An Efficient Implicit OBDD-Based Algorithm for Maximal Matchings*

Beate Bollig[1] and Tobias Pröger[2]

[1] TU Dortmund, LS2 Informatik, Germany
[2] ETH Zürich, Institut für Theoretische Informatik, Switzerland

Abstract. The maximal matching problem, i.e., the computation of a matching that is not a proper subset of another matching, is a fundamental optimization problem and algorithms for maximal matchings have been used as submodules for problems like maximal node-disjoint paths or maximum flow. Since in some applications graphs become larger and larger, a research branch has emerged which is concerned with the design and analysis of implicit algorithms for classical graph problems. Input graphs are given as characteristic Boolean functions of their edge sets and problems have to be solved by functional operations. As OBDDs, which are closely related to deterministic finite automata, are a well-known data structure for Boolean functions, OBDD-based algorithms are used as a heuristic approach to handle very large graphs. Here, an implicit OBDD-based maximal matching algorithm is presented that uses only a polylogarithmic number of functional operations with respect to the number of vertices of the input graph.

1 Introduction

Since some modern applications require huge graphs, explicit representations by adjacency matrices or adjacency lists may cause conflicts with memory limitations and even polynomial time algorithms are sometimes not fast enough. As time and space resources do not suffice to consider individual vertices and edges, one way out seems to be to deal with sets of vertices and edges represented by their characteristic functions. Ordered binary decision diagrams, denoted OBDDs, are well suited for the representation and manipulation of Boolean functions [5]. They are closely related to deterministic finite automata for Boolean languages L, where $L \subseteq \{0,1\}^n$ (see, e.g., Section 3.2 in [15]). OBDDs are able to take advantage over the presence of regular substructures which leads sometimes to sublinear graph representations. Therefore, a research branch has emerged which is concerned with the design and analysis of so-called implicit or symbolic algorithms for classical graph problems on OBDD-represented graph instances (see, e.g., [1–3], [6, 7], [9], [12, 13], and [16]). Implicit algorithms have to solve problems on a given graph instance by efficient functional operations offered by the OBDD data structure.

* The first author has been supported by DFG project BO 2755/1-1.

The maximal matching problem, i.e., the computation of a matching that is not a proper subset of another matching, is a fundamental optimization problem and algorithms for maximal matchings have been used in some maximal node-disjoint paths or maximum flow algorithms (see, e.g., [8]). The design of efficient implicit algorithms requires new paradigms and techniques but it has turned out that some methods known from the design of parallel algorithms are useful, e.g., the technique of iterative squaring is similar to the path-doubling strategy. Using an efficient degree reduction procedure, the first optimal parallel algorithm for maximal matchings has been presented by Kelsen [11]. It runs in time $\mathcal{O}(\log^3 |V|)$ using $\mathcal{O}((|E| + |V|)/\log^3 |V|)$ processors on bipartite graphs $G = (V, E)$ and is optimal in the sense that the time processor product is equal to that of the best sequential algorithm. The main result of our paper is the following one.

Theorem 1. *A maximal bipartite matching in an implicitly defined graph $G = (V, E)$ can be implicitly computed by $\mathcal{O}(\log^4 |V|)$ functional operations on Boolean functions over a logarithmic number of Boolean variables. For general graphs $\mathcal{O}(\log^5 |V|)$ functional operations are sufficient.*

For this result we make use of the algorithm presented in [11] but for the implicit setting also new ideas are necessary. Note, that our aim is not to achieve new algorithmic techniques for explicit graph representations but to demonstrate the similarity of paradigms in the design of parallel and implicit algorithms that can also be used as building blocks for the solution of other combinatorial problems on one hand and on the other hand to develop efficient algorithms for large structured graphs. The similarity between implicit and parallel algorithms has also been demonstrated by the following result. A problem can be solved in the implicit setting with a polylogarithmic number of functional operations on a logarithmic number of Boolean variables (with respect to the number of vertices of the input graph) iff the problem is in NC, the complexity class that contains all problems computable in polylogarithmic time with polynomially many processors [13, 14]. Nevertheless, this structural result does not lead directly to efficient implicit algorithms.

In order to reduce the number of functional operations, iterative squaring is used in our algorithm. One may argue against the use of iterative squaring because despite the improvement in the number of functional operations intermediate results of exponential size (with respect to the input length) can be generated. Nevertheless, Sawitzki has demonstrated that iterative squaring can also be useful in applications [12]. The maximum flow problem in 0-1 networks has been one of the first classical graph problems for which an implicit OBDD-based algorithm has been presented and Hachtel and Somenzi were able to compute a maximum flow for a graph with more than 10^{27} vertices and 10^{36} edges in less than one CPU minute [9]. To improve this algorithm Sawitzki has used iterative squaring for the computation of augmenting paths by $\mathcal{O}(\log^2 |V|)$ functional operations. If the maximum flow value is constant with respect to the network size, the algorithm performs altogether a polylogarithmic number of operations. Both max flow algorithms belong to the class of so-called layered-network methods but Sawitzki's algorithm prevents breadth-first searches by using iterative squaring and as a result overcomes the dependence on the depths of the layered networks. In order to confirm the practical relevance of his algorithm he has implemented both maximum flow algorithms and has shown that his algorithm outperforms the algorithm of Hachtel and Somenzi for very structured graphs.

