminal node *true* as 0 (in EV$^+$MDD terminology). See [10] on how to build an MDD when $\mathcal{X}_{init}$ contains a single state. In general, the MDD encoding of $\mathcal{X}_{init}$ will be derived from some other symbolic computation, e.g., it will be already available as the result of a temporal logic query.
2. Call *BuildDistance*(K, r).

Functions *CheckInUniqueTable*, *LocalsToExplore*, *UCacheFind*, *FCacheFind*, *UCacheInsert*, *FCacheInsert*, *PickAndRemoveElementFromSet*, and *CreateNode* have the intuitive semantic associated to their name (see also the comments in the pseudo–code). *Normalize*(k, s) puts node $\langle k|s\rangle$ in canonical form by computing $\mu = \min\{\langle k|s\rangle[i_k].val : i_k \in \mathcal{S}_k\}$ and subtracting μ from each $\langle k|s\rangle[i_k].val$ (so that at least one of them becomes 0), then returns μ; in particular, if all edge values in $\langle k|s\rangle$ are ∞, it returns ∞ (this is the case in Statement 17 of *RecursiveFire* if the while–loop did not manage to fire e from any of the local states in $\mathcal{L}$).

The hash–key for the firing cache does not use the value α on the incoming edge, because the node $\langle k|s\rangle$ corresponding to the result (γ, s) of *RecursiveFire* is independent of this quantity. The edge value returned by *RecursiveFire* depends instead of α: it is simply obtained by adding the result of *Normalize*(k, s) to α.

RecursiveFire may push excess values upwards when normalizing a node in line 17, that is, residual values are moved in the opposite direction as in *UnionMin*. However, the normalization procedure is called only once per node (when the node has been saturated), therefore excess values are not bounced back and forth repeatedly along edges.

4.2 Trace generation using EV$^+$MDDs

Once the EV$^+$MDD $(\rho, \langle K|r\rangle)$ encoding δ and $\mathcal{S}$ is built, a shortest–length trace from any of the states in $\mathcal{X}_{init}$ to one of the states in a set $\mathcal{X}$ (given in input as an MDD) can be obtained by backtracking. For simplicity, the following algorithm does not output the identity of the events along the trace, but this option could be easily added, if desired:

