Experimental Studies of Symbolic Shortest-Path Algorithms

Daniel Sawitzki*

University of Dortmund, Computer Science 2, D-44221 Dortmund, Germany.
daniel.sawitzki@cs.uni-dortmund.de

Abstract. Graphs can be represented symbolically by the Ordered Binary Decision Diagram (OBDD) of their characteristic function. To solve problems in such implicitly given graphs, specialized symbolic algorithms are needed which are restricted to the use of functional operations offered by the OBDD data structure. In this paper, two symbolic algorithms for the single-source shortest-path problem with nonnegative positive integral edge weights are presented which represent symbolic versions of Dijkstra's algorithm and the Bellman-Ford algorithm. They execute $\mathcal{O}\big(N\cdot\log(NB)\big)$ resp. $\mathcal{O}\big(NM\cdot\log(NB)\big)$ OBDD-operations to obtain the shortest paths in a graph with N nodes, M edges, and maximum edge weight B. Despite the larger worst-case bound, the symbolic Bellman-Ford-approach is expected to behave much better on structured graphs because it is able to handle updates of node distances effectively in parallel. Hence, both algorithms have been studied in experiments on random, grid, and threshold graphs with different weight functions. These studies support the assumption that the Dijkstra-approach behaves efficient w. r. t. space usage, while the Bellman-Ford-approach is dominant w. r. t. runtime.

1 Introduction

Algorithms on graphs G with node set V and edge set $E \subseteq V^2$ typically work on adjacency lists of size $\Theta(|V|+|E|)$ or on adjacency matrices of size $\Theta(|V|^2)$. These representations are called *explicit*. However, there are application areas in which problems on graphs of such large size have to be solved that an explicit representation on today's computers is not possible. In the area of logic synthesis and verification, state-transition graphs with for example 10^{27} nodes and 10^{36} edges occur. Other applications produce graphs which are representable in explicit form, but for which even runtimes of efficient polynomial algorithms are not practicable anymore. Modeling of the WWW, street, or social networks are examples of this problem scenario.

However, we expect the large graphs occurring in application areas to contain regularities. If we consider graphs as Boolean functions, we can represent them

* Supported by the Deutsche Forschungsgemeinschaft (DFG) as part of the Research Cluster "Algorithms on Large and Complex Networks" (1126).

C.C. Ribeiro and S.L. Martins (Eds.): WEA 2004, LNCS 3059, pp. 482–497, 2004.
© Springer-Verlag Berlin Heidelberg 2004

by *Ordered Binary Decision Diagrams* (*OBDDs*) [4,5,19]. This data structure is well established in verification and synthesis of sequential circuits [9,11,19] due to its good compression of regular structures. In order to represent a graph $G = (V, E)$ by an OBDD, its edge set E is considered as a *characteristic Boolean function* χ_E, which maps binary encodings of E's elements to 1 and all others to 0. This representation is called *implicit* or *symbolic*, and is not essentially larger than explicit ones. Nevertheless, we hope that advantageous properties of G lead to small, that is sublinear OBDD-sizes.

Having such an OBDD-representation of a graph, we are interested in solving problems on it without extracting too much explicit information from it. Algorithms that are mainly restricted to the use of functional operations are called *implicit* or *symbolic algorithms* [19]. They are considered as heuristics to save time and/or space when large structured input graphs do not fit into the internal memory anymore. Then, we hope that each OBDD-operation processes many edges in parallel. The runtime of such methods depends on the number of executed operations as well as on the efficiency of each single one. The latter in turn depends on the size of the operand OBDDs.

In general, we want heuristics to perform well on special input subsets, while their worst-case runtime is typically worse than for optimal algorithms. Symbolic algorithms often have proved to behave better than explicit methods on interesting graphs and are well established in the area of logic design and verification. Most papers on OBDD-based algorithms prove their usability just by experiments on benchmark inputs from a special application area [10,11,13]. In less application-oriented works considering more general graph problems, mostly the number of OBDD-operations is bounded as a hint on the real over-all runtime [3,8,7,14]. Only a few of them contain analyses of the over-all runtime and space usage for special cases like grids [15,16,20].

Until now, symbolic shortest-path algorithms only existed for graph representations by algebraic decision diagrams (ADDs) [1], which are difficult to analyze and are useful only for a small number of different weight values. A new OBDD-based approach to the all-pairs shortest-paths problem [18] aims at polylogarithmic over-all runtime on graphs with very special properties. In contrast to these results, the motivation of this paper's research was to transform popular methods for the single-source shortest-path problem into symbolic algorithms, and to compare their performance in experiments. This has been done for Dijkstra's algorithm as well as for the Bellman-Ford algorithm.

This paper is organized as follows: Section 2 introduces the principles of symbolic graph representation by OBDDs. Section 3 presents the symbolic shortest-path algorithms studied in this paper. Section 4 discusses experimental results of the algorithms on random, grid, and threshold graphs. Finally, Sect. 5 gives conclusions on the work.

2 Symbolic Graph Representation

We denote the class of Boolean functions $f\colon\{0,1\}^n \to \{0,1\}$ by B_n. The ith character of a binary number $x \in \{0,1\}^n$ is denoted by x_i and $|x| := \sum_{i=0}^{n-1} x_i 2^i$ identifies its value.

Consider a directed graph $G = (V, E)$ with node set $V = \{v_0, \ldots, v_{2^n-1}\}$ and edge set $E \subseteq V^2$. G can be represented by a *characteristic* Boolean function $\chi_E \in B_{2n}$ which maps pairs $(x, y) \in \{0,1\}^{2n}$ of binary node numbers of length n to 1 iff $(v_{|x|}, v_{|y|}) \in E$. We can capture more complex graph properties by adding further arguments to characteristic functions. An additional weight function $c\colon E \to \{0, \ldots, 2^m - 1\}$ is modeled by $\chi_C \in B_{2n+m}$ which maps triples (x, y, d) to 1 iff $(v_{|x|}, v_{|y|}) \in E$ and $c(v_{|x|}, v_{|y|}) = |d|$.