The rest of the paper is organized as follows. In Section 2 we define some notation and review some basics concerning OBDDs and functional operations, implicit graph algorithms and matchings. Section 3 contains the main result, an implicit algorithm for the maximal matching problem that uses only a polylogarithmic number of functional operations with respect to the number of vertices of the input graph. Finally, we finish the paper with some concluding remarks.

In order to investigate the algorithm's behavior on large and structured networks, it has been analyzed on grid graphs and it has been shown that the overall running time and the space requirement is also polylogarithmic (for these results see the full version of the paper [4]).

2 Preliminaries

We briefly recall the main notions we are dealing with in the paper.

2.1 OBDDs and Functional Operations

OBDDs are a very popular dynamic data structure in areas working with Boolean functions, like circuit verification or model checking. (For a history of results on binary decision diagrams see, e.g., [15]).

Definition 2. *Let $X_n = \{x_1, \dots, x_n\}$ be a set of Boolean variables. A variable ordering π on X_n is a permutation on $\{1, \dots, n\}$ leading to the ordered list $x_{\pi(1)}, \dots, x_{\pi(n)}$ of the variables. A π-*OBDD *on X_n is a directed acyclic graph $G = (V, E)$ whose sinks are labeled by the Boolean constants* 0 *and* 1 *and whose non-sink (or decision) nodes are labeled by Boolean variables from X_n. Each decision node has two outgoing edges, one labeled by* 0 *and the other by* 1*. The edges between decision nodes have to respect the variable ordering π, i.e., if an edge leads from an x_i-node to an x_j-node, then $\pi^{-1}(i) < \pi^{-1}(j)$ (x_i precedes x_j in $x_{\pi(1)}, \dots, x_{\pi(n)}$). Each node v represents a Boolean function $f_v \in B_n$, i.e., $f_v : \{0,1\}^n \to \{0,1\}$, defined in the following way. In order to evaluate $f_v(b)$, $b \in \{0,1\}^n$, start at v. After reaching an x_i-node choose the outgoing edge with label b_i until a sink is reached. The label of this sink defines $f_v(b)$. The size of a π-*OBDD *G, denoted by $|G|$, is equal to the number of its nodes. A π-*OBDD *of minimal size for a given function f and a fixed variable ordering π is unique up to isomorphism. The π-*OBDD *size of a function f, denoted by π-*OBDD(f), *is the size of the minimal π-*OBDD *representing f. The* OBDD *size of f is the minimum of all π-*OBDD(f).

Sometimes it is useful to have the notion of OBDDs where there are only edges between nodes labeled by neighboring variables, i.e., if an edge leads from an x_i-node to an x_j-node, then $\pi^{-1}(i) = \pi^{-1}(j) - 1$.

Definition 3. *An* OBDD *on X_n is complete if all paths from the source to one of the sinks have length n. The width of a complete* OBDD *is the maximal number of nodes labeled by the same variable.*

A variable ordering is called a natural variable ordering if π is the identity $1, 2, \ldots, n$. Complete OBDDs with respect to natural variable orderings differ from deterministic finite automata only in the minor aspect that tests may not be omitted even if the corresponding subfunction is the constant 0.

Now, we briefly describe a list of important operations on OBDDs (for a detailed discussion and the corresponding time and space requirements see, e.g., Section 3.3 in [15] and the full version of the paper [4]). Let f and g be Boolean functions in B_n on the variable set $X_n = \{x_1, \ldots, x_n\}$ and G_f and G_g be π-OBDDs for the representations of f and g, respectively.

- Negation: Given G_f, compute a π-OBDD for the function $\overline{f} \in B_n$.
- Replacement by constant: Given G_f, an index $i \in \{1, \ldots, n\}$, and a Boolean constant $c_i \in \{0, 1\}$, compute a π-OBDD for the subfunction $f_{|x_i=c_i}$.
- Equality test: Given G_f and G_g, decide, whether f and g are equal.
- Satisfiability: Given G_f, decide, whether f is not the constant function 0.
- Synthesis: Given G_f and G_g and a binary Boolean operation $\otimes \in B_2$, compute a π-OBDD G_h for the function $h \in B_n$ defined as $h := f \otimes g$.
- Quantification: Given G_f, an index $i \in \{1, \ldots, n\}$, and a quantifier $Q \in \{\exists, \forall\}$, compute a π-OBDD G_h for the function $h \in B_n$ defined as $h := (Qx_i)f$, where $(\exists x_i)f := f_{|x_i=0} \vee f_{|x_i=1}$ and $(\forall x_i)f := f_{|x_i=0} \wedge f_{|x_i=1}$. In the rest of the paper quantifications over k Boolean variables $(Qx_1, \ldots, x_k)f$ are denoted by $(Qx)f$, where $x = (x_1, \ldots, x_k)$.

