

Implicit Maximization of Flows over Time

Daniel Sawitzki*

University of Dortmund, Computer Science 2

D-44221 Dortmund, Germany

`daniel.sawitzki@cs.uni-dortmund.de`

January 7, 2004

Abstract

Flows over time may be maximized by computing a static maximum flow on the corresponding time-expanded network, which contains a copy of the original network for each discrete time step. This makes available the whole algorithmic toolbox developed for static flows. The major drawback of this method compared to a polynomial (motivated by Ford and Fulkerson [6, 7]) is that the time-expanded networks may become exponentially large. However, the interconnections between the network copies are of simple structure. Implicit network representations by ordered binary decision diagrams (OBDDs) reward such structures with good compression. Therefore, the presented heuristic algorithm transforms an implicitly given 0-1 network into a time-expanded network, which serves as input of existing implicit methods for the static maximum flow problem. After their execution, the resulting maximum static flow is converted into a flow over time in the original network. It is shown that time-expanded networks have compact OBDD-representations under certain structural assumptions, which hopefully makes the flow maximization efficient w. r. t. time and space.

1 Introduction

Flows over time have been introduced about forty years ago by Ford and Fulkerson [6, 7]. We consider the computation of maximum flows over time in a *network* N consisting of an asymmetric directed graph (V, E) with *edge capacities* $u: E \rightarrow \mathbb{N}$. In contrast to static flow problems [4, 1], we additionally incorporate *transit times* $\tau: E \rightarrow \mathbb{N}$ needed by flow units to travel along an edge e . In such

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

a network, a *maximum flow* f has to be sent from a *source node* s to a *terminal node* t within a *time horizon* \mathcal{T} . In contrast to static flow maximization, we allow storage of flow at intermediate nodes. Due to natural capacities and transit times we may consider a discrete time model with times $\{0, \dots, \mathcal{T}\}$.

Ford and Fulkerson [6, 7] presented a reduction to the min-cost flow problem that yields a polynomial algorithm. Another technique in the context of flows over time is the generation of a *time-expanded network* N^{te} [6, 7, 5], which contains a copy of the original network for every discrete time $\rho \in \{0, \dots, \mathcal{T}\}$. A maximum static flow in N^{te} can be interpreted as a maximum flow over time in N . Then, a maximum static flow in N^{te} can be computed [4, 1] and interpreted as flow over time.

If N consists of $|V|$ nodes and $|E|$ edges with time horizon \mathcal{T} , the time-expanded network N^{te} consists of $O(|V| \cdot \mathcal{T})$ nodes and $O((|V| + |E|) \cdot \mathcal{T})$ edges. Thus, this method to compute maximum flows over time is pseudo-polynomial, and may take exponential runtime. Nevertheless, time-expanded networks are used within an FPTAS [5] for the quickest multicommodity flow problem, for which no polynomial algorithm is known yet.

Anyhow, we will use this approach to solve the maximum flow problem over time in 0-1 networks. That is, we consider a simplified variant of the problem where for all edges $e \in E$ it is $u(e) = 1$. This restriction enables an *implicit* representation of E using *ordered binary decision diagrams (OBDDs)* [2, 3, 12], which are graph-based data structures for Boolean functions. They are used to represent the characteristic function $\chi_E: V^2 \rightarrow \{0, 1\}$ of E , which maps binary encodings of edges $(v, w) \in E$ to 1, while other pairs $(v, w) \notin E$ are mapped to 0.

The size of an OBDD that represents a graph $G = (V, E)$ is bounded both by $O(|V|^2 / \log |V|)$ and $O(|E| \log^2 |V|)$. Hence, it is not essentially larger than the corresponding adjacency matrix resp. the adjacency lists. Because OBDDs reward structures and regularities with good compression, we hope that advantageous properties of G lead to “small,” that is sublinear or even polylogarithmic OBDD-sizes $O(\log^k |V|)$, for a constant k . Examples are grid graphs [15, 11], which have OBDDs of logarithmic size, and cographs [13], which have OBDDs of size $O(|V| \log |V|)$.

OBDDs offer a set of functional operations which are efficient w. r. t. the sizes of the participating OBDDs. These operations like disjunction and quantification have to suffice for the design of algorithms working on implicitly defined data (called *implicit algorithm*). Although each single operations is efficient, a sequence of $O(\log |V|)$ operations may generate OBDDs of exponential size. Thus, besides of the input OBDD-sizes, the over-all runtime of an implicit algorithm is essentially influenced by the sizes of OBDDs generated during the algorithm execution. In general, it is difficult to analyze these sizes for an interesting input subset. Two results of Sawitzki [11] and Woelfel [15] represent first steps in this

area. Often, only the number of OBDD-operations is bounded as a hint on the real over-all runtime. Because implicit algorithms typically have a higher worst-case runtime than corresponding algorithms which work on traditional adjacency lists (called *explicit algorithms*), they can be considered as heuristics to save time and/or space when the input graphs are heavily structured, and maybe too large for an explicit representation (e. g., by adjacency lists).

Implicit algorithms for some particular graph problems like reachability analysis are well established in the context of sequential circuits [8]. Newer research tries to attack more general graph problems by implicit methods. The algorithm of Hachtel and Somenzi [9] represents one of the first steps in this direction. It solves the maximum flow problem on implicitly given 0–1 networks, i. e., it computes an edge-disjoint path set of maximum cardinality between a source node $s \in V$ and a terminal node $t \in V$ in a network $N = (V, E, s, t)$. In the worst case, it performs $\Omega(|V|^3 \log |V|)$ OBDD-operations and takes over-all runtime $O(|V|^9 \log |V|)$. In experiments, the algorithm performed better and was able to efficiently solve maximum flow problems on state-transition graphs with 10^{27} nodes and 10^{36} edges.

