Algorithms **2014**, *7*, 243-252; doi:10.3390/a7020243

OPEN ACCESS

algorithms

ISSN 1999-4893

www.mdpi.com/journal/algorithms

Article

Efficient Algorithms for Subgraph Listing

Niklas Zechner [1,*] and Andrzej Lingas [2]

[1] Department of Computing Science, Umea University, 901 87 Umea, Sweden

[2] Department of Computer Science, Lund University, 221 00 Lund, Sweden;
E-Mail: Andrzej.Lingas@cs.lth.se

* Author to whom correspondence should be addressed; E-Mail: zechner@cs.umu.se;
Tel.: +46-90-786-9672; Fax: +46-90-58014.

Received: 6 March 2014; in revised form: 4 May 2014 / Accepted: 7 May 2014 / Published: 15 May 2014

Abstract: Subgraph isomorphism is a fundamental problem in graph theory. In this paper we focus on listing subgraphs isomorphic to a given pattern graph. First, we look at the algorithm due to Chiba and Nishizeki for listing complete subgraphs of fixed size, and show that it cannot be extended to general subgraphs of fixed size. Then, we consider the algorithm due to Gąsieniec *et al.* for finding multiple witnesses of a Boolean matrix product, and use it to design a new output-sensitive algorithm for listing all triangles in a graph. As a corollary, we obtain an output-sensitive algorithm for listing subgraphs and induced subgraphs isomorphic to an arbitrary fixed pattern graph.

Keywords: subgraph; subgraph isomorphism; subgraph listing; clique; triangle; output-sensitive algorithm; time complexity

1. Introduction

We shall consider undirected graphs. The decision version of the subgraph isomorphism problem is to decide if a host graph has a subgraph isomorphic to a pattern graph. The search version asks for finding a subgraph of the host graph isomorphic to the pattern graph. In the counting version, the number of subgraphs of the host graph isomorphic to the pattern graph should be reported. Finally, the listing version asks for a list of all subgraphs of the host graph isomorphic to the pattern graph.

A subgraph of the host graph is induced if it retains all edges of the host graph between its nodes. By replacing "subgraph" with "induced subgraph", we obtain the corresponding versions of induced subgraph isomorphism.

The decision versions of subgraph isomorphism and induced subgraph isomorphism are NP-complete in general. Special cases of these problems include the independent set, clique, Hamiltonian cycle, and Hamiltonian path, all of which are known to be NP-complete [1]. However, for pattern graphs of fixed size, all these problems admit polynomial-time solutions. Namely, for a pattern graph with $p = \mathrm{O}(1)$ nodes and a host graph with n nodes, the brute force method yields a solution in $\mathrm{O}(n^p)$ time. Since there might be up to $\Omega(n^p)$ subgraphs or induced subgraphs isomorphic to the pattern graph, the brute force method cannot be substantially asymptotically improved in case of the listing versions in general. This however does not exclude the possibility of faster listing algorithms for restricted graph classes and output-sensitive algorithms whose complexity depends on the size of the output list of isomorphic subgraphs.

In the first part of this paper, we look at an algorithm by Chiba and Nishizeki [2], which lists all occurrences of complete subgraphs of size s. It assumes that the host graph has bounded arboricity. The arboricity of a graph is the minimum number of forests (*i.e.*, sets of edge-disjoint trees) that are needed to cover the graph. The algorithm runs in $\mathrm{O}(sma^{s-2})$ time, where a is the arboricity of the host graph and m is its number of edges. We show that this upper bound on the running time cannot be extended to general pattern graphs—for any pattern graph that is connected but not complete, the time has to be at least quadratic in the number of nodes of the host graph.