Sometimes it is useful to reverse the edges of a given graph. Therefore, we define the following operation (see, e.g., [13]).

Definition 4. *Let ρ be a permutation on $\{1, \ldots, k\}$ and $f \in B_{kn}$ be defined on Boolean variable vectors $x^{(1)}, \ldots, x^{(k)}$ of length n. The argument reordering $\mathcal{R}_\rho(f) \in B_{kn}$ with respect to ρ is $\mathcal{R}_\rho(f)(x^{(1)}, \ldots, x^{(k)}) = f(x^{(\rho(1))}, \ldots, x^{(\rho(k))})$.*

2.2 OBDD-Based Graph Algorithms and Matching Problems

Let $G = (V, E)$ be a graph with N vertices $v_0, \ldots v_{N-1}$ and $|z|_2 := \sum_{i=0}^{n-1} z_i 2^i$, where $z = (z_{n-1}, \ldots, z_0) \in \{0, 1\}^n$ and $n = \lceil \log N \rceil$. Now, E can be represented by an OBDD for its characteristic function, where $x, y \in \{0, 1\}^n$ and

$$\chi_E(x, y) = 1 \Leftrightarrow (|x|_2, |y|_2 < N) \wedge (v_{|x|_2}, v_{|y|_2}) \in E.$$

(For the ease of notation we omit the index 2 in the rest of the paper and we assume that N is a power of 2 since it has no bearing on the essence of our results.) Undirected edges are represented by symmetric directed ones. Furthermore, we do not distinguish between vertices of the input graph and their Boolean encoding since the meaning is clear from the context.

For implicit computations some Boolean functions are helpful. The equality function EQ computes 1 for two inputs x and y iff $|x| = |y|$. NEQ is the negated equality function. Since sometimes a vertex (or an edge) has to be chosen out of a given set of vertices (or edges), several priority functions $\Pi_{\prec}$ have been defined in the implicit setting (see, e.g., [9, 12]). We define $\Pi_{\prec}(x, y, z) = 1$ iff $y \prec_x z$, where $\prec_x$ is a total order on the vertex set V and x, y, z are vertices in V. In the following we only use a very simple priority function independent of the choice of x, where $\Pi_{\prec}(x, y, z) = 1$ iff $|y| < |z|$. It is easy to see that EQ, NEQ, and $\Pi_{\prec}$ can be represented by OBDDs of linear size with respect to variable orderings, where the variables with the same significance are tested one after another.

A graph $G = (V, E)$ is bipartite, if V can be partitioned into two disjoint nonempty sets U and W, such that for all edges $(u, w) \in E$ it holds $u \in U$ and $w \in W$ or vice versa. The distance between two edges on a (directed) path is the number of edges between them. The distance between two vertices on a (directed) path is the number of vertices between them plus 1. The degree of a vertex v in G is the number of edges in E incident to v. A matching in an undirected graph $G = (V, E)$ is a subset $M \subseteq E$ such that no two edges of M are adjacent. M is a maximum matching if there exists no matching $M' \subseteq E$ such that $|M'| > |M|$, where $|S|$ denotes the cardinality of a set S. A matching M is maximal if M is not a proper subset of another matching. Given a matching M a vertex v is matched if $(v, w) \in M$ for some $w \in V$ and free otherwise.

In the implicit setting the maximum (maximal) matching problem is the following one. Given an OBDD for the characteristic function of the edge set of an undirected input graph G, the output is an OBDD that represents the characteristic function of a maximum (maximal) matching in G.

3 The Maximal Matching Algorithm

In this section we prove Theorem 1 and present an implicit algorithm for the maximal bipartite matching problem. The algorithm can easily be extended for general graphs. The idea is to start with Kelsen's parallel algorithm for the computation of maximal matchings on explicitly defined graphs [11]. In the parallel setting more or less only a high-level description of the algorithm is given. Moreover, we have to add more ideas because we cannot access efficiently single vertices or edges in the implicit setting.

The algorithm `findMaximalBipartiteMatching` is simple. Step-by-step a current matching is enlarged by computing a matching in the subgraph of $G = (V, E)$ that consists only of the edges that are not incident to the current matching. The key idea is an algorithm `match` that computes a matching M', $M' \subseteq E$, adjacent to at least a fraction of 1/6 of the edges in the input graph for `match`. After removing these edges from the input graph the procedure is repeated. Therefore, after $\mathcal{O}(\log |V|)$ iterations the remaining subgraph is empty and the current matching is obviously a maximal matching in G.

