Detecting short directed cycles using rectangular matrix multiplication and dynamic programming

Raphael Yuster * *Uri Zwick* †

Abstract

We present several new algorithms for detecting short fixed length cycles in digraphs. The new algorithms utilize fast rectangular matrix multiplication algorithms together with a dynamic programming approach similar to the one used in the solution of the classical chain matrix product problem. The new algorithms are instantiations of a generic algorithm that we present for finding a directed C_k, i.e., a directed cycle of length k, in a digraph, for any fixed $k \geq 3$. This algorithm partitions the prospective C_k's in the input digraph $G = (V, E)$ into $O(\log^k V)$ classes, according to the degrees of their vertices. For each cycle class we determine, in $O(E^{c_k} \log V)$ time, whether G contains a C_k from that class, where $c_k = c_k(\omega)$ is a constant that depends only on ω, the exponent of square matrix multiplication. The search for cycles from a given class is guided by the solution of a small dynamic programming problem. The total running time of the obtained deterministic algorithm is therefore $O(E^{c_k} \log^{k+1} V)$.

For C_3, we get $c_3 = 2\omega/(\omega + 1) < 1.41$ where $\omega < 2.376$ is the exponent of square matrix multiplication. This coincides with an existing algorithm of [AYZ97].

For C_4 we get $c_4 = (4\omega-1)/(2\omega+1) < 1.48$. We can dispense, in this case, of the polylogarithmic factor and get an $O(E^{(4\omega-1)/(2\omega+1)}) = o(E^{1.48})$ time algorithm. This improves upon an $O(E^{3/2})$ time algorithm of [AYZ97].

For C_5 we get $c_5 = 3\omega/(\omega + 2) < 1.63$. The obtained running time of $O(E^{3\omega/(\omega+2)} \log^6 V) = o(E^{1.63})$ improves upon an $O(E^{5/3})$ time algorithm of [AYZ97].

Determining c_k for $k \geq 6$ is a difficult task. We *conjecture* that $c_k = (k + 1)\omega/(2\omega + k - 1)$, for every *odd* k. The values of c_k for even $k \geq 6$ seem to exhibit a much more complicated dependence on ω.

*Department of Mathematics, University of Haifa at Oranim, Tivon 36006, Israel. E–mail: raphy@research.haifa.ac.il

†Department of Computer Science, Tel Aviv University, Tel Aviv 69978, Israel. E–mail: zwick@post.tau.ac.il

1 Introduction

We present several improved algorithms for detecting fixed length (directed) cycles in digraphs. The algorithms utilize fast square and rectangular matrix multiplication algorithms due to Coppersmith and Winograd [CW90], Coppersmith [Cop97] and Huang an Pan [HP98], together with a dynamic programming approach. Although the algorithms are given explicitly, their complexity analysis is quite a difficult task. Determining this complexity requires, in general, the solution of a huge number of (constant size) linear programs.

Although our algorithm use fast *rectangular* matrix multiplication algorithms, we express most of our time bounds, for simplicity reasons, in terms of ω, the exponent of fast *square* matrix multiplications. The best bound currently available on ω is $\omega < 2.376$, obtained by Coppersmith and Winograd [CW90]. This is done by reducing each rectangular matrix product into a collection of smaller square matrix products. Slightly improved bounds can be obtained by using, directly, the best available rectangular matrix multiplication algorithms of Coppersmith [Cop97] and Huang an Pan [HP98].

As explained in the abstract, the new algorithms are instantiations of a generic algorithm that we present for finding a directed C_k, i.e., a directed cycle of length k, in a digraph, for any fixed $k \geq 3$. This algorithm partitions the prospective C_k's in the input digraph $G = (V, E)$ into a $O(\log^k V)$ classes [1], according to the degrees of their vertices. For each cycle class we determine, in $O(E^{c_k} \log V)$ time, whether G contains a C_k from that class, where $c_k = c_k(\omega)$ is a constant that depends only on ω. The search for cycles from a given class is guided by the solution of a small dynamic programming problem.

For $k = 3$, we 'rediscover' an $O(E^{2\omega/(\omega+1)})$ algorithm for finding triangles obtained by Alon *et al.* [AYZ97]. For $k = 4, 5$ we obtain algorithms that improve upon the purely combinatorial algorithms for finding C_4's and C_5's presented in [AYZ97]. This was stated there as an open problem. We conjecture that the algorithms that we obtain for $k \geq 6$ also improve on the combinatorial algorithms of [AYZ97], at least when ω is close enough to 2. (Many researchers believe that $\omega = 2 + o(1)$.)