In the second part, we consider the problem of listing all triangles, that is, complete subgraphs of size three (*cf.* [3]). Using the matrix product, we can find out which of the edges participate in any triangle, and how many triangles each of them belongs to [4]. Then, in order to list the triangles, we use the k-witness algorithm developed by Gąsieniec *et al.* [5]. It finds multiple witnesses of a Boolean matrix product (see Preliminaries), which can be used to list for each edge the triangles it participates in. The k-witness algorithm takes a parameter k and finds for each entry all the witnesses up to a maximum of k witnesses, so it is useful when no edge is in more than a small number of triangles. We devise a new algorithm, which uses the k-witness algorithm for those edges that are in few triangles, and otherwise a straightforward method checking all nodes incident to an endpoint of an examined edge. We show that the asymptotic time of this combined algorithm is less than that of the brute force algorithm in all cases except when the number of triangles is nearly cubic. For a graph with n nodes and l triangles, our combined algorithm lists all triangles in time $\widetilde{\mathrm{O}}(n^{2.3727})$ for $l < n^{1.3727}$, $\widetilde{\mathrm{O}}(n^{1.9368}l^{0.31754})$ for $n^{1.3727} \leqslant l < n^{2.3940}$, and $\widetilde{\mathrm{O}}(n^{1.5}l^{0.5})$ for $n^{2.3940} \leqslant l < n^3$, where $\widetilde{\mathrm{O}}(f)$ stands for $\mathrm{O}(f^{1+\mathrm{o}(1)})$.

Nešetřil and Poljak present a straightforward reduction of subgraph isomorphism or induced subgraph isomorphism to triangle detection [6]. The reduction also works for the counting and listing versions. By using it, we can also generalize our combined algorithm to include listing of subgraphs or induced subgraphs isomorphic to an arbitrary fixed pattern graph. For a graph with n nodes and l subgraphs (induced subgraphs, respectively) isomorphic to a fixed pattern graph with h nodes, our combined algorithm lists all the aforementioned subgraphs in time $\widetilde{\mathrm{O}}(n^{2.3727\lceil h/3\rceil})$ for $l < n^{1.3727\lceil h/3\rceil}$, $\widetilde{\mathrm{O}}(n^{1.9368\lceil h/3\rceil}l^{0.31754})$ for $n^{1.3727\lceil h/3\rceil} \leqslant l < n^{2.3940\lceil h/3\rceil}$, and $\widetilde{\mathrm{O}}(n^{1.5\lceil h/3\rceil}l^{0.5})$ for $n^{2.3940\lceil h/3\rceil} \leqslant l < n^{3\lceil h/3\rceil}$.

Other Related Work

Itai and Rodeh present several algorithms for triangle detection in their pioneering work [3], some of which can easily be adapted to include triangle listing (see Fact 1 in Preliminaries). A good survey over known algorithms and heuristics for triangle counting and listing can be found in [7].

2. Preliminaries

A *subgraph* of the graph $G = (V, E)$ is a graph $H = (V_H, E_H)$ such that $V_H \subseteq V$ and $E_H \subseteq E$.

An *induced subgraph* of the graph $G = (V, E)$ is a graph $H = (V_H, E_H)$ such that $V_H \subseteq V$ and $E_H = E \cap (V_H \times V_H)$.

The *arboricity* $\mathrm{a}(G)$ of a graph G is the minimum number of forests needed to cover all its edges.

The *adjacency matrix* A of a graph $G = (V, E)$ is the matrix where an entry $A[i, j]$ is 1 if $(i, j) \in E$, and otherwise 0.

A *witness* of an entry $C[i, j]$ of the Boolean matrix product C of two matrices A and B is any index k such that $A[i, k]$ and $B[k, j]$ are equal to 1 [5].

The following fact is implicit in [3]:

Fact 1. *All triangles in a graph with n nodes and m edges can be listed in $O(mn)$ time.*

Form a list of all edges of the graph. Scan the list. For each edge, count the number of triangles formed by it and a node incident to its endpoint that have not been counted for edges scanned before.