A Boolean function $f \in B_n$ defined on variables $x_0, \ldots, x_{n-1}$ can be represented by an OBDD $\mathcal{G}_f$ [4,5,19]. An OBDD is a directed acyclic graph consisting of *internal nodes* and *sink nodes*. Each internal node is labeled with a Boolean variable x_i, while each sink node is labeled with a Boolean constant. Each internal node is left by two edges one labeled by 0 and the other by 1. A *function pointer p* marks a special node that represents f. Moreover, a permutation $\pi \in \Sigma_n$ called *variable order* must be respected by the internal nodes' labels on every path in the OBDD.

For a given variable assignment $a \in \{0,1\}^n$, we compute the corresponding function value $f(a)$ by traversing $\mathcal{G}_f$ from p to a sink labeled with $f(a)$ while leaving a node x_i via its a_i-edge. The size $\mathrm{size}(\mathcal{G}_f)$ of $\mathcal{G}_f$ is measured by the number of its nodes. An OBDD is called *complete*, if every path from p to a sink has length n. This has not to be the case in general, because OBDDs may skip a variable test. We adopt the usual assumption that all OBDDs occurring in symbolic algorithms are minimal, since all OBDD-operations exclusively produce minimized diagrams, which are known to be canonical. There is an upper bound of $\big(2 + o(1)\big)2^n/n$ for the OBDD-size of every $f \in B_n$; hence, a graph's edge set $E \subseteq V^2$ has an OBDD of worst-case size $\mathcal{O}(V^2/\log|V|)$.

OBDD-operations. The satisfiability of f can be decided in time $\mathcal{O}(1)$. The negation $\overline{f}$ as well as the replacement of a function variable x_i by a constant c (i. e., $f_{|x_i=c}$) is computable in time $\mathcal{O}\big(\mathrm{size}(\mathcal{G}_f)\big)$. Whether two functions f and g are equivalent (i. e., $f = g$) can be decided in time $\mathcal{O}(\mathrm{size}(\mathcal{G}_f) + \mathrm{size}\big(\mathcal{G}_g)\big)$. These operations are called *cheap*. Further essential operations are the *binary synthesis* $f \otimes g$ for $f, g \in B_n$, $\otimes \in B_2$ (e. g., "$\wedge$" and "$\vee$") and the *quantification* $(\mathcal{Q}x_i)f$ for a quantifier $\mathcal{Q} \in \{\exists, \forall\}$. In general, the result $\mathcal{G}_{f\otimes g}$ has size $\mathcal{O}(\mathrm{size}(\mathcal{G}_f) \cdot \mathrm{size}\big(\mathcal{G}_g)\big)$, which is also the general runtime of this operation. The computation of $\mathcal{G}_{(\mathcal{Q}x_i)f}$ can be realized by two cheap operations and one binary synthesis in time and space $\mathcal{O}\big(\mathrm{size}^2(\mathcal{G}_f)\big)$.

Notation. The characteristic functions used for symbolic representation are typically defined on several subsets of Boolean variables, each representing a different argument. For example, a weighted graph's function χ_C is defined on

two binary node numbers $x = x_{n-1} \dots x_0$ and $y = y_{n-1} \dots y_0$ and a binary weight value $d = d_{m-1} \dots d_0$. We assume w. l. o. g. that all arguments consist of the same number of n variables. Moreover, both a function $\chi_S \in B_{kn}$ defined on k arguments $x^{(1)}, \dots, x^{(k)} \in \{0,1\}^n$ as well as its OBDD-representation $\mathcal{G}_{\chi_S}$ will be denoted by $S(x^{(1)}, \dots, x^{(k)})$ in this paper. We use an *interleaved* variable order $\pi = (x_0^{(1)}, x_0^{(2)}, \dots, x_0^{(k)}, x_1^{(1)}, \dots, x_{n-1}^{(k)})$, which enables to *swap* [19] arguments in time $\mathcal{O}\big(\text{size}(\mathcal{G}_{\chi_S})\big)$ (e. g., $F(x,y) := G(y,x)$).

The symbolic algorithms will be described in terms of functional assignments like "$F(x) := G(x) \wedge H(x)$." The quantification $(\exists y_{n-1} \dots \exists y_0)\ F(x,y)$ over the n bits of an argument y will be denoted by $(\exists y)\ F(x,y)$. Although this seems to be one OBDD-operation, this corresponds to $\mathcal{O}(n)$ quantification operations. Identifiers with braced superscripts mark additional arguments of characteristic functions occurring only temporarily in quantified formulas (e. g., $d^{(1)}$). Furthermore, the functional assignments will contain tool functions for comparisons of weighted sums like $F(x,y,z) := (|x| + |y| = |z|)$. These can be composed from *multivariate threshold functions*.

Definition 1. *Let $f \in B_{kn}$ be defined on variables $x^{(1)}, \dots, x^{(k)} \in \{0,1\}^n$. Moreover, let $\mathcal{W}$, $T \in \mathbb{Z}$, and $w_1, \dots, w_k \in \{-\mathcal{W}, \dots, \mathcal{W}\}$. f is called k-*variate threshold function *iff it is*

$$f(x^{(1)}, \dots, x^{(k)}) = \left(\sum_{i=1}^{k} w_i \cdot |x^{(i)}| \geq T \right) .$$

$\mathcal{W}$ is called the maximum absolute weight *of f.*

Besides the greater or equal comparison, the relations $>$, $\leq$, $<$, and $=$ can be realized by binary syntheses of multivariate threshold functions, too. For a constant number k of arguments and a constant maximum absolute weight $\mathcal{W}$, such a comparison function $f \in B_{kn}$ has a compact OBDD of size $\mathcal{O}(n)$ [20].