1. Transform the MDD for $\mathcal{X}$ into an EV$^+$MDD $(\rho_x, \langle K|x\rangle)$ encoding $\mathcal{X}$ and δ_x using the approach previously described for $\mathcal{X}_{init}$, where $\delta_x(i) = 0$ if $i \in \mathcal{X}$ and $\delta_x(i) = \infty$ if $i \in \widehat{\mathcal{S}} \setminus \mathcal{X}$.

```
$BuildDistance(k : level, p : index)$
if $k > 0$ then
  for $i_k = 0$ to $n_k - 1$ do
    if $\langle k|p\rangle[i_k].val < \infty$ then $BuildDistance(k-1, \langle k|p\rangle[i_k].child)$;
$Saturate(k, p)$;
```

```
$Saturate(k : level, p : index)$
repeat
  $pChanged \leftarrow false$;
  foreach $e \in \mathcal{E}_k$ do
      $\mathcal{L} \leftarrow LocalsToExplore(e, k, p)$;          • $\{i_k : \mathcal{N}_{e,k}(i_k) \neq \emptyset \wedge \langle k|p\rangle[i_k].val \neq \infty\}$
      while $\mathcal{L} \neq \emptyset$ do
        $i_k \leftarrow PickAndRemoveElementFromSet(\mathcal{L})$;
        $(\alpha, f) \leftarrow RecursiveFire(e, k-1, \langle k|p\rangle[i_k])$;
        if $\alpha \neq \infty$ then
          foreach $j_k \in \mathcal{N}_{e,k}(i_k)$ do
            $(\beta, u) \leftarrow UnionMin(k-1, (\alpha + 1, f), \langle k|p\rangle[j_k])$;
            if $(\beta, u) \neq \langle k|p\rangle[j_k]$ then
              $\langle k|p\rangle[j_k] \leftarrow (\beta, u)$;
              $pChanged \leftarrow true$;
              if $\mathcal{N}_{e,k}(j_k) \neq \emptyset$ then $\mathcal{L} \leftarrow \mathcal{L} \cup \{j_k\}$;          • remember to explore $j_k$ later
until $pChanged = false$;
$CheckInUniqueTable(k, p)$;
```

```
$RecursiveFire(e : event, k : level, (\alpha, q) : edge) : edge$
if $k < Bot(e)$ then return $(\alpha, q)$;                    • level $k$ is not affected by event $e$
if $FCacheFind(k, q, e, (\gamma, s))$ then                    • match $(k, q, e)$, return $(\gamma, s)$
  return $(\gamma + \alpha, s)$;
$s \leftarrow NewNode(k)$;          • create new node at level $k$ with edges set to $(\infty, 0)$
$sChanged \leftarrow false$;
$\mathcal{L} \leftarrow LocalsToExplore(e, k, q)$;          • $\{i_k : \mathcal{N}_{e,k}(i_k) \neq \emptyset \wedge \langle k|q\rangle[i_k].val \neq \infty\}$
while $\mathcal{L} \neq \emptyset$ do
  $i_k \leftarrow PickAndRemoveElementFromSet(\mathcal{L})$;
  $(\phi, f) \leftarrow RecursiveFire(e, k-1, \langle k|q\rangle[i_k])$;
  if $\phi \neq \infty$ then
    foreach $j_k \in \mathcal{N}_{e,k}(i_k)$ do
      $(\beta, u) \leftarrow UnionMin(k-1, (\phi, f), \langle k|s\rangle[j_k])$;
      if $(\beta, u) \neq \langle k|s\rangle[j_k]$ then
        $\langle k|s\rangle[j_k] \leftarrow (\beta, u)$;
        $sChanged \leftarrow true$;
if $sChanged$ then $Saturate(k, s)$;
$\gamma \leftarrow Normalize(k, s)$;
$s \leftarrow CheckInUniqueTable(k, s)$;
$FCacheInsert(k, q, e, (\gamma, s))$;
return $(\gamma + \alpha, s)$;
```

Fig. 7. *BuildDistance*, our saturation–based algorithm using EV$^+$MDDs.

2. Compute *IntersectionMax*$(K, (\rho, r), (\rho_x, x))$, which is the dual of *UnionMin*, and whose pseudo–code is exactly analogous; let $(\mu, \langle K|m\rangle)$ be the resulting EV$^+$MDD, which encodes $\mathcal{X} \cap \mathcal{S}$ and the restriction of δ to this set (μ is then the length of one of the shortest–paths we are seeking).
3. Extract from $(\mu, \langle K|m\rangle)$ a state $j^{[\mu]} = (j_K^{[\mu]}, \ldots, j_1^{[\mu]})$ encoded by a path from $\langle K|m\rangle$ to $\langle 0|0\rangle$ labelled with 0 values ($j^{[\mu]}$ is a state in $\mathcal{X}$ at the desired minimum distance μ from $\mathcal{X}_{init}$). The algorithm proceeds now with an explicit flavor.
4. Initialize ν to μ and iterate:
   (a) Find all states $i \in \widehat{\mathcal{S}}$ such that $j^{[\nu]} \in \mathcal{N}(i)$. With our boolean Kronecker encoding of $\mathcal{N}$, this "one step backward" is easily performed: we simply have to use the transpose of the matrices $\mathcal{N}_{e,k}$.
   (b) For each such state i, compute $\delta(i)$ using $(\rho, \langle K|r\rangle)$ and stop on the first i such that $\delta(i) = \nu - 1$ (there exists at least one such state i^*).
   (c) Decrement ν.
   (d) Let $j^{[\nu]}$ be i^*.
5. Output $j^{[0]}, \ldots, j^{[\mu]}$.