Sawitzki [10] presented an implicit maximum flow algorithm which uses the technique of iterative squaring to compute at least one augmenting path by $O(\log |V|)$ OBDD-operations. This enables exponential less OBDD-operations on graphs with long augmenting paths from s to t . This approach hopefully leads to a better over-all runtime on specially structured networks. This has been shown for the case of grid networks [11], on which the algorithm takes over-all runtime $O(\log^3 |V|)$, while Hachtel and Somenzi’s method needs $\Omega(|V|^{1/2} \log |V|)$ operations.

Using implicit time-expanded networks, these maximum flow algorithms can directly be used to compute flows over time. In contrast, there is no implicit version of the polynomial algorithm using min-cost flows yet. Thus, the presented algorithm may be considered as a heuristic method to solve the maximum flow over time problem if the network size does not allow an explicit representation. Furthermore, the regular interconnection structure of time-expanded networks enables compact OBDD-representations. We hope that this leads to better over-all runtimes than the polynomial algorithm on special networks.

The paper is organized as follows. Section 2 formally introduces the problem of computing maximum flows over time as well as the detailed structure of time-expanded networks. Section 3 gives some foundations on implicit network representations by OBDDs. The implicit flow over time algorithm is described in Section 4. In Section 5, the consequences of different properties of the input network and the transit time function τ on the OBDD-sizes are considered. Finally, Section 6 gives conclusions, and hints to possible future work.

2 Maximum Flows over Time

We consider a *network* $N = (V, E, s, t, u, \tau)$, where (V, E) is a directed asymmetric graph, $s \in V$ the source, and $t \in V$ the terminal node. Function $u: E \rightarrow \mathbb{N}$ assigns *capacities* to the edges, while $\tau: E \rightarrow \mathbb{N}$ assigns *transit times* to them. In the *maximum flow over time problem*, our aim is to send as much flow as possible from source s to terminal t within a time horizon \mathcal{T} while respecting the edge capacities. We consider a problem variant that allows storage of flow at intermediate nodes.

Formally, we search a *flow over time* $f: E \times \{0, \dots, \mathcal{T}\} \rightarrow \mathbb{N}_0$, which assigns $f(e, \rho)$ units of flow to every edge e and time ρ . This means that $f(e, \rho)$ flow units are sent into edge e during the time interval $[\rho, \rho + 1[$. f has to respect the edge capacities, i.e., $f(e, \rho) \leq u(e)$, $e \in E$, $0 \leq \rho \leq \mathcal{T}$. In order to obey the time horizon \mathcal{T} , we require that $f(e, \rho) = 0$ for all $e = (v, w) \in E$, $w \neq t$, and $\mathcal{T} - \tau(e) \leq \rho \leq \mathcal{T}$. Furthermore, the flow has to fulfill the *flow conservation constraints*

$$\text{pot}(v, \rho) := \sum_{\xi=0}^{\rho} \left(\sum_{e=(u,v) \in E} f(e, \xi - \tau(e)) - \sum_{e=(v,w) \in E} f(e, \xi) \right) \geq 0 \quad (1)$$

for all $0 \leq \rho \leq \mathcal{T}$ and $v \in V \setminus \{s, t\}$. This enables flow storage at intermediate nodes, and guarantees for any time ρ that every node (except source and terminal) received at least as much flow as it gave away. Moreover, for $\rho = \mathcal{T}$ we require that equality holds in (1), meaning that no flow remains in the network after time \mathcal{T} . Finally, the *flow value* that has to be maximized is defined as $\text{pot}(t, \mathcal{T})$. We do not consider cost bounds in this paper.

The corresponding time-expanded network N^{te} contains a copy of N for every discrete time $\{0, \dots, \mathcal{T}\}$, and enables the application of static maximum flow algorithms. It consists of the nodes $V^{\text{te}} := V \times \{0, \dots, \mathcal{T}\}$, where $s^{\text{te}} := (s, 0)$ becomes the new source and $t^{\text{te}} := (t, \mathcal{T})$ becomes the new terminal. N^{te} contains the edges $E^{\text{te}} := E_1^{\text{te}} \cup E_2^{\text{te}}$, with

$$E_1^{\text{te}} := \left\{ ((v, \rho_v), (w, \rho_w)) \mid (v, w) \in E, \right. \\ \left. (\rho_w - \rho_v = \tau(v, w)), 0 \leq \rho_v, \rho_w \leq \mathcal{T} \right\} \quad (2)$$

and

$$E_2^{\text{te}} := \left\{ ((v, \rho), (v, \rho + 1)) \mid v \in V, 0 \leq \rho < \mathcal{T} \right\}.$$

Flow on an edge $((v, \rho_v), (w, \rho_w)) \in E_1^{\text{te}}$ corresponds to sending flow on $(v, w) \in E$ in the time interval $[\rho, \rho + 1[$. Therefore, all v - w -edges in E^{te} have the same capacity as in E .

The edges E_2^{te} are called *holdover edges*, and correspond to flow storage in time interval $[\rho, \rho + 1[$ in a node v . Therefore, they get unlimited capacity.

This paper’s algorithm solves a simplified variant of the problem where all edges $e \in E$ have the same capacity $u(e) = 1$. Because also the time-expanded network N^{te} may only contain edges of capacity 1, we will add extra-nodes and -edges to simulate edges with higher capacity (see Section 4).

3 Implicit Network Representation

In the following, the class of Boolean functions $f: \{0, 1\}^n \rightarrow \{0, 1\}$ is denoted by B_n . The i th character of a binary string x is denoted by x_i , while $|x| := \sum_{i=0}^{n-1} x_i 2^i$ identifies its value. If not stated otherwise, x has length n , and encodes the graph node number $|x|$.