The algorithm `match` makes use of another algorithm `halve` that halves approximately the degree of each vertex in a bipartite graph. The idea is to compute

Algorithm 1. `findMaximalBipartiteMatching`

Input: $\chi_E(x, y)$

```
(1) ▷ Initialize. Start with the empty matching.
    $M(x, y) \leftarrow 0$
(2) while $\chi_E(x, y) \neq 0$ do
(3)     ▷ Compute a matching $M'$.
        $M'(x, y) \leftarrow$ match($\chi_E(x, y)$)
(4)     ▷ Delete the edges incident to a matched vertex in $M'$.
        $\mathtt{INCNODE}(x) \leftarrow (\exists y)(M'(x, y))$
        $\chi_E(x, y) \leftarrow \chi_E(x, y) \wedge \overline{\mathtt{INCNODE}(x)} \wedge \overline{\mathtt{INCNODE}(y)}$
(5)     ▷ Add the edges from $M'$ to $M$.
        $M(x, y) \leftarrow M(x, y) \vee M'(x, y)$
(6) return $M(x, y)$
```

an Euler partition of the input graph such that the graph is decomposed into edge-disjoint paths. Each vertex of odd (even) degree is the endpoint of exactly 1 (0) open path. By two-coloring the edges on each path in the Euler partition and deleting all edges of one color, the degree of each vertex in the input graph is approximately halved. Here, we use the fact that bipartite graphs have no cycles of odd length. Therefore, for each path, where a vertex v is not an endpoint, and for each cycle, the number of edges incident to v colored by one of the two colors is equal to the number of edges colored by the other one. In fact in the algorithm `halve` we only use the color red and delete all red edges after the coloring. A precondition of the parallel algorithm `halve` is that for each vertex its incident edges have been paired [11]. Here, we present a new algorithm called `calculatePairing` which computes implicitly a pairing of the edges with $\mathcal{O}(\log^2 |V|)$ functional operations. This algorithm together with the degree reduction procedure in `halve` can possibly be used as building blocks for the solution of other combinatorial problems in the implicit setting. We assume that there is for each vertex an ordering on its incident edges given by a priority function (see Section 2). In the first step of the algorithm `calculatePairing` for each vertex the neighborhood of its incident edges is determined. Almost all edges have two neighbors with respect to one of its endpoints (all but the first and the last one). In order to compute a symmetric pairing every other edge incident to the same vertex is colored red. This is realized by an indicator function called RED. Afterwards, two neighboring edges $(x, y), (x, z)$ are paired iff (x, y) is red, i.e., $\mathtt{RED}(x, y) = 1$, and (x, y) has a higher priority than (x, z), i.e., $\Pi_{\prec}(x, y, z) = 1$. Therefore, each edge has at most one chosen neighbor with respect to one of its endpoints and at most one of the edges incident to the same vertex is not paired. The output of `calculatePairing` is a function which depends on three vertex arguments x, y, and z and whose function value is 1 iff y and z are two vertices adjacent to x which have been paired. Therefore, the function is symmetric in the second and third argument. For the computation of the pairing we determine for each edge its position with respect to all edges incident to the same vertex according to a priority function. This procedure is similar to the well-known list ranking algorithm: given a linked list for each member of the list the number of its position in the list has to be calculated (for a nice introduction into design and analysis of parallel algorithms see, e.g., [10]). In our case we are only interested in whether the number of the position of an edge according to a priority function is odd or even.

Algorithm 2. `calculatePairing`

```
Input: $\chi_E(x, y)$

(1) ▷ Determine the neighborhood of the edges.
    $\mathtt{ORDER}(x,y,z) \leftarrow \chi_E(x,y) \wedge \chi_E(x,z) \wedge \Pi_\prec(x,y,z) \wedge \overline{(\exists\xi)(\chi_E(x,\xi) \wedge \Pi_\prec(x,y,\xi) \wedge \Pi_\prec(x,\xi,z))}$
(2) ▷ Compute the distance between edges incident to the same vertex
    using iterative squaring.
    $\mathtt{DIST}_0(x,y,z) \leftarrow \mathtt{ORDER}(x,y,z)$
    for $i = 1, 2, ..., \log|V|$ do
        $\mathtt{DIST}_i(x,y,z) \leftarrow (\exists\xi)(\mathtt{DIST}_{i-1}(x,y,\xi) \wedge \mathtt{DIST}_{i-1}(x,\xi,z))$
(3) ▷ Color for each vertex its incident edges alternately.
    $\mathtt{RED}(x,y) \leftarrow \chi_E(x,y) \wedge \overline{(\exists\xi)(\chi_E(x,\xi) \wedge \Pi_\prec(x,\xi,y))}$
    for $i = 1, 2, ..., \log|V|$ do
        $\mathtt{RED}(x,y) \leftarrow \mathtt{RED}(x,y) \vee (\exists\xi)(\mathtt{RED}(x,\xi) \wedge \mathtt{DIST}_i(x,\xi,y))$
(4) ▷ Select only edge pairs $((x,y),(x,z))$, where the first edge is red.
    return $(\mathtt{ORDER}(x,y,z) \wedge \mathtt{RED}(x,y)) \vee (\mathtt{ORDER}(x,z,y) \wedge \mathtt{RED}(x,z))$
```