The cost of obtaining $j^{[\mu]}$ as the result of the *IntersectionMax* operation is $O(\#\langle K|r\rangle \cdot \#\langle K|x\rangle)$, where $\#$ indicates the number of EV$^+$MDD nodes. The complexity of the rest of the algorithm is then simply $O(\mu \cdot M \cdot K)$, where M is the maximum number of incoming arcs to any state in the reachability graph of the model, i.e., $M = \max\{|\mathcal{N}^{-1}(j)| : j \in \mathcal{S}\}$, and K comes from traversing one path in the EV$^+$MDD. In practice M is small but, if this were not the case, the set $\mathcal{N}^{-1}(j^{[\nu]})$ could be computed symbolically at each iteration instead.

Generating the same trace using traditional symbolic approaches could follow a similar idea. If we used ADDs, we would start with an ADD encoding the same information as the EV$^+$MDD $(\rho_x, \langle K|x\rangle)$, compute the ADD equivalent to the EV$^+$MDD $(\mu, \langle K|m\rangle)$ using a breadth–first approach, and pick as $j^{[\mu]}$ any state leading to a terminal with minimal value μ. If we used a forest of MDDs, we would compute $\mu = \min\{d : \mathcal{N}^d(\mathcal{X}_{init}) \cap \mathcal{X} \neq \emptyset\}$, and pick as $j^{[\mu]}$ any state in $\mathcal{N}^\mu \cap \mathcal{X}$. Then, the backtracking would proceed in exactly the same way.

In either case, however, we are discovering states symbolically in breadth–first order, thus we could choose to perform an intersection with $\mathcal{X}$ after finding each additional set of states $\mathcal{N}^d$, and stop as soon as $\mathcal{N}^d(\mathcal{X}_{init}) \cap \mathcal{X} \neq \emptyset$. Overall, we would then have explored only $\{i : \delta(i) \leq \mu\}$, which might be a strict subset of the entire state space $\mathcal{S}$. However, two observations are in order. First, while this "optimization" manages fewer *states*, it may well require many more *nodes* in the symbolic representation: decision diagrams are quite counter–intuitive in this respect. Second, in many verification applications, the states in $\mathcal{X}$ satisfy some property, e.g., "being a deadlock", and they can only be reached in some obscure and tortuous way, so that the minimum distance μ to any state in $\mathcal{X}$ is in practice close, if not equal, to the maximum distance ρ to any of the states in $\mathcal{S}$.

The advantage of our approach is that, while it must explore the entire $\mathcal{S}$, it can do so using the idea of saturation, thus the resulting decision diagrams are

Table 1. Comparison of the five approaches ("—" means "out of memory").