Node resp. edge sets S handled by implicit algorithms are represented by their characteristic Boolean function χ_S , which maps binary encodings of elements $e \in S$ to 1 and all others to 0. Therefore, the nodes get numbers $|x| \in \{0, \dots, |V| - 1\}$, whose binary encoding x is passed to the function as $n := \lceil \log |V| \rceil$ Boolean variables. Edges are passed as pairs of node numbers by $2n$ variables.

The Boolean functions in turn are represented by OBDDs [2, 3, 12]. In particular, the π -OBDD G_f for the function $f \in B_n$ defined on the variable set $X := \{x_0, \dots, x_{n-1}\}$ is an acyclic directed graph containing two kinds of nodes: *inner nodes* and *sinks*. Inner nodes are labeled with variables, and have two outgoing edges: the *0-edge* and the *1-edge*. Sinks are labeled with one of the function values 0 or 1 and have no outgoing edges. Finally, the *function pointer* points to the node that represents f . The variable labels at inner nodes have to respect the permutation $\pi \in \Sigma_n$ (called *variable ordering*) on every path from the function pointer to a sink. This implies that every variable occurs at most once on these paths.

In order to evaluate the function according to a given variable assignment, we start at the function pointer. If we reach an inner node v labeled with variable x_i , we follow v ’s 0-edge for $x_i = 0$ respectively its 1-edge for $x_i = 1$. If we reach a sink, its label gives us the desired function value. Each variable is tested once at most, why f can be evaluated in time $O(n)$.

The *size* $|G_f|$ of an OBDD G_f is measured by the number of its nodes. In general, the OBDD-representation of a function $f \in B_n$ is not canonical. Though, for a fixed variable ordering π , we can minimize a π -OBDD F for f in linear time $|F|$. In this paper, we adopt the usual assumption that all occurring OBDDs are minimized. This is reasonable since all mentioned OBDD-operations produce only minimized diagrams, which are known to be canonical.

Whether a function f is satisfiable (i. e. $f \neq 0$) can be decided in time $O(1)$. The negation \bar{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 $O(|G_f|)$. Whether two functions f and g are equivalent (i. e. $f = g$) can be decided in time $O(|G_f| + |G_g|)$. These operations

are considered as cheap.

Further essential operations offered by OBDDs are the *binary synthesis* $f \otimes g$ for $f, g \in B_n$, $\otimes \in B_2$ and the *quantification* $(Qx_i)f$ for $Q \in \{\exists, \forall\}$. The computation of $G_{f \otimes g}$ takes time $O(|G_f| \cdot |G_g|)$, and, therefore, is considered as an expensive operation. The quantification $(Qx_i)f$ can be realized as two cheap operations and one binary synthesis, why it is considered as an expensive operation.

Besides implicit node sets $S \subseteq V$ and edge sets $T \subseteq V^2$ there may be functions created within an algorithm which receive more than two node arguments. Let k be the maximum number of node encodings passed to a characteristic function along the algorithm execution. The OBDDs are then defined on the variable set $X_{k,n} := \{x_j^i \mid 1 \leq i \leq k, 0 \leq j \leq n-1\}$, which contains k subsets $x^i \in \{0, 1\}^n$ corresponding to node numbers $|x^i|$, $1 \leq i \leq k$.

The *interleaved variable ordering* $\pi_{k,n} \in \Sigma_{kn}$ defined on $X_{k,n}$ maps variable x_j^i to position $jk + i$. Accordingly, $\pi_{k,n}$ tests the node numbers' bits with increasing significance, and has the appearance

$$(x_0^1, \dots, x_0^k, x_1^1, \dots, x_1^k, \dots, x_{n-1}^1, \dots, x_{n-1}^k).$$

The use of $\pi_{k,n}$ as the variable ordering for an OBDD G_f allows to swap node arguments in time and space $O(|G_f|)$. Furthermore, it leads to small OBDDs of size $O(n)$ for important functions like value comparisons (see Section 5.1). Due to these convenient properties, this paper's algorithm uses $\pi_{k,n}$ as variable ordering.

While the characteristic function of a set S is usually denoted by $\chi_S \in B_n$ and G_{χ_S} identifies the OBDD for χ_S , we simplify the notation, and just write S or $S(x)$ for all three kinds of representations. Furthermore, both the binary encoding x of the node $v_{|x|}$ as well as the node itself are just denoted by x . For example, the OBDD G_{χ_E} representing a graph's edge set E is denoted by $E(x, y)$; for the singletons $\{s\}$ and $\{t\}$ of source and terminal node this is $s(x)$ resp. $t(x)$. In the following, the OBDD-representation of node and edge sets and the operations offered by OBDDs will be considered as black boxes. In Section 4, the algorithm will be described mainly by means of functional assignments like " $F(x) := G(x) \wedge H(x)$." This example corresponds to the computation of a new OBDD called $F(x)$ for the function $g \wedge h$, where $G(x)$ represent g , $H(x)$ represents h , and all three functions are defined on the variables $x = x_0, \dots, x_{n-1}$ that encode one graph node number $|x|$.

4 The Algorithm

The algorithm receives the input network $N = (V, E, s, t, \tau)$ in form of four OBDDs $E(x, y)$, $s(x)$, $t(x)$, and $TAU(x, y, \rho)$. $E(x, y)$ is the characteristic function of the edge set E , $s(x)$ and $t(x)$ represent the singletons $\{s\}$ and $\{t\}$, and $TAU(x, y, \rho)$ corresponds to the transit time function $\tau: E \rightarrow \mathbb{N}$. In contrast to

the characteristic functions considered so far, TAU depends also on a variable set (namely ρ) that represents no node number but the natural function value $\tau(e)$. TAU 's first two arguments x and y match the start- resp. end-node of τ 's edge argument.