Lemma 5. *The algorithm* `calculatePairing` *computes for all vertices a pairing of its incident edges respectively with* $\mathcal{O}(\log^2|V|)$ *functional operations.*

Proof. The correctness of `calculatePairing` follows from the following observations: the function $\mathtt{ORDER}(x,y,z)$ computes the output 1 iff y and z are adjacent to the vertex x, the edge (x,y) is smaller than the edge (x,z) according to the chosen priority function, and there is no edge between (x,y) and (x,z) (with respect to the priority function). In step 2 the function $\mathtt{DIST}_i(x,y,z)$ computes 1 iff the distance, i.e., the number of edges between the edge (x,y) and (x,z) with respect to the priority function, is $2^i - 1$. Afterwards for each vertex the first of its incident edges according to the priority function is colored red and then all edges which have an odd distance to the first one are also colored red. Now, the output of `calculatePairing` is a function on three vertex arguments x, y and z, where the value is 1 iff the edges (x,y) and (x,z) are neighbored and the first one with respect to the priority function is red.

The most time consuming steps during `calculatePairing` are (2) and (3), where the position of the edges and the coloring of the incident edges are calculated. Traversing the incident edges of a vertex needs $\mathcal{O}(\log|V|)$ iterations each using $\mathcal{O}(\log|V|)$ operations for the quantification over $\mathcal{O}(\log|V|)$ variables. □

Algorithm 3. `halve`

```
Input: $\chi_E(x,y)$

(1) ▷ Compute the successor relation.
    $\mathtt{PAIRING}(x,y,z) \leftarrow \mathtt{calculatePairing}(\chi_E(x,y))$
    $\mathtt{SUCC}(x,y,z) \leftarrow \mathtt{PAIRING}(y,x,z)$
(2) ▷ Compute distance and reachability relations on the directed edges
    defined by the successor relation.
    $i \leftarrow 0$
    $\mathtt{DIST}_0(v,w,x,y) \leftarrow EQ(w,x) \wedge \mathtt{SUCC}(v,w,y)$
    $\mathtt{REACHABLE}(v,w,x,y) \leftarrow (EQ(v,x) \wedge EQ(w,y)) \vee (EQ(w,x) \wedge \mathtt{SUCC}(v,w,y))$
    repeat
       $i \leftarrow i+1$
       $\mathtt{REACHABLE}'(v,w,x,y) \leftarrow \mathtt{REACHABLE}(v,w,x,y)$
       $\mathtt{REACHABLE}(v,w,x,y) \leftarrow \mathtt{REACHABLE}(v,w,x,y) \vee$
            $(\exists\xi,\theta)(\mathtt{REACHABLE}(v,w,\xi,\theta) \wedge \mathtt{REACHABLE}(\xi,\theta,x,y))$
       $\mathtt{DIST}_i(v,w,x,y) \leftarrow (\exists\xi,\theta)(\mathtt{DIST}_{i-1}(v,w,\xi,\theta) \wedge \mathtt{DIST}_{i-1}(\xi,\theta,x,y))$
    until $\mathtt{REACHABLE}'(v,w,x,y) = \mathtt{REACHABLE}(v,w,x,y)$
(3) ▷ On each path, color an appropriate edge red.
    $\mathtt{RED}(x,y) \leftarrow \chi_E(x,y) \wedge (\forall\xi,\theta)(\overline{\mathtt{REACHABLE}(\xi,\theta,x,y)} \vee EQ(\xi,x) \vee \Pi_{\prec}(x,x,\xi)) \wedge$
          $(\forall\theta,\xi)(\overline{\mathtt{REACHABLE}(\theta,\xi,y,x)} \vee EQ(\theta,x) \vee \Pi_{\prec}(x,x,\theta)) \wedge$
          $(\forall\xi)((\overline{\mathtt{REACHABLE}(x,y,\xi,x)} \wedge \overline{\mathtt{REACHABLE}(\xi,x,x,y)}) \vee \Pi_{\prec}(x,y,\xi))$
    $\mathtt{RED}(x,y) \leftarrow \mathtt{RED}(x,y) \vee \mathtt{RED}(y,x)$
(4) ▷ Color the edges alternately.
    for $j = 1,2,...,i$ do
       $\mathtt{RED}(x,y) \leftarrow \mathtt{RED}(x,y) \vee (\exists\xi,\theta)(\mathtt{RED}(\xi,\theta) \wedge \mathtt{DIST}_j(\xi,\theta,x,y))$
(5) ▷ Delete the red edges.
    return $\chi_E(x,y) \wedge \overline{\mathtt{RED}(x,y)} \wedge \overline{\mathtt{RED}(y,x)}$
```