N	$\lvert\mathcal{S}\rvert$	Time (in seconds)					Number of nodes							
							final			peak				
		E_s	E_b	M_b	A_s	A_b	E_sE_b	M_b	A_sA_b	E_s	E_b	M_b	A_s	A_b
Dining philosophers: $D=2N$, $K=\lceil N/2\rceil$, $\lvert\mathcal{S}_k\rvert=34$ for all k except $\lvert\mathcal{S}_1\rvert=8$ when N is odd														
5	$1.3\cdot10^{3}$	0.00	0.01	0.01	0.01	0.03	11	83	38	11	155	172	48	434
10	$1.9\cdot10^{6}$	0.01	0.06	0.05	0.12	0.46	21	255	170	21	605	644	238	4022
20	$3.5\cdot10^{12}$	0.01	0.34	0.28	1.64	9.00	46	1100	740	46	2990	3079	1163	38942
25	$4.7\cdot10^{15}$	0.01	0.59	0.47	4.09	26.08	61	1893	1178	61	5215	5334	1958	79674
30	$6.4\cdot10^{18}$	0.02	0.86	0.70	7.39	56.80	71	2545	1710	71	7225	7364	2788	140262
1000	$9.2\cdot10^{626}$	0.48	—	—	—	—	2496	—	—	2496	—	—	—	—
Kanban system: $D=14N$, $K=4$, $\lvert\mathcal{S}_k\rvert=(N+3)(N+2)(N+1)/6$ for all k														
3	$5.8\cdot10^{4}$	0.01	0.02	0.02	0.04	0.17	7	180	68	29	454	464	284	3133
5	$2.5\cdot10^{6}$	0.02	0.14	0.12	0.24	1.55	9	444	133	57	1132	1156	776	13241
7	$4.2\cdot10^{7}$	0.04	0.51	0.42	0.94	7.79	11	848	218	93	2112	2166	1600	35741
10	$1.0\cdot10^{9}$	0.16	2.10	1.68	4.68	48.86	14	1673	383	162	4041	4160	3616	98843
12	$5.5\cdot10^{9}$	0.34	4.34	3.45	11.08	129.46	16	2368	518	218	5633	5805	5585	165938
50	$1.0\cdot10^{16}$	179.48	—	—	—	—	58	—	—	2802	—	—	—	—
Flex. manuf. syst.: $D=14N$, $K=19$, $\lvert\mathcal{S}_k\rvert=N+1$ for all k except $\lvert\mathcal{S}_{17}\rvert=4, \lvert\mathcal{S}_{12}\rvert=3, \lvert\mathcal{S}_2\rvert=2$														
3	$4.9\cdot10^{4}$	0.00	0.12	0.09	0.26	1.58	88	1925	1191	116	5002	5187	2075	37657
5	$2.9\cdot10^{6}$	0.01	0.42	0.34	0.88	11.78	149	5640	2989	211	15205	15693	4903	179577
7	$6.6\cdot10^{7}$	0.02	1.05	0.85	2.08	65.32	222	12070	5739	326	32805	33761	9027	523223
10	$2.5\cdot10^{9}$	0.04	2.96	2.40	5.79	608.92	354	28225	11894	536	76676	78649	17885	1681625
140	$2.0\cdot10^{23}$	20.03	—	—	—	—	32012	—	—	52864	—	—	—	—
Round–robin mutex protocol: $D=8N-6$, $K=N+1$, $\lvert\mathcal{S}_k\rvert=10$ for all k except $\lvert\mathcal{S}_1\rvert=N+1$														
10	$2.3\cdot10^{4}$	0.01	0.06	0.05	0.22	0.50	92	1038	1123	107	1898	1948	1210	9245
15	$1.1\cdot10^{6}$	0.01	0.15	0.14	1.00	2.93	177	2578	3136	212	4774	4885	3308	34897
20	$4.7\cdot10^{7}$	0.02	0.32	0.31	3.10	12.62	287	4968	6619	322	9270	9467	6901	92140
25	$1.8\cdot10^{9}$	0.03	0.59	0.54	7.89	52.29	422	8333	11947	477	15636	15944	12364	198839
30	$7.2\cdot10^{10}$	0.05	0.95	0.89	16.04	224.83	582	12798	19495	637	24122	24566	20072	376609
200	$7.2\cdot10^{62}$	1.63	—	—	—	—	20897	—	—	21292	—	—	—	—

built much more efficiently and require much less memory than with breadth–first approaches. The following section confirms this, focusing on the first and expensive phase of trace generation, the computation of the distance information, since the backtracking phase has negligible cost in comparison and is in any case essentially required by any approach.

5 Results

To stress the importance of using a saturation–based approach, we compare the three types of encodings for the distance function we have discussed, EV$^+$MDDs, forests of MDDs, and ADDs, in conjunction with two iteration strategies, based on breadth–first and saturation, respectively (see Table 1). Since only breadth–first is applicable in the case of forests of MDDs, this leads to five cases: EV$^+$MDD with saturation (E_s), EV$^+$MDD with breadth–first (E_b), forest of MDDs with breadth–first (M_b), ADD with saturation (A_s), and ADD with breadth–first (A_b). Note that only M_b and A_b have been used in the literature before, while

E_s and E_b use our new data structure and A_s (which we cannot discuss in detail for lack of space) applies the idea of saturation to ADDs, thus it is also a new approach.

We implemented the five algorithms (their MDD, not BDD, version) in our tool SMART [8] and used them to generate the distance function for the entire state space. The suite of examples is chosen from the same benchmark we used in [10]; each model is scalable by a parameter N. All experiments were ran on a 800 MHz Pentium III workstation with 1GB of memory. For each model, we list the maximum distance D, the number K of levels in the decision diagram, and the sizes of the local state spaces. For each experiment we list the maximum distance to a reachable state, which is also the number of iterations in the breadth–first approaches, the runtime, and the number of nodes (both final and peak).