Our concrete results for $k = 4, 5$ are:

Theorem 1.1 *There exists an algorithm that given a digraph $G = (V, E)$ determines whether G has a C_4, and outputs one if it does, in $O(E^{(4\omega-1)/(2\omega+1)}) = o(E^{1.48})$ time.*

Theorem 1.2 *There exists an algorithm that given a digraph $G = (V, E)$ determines whether G has a C_5, and outputs one if it does, in $O(E^{3\omega/(\omega+2)} \log^6 V) = o(E^{1.63})$ time.*

[1] Throughout this paper we write V and E, instead of $|V|$ and $|E|$, when these terms appear inside a big-O notation.

cycle	new algorithm	existing algorithms
C_4	$O(E^{(4\omega-1)/(2\omega+1)}) = o(E^{1.48})$	$O(E^{3/2})$ [AYZ97], $O(E^{2-2/\omega}V^{1/\omega})$ [EG03], $O(V^\omega)$ [AYZ97]
C_5	$\tilde{O}(E^{3\omega/(\omega+2)}) = o(E^{1.63})$	$O(E^{5/3})$ [AYZ97], $O(V^\omega)$ [AYZ97]

Table 1: New and old algorithms for detecting C_4's and C_5's.

A summary of all available algorithms for finding C_4's and C_5's in digraphs is given in Table 1. Alon *et al.* [AYZ97] obtained two algorithms for finding C_4's. The first has a running time of $O(E^{3/2})$, and the second a running time of $O(V^\omega)$. The new $O(E^{(4\omega-1)/(2\omega+1)}) = o(E^{1.48})$ is clearly better than the old $O(E^{3/2})$ algorithm. It is better than the $O(V^\omega)$ algorithm when the graph is sufficiently sparse. A recent algorithm of Eisenbrand and Grandoni [EG03] detects C_4's in $O(E^{2-2/\omega}V^{1/\omega})$ time. This algorithm is inferior to the algorithm of Theorem 1.1 for sparse graphs and inferior to the $O(V^\omega)$ algorithm for dense graphs, but is better than the two of them in some intermediate density interval. Using directly a fast rectangular matrix multiplication algorithm, the running time of the algorithm of Theorem 1.1 can be slightly improved to $O(E^{1.474})$.

The algorithm of Theorem 1.2 is clearly faster than the $O(E^{5/3})$ algorithm Alon *et al.* [AYZ97]. It is faster than the $O(V^\omega)$ algorithm of [AYZ97] when $|E| = o(V^{(\omega+2)/3})$.

Determining the exact value of c_k for $k \geq 6$ is a difficult task. For *odd* k, it is not difficult to show that $c_k \geq (k+1)\omega/(2\omega+k-1)$. We *conjecture* that for odd k we in fact have $c_k = (k+1)\omega/(2\omega+k-1)$. The values of c_k for even $k \geq 6$ seem to exhibit a much more complicated dependence on ω. Numerical experiments suggest that $c_6 = (10-\omega)/(7-\omega) < 1.6488$, when ω is near 2.376. This, however, is *not* true for every ω.

In this extended abstract we concentrate on obtaining algorithms whose running time can be expressed solely in terms of $|E|$. Our approach can be easily extended, however, to yield possibly improved algorithms whose running times are expressed in terms of both $|V|$ and $|E|$, like the $O(V^{1/\omega}E^{2-2/\omega})$ time C_4 algorithm of Eisenbrand and Grandoni [EG03]. The details will appear in the full version of the paper.

The rest of this paper is organized as follows. In Section 2 we present, as a warm-up, the $O(E^{2\omega/(\omega+1)})$ time algorithm of [AYZ97] for detecting C_3's. In Section 3 we give a direct proof of Theorem 1.1. This section is quite technical and may be skipped in a first reading. The concrete algorithm for detecting C_4's presented in this section partitions the cycles into only a finite number of classes, thus saving a polylogarithmic factor in the running time. In Section 4 we present our general procedure for finding a C_k. In Section 5 we analyze the algorithm obtained for $k = 5$ and prove Theorem 1.2. In Section 6 we discuss our conjectures for $k \geq 6$. The final section contains some concluding remarks.