3 Symbolic Shortest-Path Algorithms

In this section, symbolic versions of two popular shortest-path algorithms are presented: Dijkstra's algorithm [6] and the Bellman-Ford algorithm [2]. We assume that the reader is familiar with these two methods, and describe their symbolic versions in separate sections. Both solve the single-source shortest-path problem in symbolically represented directed graphs $G = (V, E, s, c)$ with node set $V = \{v_0, \dots, v_{N-1}\}$, edge set $E \subseteq V^2$ of cardinality M, source node $s \in V$, edge weight function $c\colon E \to \mathbb{N}_0$, and $B := \max\{c(e) \mid e \in E\}$. The maximum path length from s to any node v is $B(N-1) =: L$. Let $n := \lceil \log(L+1) \rceil = \Theta(\log N + \log B)$ be the number of bits necessary to encode one node number or distance value. The algorithms receive the input graph in form of two OBDDs for the characteristic functions $C(x,y,d)$ and $s(x)$ with

$$C(x,y,d) = 1 \Leftrightarrow \big[(v_{|x|}, v_{|y|}) \in E\big] \wedge \big[c(v_{|x|}, v_{|y|}) = |d|\big] \ ,$$
$$s(x) = 1 \Leftrightarrow v_{|x|} = s \ .$$

The output is the distance function dist: $V \to \mathbb{N}_0 \cup \{\infty\}$ which maps a node $v \in V$ to the length of a shortest path from s to v, given as an OBDD $DIST(x, d)$ with

$$DIST(x, d) = 1 \Leftrightarrow \text{dist}(v_{|x|}) = |d| \ .$$

Both algorithms maintain a temporary distance function $\Delta: E \to \mathbb{N}_0 \cup \{\infty\}$ represented by an OBDD $D(x, d)$, which is updated until it equals dist.

3.1 The Dijkstra-Approach

Dijkstra's algorithm [6] stores a node set $A \subseteq V$ of nodes for which the shortest-path length is already known. At the beginning, it is $A = \{s\}$, $\Delta(s) = 0$, and $\Delta(v) = \infty$ for all nodes $v \neq s$. In each iteration, we add one node to A. Let u be the last node added to A. For each edge (u, v) it is checked whether $\Delta(u) + c(u, v) < \Delta(v)$. If this is the case, we update $\Delta(v)$ to $\Delta(u) + c(u, v)$. After this relaxation step, we add a node $v^{\min} \in V \backslash A$ to A whose value $\Delta(v^{\min})$ is minimal. If $\{v \in V \backslash A \mid \Delta(v) \neq \infty\} = \emptyset$, the actual distances Δ correspond to dist and we terminate. If the nodes are stored in a priority heap with access time $\mathcal{O}(\log N)$, this explicit algorithm needs time $\mathcal{O}\big((N + M) \cdot \log N\big)$.

This approach is now transformed into a symbolic algorithm that works with corresponding OBDDs $A(x)$ and $D(x, d)$ for the characteristic functions of A and Δ. At the beginning, they are initialized to the source node:

$$\begin{aligned} A(x) &:= s(x) \ , \\ D(x, d) &:= s(x) \wedge (|d| = 0) \ . \end{aligned}$$

$x^{\min}$ and $d^{\min}$ are bit strings representing the node $v^{\min} = v_{|x^{\min}|}$ lastly added to A with $\Delta(v^{\min}) = |d^{\min}|$. Initially, $x^{\min}$ represents s and it is $v_{|x^{\min}|} = s$ and $|d^{\min}| = 0$.

Now all edges leaving $v^{\min}$ have to be relaxed. We introduce three helping functions which will be used to update $D(x, d)$: Function $RELAX(x, d)$ represents pairs (x, d) such that $(v^{\min}, v_{|x|}) \in E$ and $\Delta(v^{\min}) + c(v^{\min}, v_{|x|}) = |d|$. $D_1(x, d)$ and $D_2(x, d)$ represent the two possibilities for nodes $v_{|x|} \notin A$: 1. It is $D_1(x, d) = 1$ iff distance $|d|$ is the relaxed distance of $v_{|x|}$ not being larger than the actual distance $\Delta(v_{|x|})$. 2. It is $D_2(x, d) = 1$ iff distance $|d|$ is the actual distance $\Delta(v_{|x|})$ not being larger than the relaxed distance of $v_{|x|}$. Case 1 represents the update of $\Delta(v_{|x|})$, while Case 2 represents its retention. Finally, the new $D(x, d)$ equals the actual $D(x, d)$ for nodes $v_{|x|} \in A$, while for nodes $v_{|x|} \notin A$ Case 1 ($D_1(x, d)$) or Case 2 ($D_2(x, d)$) applies. This leads to the following symbolic formulation:

$$\begin{aligned} RELAX(x, d) &:= (\exists d^{(1)}) \left[C(x^{\min}, x, d^{(1)}) \wedge (|d| = |d^{\min}| + |d^{(1)})|\right] \ , \\ D_1(x, d) &:= RELAX(x, d) \wedge \overline{(\exists d^{(1)}) \left[D(x, d^{(1)}) \wedge (|d^{(1)}| < |d|)\right]} \ , \\ D_2(x, d) &:= D(x, d) \wedge \overline{(\exists d^{(1)}) \left[RELAX(x, d^{(1)}) \wedge (|d^{(1)}| < |d|)\right]} \ , \\ D(x, d) &:= \left[A(x) \wedge D(x, d)\right] \vee \left[\overline{A(x)} \wedge [D_1(x, d) \vee D_2(x, d)]\right] \ . \end{aligned}$$

At next, we select the new minimal node $v^{\min}$. If there are several nodes with minimal value $\Delta(v)$, we select the node $v_{|x|}$ with the smallest node number $|x|$. Hence, we need a comparison function for two node–distance-pairs denoted by $LESS(x, d, y, d')$.

$$LESS(x, d, y, d') := (|d| < |d'|) \vee \left[(d = d') \wedge (|x| < |y|)\right]$$

The facts on multivariate threshold functions in Sect. 2 imply that comparisons like $LESS(x, d, y, d')$ and $(|d| = |d^{\min}| + |d^{(1)}|)$ have OBDD-size $\mathcal{O}(n)$. Now we define the selection function $SEL(x, d)$.

$$\begin{aligned} SEL(x,d) := {} & \overline{A(x)} \wedge D(x,d) \\ & \wedge \overline{(\exists x^{(1)}, d^{(1)}) \left[\overline{A(x^{(1)})} \wedge D(x^{(1)}, d^{(1)}) \wedge LESS(x^{(1)}, d^{(1)}, x, d)\right]} \end{aligned}$$

The interpretation of this functional assignment is that the node–distance-pair $(v_{|x|}, |d|)$ is selected iff $v_{|x|} \notin A$, $\Delta(v_{|x|}) = |d|$, and there is no other node-distance pair $(v_{|x^{(1)}|}, |d^{(1)}|)$ with these properties and $|d^{(1)}| < |d|$ or $(d = d') \wedge (|x| < |y|)$. If $SEL(x, d) \equiv 0$, all nodes reachable from s have been added to A and we may terminate with output $DIST(x, d) = D(x, d)$. Otherwise, $SEL(x, d)$ contains exactly one satisfying assignment for x and d. This can be extracted in linear time w. r. t. size(SEL) [19]. Finally, we just need to add $x^{\min}$ to $A(x)$.

$$\begin{aligned} (x^{\min}, d^{\min}) &:= SEL^{-1}(1) \ , \\ A(x) &:= A(x) \vee (x = x^{\min}) \end{aligned}$$

Afterwards, we jump to the relaxation step. The correctness of this symbolic procedure follows from the correctness of Dijkstra's algorithm, while we now consider the number of executed OBDD-operations.