Fact 2. *The fast matrix multiplication algorithm runs in $\mathrm{O}(n^{\omega})$ time, where ω is not greater than 2.3727 [8] (cf. [9]).*

Fact 3. *The k-witness algorithm from [5] takes as input an integer k and two $n \times n$ Boolean matrices, and returns a list of all witnesses to each entry in the Boolean matrix product of those matrices, up to a maximum of k witnesses for an entry. It runs in $\widetilde{\mathrm{O}}(n^{\omega}k^{(3-\omega-\alpha)/(1-\alpha)} + n^2k)$ time, where $\alpha \approx 0.30298$ (see [13]). One can rewrite the upper time bound as $\widetilde{\mathrm{O}}(n^{\omega}k^{\mu} + n^2k)$, where $\mu \approx 0.46530$ [5].*

3. A Lower Bound on Listing

Chiba and Nishizeki [2] describe an algorithm for finding complete subgraphs of size s. If the host graph has an arboricity a, they show that their algorithm runs in $\mathrm{O}(sma^{s-2})$ time, where m is the number of edges in the host graph. Since a tree with n nodes always has $n - 1$ edges, a graph with arboricity a always has less than an edges, so for a constant a, this is essentially linear time. We show that this upper bound for the running time cannot be extended to general pattern graphs. If the pattern graph is connected but not complete, it is possible to find a sequence of host graphs containing a number of induced subgraphs isomorphic to the pattern graph that is quadratic in the number of nodes. It follows that there is no algorithm that lists all of them in less than quadratic time.

We start by proving that every graph, unless it is complete or disconnected, has three nodes connected in a V-shape—two of them connect to the third but not to each other. Knowing that, we can create multiple copies of the end nodes of the V-shape, in a way that does not increase the arboricity. That gives us a graph with many subgraphs isomorphic to the original graph. If there are n copies of each of those two nodes, we can combine them in n^2 ways.

Theorem 1. *If a pattern graph is connected and not complete, there is no algorithm that lists all occurrences of that pattern in a host graph with constant arboricity in a time that is subquadratic in the number of nodes.*

Lemma 2. *If a graph is connected but not complete, there is always a V-shaped induced subgraph; that is, there are always nodes A, B and C such that there are edges AB and BC but not AC.*

Proof. Let G be a graph with n nodes. For $n \leq 3$, the theorem holds trivially.

For $n > 3$, inductively assume that the lemma holds for $n-1$. G can be seen as the union of a graph γ with $n-1$ nodes and a single node ν. We may assume w.l.o.g. that γ is either complete or not connected.

Suppose γ is complete. If ν connects to all nodes in γ, then G is complete. If ν does not connect to any node in γ, then G is not connected. If there is a node α that ν connects to, and a node β that ν does not connect to, then $\{\nu, \alpha, \beta\}$ induces a V-shaped subgraph in G.

Suppose γ is not connected. There are two subgraphs γ_1 and γ_2 in γ with no edge between them. If ν is not connected to any of these two subgraphs, then G is not connected. Otherwise, there is a node α in γ_1 and a node β in γ_2, both connected to ν. Then $\{\nu, \alpha, \beta\}$ induces a V-shaped subgraph in G.

By induction the lemma holds for all G. □

Lemma 3. *If a graph P has a V-shaped induced subgraph, there exists a sequence of graphs* $\mathrm{G}(n)$ *with constant arboricity and* $\mathrm{O}(n)$ *nodes such that the number of induced subgraphs of* $\mathrm{G}(n)$ *isomorphic to P is* $\Omega(n^2)$.

Proof. Suppose that $\{A_1, B, C_1\}$ induce a V-shaped subgraph in P; A_1 and C_1 form edges with B but not with each other. Let D be the subgraph of P resulting from removing the nodes A_1 and C_1, and let G(1) = P. Next, let $\mathrm{G}(n+1)$ be a graph based on $\mathrm{G}(n)$, with two more nodes; one node A_{n+1} that connects to the same nodes as A_1 in $\mathrm{G}(n)$, and one node C_{n+1} that connects to the same nodes as C_1 in $\mathrm{G}(n)$. The number of nodes in $\mathrm{G}(n)$ is then $|P| + 2n - 2$, which is $\mathrm{O}(n)$. Let $\mathrm{H}(a, c)$ be the subgraph of $\mathrm{G}(n)$ induced by $\{A_a, C_c\}$ and the nodes of D. $\mathrm{H}(a, c)$ is isomorphic to P. In $\mathrm{G}(n)$, all $\mathrm{H}(a, c)$ with a and c in the interval $[1, n]$ are induced subgraphs. (a, c) can be chosen in $\Omega(n^2)$ ways. Therefore there are $\Omega(n^2)$ induced subgraphs in $\mathrm{G}(n)$ that are isomorphic to P.