The pairing computed by `calculatePairing` is symmetric and it is used by the algorithm `halve` to define (directed) paths in the (undirected) input graph. An edge (y,z) is a successor edge of an edge (x,y) and $\mathtt{SUCC}(x,y,z)=1$ iff the edges (y,z) and (y,x) are paired according to `calculatePairing`. Using this successor relation `SUCC` the undirected input graph is decomposed into directed edge-disjoint paths. Since the pairing is symmetric, (y,x) is also a successor of (z,y). Therefore, for each directed path from a vertex u' to a vertex u'' defined by the successor relation `SUCC`, there exists also a directed path from u'' to u'. This property is important in order to guarantee that a coloring of the directed edges can be used for an appropriate coloring of the undirected edges in the input graph. For each directed path in the decomposition every other edge is colored red and a directed edge (u,v) is red iff the directed edge (v,u) is red. Therefore, for each pair of (undirected) edges computed by `calculatePairing` exactly one edge is red and by deleting the red edges the degree of each vertex is approximately halved. A crucial step, which is new in the implicit setting, is the choice of the first edges that are colored red on a directed path, because all directed paths are investigated simultaneously, i.e., for each directed path from a vertex s to a vertex t the directed path from t to s is considered at the same time. We have to avoid the situation that two edges (x, y) and (x, z), where (x, y) is a directed edge on a directed path from a vertex s to a vertex t and (x, z) a directed edge on the reversed path from t to s, are colored red at the same time, because otherwise all edges from s to t and from t to s would be red after the coloring procedure. Therefore, we ensure that in the beginning either directed edges on the path from s to t or on the corresponding path from t to s are colored. For this reason the edge relation `Reachable` is used, where $\mathtt{REACHABLE}(v, w, x, y)$ is 1 iff there exists a directed path from the edge (v, w) to the edge (x, y) defined by the successor relation `SUCC`. Due to the symmetry of `SUCC` the relation `REACHABLE` is also symmetric in the following way: iff $\mathtt{REACHABLE}(v, w, x, y) = 1$ then $\mathtt{REACHABLE}(y, x, w, v) = 1$. Therefore, using `REACHABLE` it is also possible to determine the predecessors of a directed edge. Now, the first red edges on a directed path are the edges with the highest priority: for each directed path the smallest vertex v on the path according to a priority function together with a successor u is chosen if there exists no predecessor of v which has a higher priority than u according to the chosen priority function. This procedure ensures that either for a directed path starting from a vertex s and ending in a vertex t an edge (v, u) is chosen or for the reversed directed path from t to s. Afterwards an edge (u, v) is colored red iff the edge (v, u) is red. Next, each edge on a directed path for which the distance to one of the first red edges on the path is odd is also colored red and all red edges are deleted from the input graph. Note, that it is possible for a directed path that more than one edge is chosen in the beginning but these edges have the same starting point, therefore the distance between these edges is even because the input graph is bipartite such that no problem occurs.

Lemma 6. *Let $d(v)$ and $d'(v)$ denote the degree of a vertex v in the graph given by $\chi_E(x, y)$ before and after running procedure* `halve` *on $\chi_E(x, y)$. Then $d'(v) \in \{\lfloor d(v)/2 \rfloor, \lceil d(v)/2 \rceil\}$. The algorithm* `halve` *uses $\mathcal{O}(\log^2 |V|)$ functional operations.*

Proof. For the number of functional operations step (2) and step (4) are the most expensive ones. The graph is traversed via iterative squaring in the second step. Since the length of a path is $\mathcal{O}(|E|)$, the number of iterations is $\mathcal{O}(\log |E|) = \mathcal{O}(\log |V|)$, and the quantification over the Boolean variables that encode an edge can also be done using $\mathcal{O}(\log |V|)$ operations ($\mathcal{O}(1)$ functional operations for the quantification of each variable). The number of functional operations in step (4) can be calculated in a similar way.

For the correctness of the algorithm `halve` step (3) is the most interesting one. The directed paths according to the successor relation `SUCC` are edge-disjoint but not vertex disjoint. In step (3) for each directed path according to the successor relation at least one edge is carefully chosen and colored red. The first condition ensures that only edges that belong to the input graph can be chosen. The second and third requirements guarantee that a first red edge is incident to the vertex x with the highest priority on the path. The fourth condition ensures that two arbitrary edges incident to x that are on the same directed path and have an even distance are not both colored red. This condition is sufficient, we do not have to choose only the neighbor of x which has the highest priority because we color the edges afterwards alternately and edges whose distance is odd get the same color anyway. Together with our considerations above we are done. □

The idea for the correctness of `match` is that a (directed) subgraph $P(x, y)$ of the input $\chi_E(x, y)$ is computed for which each vertex has indegree and outdegree at most 1. It consists only of vertex-disjoint open simple paths and trivial cycles. Therefore, the subgraph can be seen as the union of two matchings. By choosing the matching which is the larger one we are done. For this reason we color each path in $P(x, y)$ alternately and we remove the edges that are not red. Since the paths are vertex-disjoint the coloring is easier than the coloring of the edges in the algorithm `halve`. In fact we color the vertices and not the edges. We choose for each directed simple path the first vertex and color afterwards all vertices for which the distance to the first one is even. Then we choose the (directed) edges (x, y) iff the vertex x is red. Finally, we traverse the computed directed subgraph into the corresponding undirected one.