Theorem 1. *The symbolic Dijkstra-approach computes the output OBDD $DIST(x, d)$ by $\mathcal{O}\big(N \cdot \log(NB)\big)$ OBDD-operations.*

Proof. All nodes reachable from s are added to $A(x)$. That is, at most N relaxation and selection iterations are executed. In each iteration, the algorithm performs a constant number of cheap operations, argument swaps, binary syntheses, and quantifications over node or distance arguments. Each of the latter corresponds to $\mathcal{O}(n) = \mathcal{O}\big(\log(NB)\big)$ quantifications over single Boolean variables. Altogether, $\mathcal{O}\big(N \cdot \log(NB)\big)$ OBDD-operations are executed. □

We have also studied a parallelized symbolic version of Dijkstra's algorithm, which selects not only one distance-minimal node to be handled in each iteration, but a maximal set of independent nodes not interfering by adjacency. Experiments showed that the parallelization could compensate the overhead caused by the more complex symbolic formulation only for graphs of very special structure, why this approach is not discussed in this work.

3.2 The Bellman-Ford-Approach

In contrast to Dijkstra's algorithm, the Bellman-Ford algorithm [2] does not select special edges to relax, but performs N iterations over all edges $(u, v) \in E$ to check the condition $\Delta(u) + c(u, v) < \Delta(v)$ and to update $\Delta(v)$ eventually. Therefore, its explicit runtime is $\mathcal{O}(NM)$. In contrast to Dijkstra's algorithm, Bellman-Ford is able to handle graphs with negative edge weights if they do not contain negative cycles. Furthermore, it is easy to parallelize, which motivated the development of a symbolic version: Few OBDD-operations hopefully perform many edge relaxations at once.

Again, the actual distance function $D(x, d)$ is only known for the source s at the beginning:

$$D(x, d) := s(x) \wedge (|d| = 0) \ .$$

We again need a function $RELAX(x, y, d)$ representing the candidates for edge relaxation. Let $RELAX(x, y, d) = 1$ iff $\Delta(v_{|x|}) + c(v_{|x|}, v_{|y|}) = |d|$ and $|d|$ is not larger than the actual $\Delta(v_{|y|})$.

$$\begin{aligned} RELAX(x, y, d) := {} & (\exists d^{(1)}, d^{(2)}) \left[D(x, d^{(1)}) \wedge C(x, y, d^{(2)}) \wedge (|d| = |d^{(1)}| + |d^{(2)}|)\right] \\ & \wedge \overline{(\exists d^{(1)}) \left[D(y, d^{(1)}) \wedge (|d^{(1)}| \leq |d|)\right]} \end{aligned}$$

If $RELAX(x, y, d) \equiv 0$, no relaxations are applicable and $D(x, d) = DIST(x, d)$ represents the correct output—we may terminate. Otherwise, we use the comparison function $LESS(x, d, y, d')$ to choose the subset that is minimal w. r. t. distance $|d|$ and, secondly, the node number $|x|$:

$$\begin{aligned} SEL(x, y, d) := {} & RELAX(x, y, d) \\ & \wedge \overline{(\exists x^{(1)}, d^{(1)}) \left[RELAX(x^{(1)}, y, d^{(1)}) \wedge LESS(x^{(1)}, d^{(1)}, x, d)\right]} \ . \end{aligned}$$

Finally, we compute the symbolic set $U(x, d)$ of node-distance pairs that have to be updated in $D(x, d)$ because they were part of a selected relaxation:

$$\begin{aligned} U(x, d) &:= (\exists x^{(1)}) \ SEL(x^{(1)}, x, d) \ , \\ D(x, d) &:= U(x, d) \vee \left[\overline{U(x, d)} \wedge D(x, d)\right] \ . \end{aligned}$$

In this way, the new distances of $U(x, y)$ are taken over into $D(x, d)$, while the other nodes keep their distance value. The new iteration starts with computing $RELAX(x, y, d)$. Again, the correctness follows from the correctness of the explicit Bellman-Ford algorithm.

Theorem 2. *The symbolic Bellman-Ford-approach computes the output OBDD $DIST(x, d)$ by $\mathcal{O}\big(NM \cdot \log(NB)\big)$ OBDD-operations.*

Proof. Every implementation of the Bellman-Ford-algorithm performs at most $\mathcal{O}(NM)$ edge relaxations. In each iteration, the symbolic method relaxes at least one edge, and executes a constant number of cheap operations, argument swaps, binary syntheses, and quantifications over node and distance arguments. Each of the latter corresponds to $\mathcal{O}(n) = \mathcal{O}\big(\log(NB)\big)$ quantifications over single Boolean variables. Altogether, $\mathcal{O}\big(NM \cdot \log(NB)\big)$ OBDD-operations are executed. □

3.3 Computing the Predecessor Nodes

Besides dist, explicit shortest-path algorithms return for each node $v \in V$ a predecessor node $\mathrm{pred}(v) =: u$, such that there is a shortest path from s to v which uses the edge (u, v). Analogue, the symbolic approaches can be modified such that they also compute the predecessor nodes on shortest paths.