To enable a small OBDD-size, TAU does not exactly reflect τ ; we relax its definition so that $TAU(x, y, \rho) = 1$ does not necessarily imply $(x, y) \in E$. Instead, we define it as

$$TAU(x, y, \rho) = 1 \Leftrightarrow ((x, y) \in E \Rightarrow \tau(x, y) = \rho).$$

This enables us to represent constant transit times $\tau \equiv c$, $c \in \mathbb{N}$, by the simple check $(\rho = c) =: TAU(x, y, \rho)$, which has an OBDD of size $O(\log(|V| \cdot \mathcal{T}))$ (see Section 5.1).

The result of the algorithm will be an OBDD $FOT(x, y, \rho)$ which represents the set of all edges $e = (x, y)$ and times ρ such that the maximum flow over time lays flow on e in the time interval $[\rho, \rho + 1[$.

4.1 Replacing the hold-over arcs

Holdover edges $((v, \rho), (v, \rho + 1))$ model the storage of flow in a node $v \in V$ from time ρ to $\rho + 1$. Intuitively, they have unlimited capacity due to the unlimited storage capacity of nodes. Both implicit algorithms [9, 10] which may be used to solve the static flow problem on the time-expanded network are only able to handle 0-1 networks. Anyhow, in a 0-1 network, at most $(|V| - 1) \cdot \mathcal{T} \leq |V| \cdot \mathcal{T} - 1$ units of flow can enter a node $v \in V$ until time \mathcal{T} . Therefore, we simulate the behavior of an unlimited edge by introducing extra-nodes and -edges. We extend the node set V^{te} by building the Cartesian product with the set $\{0, \dots, |V| \cdot \mathcal{T} - 1\}$, i.e.,

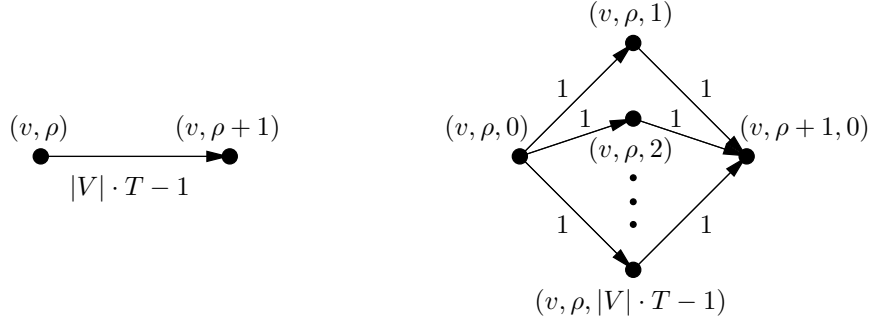
$$V^{\text{te}} := V \times \{0, \dots, \mathcal{T}\} \times \{0, \dots, |V| \cdot \mathcal{T} - 1\}.$$

We will use nodes $(v, \rho, \sigma) \in V^{\text{te}}$ with $\sigma > 0$ as extra-nodes, while node $(v, \rho, 0) \in V^{\text{te}}$ will represent the copy of v for time ρ . Therefore, the new definition of E_1^{te} demands $\sigma = 0$ and it is $s^{\text{te}} := (s, 0, 0)$ and $t^{\text{te}} := (t, \mathcal{T}, 0)$:

$$E_1^{\text{te}} := \left\{ ((v, \rho_v, 0), (w, \rho_w, 0)) \mid (v, w) \in E, \right. \\ \left. (\rho_w - \rho_v = \tau(v, w)), 0 \leq \rho_v, \rho_w \leq \mathcal{T} \right\}. \quad (3)$$

Instead of holdover edges $((v, \rho), (v, \rho + 1))$ with capacity $|V| \cdot \mathcal{T} - 1$, we use edges

$$E_2^{\text{te}} := \left\{ ((v, \rho, 0), (v, \rho, \sigma)), ((v, \rho, \sigma), (v, \rho + 1, 0)) \mid \right. \\ \left. v \in V, \rho \in \{0, \dots, \mathcal{T} - 1\}, \sigma \in \{1, \dots, |V| \cdot \mathcal{T} - 1\} \right\} \quad (4)$$



(a) Holdover edge $((v, \rho), (v, \rho + 1))$. (b) Subgraph that simulates the holdover edge.

Figure 1: Holdover edge of capacity $|V| \cdot \mathcal{T} - 1$ versus an equivalent construction with $|V| \cdot \mathcal{T} - 1$ extra-nodes.

with capacity 1. Figure 1 illustrates that this replacement does not influence the maximum flow value.

4.2 Building the time-expanded network

In order to apply an implicit static maximum flow method, we have to generate an OBDD $E^{\text{te}}(x, \rho_x, \sigma_x, y, \rho_y, \sigma_y)$ for the time-expanded network's edge set E^{te} , which depends on binary encodings of all three components of both the start- and the end-edge. We compose it from two functions that correspond to the edges E_1^{te} and E_2^{te} , which in turn are obtained by Boolean equations that correspond to (3) and (4):

$$E_1^{\text{te}}(x, \rho_x, \sigma_x, y, \rho_y, \sigma_y) := E(x, y) \wedge (\exists \rho) [(\rho_y - \rho_x = \rho) \wedge TAU(x, y, \rho)] \wedge (\sigma_x = \sigma_y = 0). \quad (5)$$

This equation corresponds to (3), and explains as follows: To be connected in N^{te} , (x, y) has to be an original edge in E . Furthermore, there has to be a time ρ which is the difference between the times ρ_y of y and ρ_x of x and which equals to $\tau(x, y)$. This is the Boolean formulation of the condition $\rho_y - \rho_x = \tau(x, y)$. Finally, these edges are not defined on the extra-nodes but on the copies of V , why the σ -values have to be 0.