Each $\mathrm{G}(n)$ can be covered by no more than $|P| - 2$ trees. For each node in D, create a tree where that node is the parent and all adjacent nodes are children. This set of trees will cover any edge with at least one endpoint in D. The nodes in $\mathrm{G}(n) \setminus D$ have no internal edges, so the trees cover all the edges of $\mathrm{G}(n)$. This shows that the arboricity of $\mathrm{G}(n)$ is bounded by a constant, which proves the lemma. □

Proof of Theorem 1. Lemma 1 shows that a non-complete connected pattern graph must have a V-shaped induced subgraph. Lemma 2 shows that for any such pattern graph, there is a sequence of host graphs in which the number of occurrences of the pattern increases quadratically with the number of nodes. Since the number of occurrences can be quadratic, it is impossible to list them in less than quadratic time. □

Algorithm 1 A combined algorithm for listing of triangles

```
Require: $A$ is the adjacency matrix for a graph with $n$ nodes
 1: $U \leftarrow$ a list of pairs composed of a pair of nodes and an integer (initially empty)
 2: $R \leftarrow$ a list of triples of nodes (initially empty)
 3: calculate $A^2$
 4: for $x = 0$ to $n - 1$ do
 5:     for $y = 0$ to $n - 1$ do
 6:         if $A(x, y) = 1$ & $A^2(x, y) > 0$ then
 7:             add $((x, y), A^2(x, y))$ to $U$
 8:         end if
 9:     end for
10: end for
11: sort $U$ in descending order of $A^2(x, y)$
12: $l \leftarrow$ the sum of all $A^2(x, y)$ in $U$
13: if $l < n^{1.376}$ then
14:     $k \leftarrow 0$
15: else if $l < n^{2.3940}$ then
16:     $k \leftarrow l^{0.68245}/n^{0.93681}$
17: else
18:     $k \leftarrow \sqrt{l/n}$
19: end if
20: for all $((x, y), A^2(x, y))$ in $U$ while $A^2(x, y) > k$ do
21:     for $j$ = 0 to $n - 1$ do
22:         if $A(x, j) = 1$ & $A(y, j) = 1$ then
23:             add $(x, y, j)$ to $R$
24:         end if
25:     end for
26: end for
27: run the $k$-witness algorithm on $(k, A, A)$
28: for all remaining elements $(x, y)$ in U do
29:     for each witness $j$ of $A^2(x, y)$ listed by the algorithm, add $(x, y, j)$ to $R$
30: end for
31: print $R$
```

4. A Combined Algorithm for Listing of Triangles

In this chapter, we present an algorithm for listing triangles that combines two different methods. Using matrix multiplication, we can find out for each edge the number of triangles it belongs to, and use one method for those edges that are in many triangles and another one for those that are in few.

For those edges (x, y) that are in no more than k triangles, we make use of the k-witness algorithm [5]. For a given integer k, it finds all witnesses to the matrix product, up to a maximum of k witnesses for each entry in the product matrix. In this case, a witness j to an entry $A^2(x, y)$ of the square product of the adjacency matrix A corresponds to the triangle (x, y, j) including the edge (x, y). For those edges that are in more than k triangles, we let the algorithm examine each node once for every such edge (Fact 1).

Theorem 4. *There is an algorithm that, for a given graph with n nodes and l triangles, lists all triangles, in time*
$\widetilde{O}(n^{2.3727})$*, if* $l < n^{1.3727}$
$\widetilde{O}(n^{1.9368} l^{0.31754})$*, if* $n^{1.3727} \leqslant l < n^{2.3940}$
$\widetilde{O}(n^{1.5} l^{0.5})$*, if* $n^{2.3940} \leqslant l < n^3$*.*

4.1. Correctness

Lemma 5. *The algorithm is correct.*

Proof. If $A(x, y) = 1$, there is an edge between x and y. If $A^2(x, y) > 0$, there is a two-edge path between x and y. If and only if both these conditions are satisfied, there is at least one triangle that includes the nodes x and y. $A^2(x, y)$ is the number of two-edge paths from x to y [4]. Therefore, if $A(x, y) = 1$, $A^2(x, y)$ is the number of (ordered) triangles (x, y, z), where z is any other node. After the loop on Line 4, U contains the list of all (ordered) pairs of nodes (x, y) that are in a triangle, and the number of triangles (x, y, z) that they are in.