The following method computes these just from the final $DIST(x, d)$ and is independent of the considered symbolic algorithm. It uses the helping function $P(x, y, d^{(1)}, d^{(2)}, d^{(3)})$ which is satisfied iff $\mathrm{dist}(v_{|x|}) = |d^{(1)}|$, $c(v_{|x|}, v_{|y|}) = |d^{(2)}|$, $\mathrm{dist}(v_{|y|}) = |d^{(3)}|$, and $(|d^{(1)}| + |d^{(2)}| = |d^{(3)}|)$. By existential quantification over the distances $d^{(1)}$, $d^{(2)}$, and $d^{(3)}$, we obtain the function $PREDS(x, y)$, which represents exactly all edges being part of some shortest path (i. e., for which $v_{|x|}$ is a predecessor of $v_{|y|}$).

$$
\begin{aligned}
P(x, y, d^{(1)}, d^{(2)}, d^{(3)}) &:= DIST(x, d^{(1)}) \wedge C(x, y, d^{(2)}) \\
&\qquad \wedge DIST(y, d^{(3)}) \wedge (|d^{(1)}| + |d^{(2)}| = |d^{(3)}|) \ , \\
PREDS(x, y) &:= (\exists d^{(1)}, d^{(2)}, d^{(3)}) \ P(x, y, d^{(1)}, d^{(2)}, d^{(3)})
\end{aligned}
$$

If we are only interested in an arbitrary predecessor of a concrete node $v_{|y^*|}$, we may omit the computation of $PREDS(x, y)$ by replacing argument y of P by y^* and extracting an arbitrary satisfying variable assignment x^* of P. Therefore, computing $DIST(x, d)$ is the essential part of symbolic shortest-path algorithms, which has been analyzed by means of the experiments documented in Sect. 4.

Remark 1. The worst-case behavior of a particular OBDD-operation executed by a symbolic algorithm can be obtained from the general bound $\big(2 + o(1)\big)2^n/n$ for the OBDD-size of any function $f \in B_n$ together with the worst-case bounds for runtime and space in Sect. 2.

Analogue to Theorem 2 in [18], it can be shown that constant width bounds of input OBDD $C(x, y, d)$ and output OBDD $DIST(x, d)$ imply a polylogarithmic upper bound on time and space for each operation. However, we did not want to restrict ourselves to such special cases and applied the Dijkstra-approach as well as the Bellman-Ford-approach in experiments to obtain more general results.

4 Experimental Results

Although the symbolic Bellman-Ford-approach has a higher worst-case bound for the number of OBDD-operations than the Dijkstra-approach, we hope that each of its iterations relaxes many edges in parallel leading to a sublinear operation number. On the other hand, representing symbolic sets like $RELAX(x, y, d)$ may involve many little structured information causing larger OBDDs than Dijkstra. That is, we expect the Bellman-Ford method to need more space, while hoping that the smaller operation count results in less over-all runtime.

In order to check these hypotheses, the symbolic shortest-path algorithms have been applied in experiments on random, grid, and threshold graphs. Because the OBDD-size $\mathrm{size}(\mathcal{G})$ of a symbolic algorithm's input graph G is a natural

lower bound for its resource usage, we investigate experimental behaviors also w. r. t. these input sizes. This allows to measure performances independently of how well an input G is suited for OBDD-representation.

Experiment setting. Both symbolic algorithms have been implemented[1] in C++ using the OBDD package CUDD 2.3.1 by Fabio Somenzi[2]. An interleaved variable order with increasing bit significance has been used for the Boolean variables of each function argument. The experiments took place on a PC with Pentium 4 2GHz processor and 512 MB of main memory. The runtime has been measured by seconds of process time, while the space usage is given as the maximum number of OBDD-nodes present at any time during an algorithm execution. The latter is of same magnitude as the over-all space usage and independent of the used computer system.

4.1 Random Graphs

Random graphs possess no regular structure and, therefore, are pathological cases for symbolic representations—they have expected OBDD-size $\Theta(N^2/\log N)$. Just for dense graphs some compression is achieved because, intuitively spoken, the OBDD stores the smaller number of missing edges instead of all existing ones. However, we cannot hope symbolic methods to beat explicit algorithms on random graphs. But even in such worst cases their runtime and space usage may be only linear w. r. t. the (correspondingly large) input OBDD-sizes, which is the best we can expect from symbolic methods in general.

Both presented symbolic shortest-path algorithms have been tested on random graphs with 100, 200, 300, and 400 nodes and edge probabilities from 0.05 to 1 in steps of 0.05 influencing the observed edge density. Node 0 served as source. Moreover, three edge weight functions of different regularity have been considered. The documented experimental results are the averages of results of 10 independent experiments for each parameter setting. The particular results merely deviated from their averages.

Constant edge weights. At first, the constant edge weight function $c(e) = 1$ has been considered. This structural assumption causes a slightly sublinear growth of the symbolic representation's OBDD-size w. r. t. the edge probability (see Fig. 1(a)). Figures 1(b) to 1(e) show the observed runtimes and space usage of both algorithms w. r. t. the edge probabilities, where "ParBF" identifies the symbolic (parallelized) Bellman-Ford-approach.

As expected, the Dijkstra method uses less space, while Bellman-Ford has lower runtimes. Figures 1(f) and 2(a) integrate all runtimes resp. space usage into one plot w. r. t. the input graphs' OBDD-sizes, which constitutes the Dijkstra–space resp. Bellman-Ford–time connection: The space usage of the Dijkstra-approach grows linearly with the input graph's OBDD-size with same offset and

[1] Implementation and experiments available at `http://thefigaro.sourceforge.net/`.
[2] CUDD is available at `http://vlsi.colorado.edu/`.

gradient for all considered numbers of nodes N, while this is not the case for the Bellman-Ford-approach. For the runtime, the situation is vice versa: Only the Bellman-Ford approach shows unique linear runtime-growth.

Difference edge weights. In order to proceed to a less simple weight function than constant weights, *difference edge weights* have been considered:

$$c(v_a, v_b) := |a - b| \bmod 200 \ .$$

The modulo-operation was used to bound the gap between maximum weights for the different numbers of nodes $N = 100$ to 400.

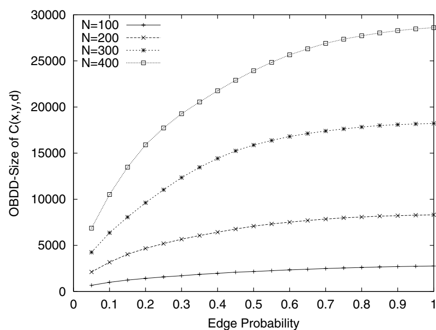
This weight function can be composed of multivariate threshold functions, and has OBDD-size $\mathcal{O}(\log N)$. Accordingly, the OBDD-sizes of random graphs with difference weights are not essentially larger than for constant weights (see Fig. 2(b)). Again, Dijkstra dominates w.r.t. space usage and Bellman-Ford dominates w.r.t. runtime, while the general resource usage is higher than for constant weights (see Tables 2(c) and 2(d)). The dependence of time and space on the input OBDD-sizes is given by Figs. 2(f) and 3(a): While Dijkstra's space still grows linearly with the same offset and gradient for all node numbers, the Bellman-Ford's runtime behavior now differs for different N.