The following equation defines the edges E_2^{te} that simulate holdover edges

corresponding to (4):

$$\begin{aligned}
E_2^{\text{te}}(x, \rho_x, \sigma_x, y, \rho_y, \sigma_y) &:= (x = y) \\
&\wedge \left[((\rho_x = \rho_y) \wedge (\sigma_x = 0) \wedge (\sigma_y > 0)) \right. \\
&\quad \left. \vee ((\rho_y - \rho_x = 1) \wedge (\sigma_x > 0) \wedge (\sigma_y = 0)) \right]. \quad (6)
\end{aligned}$$

Because the extra-edges of E_2^{te} are used to model the storage of flow at nodes $v \in V$ over time, their node components x and y have to be equal. The case $\rho_x = \rho_y$ corresponds to edges that leave a copy node ($\sigma_x = 0$) and enter an extra-node ($\sigma_y > 0$). Analogously, the case $\rho_y - \rho_x = 1$ corresponds to edges that leave an extra-node and enter a copy node.

Finally, we have to conjoin E_1^{te} and E_2^{te} :

$$E^{\text{te}}(x, \rho_x, \sigma_x, y, \rho_y, \sigma_y) := E_1^{\text{te}}(x, \rho_x, \sigma_x, y, \rho_y, \sigma_y) \vee E_2^{\text{te}}(x, \rho_x, \sigma_x, y, \rho_y, \sigma_y).$$

Note that the argument values are only bounded by the number of bits representing them. Therefore, the OBDD for E^{te} may contain more edges than provided by (3) and (4): If ρ_x and ρ_y are encoded by n bits and \mathcal{T} is smaller than $2^n - 1$, then there are copies V corresponding to times $\rho > \mathcal{T}$. However, this does not influence the maximum flow in N^{te} , because $t^{\text{te}} := (t, \mathcal{T}, 0)$ has no ingoing edge from nodes (v, ρ, σ) with $\rho > \mathcal{T}$.

The same reason may cause that the amount of extra nodes (v, ρ, σ) , $\sigma > 0$, per node $v \in V$ and time $\rho \in \{0, \dots, 2^n - 1\}$ is higher than the largest possible flow storage at v . Of course, this neither changes the maximum flow.

Source and terminal of N^{te} are defined straight-forward:

$$s^{\text{te}}(x, \rho_x, \sigma_x) := s(x) \wedge (\rho_x = \sigma_x = 0), \quad (7)$$

$$t^{\text{te}}(x, \rho_x, \sigma_x) := t(x) \wedge (\rho_x = \mathcal{T}) \wedge (\sigma_x = 0). \quad (8)$$

Equations (5), (6), (7), and (8) for E^{te} , s^{te} , and t^{te} are build from basic operations of B_2 (disjunction, conjunction) and existential quantifications on the input functions $E(x, y)$, $s(x)$, and $t(x)$ as well as on supporting functions that realize comparisons like $\rho_x = \mathcal{T}$. The functional operations are provided by the OBDD data structure, while the comparison functions have small OBDDs, and, therefore, can be efficiently generated. They are composable of multivariate threshold functions introduced by Woelfel [15], and are handled more detailed in Section 5.1. Altogether, the equations that lead to the characteristic functions of N^{te} can be generated by OBDD-operations.

4.3 Computing the maximum flow over time

We now apply an implicit algorithm on the time-expanded network that solves the static maximum flow problem on it. One can use Hachtel and Somenzi's

algorithm [9] as well as the iterative squaring approach of Sawitzki [10]. These methods return an OBDD for the function $F(x, \rho_x, \sigma_x, y, \rho_y, \sigma_y)$ which implicitly represents the set of all edges carrying a flow unit. From this flow we have to compute the final result $FOT(x, y, \rho)$. Note that flow on holdover edges does not correspond to flow in the final maximum flow over time. Only flow on edges of E_1^{te} contributes to FOT . This leads to the following equation:

$$FOT(x, y, \rho) := (\exists \rho') F(x, \rho, 0, y, \rho', 0).$$

The maximum flow over time lays flow on (x, y) from time ρ to $\rho + 1$ if and only if the maximum flow in the time-expanded network carries flow from node $(x, \rho, 0)$ to some node $(y, \rho', 0)$. Therefore, it suffices to replace σ_x and σ_y by 0 and afterwards to quantify existentially over the end-edge's time component ρ' in order to obtain FOT .

5 OBDD-Sizes of Time-Expanded Networks

We now investigate upper bounds for the OBDD-size of time-expanded networks. In general, the size of an OBDD depending on n variables is bounded by $(2 + o(1))2^n/n$ [12]. E^{te} depends on the six variable sets $x, \rho_x, \sigma_x, y, \rho_y,$ and σ_y . Both node numbers x and y can be represented by $\lceil \log |V| \rceil$ variables each. ρ_x and ρ_y are bounded by \mathcal{T} , why $\lfloor \log \mathcal{T} \rfloor + 1$ variables suffices to represent each of these time values. Finally, each σ -value is bounded by $|V| \cdot \mathcal{T} - 1$, why $\lceil \log(|V| \cdot \mathcal{T}) \rceil$ variables suffice. This leads to the worst-case size

$$|E^{\text{te}}| = O\left(\frac{2^{4(\log |V| + \log \mathcal{T})}}{\log |V| + \log \mathcal{T}}\right) = O\left(\frac{|V|^4 \cdot \mathcal{T}^4}{\log |V| + \log \mathcal{T}}\right).$$

Of course, the use of heuristics like implicit flow algorithms is motivated by the hope that the worst case does not apply. In contrast, we hope that OBDDs are suited to catch the structures of the considered time-expanded networks. In the following, we will present a function property leading to small OBDD-sizes, and analyze E^{te} 's size under different degrees of structural assumptions. For the singletons $s(x)$, $t(x)$, $s^{\text{te}}(x, \rho_x, \sigma_x)$, and $t^{\text{te}}(x, \rho_x, \sigma_x)$ we will obtain trivial size bounds without any assumptions.