Consider a triple of nodes (x, y, z) such that $A^2(x, y) > k$.
If (x, y, z) is not a triangle, then at least one of (x, y), (x, z) and (y, z) is not an edge.
If there is no edge (x, y), then (x, y) will not be in U, so the test on Line 22 will not be performed for those values and the triple will not be counted.
If there is no edge (x, z) or no edge (y, z), the test on Line 22 will not return true for those values and the triple will not be counted.
If (x, y, z) is a triangle, then (x, y) will be in U, and the test on Line 22 will return true, so the triple will be counted.
Thus, all triangles (x, y, z), where (x, y) belongs to more than k triangles, are listed by this part of the algorithm, and no non-triangles are listed.

All triangles (x, y, z) where (x, y) is in no more than k triangles are listed by the k-witness algorithm (Fact 2). The loop on Line 28 merges the two lists, so that R contains all triangles of the graph, and only triangles. □

4.2. Time Complexity

Lemma 6. *The algorithm runs in time*
$\widetilde{O}(n^{2.3727})$*, if* $l < n^{1.3727}$
$\widetilde{O}(n^{1.9368} l^{0.31754})$*, if* $n^{1.3727} \leqslant l < n^{2.3940}$
$\widetilde{O}(n^{1.5} l^{0.5})$*, if* $n^{2.3940} \leqslant l < n^3$
where n is the number of nodes, and l is the number of triangles.

Proof. The calculation on Line 3 can be implemented by using the fast matrix multiplication algorithm in $\mathrm{O}(n^{\omega})$ time (Fact 1).

The loop on Line 4 runs in $\mathrm{O}(n^2)$ time.

The sorting on Line 11 can be done in $\mathrm{O}(n \log n)$ time. We might for example use radix sort [10].

The sum on Line 12 takes O(n) time.

Recall that l is the number of triangles in the graph. Let p be the number of elements that the loop on Line 20 goes through. For each of them, there are more than k triangles. Hence, we obtain $pk \leqslant l$ and consequently $p \leqslant l/k$.

The loop on Line 21 has n iterations, and each iteration takes $\mathrm{O}(1)$ time, so the loop takes $\mathrm{O}(n)$ time. The loop on Line 20 therefore takes $\mathrm{O}(np)$ time, which is $\mathrm{O}(nl/k)$, for $k > 0$. In the special case where $k = 0$, we know that $p \in \mathrm{O}(l)$ so the loop on Line 20 takes $\mathrm{O}(nl)$ time. The k-witness algorithm runs in $\widetilde{\mathrm{O}}(n^{\omega}k^{\mu} + n^2k)$ time (for $k > 0$), where $\mu \approx 0.46530$ (Fact 2). The total time is the sum of these contributions, $\widetilde{\mathrm{O}}(nl/k + n^{\omega}k^{\mu} + n^2k + n^{\omega})$.

In the case $l < n^{1.3727}$, $k = 0$:
The first term of the sum can be replaced by nl, so we have

$$\widetilde{\mathrm{O}}(nl + n^{\omega} * k^{\mu} + n^2k + n^{\omega}) =$$

$$= \widetilde{\mathrm{O}}(n^{2.3727} + n^{2.3727}) =$$

$$= \widetilde{\mathrm{O}}(n^{2.3727})$$

In the case $n^{1.3727} \leqslant l < n^{2.3940}$, $k = l^{0.68245}/n^{0.93681}$:

$$\widetilde{\mathrm{O}}(\frac{nl}{k} + n^{\omega}k^{\mu} + n^2k + n^{\omega}) =$$

$$= \widetilde{\mathrm{O}}(n^{1.9368}l^{0.31754} + n^{1.9368}l^{0.31754} + n^{1.0632}l^{0.68245} + n^{2.3727})$$

We have $n^{1.3727} \leqslant l$; therefore $n^{2.3727} \leqslant n^{1.9368}l^{0.31754}$.
We have $l < n^{2.3940}$; therefore $n^{1.0632}l^{0.68245} < n^{1.9368}l^{0.31754}$.
This means that the first two terms dominate, so the time is $\widetilde{\mathrm{O}}(n^{1.9368}l^{0.31754})$.