Random edge weights. Finally, random graphs with random edge weights between 1 and 200 have been considered in experiments. Figure 2(e) shows that their OBDD-sizes grow linearly with the edge density, because the random weights prohibit the space savings observed for the two other weight functions. The runtimes w.r.t. edge probabilities and numbers of nodes N are given by Tables 2(c) and 2(d), while the dependence of time and space on the input OBDD-sizes is given by Figs. 2(f) and 3(a). The general resource usage further increased in comparison to difference weights. The missing structure of the inputs leads to nearly the same runtime for Dijkstra and Bellman-Ford—the latter is not able to compensate the larger space requirements by less operations anymore. In contrast, the advantage of the Dijkstra-approach still remains: Its space usage grows linearly with the same offset and gradient for all considered edge probabilities p and numbers of nodes N.

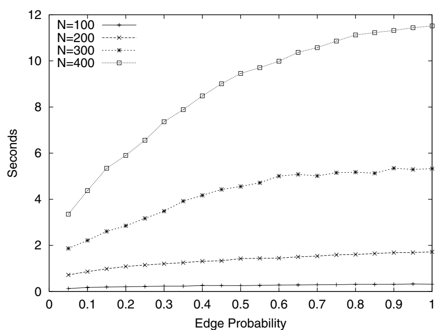
4.2 Grid and Threshold Graphs

In contrast to random graphs, the grid and threshold graphs considered in this section are examples of structured inputs with logarithmic OBDD-size $\mathcal{O}(\log N)$ [16,20], whose OBDDs can be constructed efficiently. Hence, we hope a useful symbolic algorithm to use only polylogarithmic resources in these cases.

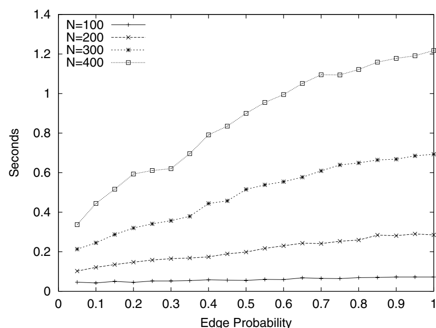
Grid graphs. Both algorithms have been applied to $2^{n/2} \times 2^{n/2}$-grid graphs, which are quadratic node matrices of 2^n nodes (i, j), $i, j \in \{0, \ldots, 2^{n/2}-1\}$, with vertical edges $\big((i, j),\ (i+1, j)\big)$ and horizontal edges $\big((i, j),\ (i, j+1)\big)$. Grids of



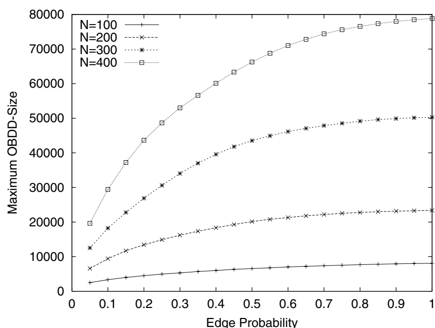
(a) Random edges, constant weights.



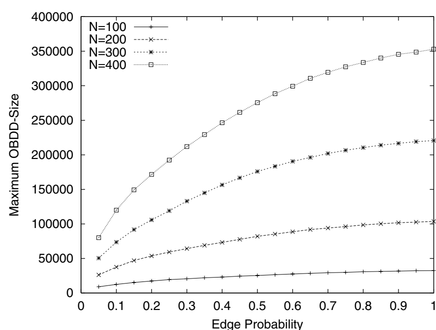
(b) Dijkstra, random edges, constant weights.



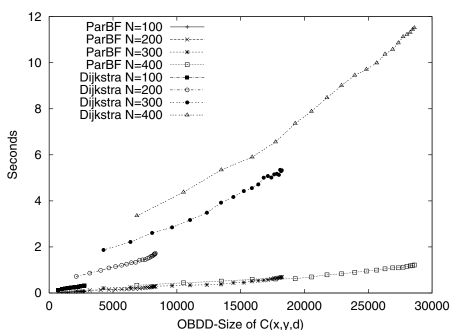
(c) ParBF, random edges, constant weights.



(d) Dijkstra, random edges, constant weights.

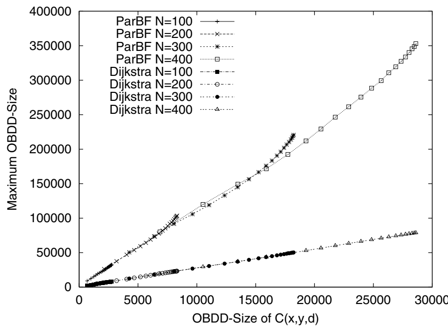


(e) ParBF, random edges, constant weights.

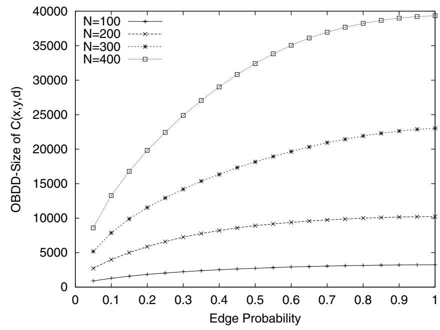


(f) Random edges, constant weights.

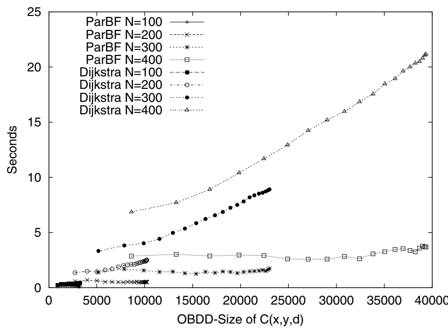
Fig. 1. Experimental results on random graphs.