5.1 Functions with complete OBDDs of constant width

An OBDD F defined on n variables is called *complete* if every path from F 's function pointer to a sink has length n [12]. This may cause that the OBDD is not minimal. The *width* of an OBDD refers to the maximum numbers of nodes labeled with same variable.

Woelfel [15] has introduced the class of k -variate threshold functions.

Definition 1 A function $f \in B_{kn}$, $k, n \in \mathbb{N}$, defined on the variable set $X_{k,n} := \{x_j^i \mid 1 \leq i \leq k, 0 \leq j < n\}$ is called k -variate threshold function if there are weights $w_1, \dots, w_k \in \mathbb{Z}$ and a threshold $T \in \mathbb{Z}$ such that

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

The maximum absolute weight of f is defined as $w(f) := \max\{|w_1|, \dots, |w_k|\}$.

An interesting property of k -variate threshold functions f with constant $k = O(1)$ and $w(f) = O(1)$ is that they have complete $\pi_{k,n}$ -OBDDs \mathcal{F} of constant width, where $\pi_{k,n}$ is the interleaved variable ordering (see Section 3). That is, every variable $x_j^i \in X_{k,n}$ labels only $O(1)$ nodes and \mathcal{F} has size $O(n)$. This holds in particular for the corresponding minimized OBDD F as well as for any formula over a constant number of threshold functions. The latter follows from Lemma 1 which is proven in Appendix A.

Lemma 1 Let F and G minimal π -OBDDs of functions $f, g \in B_n$, $\pi \in \Sigma_n$, which have complete π -OBDDs \mathcal{F} and \mathcal{G} of constant width k . The binary synthesis [12] $F \otimes G$, $\otimes \in B_2$, generates a minimal π -OBDD H of size $O(n)$ in time and space $O(n)$ which also has a complete π -OBDD \mathcal{H} of constant width k^2 .

It is easy to see that also the comparisons $=$, $>$, \leq , and $<$ can be expressed by a formula of constant size over threshold functions. For example, $\rho_x = \rho_y$ corresponds to the formula

$$[1 \cdot \rho_x + (-1) \cdot \rho_y \geq 0] \wedge [(-1) \cdot \rho_x + 1 \cdot \rho_y \geq 0].$$

In general, applying $O(n)$ existential quantifications on an OBDD of size $O(n)$ may cause an exponential blow-up. Lemma 2 [14] states that this worst cases does not apply for functions with complete OBDDs of constant width. Its proof has been moved to Appendix A.

Lemma 2 Let $f \in B_n$ be a Boolean function defined on the variables $\{x_0, \dots, x_{n-1}\}$ that has a complete π -OBDD \mathcal{F} , $\pi \in \Sigma_n$, of constant width k . Then, for every sequence x_{i_1}, \dots, x_{i_r} of c variables and a quantifier $Q \in \{\exists, \forall\}$, the function $g := (Qx_{i_r} \dots Qx_{i_1})f$, $r \leq c$, has a complete π -OBDD \mathcal{G} of width $2^k = O(1)$.

That is, also the existential quantification of the minimized OBDD F of f over an arbitrary variable subset produces intermediate OBDDs of the same asymptotical size $O(|\mathcal{F}|) = O(n)$.

Definition 1 assumes that all arguments x^i consists of the same number n of variables. In our case, n is dominated by the σ -values, which are encoded by $n := \left\lceil \log(|V| \cdot \mathcal{T}) \right\rceil$ variables. The functions $s(x)$ and $t(x)$ resp. $s^{\text{te}}(x, \rho_x, \sigma_x)$ and

$t^{\text{te}}(x, \rho_x, \sigma_x)$ represent singletons, that is, each of them maps exactly one variable assignment to 1. The minimal OBDD for such functions consists of just one path to the 1-sink, and has size $O(n)$.

At next, we will consider assumptions of different strength and their consequences on the size of E^{te} . In detail, we will assume that both E and TAU have complete OBDDs of constant width respectively that only one of both has this property.

5.2 Assuming a compact edge relation

Here, we consider the case that $E(x, y)$ is compact by means of having a complete OBDD of constant width, while the size of $TAU(x, y, v)$ is only bounded by the worst case $O(|V|^2 \cdot \mathcal{T}/n)$. Grid networks for example have such compact OBDDs [15, 11]. In order to analyze the size of E_1^{te} , we rewrite its definition:

$$\begin{aligned} I_1(x, \rho_x, y, \rho_y) &:= (\exists \rho) [(\rho_y - \rho_x = \rho) \wedge TAU(x, y, \rho)], \\ I_2(x, \sigma_x, y, \sigma_y) &:= E(x, y) \wedge (\sigma_x = \sigma_y = 0), \\ E_1^{\text{te}}(x, \rho_x, \sigma_x, y, \rho_y, \sigma_y) &:= I_1(x, \rho_x, y, \rho_y) \wedge I_2(x, \sigma_x, y, \sigma_y). \end{aligned}$$

The intermediate OBDD for I_1 occurring during the computation of (5) depends on x, y, ρ_x , and ρ_y . We can bound its size only by the worst case $O(|V|^2 \cdot \mathcal{T}^2/n)$. Being a formula over a constant number of threshold functions, I_2 has size $O(n)$. Therefore, E_1^{te} has size $O(|I_1| \cdot |I_2|) = O(|V|^2 \cdot \mathcal{T}^2 \cdot n/n) = O(|V|^2 \cdot \mathcal{T}^2)$.