Throughout the rest of this paper we use the notation $\omega(a,b,c)$ to denote the exponent of the multiplication of an $N^a \times N^b$ matrix by an $N^b \times N^c$ matrix, where a,b,c are positive constants; namely, the number of arithmetic operations required to perform this multiplication is $O(N^{\omega(a,b,c)})$. It is not difficult to see (see, e.g., Hunag and Pan [HP98]), that $\omega(a,b,c) \leq a+b+c-(3-\omega)\min\{a,b,c\}$, where $\omega = \omega(1,1,1)$ is the exponent of square matrix multiplication.

2 A warm up: Finding triangles (C_3's)

To put the results obtained in this paper in context we use this short section to remind the reader of the very simple $O(E^{2\omega/(\omega+1)})$ algorithm of [AYZ97] for finding triangles. Implementing our general procedure in the case $k = 3$ coincides with this algorithm.

Let $G = (V,E)$ be the input graph. Set $\Delta = |E|^{(\omega-1)/(\omega+1)}$. Let $H = \{v \in V \mid \deg(v) \geq \Delta\}$ be the set of high degree vertices of the graph. (For the rest of this paper the notations $\deg(v)$ or *degree* refer to the sum of the indegree and the outdegree of a vertex.) Clearly $|H| \leq 2|E|/\Delta$. Let $L = V - H$ be the set of low degree vertices. A triangle consisting of three high degree vertices can be easily found, using matrix multiplication, in $O(H^\omega) = O((E/\Delta)^\omega) = O(E^{2\omega/(\omega+1)})$ time. A triangle containing a low degree vertex can be easily found, without matrix multiplication, in $O(E\Delta)$ time, as this is a bound on the number of paths of length two that pass through a low degree vertex. As $O(E\Delta) = O(E^{2\omega/(\omega+1)})$, and as each triangle is of one of these two types, the result follows.

3 A more complicated case: Finding quadrilaterals (C_4's)

In this section we prove Theorem 1.1. Denote $\alpha = 2(\omega-1)/(2\omega+1)$ and let $\Delta = |E|^\alpha$. We partition the vertex set V into three parts. Let H denote the vertices having degree greater than Δ in G. Let M denote the vertices having degree at most Δ and greater than $\sqrt{\Delta}$. Let $L = V \setminus (H \cup M)$. Clearly, $|H| < 2|E|/\Delta$ and $|M| < 2|E|/\sqrt{\Delta}$. We distinguish between five different types of C_4.
(i) The C_4 whose vertices are all in H.
(ii) The C_4 with two opposite vertices in $M \cup L$.
(iii) The C_4 with only one vertex in $M \cup L$.
(iv) The C_4 with only two consecutive vertices in $M \cup L$, at least one of them being from M.
(v) The C_4 with only two consecutive vertices in L.
Clearly, any C_4 in G is of one of these types. We show that the running time required to detect a C_4 of a given type is $O(E^{(4\omega-1)/(2\omega+1)})$.

We can detect a C_4 of type (i) using the $O(V^\omega)$ algorithm from [AYZ97] mentioned in the introduction in $O(H^\omega)$ time. Since $|H| < 2|E|/\Delta = 2|E|^{1-\alpha}$ we can detect a C_4 of type (i) in $O(E^{\omega(1-\alpha)}) \le O(E^{(4\omega-1)/(2\omega+1)})$ time.

Any C_4 of type (ii) is composed of two directed paths of length 2 with an intermediate vertex in $M \cup L$. As in [AYZ97] we can detect all such directed 2-paths in $O(E\Delta)$ time, and use radix sort to sort them according to their start and end vertices in linear time (namely, in $O(E\Delta)$ time). It is then straightforward to check whether there are two such directed paths of the form $u - x - v$ and $v - y - u$ where $x \neq y$ in linear time. Thus, we can detect cycles of type (ii) in $O(E\Delta) = O(E^{1+\alpha}) = O(E^{(4\omega-1)/(2\omega+1)})$ time.