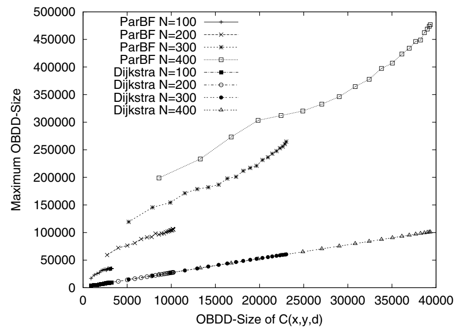
(a) Random edges, constant weights.



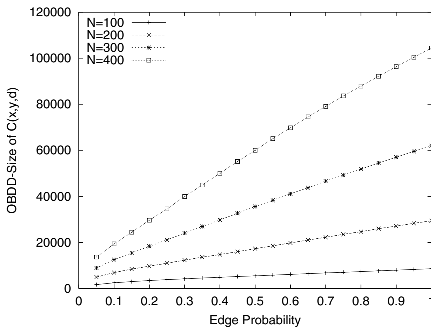
(b) Random edges, difference weights.



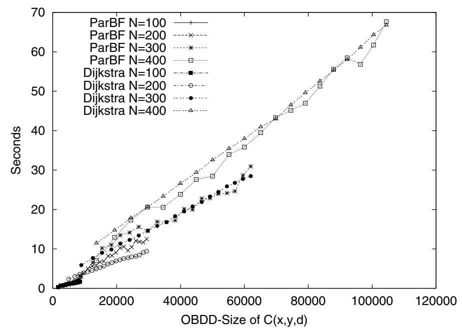
(c) Random edges, difference weights.



(d) Random edges, difference weights.

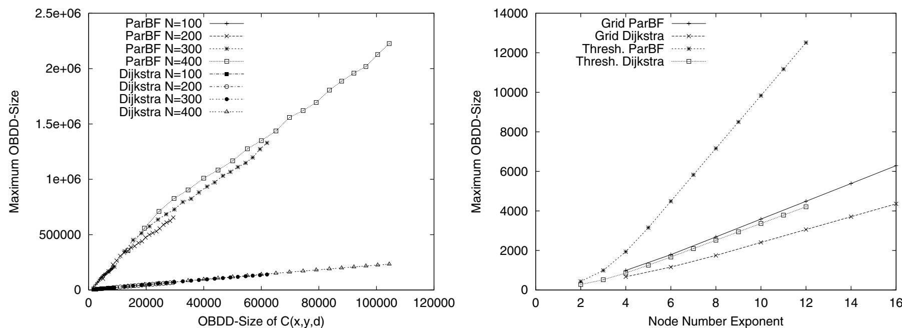


(e) Random edges, random weights.



(f) Random edges, random weights.

Fig. 2. Experimental results on random graphs.



(a) Random edges, random weights.

(b) Grid and threshold graphs.

Fig. 3. Experimental results on random, grid, and threshold graphs.

size 2^2 to 2^{16} with source node $(0,0)$ and constant edge weight 1 have been considered. Because these should be examples for graphs of optimal symbolic representation, no random weights were used in the experiments.

Threshold graphs. *Threshold graphs* [12] have compact OBDDs of size $\mathcal{O}(\log N)$ if their degree sequence or construction sequence has a compact symbolic representation [17]. In particular, this is the case for graphs with nodes $v_0, \ldots, v_{N-1}$ and edges

$$(v_a, v_b) \in E \Leftrightarrow a + b \geq T \qquad , T \in \mathbb{N} \ .$$

Both symbolic shortest-path algorithms have been applied on such threshold graphs for $N = 2^n$, $T = 2^{n-1}$, and $n := 2, \ldots, 12$. As edge weight, the difference of Sect. 4.1 without modulo-operation has been chosen.

Results. For both algorithms on grid and threshold graphs, Fig. 3(b) shows the dependency of space usage on the node number exponent n, where the Dijkstra-approach is again dominating. Nevertheless, the linear growth of *all* four plots implies logarithmic growth w. r. t. N, which is the best case for symbolic algorithms' behavior. In general, this convenient property cannot be deduced just from logarithmic input OBDD-size.

Because Dijkstra's runtime is at least linear in the number of reachable nodes and the Bellman-Ford's runtime is at least linear in the minimum number of edges on any s–v-path, no polylogarithmic runtime can be obtained on grid graphs. Despite this theoretical fact, Bellman-Ford performed very efficiently on grids, while Dijkstra's runtime got very inefficient for the exponentially growing grid sizes (see Table 3(a)). Moreover, in experiments on grids with a number

Table 1. Experimental runtime results on random graphs.

p/N	100	200	300	400
0.1	0.332	1.49	3.828	7.715
0.2	0.34	1.611	4.423	10.405
0.3	0.361	1.891	5.362	12.934
0.4	0.371	2.07	6.232	15.192
0.5	0.394	2.121	6.869	16.843
0.6	0.401	2.256	7.508	18.468
0.7	0.393	2.342	8.191	19.621
0.8	0.406	2.409	8.529	20.347
0.9	0.415	2.479	8.724	20.794
1	0.415	2.497	8.912	21.156

(a) Dijkstra, random edges, difference weights.

p/N	100	200	300	400
0.1	0.148	0.693	1.691	3.032
0.2	0.148	0.535	1.433	2.961
0.3	0.165	0.504	1.313	2.601
0.4	0.154	0.538	1.441	2.593
0.5	0.155	0.504	1.472	2.624
0.6	0.13	0.483	1.33	3.267
0.7	0.135	0.509	1.484	3.541
0.8	0.133	0.506	1.541	3.245
0.9	0.135	0.553	1.525	3.773
1	0.134	0.508	1.75	3.693

(b) ParBF, random edges, difference weights.

p/N	100	200	300	400
0.1	0.654	3.048	7.669	14.76
0.2	0.783	3.932	9.849	20.64
0.3	0.959	4.535	12.281	26.619
0.4	1.054	5.268	14.664	32.572
0.5	1.184	6.169	16.796	37.954
0.6	1.244	6.919	19.521	42.984
0.7	1.433	7.567	21.875	49.621
0.8	1.457	7.977	24.446	55.312
0.9	1.615	8.454	26.745	61.136
1	1.674	9.415	28.467	66.74

(c) Dijkstra, random edges, random weights.

p/N	100	200	300	400
0.1	0.429	1.949	6.554	12.928
0.2	0.618	3.882	11	20.59
0.3	1.076	5.622	14.14	23.841
0.4	1.204	6.662	14.553	28.471
0.5	1.354	8.192	16.832	35.826
0.6	1.603	9.093	20.15	43.281
0.7	1.928	10.561	22.796	46.972
0.8	1.976	10.457	23.986	55.561
0.9	2.237	11.819	24.637	56.82
1	2.541	12.502	30.942	67.631

(d) ParBF, random edges, random weights.

of $20 \log N$ randomly added edges, both algorithms' runtime did not change essentially in comparison to unmodified grids.