E_2^{te} is composed of threshold functions, and, therefore, has size $O(n)$. The union with E_1^{te} leads to $|E^{\text{te}}| = O(|V|^2 \cdot \mathcal{T}^2 \cdot n)$. Although this size is smaller than the general worst-case size of E^{te} without assumptions, the flow maximization has still pseudopolynomial runtime.

5.3 Assuming compact transit times

Here, we consider the case that $TAU(x, y, v)$ is compact by means of having a complete OBDD of constant width, while the size of $E(x, y)$ is only bounded by the worst case $O(|V|^2/\log |V|)$. Constant transit times $\tau \equiv c, c \in \mathbb{N}$, for example have such compact OBDDs $TAU(x, y, v) := (v = c)$. In order to analyze the size of E_1^{te} , we rewrite its definition:

$$\begin{aligned} I_3(x, \rho_x, \sigma_x, y, \rho_y, \sigma_y) &:= (\exists \rho) [(\rho_y - \rho_x = \rho) \wedge TAU(x, y, \rho)] \wedge (\sigma_x = \sigma_y = 0), \\ E_1^{\text{te}}(x, \rho_x, \sigma_x, y, \rho_y, \sigma_y) &:= E(x, y) \wedge I_3(x, \rho_x, \sigma_x, y, \rho_y, \sigma_y). \end{aligned}$$

For $\rho_y - \rho_x = \rho$, TAU , and $(\sigma_x = \sigma_y = 0)$ the constant width property holds. Due to Lemmas 1 and 2, it still holds for I_3 being the result of $O(1)$ binary synthesis and one quantification. Altogether, I_3 has size $O(n)$, so that E_1^{te} has size $O(|V|^2 \cdot n/\log |V|)$. The conjoin with E_2^{te} leads to $|E^{\text{te}}| = O(|V|^2 \cdot n^2/\log |V|)$.

Hence, the size of the implicit time-expanded network is polynomial in the input size $|V| + |E| + \log T$, and we may hope that this holds also for the over-all runtime of the flow maximization.

5.4 Assuming a compact edge relation and compact transit times

If we assume that both $E(x, y)$ and $TAU(x, y, v)$ have the constant width property, it follows from above that E^{te} has also a complete OBDD of constant width. Therefore, the implicit time-expanded network has size $|E^{te}| = O(n) = O(\log |V| + \log T)$, which is linear in the input size $|V| + |E| + \log T$.

6 Conclusions

An implicit algorithm for the maximum flow over time problem has been presented which works on OBDD-representations of the input network N and the transit times function τ . It generates an implicit time-expanded network N^{te} , and incorporates existing implicit algorithms for the static maximum flow problem. Due to the simple interconnection structure of time-expanded networks, a compact OBDD-representation of τ suffices to obtain also a compact OBDD for N^{te} , which is a necessary (though not sufficient) condition for the efficiency of OBDD-algorithms.

Altogether, we hope that the algorithm, being a heuristic, represents a more efficient alternative to explicit algorithms w. r. t. time and/or space on large, but heavily structured networks that arise in practical applications.

At the moment, there exist only implicit maximum flow algorithms for the special case of 0-1 networks. Areas of future research could be the extension to networks with arbitrary edge capacities as well as attacking further flow problems like the computation of multicommodity flows with implicit methods. This would enable the generation of implicit condensed time-expanded networks [5], because these contain scaled edge capacities.

Acknowledgments

Thanks to Heiko Schilling for drawing our attention to the maximum flows over time problem.

References

- [1] R.K. Ahuja, T.L. Magnanti, and J.B. Orlin. *Network Flows*. Prentice Hall, Englewood Cliffs, 1993.

- [2] R.E. Bryant. Symbolic manipulation of Boolean functions using a graphical representation. In *Design Automation Conference*, pages 688–694. ACM Press, 1985.
- [3] R.E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers*, 35:677–691, 1986.
- [4] T.H. Cormen, C.E. Leiserson, and R.L. Rivest. *An Introduction to Algorithms*. McGraw-Hill, New York, 1990.
- [5] L. Fleischer and M. Skutella. The quickest multicommodity flow problem. In *Integer Programming and Combinatorial Optimization*, volume 2337 of *Lecture Notes in Computer Science*, pages 36–53. Springer, 2002.
- [6] L.R. Ford and D.R. Fulkerson. Constructing maximal dynamic flows from static flows. *Operations Research*, 6:419–433, 1958.
- [7] L.R. Ford and D.R. Fulkerson. *Flows in Networks*. Princeton University Press, Princeton, 1962.
- [8] G.D. Hachtel and F. Somenzi. *Logic Synthesis and Verification Algorithms*. Kluwer Academic Publishers, Boston, 1996.
- [9] 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.
- [10] D. Sawitzki. Implicit flow maximization by iterative squaring. In *SOFSEM 2004: Theory and Practice of Computer Science*, volume 2932 of *Lecture Notes in Computer Science*, pages 301–313. Springer, 2004.
- [11] D. Sawitzki. Implicit flow maximization on grid networks. Technical report, Universität Dortmund, 2004. URL <http://ls2-www.cs.uni-dortmund.de/~sawitzki/IFMoGN.pdf>.
- [12] I. Wegener. *Branching Programs and Binary Decision Diagrams*. SIAM, Philadelphia, 2000.
- [13] P. Woelfel. The OBDD-size of cographs. Internal report, Universität Dortmund, 2003.
- [14] P. Woelfel. Private communication, 2003.
- [15] P. Woelfel. Symbolic topological sorting with OBDDs. In *Mathematical Foundations of Computer Science*, volume 2747 of *Lecture Notes in Computer Science*, pages 671–680. Springer, 2003.

A Proofs

Because there is a variety of possible implementations of the binary synthesis, we adopt the usual assumption that the algorithm recursively constructs the reachable part of F 's and G 's cross product graph before it minimizes the result in linear time.