In order to detect a C_4 of type (iii) we use the following idea from [EG03]. Let A denote the square matrix of order $|H|$, with $A(x, y)$ equal to the number of directed paths of length two from $x \in H$ to $y \in H$ that pass through a vertex of H. Clearly, A is the result of multiplying two square matrices of order $|H|$ and hence A can be computed in time $O(H^\omega)$. Notice that there are at most $|E|\Delta = |E|^{1+\alpha}$ directed paths of length 2 that begin and end in vertices of H and their middle vertex is from $M \cup L$. These paths can easily be traversed in $O(E^{1+\alpha})$ time by simply traversing the edges of G one by one, and noticing that for each edge (x, y), if $x \in M \cup L$ and $y \in H$ there are at most Δ edges of the form (z, x) where $z \in H$, and if $x \in H$ and $y \in M \cup L$ there are at most Δ edges of the form (y, z) where $z \in H$. For each such path, say, (x, y, z) we only need to check whether $A(z, x) > 0$. Furthermore, if $A(z, x) > 0$ it is straightforward to find a directed path of length 2 from z to x in $O(E)$ time. Hence, finding a C_4 of type (iii) requires $O(H^\omega + E^{1+\alpha}) = O(E^{(4\omega-1)/(2\omega+1)})$ time.

In order to detect a C_4 of type (iv) we proceed as follows. Let A_1 denote the rectangular matrix with $|M|$ rows and $|H|$ columns where $A_1(x, y)$ is the number of directed paths of length 2 from $x \in M$ to $y \in H$ and that pass through a vertex of H. Clearly, A_1 is the result of multiplying an $|M| \times |H|$ matrix with an $|H| \times |H|$ matrix. Hence, using $O(M/H)$ square matrix multiplications we can generate A_1 in $O(MH^{\omega-1})$ time. Similarly, we can generate the matrix A_2 with $|H|$ rows and $|M|$ columns where $A_2(y, x)$ is the number of directed paths of length 2 from $y \in H$ to $x \in M$ and that pass through a vertex of H. Note that A_2 is the result of multiplying an $|H| \times |H|$ matrix with an $|H| \times |M|$ matrix and hence can be generated in $O(MH^{\omega-1})$ time. Now, as in case (iii) there are at most $|E|^{1+\alpha}$ directed paths of the form (w, u, x) where $w \in H$, $u \in M \cup L$ and $x \in M$ and they can be generated in $O(E^{1+\alpha})$ time. For each such path we only need to check whether $A_1(x, w) > 0$ and, as in the previous case, if $A_1(x, w) > 0$ we can find a path of length 2 from x to w in $O(E)$ time. Similarly, there are at most $|E|^{1+\alpha}$ directed paths of the form (x, u, w) where $w \in H$, $u \in M \cup L$ and $x \in M$ and they can be generated in $O(E^{1+\alpha})$ time. For each such path we only need to check whether $A_2(w, x) > 0$. The overall running time needed to find a C_4 of type (iv) is therefore

$$O(E^{1+\alpha} + MH^{\omega-1}) \le O(E^{1+\alpha} + E^{1-\alpha/2}E^{(1-\alpha)(\omega-1)}) = O(E^{(4\omega-1)/(2\omega+1)}).$$

In order to detect a C_4 of type (v) we first throw away the vertices of M from the graph, as they are irrelevant. Thus, in the new graph the vertices of L have maximum indegree or outdegree at most $\Delta^{1/2} = |E|^{\alpha/2}$. Thus, there are at most $|E|^{1+\alpha}$ directed paths of length 3 that begin and end in vertices of H and their middle vertices are from L. These paths can be generated in $O(E^{1+\alpha})$ time by traversing all edges of the graph, and for each edge (u, u') where both $u, u' \in L$ we can generate all possible paths of length 3 of the form (w, u, u', w') where $w, w' \in H$. Notice that there are at most $|E|^{\alpha/2}$ choices for w and at most $|E|^{\alpha/2}$ choices for w' and hence there are at most $|E|^{\alpha}$ such paths for each (u, u'). Now, for each generated path (w, u, u', w') we check in constant time whether $(w', w) \in E$ (we can prepare the adjacency matrix of the subgraph induced by H in $O(E+H^2)$ time in the beginning of the algorithm). The overall running time required for detecting C_4 of type (v) is therefore $O(E^{1+\alpha}) = O(E^{(4\omega-1)/(2\omega+1)})$. $\square$