In the case $l > n^{2.3940}$, $k = \sqrt{l/n}$:

$$\widetilde{\mathrm{O}}(\frac{nl}{k} + n^{\omega}k^{\mu} + n^2k + n^{\omega}) =$$

$$= \widetilde{\mathrm{O}}(n^{1.5}l^{0.5} + n^{2.1401}l^{0.23265} + n^{1.5}l^{0.5} + n^{2.3727})$$

We have $l > n^{2.3940}$; therefore $n^{1.5}l^{0.5} > n^{2.1401}l^{0.23265} > n^{2.3727}$. This means that the first and third terms dominate, so the time is $\widetilde{\mathrm{O}}(n^{1.5}l^{0.5})$. $\square$

Proof of Theorem 4. As follows from Lemma 5 and Lemma 6, the algorithm satisfies the statements of the theorem. $\square$

4.3. Listing Subgraphs Isomorphic to an Arbitrary Pattern Graph

The reduction of subgraph isomorphism or induced subgraph isomorphism to triangle detection from [6] also works for the counting and listing versions. By using the aforementioned reduction, we can also generalize our combined algorithm to include listing of subgraphs or induced subgraphs isomorphic to an arbitrary fixed pattern graph. For the sake of completeness, we shall outline the reduction for listing the subgraphs isomorphic to the pattern graph.

Let h be the number of nodes in the fixed pattern graph H, and let $f = \lceil h/3 \rceil$. Since F is fixed, we have $h = \mathrm{O}(1)$ and $f = \mathrm{O}(1)$. Divide the pattern graph into three subgraphs H_i, $i = 1, 2, 3$, each having f or $\lfloor h/3 \rfloor$ nodes. Form a new graph G' in which each node v corresponds one to one to a pair (G_v, ϕ_v), where G_v is a subgraph of the original graph G and ϕ_v is an isomorphism between H_i for some $i \in \{1, 2, 3\}$ and G_v. Two nodes v and u in G' are connected by an edge if and only if

(1) G_v and G_u are node-disjoint and the isomorphisms $\phi_v,\ \phi_u$ are defined on two distinct parts of H, H_{i_v} and H_{i_u}, and
(2) the union of the isomorphisms $\phi_v,\ \phi_u$ yields an isomorphism between the subgraph of H induced by the nodes of H_{i_v} and H_{i_u}, and the subgraph of G consisting of G_v and G_u, and some edges of G between G_v and G_u.

Observe that G' has $\mathrm{O}(n^f f!) = \mathrm{O}(n^f)$ nodes. By $f = \mathrm{O}(1)$, they can be be listed in $\mathrm{O}(n^f)$ time. For a given pair of nodes $v,\ u$ of G', the necessary and sufficient conditions for making them adjacent can also be verified in $\mathrm{O}(1)$ time. It follows that G' can be constructed in $\mathrm{O}(n^{2f})$ time. Furthermore, triangles in G' are in one-to-one correspondence with isomorphisms between H and a subgraph of G isomorphic to H. Therefore, by using Theorem 4 to list all triangles in G', we can determine and list all subgraphs of G isomorphic to H in time resulting from the substitution of $\mathrm{O}(n^f)$ for n in the upper bounds of Theorem 4. By replacing "subgraph" with "induced subgraph", we obtain the analogous reduction for listing induced subgraphs isomorphic to H.