Proof of Lemma 1. We assume w.l.o.g. $\pi = id$ for the variable ordering π . At first, we convince ourselves that the binary synthesis $\mathcal{F} \otimes \mathcal{G}$ on the complete OBDDs \mathcal{F} and \mathcal{G} takes runtime $O(n)$. Let v and w be two OBDD nodes from \mathcal{F} resp. \mathcal{G} which represent two subfunctions f' resp. g' . Due to the completeness of \mathcal{F} and \mathcal{G} , f' and g' participate in a recursive computation $f' \otimes g'$ if and only if v and w are labeled with the same variable. Hence, \mathcal{F} 's and \mathcal{G} 's cross product graph contains at most k^2 nodes in each layer and the OBDD $\mathcal{H} := \mathcal{F} \otimes \mathcal{G}$ is computed in time and space $O(n)$.

We now investigate the synthesis of the minimal OBDDs F and G . These are obtained from \mathcal{F} resp. \mathcal{G} through successive applications of two minimization operators [12]:

- Eliminate nodes v whose 0- and 1-edge point to the same node.
- Merge nodes v and w which are labeled with the same variable, and which have the same a -successors $v_a = w_a$, $a \in \{0, 1\}$.

For general OBDDs A and B , the result $A \otimes B$ may have size $O(|A| \cdot |B|)$, because recursive computations may combine nodes on different layers. This problem does not occur in $F \otimes G$: a -edges (v, v_a) in F or G , $a \in \{0, 1\}$, which skip layers are the result of elimination operators that have been applied on \mathcal{F} resp. \mathcal{G} . Let v of F lie in layer i , while w of G lies in $j > i$. When a recursive computation $f' \otimes g'$ combines v and w , this corresponds to the combination of two nodes v' and w' in the computation of $\mathcal{F} \otimes \mathcal{G}$ that lie in the same layer $\ell \leq i < j$, and whose successive elimination redirected their ingoing edges to v and w (see Figure 2). Therefore, every recursive binary synthesis within $F \otimes G$ can be associated with one within $\mathcal{F} \otimes \mathcal{G}$. Hence, the OBDD $H := F \otimes G$ contains not more nodes than \mathcal{H} , and can be computed in time and space $O(n)$. \square

Proof of Lemma 2. We assume w.l.o.g. that \mathcal{F} contains exactly k nodes in each layer $\ell \in \{0, \dots, n-1\}$ as well as $\pi = id$ for the variable ordering π . At first, we construct an OBDD \mathcal{P} of width 2^k that contains 2^k nodes V_1, \dots, V_{2^k} in layer ℓ corresponding to the 2^k subsets of \mathcal{F} 's nodes $\{v_1, \dots, v_k\}$ in layer ℓ . Moreover, we define the a -successor V_a , $a \in \{0, 1\}$, of a node V in \mathcal{P} as the union of the a -successors of all original nodes $v \in V$, i. e., $V_a := \bigcup_{w \in V} \{w_a\}$.

We now successively perform the quantifications on \mathcal{P} in order to construct \mathcal{G} , where each constructed OBDD \mathcal{G}_r for $g_r := (Qx_{i_r} \dots Qx_{i_1})f$ will not be larger than \mathcal{P} .

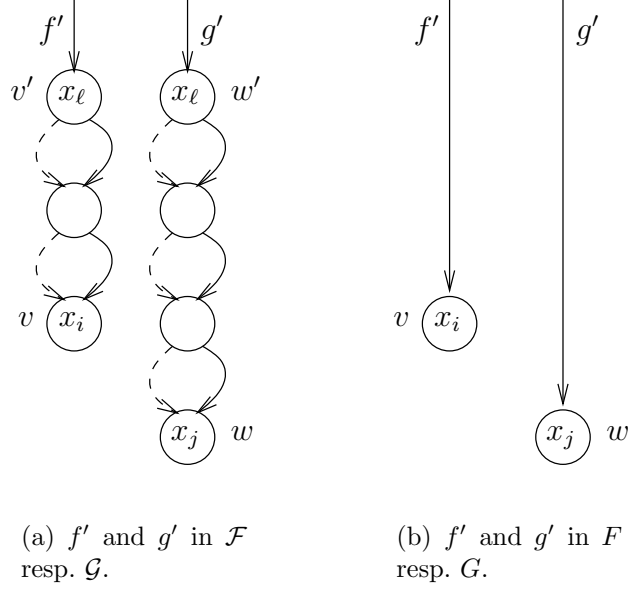


Figure 2: Examples of OBDD-representations of subfunctions f' and g' .

Let V_0 and V_1 be the 0- resp. 1-successor of V . We perform the quantification over variable x_ℓ by replacing all edges (U, V) (which point to nodes V in layer ℓ) by $(U, V_0 \cup V_1)$.

If the evaluation traversal on \mathcal{G}_r for a given input assignment x visits $V \subseteq \{v_1, \dots, v_k\}$, this node represents the union of nodes reachable in the original OBDD \mathcal{F} for any assignment of the quantified variables x_{i_1}, \dots, x_{i_r} .

In generating \mathcal{P} and the OBDDs $\mathcal{G}_1, \dots, \mathcal{G}_c := \mathcal{G}$, we also create terminal subsets of $\{0, 1\}$. When applying a sequence of existential quantifiers ($Q = \exists$), the sets $\{1\}$ and $\{0, 1\}$ are replaced by the terminal 1 and all others by 0—it suffices if one assignment of the quantified variables leads to 1. When applying universal quantifiers ($Q = \forall$), only the set $\{1\}$ is replaced by 1—all assignments must lead to 1.

We have constructed a complete OBDD \mathcal{G} for g of width $2^k = O(1)$. \square