Notice that in one of its bottleneck phases, our algorithm computes the product of an $|M| \times |H|$ matrix with an $|H| \times |H|$ matrix, and the product of an $|H| \times |H|$ matrix with an $|H| \times |M|$ matrix using $O(M/H)$ square matrix multiplications. This requires $O(MH^{\omega-1})$ time. Since $|H| < 2|E|^{1-\alpha}$ and $|M| < 2|E|^{1-\alpha/2}$ we can use rectangular matrix multiplication and perform this phase in $O(E^{(1-\alpha)\omega(1,1,(2-\alpha)/(2-2\alpha)})$ time which is slightly better than $O(E^{1+\alpha})$. In fact, we can choose α which satisfies

$$(1+\alpha) = (1-\alpha)\omega(1, 1, \frac{2-\alpha}{2-2\alpha})$$

and the overall running time would still be $O(E^{1+\alpha})$. Using a computer program that implements the formula from [HP98] we have found that $\alpha = 0.474$ suffices.

4 The general case: Finding C_k's

In this section we describe a generic algorithm for finding a C_k in a directed graph, for any *fixed* $k \geq 3$. The C_3 algorithm of Section 2 partitioned the vertices into two degree classes. The C_4 algorithm of the previous section used three degree classes. It turns out that to obtain improved algorithms for finding larger C_k's, we need an unbounded number of degree classes. For every $0 \leq i < \log |V|$, let

$$V_i \;=\; \{v \in V \mid 2^i \leq \deg(v) < 2^{i+1}\}$$

Clearly $|V_i| \leq |E|/2^{i-1}$. There are therefore $\log |V|$ vertex classes, and the C_k's of the input graph can be classified, according to the classes of their ordered set of vertices, into $O(\log^k V)$ cycle classes. The algorithm will consider each such class separately. This will only increase the running time of the algorithm by a polylogarithmic factor.

We are usually only interested in finding *simple* C_k's, i.e., C_k's that do not contain repeated vertices. We can, however, use the *color coding* technique of [AYZ95] to reduce the problem of finding a simple C_k in a given input graph into the problem of finding a non-necessarily simple C_k in a collection of $O(\log V)$ graphs. For completeness we describe a simple randomized version of the color coding technique that is sufficient for our purposes. We simply choose a random coloring $c : V \to \{0, 1, \ldots, k-1\}$ of the vertices of the graph and construct the graph $G_c = (V, E_c)$, where $E_c = \{\, (u, v) \in E \mid c(v) - c(u) \equiv 1 \pmod{k}\}$. Clearly, any C_k in G_c is simple. Also, if G contains a simple C_k, then this C_k is present in G_c with probability $1/k^{k-1}$. As k is assumed to be fixed, this is enough for our purposes. It is possible to derandomize this technique. This is done by constructing, deterministically, a sequence $c_1, c_2, \ldots$ of $O(\log V)$ colorings such that if G contains a simple C_k, then at least one of the subgraphs G_{c_i} also contains one. For the details the reader is referred to [AYZ95]. In the sequel we can therefore ignore the simplicity constraint.

Let $0 \le f_0, f_1, \ldots, f_{k-1} < \log |V|$. How do we find a cycle $v_0, v_1, \ldots, v_{k-1}$ in the graph for which $v_i \in V_{f_i}$, for $0 \le i < k$? Let A be the adjacency matrix of the input graph. For $0 \le i, j < \log |V|$, let $A_{i,j}$ be the $|V_i| \times |V_j|$ submatrix of A representing the edges directed from V_i to V_j. It should be noted that we sometimes prefer to represent A or $A_{i,j}$ as a *sparse* matrix, e.g., with adjacency lists. Clearly, the graph contains a cycle of the required form if and only if the matrix obtained as a result of the Boolean chain product $A_{f_0,f_1} A_{f_1,f_2} \cdots A_{f_{k-2},f_{k-1}} A_{f_{k-1},f_0}$ contains a 1 on its diagonal. Or, more efficiently, if there exist $0 \le i < j < k$ such that the matrix

$$A_{f_i,f_{i+1}} \cdots A_{f_{j-1},f_j} \wedge (A_{f_j,f_{j+1}} \cdots A_{f_{i-1},f_i})^T ,$$

contains a 1 anywhere. (In the above expression indices are interpreted modulo k, $A \wedge B$ is the matrix obtained by logically and'ing the corresponding elements of the matrices A and B, and A^T is the transpose of the matrix A.)