Table 3(b) shows the observed runtimes on the considered threshold graphs. Due to the very small runtimes of the Bellman-Ford-approach, we cannot deduce any assumptions about its behavior besides that its again performing much more efficient than the Dijkstra-approach. Both on grids and threshold graphs with $n \geq 8$, it was even able to beat an explicit shortest-path algorithm implemented in LEDA[3] version 4.3.

[3] Available at `http://www.algorithmic-solutions.com/`.

Table 2. Experimental runtime results on grid and threshold graphs.

n	ParBF	Dijkstra
4	0.00	0.01
6	0.01	0.03
8	0.04	0.20
10	0.12	0.97
12	0.28	5.29
14	0.71	26.40
16	1.76	127.02

(a) Grid graphs.

n	ParBF	Dijkstra
2	0.00	0.00
3	0.00	0.00
4	0.00	0.01
5	0.00	0.02
6	0.01	0.07
7	0.01	0.18
8	0.02	0.42
9	0.02	1.01
10	0.02	2.38
11	0.02	5.47
12	0.03	12.25

(b) Threshold graphs.

5 Conclusions

Two symbolic algorithms for the single-source shortest-path problem on OBDD-represented graphs with nonnegative integral edge weights have been presented which execute $\mathcal{O}\big(N \cdot \log(NB)\big)$ resp. $\mathcal{O}\big(NM \cdot \log(NB)\big)$ OBDD-operations. Although Bellman-Ford's worst-case bound is the larger one, this symbolically parallelized approach was expected to have better runtime but higher space usage than the Dijkstra-approach. This was confirmed by experiments on random graphs with constant and difference weights as well as on grid and threshold graphs. Dijkstra's space usage was always of linear magnitude w. r. t. the size of its input OBDDs with a relative error of less than 0.06. For the Bellman-Ford-approach, this property was only observed on grid and threshold graphs as well as for the runtime on random graphs with constant edge weights.

Altogether, experiments both on pathological instances (random graphs) and structured graphs well-suited for symbolic representation (grid and threshold graphs) show that for each of the resources time resp. space at least one algorithm performs well or even asymptotically optimal w. r. t. the input OBDD-size. Hence, both shortest-path algorithms can be considered as useful symbolic methods with individual strengths.

Acknowledgment. Thanks to Ingo Wegener for proofreading and helpful discussions.

References

[1] R.I. Bahar, E.A. Frohm, C.M. Gaona, G.D. Hachtel, E. Macii, A. Pardo, and F. Somenzi. Algebraic decision diagrams and their applications. In *ICCAD'93*, pages 188–191. IEEE Press, 1993.

[2] R. Bellman. On a routing problem. *Quarterly of Applied Mathematics*, 16:87–90, 1958.

[3] R. Bloem, H.N. Gabow, and F. Somenzi. An algorithm for strongly connected component analysis in $n \log n$ symbolic steps. In *FMCAS'00*, volume 1954 of *LNCS*, pages 37–54. Springer, 2000.

[4] R.E. Bryant. Symbolic manipulation of Boolean functions using a graphical representation. In *DAC'85*, pages 688–694. ACM Press, 1985.

[5] R.E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers*, 35:677–691, 1986.

[6] E.W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1:269–271, 1959.

[7] R. Gentilini, C. Piazza, and A. Policriti. Computing strongly connected components in a linear number of symbolic steps. In *SODA'03*, pages 573–582. ACM Press, 2003.

[8] R. Gentilini and A. Policriti. Biconnectivity on symbolically represented graphs: A linear solution. In *ISAAC'03*, volume 2906 of *LNCS*, pages 554–564. Springer, 2003.

[9] G.D. Hachtel and F. Somenzi. *Logic Synthesis and Verification Algorithms*. Kluwer Academic Publishers, Boston, 1996.

[10] G.D. Hachtel and F. Somenzi. A symbolic algorithm for maximum flow in 0–1 networks. *Formal Methods in System Design*, 10:207–219, 1997.

[11] R. Hojati, H. Touati, R.P. Kurshan, and R.K. Brayton. Efficient ω-regular language containment. In *CAV'93*, volume 663 of *LNCS*, pages 396–409. Springer, 1993.

[12] N.V.R. Mahadev and U.N. Peled. *Threshold Graphs and Related Topics*. Elsevier Science, Amsterdam, 1995.

[13] I. Moon, J.H. Kukula, K. Ravi, and F. Somenzi. To split or to conjoin: The question in image computation. In *DAC'00*, pages 23–28. ACM Press, 2000.

[14] K. Ravi, R. Bloem, and F. Somenzi. A comparative study of symbolic algorithms for the computation of fair cycles. In *FMCAD'00*, volume 1954 of *LNCS*, pages 143–160. Springer, 2000.

[15] D. Sawitzki. Implicit flow maximization by iterative squaring. In *SOFSEM'04*, volume 2932 of *LNCS*, pages 301–313. Springer, 2004.

[16] D. Sawitzki. Implicit flow maximization on grid networks. Technical report, Universität Dortmund, 2004.

[17] D. Sawitzki. On graphs with characteristic bounded-width functions. Technical report, Universität Dortmund, 2004.

[18] D. Sawitzki. A symbolic approach to the all-pairs shortest-paths problem. Submitted, 2004.

[19] I. Wegener. *Branching Programs and Binary Decision Diagrams*. SIAM, Philadelphia, 2000.

[20] P. Woelfel. Symbolic topological sorting with OBDDs. In *MFCS'03*, volume 2747 of *LNCS*, pages 671–680. Springer, 2003.