The directed subgraph P is computed in the following way. In each iteration the vertices with degree 1 are determined. For each vertex x adjacent to vertices with degree 1 in the remaining graph one of these vertices y is chosen according to a priority function and (x, y) is added to P. Afterwards all edges incident to vertices with degree 1 are deleted and the degree of all vertices is (approximately) halved. Note, that during the computation an edge (x, y) in P can be eliminated later on if x gets another partner that has a higher priority than y. At any time each vertex x has at most one partner.

It can be shown that at the beginning of step (8) in `match` at least 1/3 of the input edges are incident to edges defined via $P(x, y)$. The intuition is the following one. An edge (u, v) is not incident to the computed matching iff the edge (u, v) is deleted during the algorithm `halve` and the last edges (u, u') and (v, v') incident to u and v during the **while** loop are eliminated in step (5) of `match` because u' and v' are at the same time adjacent to vertices x and y which have degree 1 and are chosen as partners in the respective iteration of the **while** loop. As a consequence we can conclude that the degree of u' and v' is (approximately) at least twice the degree of u and v in the input graph because Lemma 6 ensures that the degree of each node is (almost) regularly halved in each iteration. Therefore the output of the algorithm `match` is a matching incident to at least 1/6 of the input edges.

Lemma 7. *The algorithm* `match` *implicitly computes a matching in an implicitly defined input graph* $G = (V, E)$ *incident to at least* $1/6$ *of the edges in* E. *It needs* $\mathcal{O}(\log^3 |V|)$ *functional operations.*

Proof. There are $\mathcal{O}(\log |V|)$ iterations of the **while** loop, each of them costs $\mathcal{O}(\log^2 |V|)$ functional operations. The algorithm `halve` is the dominating step during the **while** loop of `match`. Therefore, $\mathcal{O}(\log^3 |V|)$ functional operations are sufficient. The correctness follows from our considerations above. (See also [11].) □

Algorithm 4. `match`

Input: $\chi_E(x,y)$

```
(1) ▷ Initialize.
    $\chi_{E'}(x,y) \leftarrow \chi_E(x,y)$; $P(x,y) \leftarrow 0$
(2) while $\chi_{E'}(x,y) \neq 0$ do
(3)     ▷ Determine the vertices of degree at least 2.
        $\mathrm{TwoOrMoreNeighbors}(x) \leftarrow (\exists y,z)(NEQ(y,z) \wedge \chi_{E'}(x,y) \wedge \chi_{E'}(x,z))$
(4)     ▷ Set $P(x,y) = 1$ iff $y$ has degree 1 and is the partner of $x$.
        $Q(x,y) \leftarrow \chi_{E'}(x,y) \wedge \overline{\mathrm{TwoOrMoreNeighbors}(y)}$
        $Q'(x,y) \leftarrow Q(x,y) \wedge \overline{(\exists z)(Q(x,z) \wedge \Pi_{\prec}(x,z,y))}$
        $P(x,y) \leftarrow (P(x,y) \wedge \overline{(\exists z)(Q'(x,z))}) \vee Q'(x,y)$
(5)     ▷ Delete edges incident to vertices of degree 1.
        $\chi_{E'}(x,y) \leftarrow \chi_{E'}(x,y) \wedge \mathrm{TwoOrMoreNeighbors}(x) \wedge \mathrm{TwoOrMoreNeighbors}(y)$
(6)     ▷ Halve (approximately) the degree of each vertex.
        $\chi_{E'}(x,y) \leftarrow$ halve($\chi_{E'}(x,y)$)
(7) ▷ Add trivial cycles to the computed matching.
    $M_1(x,y) \leftarrow P(x,y) \wedge P(y,x)$
(8) ▷ Color the vertices in the graph given by $P(x,y)$ alternately and
    choose an edge $(x,y)$ iff $x$ is red.
    $\mathrm{RED}(x) \leftarrow (\forall \xi)\overline{(P(\xi,x))}$; $\mathrm{DIST}_0(x,y) \leftarrow P(x,y)$
    for $i = 1, 2, ..., \log|V|$ do
        $\mathrm{DIST}_i(x,y) \leftarrow (\exists \xi)(\mathrm{DIST}_{i-1}(x,\xi) \wedge \mathrm{DIST}_{i-1}(\xi,y))$
        $\mathrm{RED}(x) \leftarrow \mathrm{RED}(x) \vee (\exists \xi)(\mathrm{RED}(\xi) \wedge \mathrm{DIST}_i(\xi,x))$
    $M_2(x,y) \leftarrow P(x,y) \wedge \mathrm{RED}(x)$
(9) return $M_1(x,y) \vee M_2(x,y) \vee M_2(y,x)$
```