In terms of runtime, there is a clear order: $E_s < E_b < M_b < A_s < A_b$, with E_s easily managing much larger systems; $E_s, E_b < M_b < A_s, A_b$ clearly attests to the effectiveness of the data structures, while $E_s < E_b$ and $A_s < A_b$ attest to the improvements obtainable with saturation–based approaches.

With EV$^+$MDDs, in particular with E_s, we can scale up the models to huge parameters. The other two data structures do not scale up nearly as well and run out of memory. In terms of memory consumption: $E_s < A_s < E_b \approx M_b < A_b$ for the peak number of nodes, while $E_s = E_b < A_s = A_b \approx M_b$ for the final number of nodes. The key observation is that E_s substantially outperforms all other methods. Compared to A_b, it is over 1,000 times faster and uses fewer peak nodes, also by a factor of 1,000.

6 Conclusion

We introduced EV$^+$MDDs, a new canonical variation of EVBBDs, which can be used to store the state space of a model and the distance of every state form the initial set of states within a single decision diagram. A key contribution is that we extend the *saturation* approach we previously introduced for state–space generation alone, and apply it to this data–structure, resulting in a very fast and memory–efficient algorithm for joint state–space and distance generation.

One conclusion of our research is a clear confirmation of the effectiveness of saturation as opposed to a traditional breadth–first iteration, not just when used in conjunction with our EV$^+$MDDs, but even with ADDs. A second orthogonal conclusion is that edge–valued decision diagrams in general are much more suited than ADDs to the task at hand, because they *implicitly* encode the possible distance values, while ADDs have an explicit terminal node for each possible value, greatly reducing the degree of node merging in the diagram.

Future work along these research lines includes exploring smarter cache management policies that exploit properties of the involved operators (e.g., additivity), extending the idea to *EU* and *EG* operators (probably a major challenge), comparing the performance of our method with that of non BDD–based techniques (such as using SAT solvers [4]), and investigate other fields of application for EV$^+$MDDs.