Theorem 7. *Let H be an arbitrary fixed pattern graph with h nodes. For a graph with n nodes and l subgraphs (induced subgraphs, respectively) isomorphic to H, all the subgraphs can be listed in time*
$\widetilde{\mathrm{O}}(n^{2.3727\lceil h/3 \rceil})$, *if* $l < n^{1.3727\lceil h/3 \rceil}$
$\widetilde{\mathrm{O}}(n^{1.9368\lceil h/3 \rceil} l^{0.31754})$, *if* $n^{1.3727\lceil h/3 \rceil} \leqslant l < n^{2.3940\lceil h/3 \rceil}$
$\widetilde{\mathrm{O}}(n^{1.5\lceil h/3 \rceil} l^{0.5})$, *if* $n^{2.3940\lceil h/3 \rceil} \leqslant l < n^{3\lceil h/3 \rceil}$.

5. Final Remarks

Our lower bound (Theorem 1) does not exclude the possibility of the existence of more efficient algorithms for the related problems of counting occurrences of a non-complete pattern or finding the most frequent (complete or non-complete) patterns in sparse graphs [11,12].

The method of Theorem 7 requires $\widetilde{\mathrm{O}}(n^{2\lceil h/3 \rceil})$ space in the worst case. If each of the subgraphs H_i ($i = 1, 2, 3$) of H occurs sparsely in G, the auxiliary graph G' and consequently the space requirements of the method are substantially smaller.

Acknowledgments

The authors are grateful to Mirosław Kowaluk for preliminary discussion on triangle listing.

Author Contributions

Andrzej Lingas as a supervisor provided the general idea for the paper, which after joint discussions has been implemented and written down by Niklas Zechner, with the exception of Section 4.3 and some introductory fragments.

Conflicts of Interest

The authors declare no conflict of interest.

References

1. Garey, M.R.; Johnson, D.S. *Computers and Intractability—A Guide to the Theory of NP-Completeness*; Bell Laboratories: Murray Hill, NJ, USA, 1979.
2. Chiba, N.; Nishizeki, T. Arboricity and subgraph listing algorithms. *SIAM J. Comput.* **1985**, *14*, 210–223.
3. Itai, A.; Rodeh, M. Finding a minimum circuit in a graph. *SIAM J. Comput.* **1978**, *7*, 413–423.
4. Gibbons, A. *Algorithmic Graph Theory*; Cambridge University Press: Cambridge, UK, 1985.
5. Gąsieniec, L.; Kowaluk, M.; Lingas, A. Faster multi-witnesses for Boolean matrix multiplication. *Inf. Process. Lett.* **2009**, *109*, 242–247.
6. Nešetřil, J.; Poljak, S. On the complexity of the subgraph problem. *Comment. Math. Univ. Carol.* **1985**, *26*, 415–419.
7. Schank, T.; Wagner, D. Finding, counting and listing all triangles in large graphs, an experimental study. In Proceedings of the 4th International Workshop, WEA 2005, Santorini Island, Greece, 10–13 May 2005; pp. 606–609.
8. Vassilevska Williams, V. Multiplying matrices faster than coppersmith-winograd. In Proceedings of the 44th Symposium on Theory of Computing Conference (STOC 2012), New York, NY, USA, 19–22 May 2012; pp. 887–898.
9. Coppersmith, D.; Winograd, S. Matrix multiplication via arithmetic progressions. *J. Symb. Comput.* **1990**, *9*, 251–280.
10. Cormen, T.; Leiserson, C.; Rivest, R. *Introduction to Algorithms*; MIT Press: Cambridge, MA, USA, 1990.
11. Dvorak, Z.; Tuma, V. A dynamic data structure for counting subgraphs in sparse graphs. In Proceedings of the 13th International Symposium, WADS 2013, London, ON, Canada, 12–14 August 2013; Lecture Notes in Computer Science; Springer: Berlin, Heidelberg, Germany, 2013; pp. 304–315.
12. Kuramochi, M.; Karypis, G. Finding frequent patterns in a large sparse graph. *Data Min. Knowl. Discov.* **2005**, *11*, 243–271.

13. Le Gall, F. Faster algorithms for rectangular matrix multiplication. In Proceedings of the 53rd Annual IEEE Symposium on Foundations of Computer Science (FOCS 2012), New Brunswick, NJ, USA, 20–23 October 2012; pp. 514–523.