Summarizing, we have shown that `findMaximalBipartiteMatching` uses $\mathcal{O}(\log^4|V|)$ functional operations for the computation of a maximal matching in an implicitly defined input graph $G = (V, E)$. Adapting the ideas for the decomposition of general graphs into a logarithmic number of bipartite subgraphs [11], our algorithm can be similarly generalized with an additional factor of a logarithmic number of functional operations.

4 Concluding Remarks

Our maximal matching algorithm seems to be simple enough to be useful in practical applications. We have shown that maximal matchings can be computed with a polylogarithmic number of functional operations in the implicit setting. Moreover, in [4] we have proved that there exists a graph class for which even the overall running time of our maximal matching algorithm is $\mathcal{O}(\log^3|V|\log\log|V|)$ and the space usage is $\mathcal{O}(\log^2|V|)$, where V is the set of vertices of the input graph. One direction for future work is to implement the algorithm and to perform empirical experiments to determine its practical value. It would be interesting to investigate how the performance of the maximal matching algorithm depends on the chosen priority function. Here, we have used a very simple one. The maximal number of Boolean variables on which a function in the maximal matching algorithm depends dominates the overall worst-case bounds for the running time and the space usage. Therefore, another open question is whether we can reduce this number without increasing significantly the number of functional operations. Experimental evaluation of different maximal matching algorithms might be revealing.

The authors would like to thank the anonymous referees for carefully reading the manuscript and making helpful suggestions.

References

1. Bollig, B.: Exponential space complexity for OBDD-based reachability analysis. Information Processing Letters 110, 924–927 (2010)
2. Bollig, B.: Exponential Space Complexity for Symbolic Maximum Flow Algorithms in 0-1 Networks. In: Hliněný, P., Kučera, A. (eds.) MFCS 2010. LNCS, vol. 6281, pp. 186–197. Springer, Heidelberg (2010)
3. Bollig, B.: On Symbolic OBDD-Based Algorithms for the Minimum Spanning Tree Problem. In: Wu, W., Daescu, O. (eds.) COCOA 2010, Part II. LNCS, vol. 6509, pp. 16–30. Springer, Heidelberg (2010)
4. Bollig, B., Pröger, T.: An efficient implicit OBDD-based algorithm for maximal matchings (2011), `http://ls2-www.cs.tu-dortmund.de/~bollig/maxMatching.pdf`
5. Bryant, R.E.: Graph-based algorithms for boolean function manipulation. IEEE Trans. Computers 35(8), 677–691 (1986)
6. Gentilini, R., Piazza, C., Policriti, A.: Computing strongly connected components in a linear number of symbolic steps. In: Proc. of SODA, pp. 573–582 (2003)
7. Gentilini, R., Piazza, C., Policriti, A.: Symbolic graphs: linear solutions to connectivity related problems. Algorithmica 50(1), 120–158 (2008)
8. Goldberg, A.V., Plotkin, S.K., Vaidya, P.M.: Sublinear time parallel algorithms for matching and related problems. Journal of Algorithms 14(2), 180–213 (1993)
9. Hachtel, G.D., Somenzi, F.: A symbolic algorithm for maximum flow in $0-1$ networks. Formal Methods in System Design 10, 207–219 (1997)
10. Jájá, J.: An introduction to parallel algorithms. Addison-Wesley Publishing Company (1992)
11. Kelsen, P.: An optimal parallel algorithm for maximal matching. Information Processing Letters 52(4), 223–228 (1994)
12. Sawitzki, D.: Implicit Flow Maximization by Iterative Squaring. In: Van Emde Boas, P., Pokorný, J., Bieliková, M., Štuller, J. (eds.) SOFSEM 2004. LNCS, vol. 2932, pp. 301–313. Springer, Heidelberg (2004)
13. Sawitzki, D.: The Complexity of Problems on Implicitly Represented Inputs. In: Wiedermann, J., Tel, G., Pokorný, J., Bieliková, M., Štuller, J. (eds.) SOFSEM 2006. LNCS, vol. 3831, pp. 471–482. Springer, Heidelberg (2006)
14. Sawitzki, D.: Implicit simulation of FNC algorithms. Tech. Rep. TR07-028, ECCC Report (2007)
15. Wegener, I.: Branching programs and binary decision diagrams: theory and applications. SIAM (2000)
16. Woelfel, P.: Symbolic topological sorting with OBDDs. J. Discrete Algorithms 4(1), 51–71 (2006)