References

1. P. A. Abdulla, P. Bjesse, and N. Eén. Symbolic reachability analysis based on SAT-solvers. In S. Graf and M. Schwartzbach, editors, *Proc. Tools and Algorithms for the Construction and Analysis of Systems TACAS, Berlin, Germany*, volume 1785 of *LNCS*, pages 411–425. Springer-Verlag, 2000.
2. V. Amoia, G. De Micheli, and M. Santomauro. Computer-oriented formulation of transition-rate matrices via Kronecker algebra. *IEEE Trans. Rel.*, 30:123–132, June 1981.
3. R. I. Bahar, E. A. Frohm, C. M. Gaona, G. D. Hachtel, E. Maciii, A. Pardo, and F. Somenzi. Algebraic decision diagrams and their applications. *Formal Methods in System Design*, 10(2/3):171–206, Apr. 1997.
4. A. Biere, A. Cimatti, E. Clarke, and Y. Zhu. Symbolic model checking without BDDs. *LNCS*, 1579:193–207, 1999.
5. R. E. Bryant. Graph-based algorithms for boolean function manipulation. *IEEE Trans. Comp.*, 35(8):677–691, Aug. 1986.
6. R. E. Bryant and Y.-A. Chen. Verification of arithmetic circuits with binary moment diagrams. In *Proc. of Design Automation Conf. (DAC)*, pages 535–541, 1995.
7. J. R. Burch, E. M. Clarke, K. L. McMillan, D. L. Dill, and L. J. Hwang. Symbolic model checking: 10^{20} states and beyond. In *Proc. 5th Annual IEEE Symp. on Logic in Computer Science*, pages 428–439, Philadelphia, PA, 4–7 June 1990. IEEE Comp. Soc. Press.
8. G. Ciardo, R. L. Jones, A. S. Miner, and R. Siminiceanu. SMART: Stochastic Model Analyzer for Reliability and Timing. In P. Kemper, editor, *Tools of Aachen 2001 Int. Multiconference on Measurement, Modelling and Evaluation of Computer-Communication Systems*, pages 29–34, Aachen, Germany, Sept. 2001.
9. G. Ciardo, G. Luettgen, and R. Siminiceanu. Efficient symbolic state-space construction for asynchronous systems. In M. Nielsen and D. Simpson, editors, *Application and Theory of Petri Nets 2000 (Proc. 21th Int. Conf. on Applications and Theory of Petri Nets, Aarhus, Denmark)*, LNCS 1825, pages 103–122. Springer-Verlag, June 2000.
10. G. Ciardo, G. Luettgen, and R. Siminiceanu. Saturation: An efficient iteration strategy for symbolic state space generation. In T. Margaria and W. Yi, editors, *Proc. Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, LNCS 2031, pages 328–342, Genova, Italy, Apr. 2001. Springer-Verlag.
11. E. Clarke, E. Emerson, and A. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Trans. Progr. Lang. and Syst.*, 8(2):244–263, Apr. 1986.
12. E. Clarke and X. Zhao. Word level symbolic model checking: A new approach for verifying arithmetic circuits. Technical Report CS-95-161, Carnegie Mellon University, School of Computer Science, May 1995.
13. E. M. Clarke, O. Grumberg, and D. A. Peled. *Model Checking.* MIT Press, 1999.
14. R. Drechsler and B. Becker. Overview of decision diagrams. *IEE Proc.-Comput. Digit. Tech.*, 144(3):187–193, May 1997.
15. E.M. Clarke, O. Grumberg, K.L. McMillan, and X. Zhao. Efficient generation of counterexamples and witnesses in symbolic model checking. In *32nd Design Automation Conference (DAC 95)*, pages 427–432, San Francisco, CA, USA, 1995.
16. A. Geser, J. Knoop, G. Lüttgen, B. Steffen, and O. Rüthing. Chaotic fixed point iterations. Technical Report MIP-9403, Univ. of Passau, 1994.

17. R. Hojati, R. K. Brayton, and R. P. Kurshan. BDD-based debugging of designs using language containment and fair CTL. In C. Courcoubetis, editor, *Computer Aided Verification (CAV'93)*, volume 697 of *LNCS*, pages 41–58, Elounda, Greece, June/July 1993. Springer-Verlag.
18. 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, *Int. Conference on Very Large Scale Integration*, pages 49–58, Edinburgh, Scotland, Aug. 1991. IFIP Transactions, North-Holland.
19. T. Kam, T. Villa, R. Brayton, and A. Sangiovanni-Vincentelli. Multi-valued decision diagrams: theory and applications. *Multiple-Valued Logic*, 4(1–2):9–62, 1998.
20. Y.-T. Lai, M. Pedram, and B. K. Vrudhula. Formal verification using edge-valued binary decision diagrams. *IEEE Trans. Comp.*, 45:247–255, 1996.
21. Y.-T. Lai and S. Sastry. Edge-valued binary decision diagrams for multi-level hierarchical verification. In *Proceedings of the 29th Conference on Design Automation*, pages 608–613, Los Alamitos, CA, USA, June 1992. IEEE Computer Society Press.
22. A. S. Miner and G. Ciardo. Efficient reachability set generation and storage using decision diagrams. In H. Kleijn and S. Donatelli, editors, *Application and Theory of Petri Nets 1999 (Proc. 20th Int. Conf. on Applications and Theory of Petri Nets, Williamsburg, VA, USA)*, LNCS 1639, pages 6–25. Springer-Verlag, June 1999.
23. T. Murata. Petri Nets: properties, analysis and applications. *Proc. of the IEEE*, 77(4):541–579, Apr. 1989.
24. P. F. Williams, A. Biere, E. M. Clarke, and A. Gupta. Combining Decision Diagrams and SAT Procedures for Efficient Symbolic Model Checking. In *Proceedings of CAV'00*, pages 124–138, 2000.