This immediately reminds us of the classical *matrix chain product* problem, one of the most famous examples used to explain the dynamic programming technique (see, e.g., [CLRS01]), with several twists. In the classical problem, the rectangular matrix products are performed using the naive algorithm, i.e., the cost of multiplying an $a \times b$ matrix by a $b \times c$ matrix is abc. Using a standard reduction of rectangular matrix products to square matrix products (see, e.g., Huang and Pan [HP98]), the cost of such a product is reduced to $abc/\min\{a, b, c\}^{3-\omega}$, where ω is the exponent of square matrix multiplication. More importantly, as noted earlier, the matrices that we are considering are sometimes sparse. This can be used, in certain cases, to speed up the computation.

As we are only interested in the asymptotic complexity of the computation, it is convenient to switch to a 'logarithmic scale' and consider all quantities as powers of $|E|$. In particular, let $d_i = f_i/\log_2 |E|$, so that $|E|^{d_i} = 2^{f_i}$, for $0 \le i < k$. Notice that $0 \le d_i \le 1$.

Let $C_k(d_0, d_1, \ldots, d_{k-1})$ be such that $O(E^{C_k(d_0,d_1,\ldots,d_{k-1})})$ is a bound on the number of operations needed to find a C_k of the class we are considering. Let $P_{i,j}(d_i, \ldots, d_j)$ be such that $O(E^{P_{i,j}(d_i,\ldots,d_j)})$ is a bound on the complexity of computing the product $A_{f_i,f_{i+1}} \ldots A_{f_{j-1},f_j}$. The following two lemmas, the first of which using the dynamic programming spirit, establish suitable values for $P_{i,j}(d_i,\ldots,d_j)$ and $C_k(d_0,d_1,\ldots,d_{k-1})$. To keep the following formulae concise, we omit the arguments of expressions of the form $P_{i,j}$. Indices here, and in what follows, are considered modulo k.

Lemma 4.1

$$P_{i,i+1} \;=\; 1$$

$$P_{i,j} \;=\; \min\left\{ \begin{array}{c} P_{i,j-1}+d_{j-1}\ ,\ P_{i+1,j}+d_{i+1} \\ \min\limits_{i<r<j}\ \max\{P_{i,r},P_{r,j},M(1-d_i,1-d_r,1-d_j)\} \end{array} \right\} \quad , \quad j\neq i+1\ ,$$

where

$$M(s_0,s_1,s_2) \;=\; s_0+s_1+s_2-(3-\omega)\min\{s_0,s_1,s_2\}\ .$$

Proof: We can create a sparse representation of $A_{f_i,f_{i+1}}$ in $O(E^1)$ time and therefore $P_{i,i+1}=1$ is suitable. Suppose that we have computed a sparse representation of $A_{f_i,f_{i+1}}\cdots A_{f_{j-2},f_{j-1}}$ in $O(E^{P_{i,j-1}})$ time. By traversing the $O(E^{d_{j-1}})$ edges incident with each $v\in V_{f_{j-1}}$ we can compute a sparse representation of $A_{f_i,f_{i+1}}\cdots A_{f_{j-1},f_j}$ in $O(E^{P_{i,j-1}+d_{j-1}})$ time. Thus, $P_{i,j}\le P_{i,j-1}+d_{j-1}$. A similar argument shows that $P_{i,j}\le P_{i+1,j}+d_{i+1}$. For any $i<r<j$, suppose that we have computed a sparse representation of $A_{f_i,f_{i+1}}\cdots A_{f_{r-1},f_r}$ in $O(E^{P_{i,r}})$ time, and a sparse representation of $A_{f_r,f_{r+1}}\cdots A_{f_{j-1},f_j}$ in $O(E^{P_{r,j}})$ time. It takes at most $O(E^{P_{i,r}}+E^{1-d_i}E^{1-d_r})$ time to create the (non sparse) matrix representation of $A_{f_i,f_{i+1}}\cdots A_{f_{r-1},f_r}$, and at most $O(E^{P_{r,j}}+E^{1-d_r}E^{1-d_j})$ time to create the (non sparse) matrix representation of $A_{f_r,f_{r+1}}\cdots A_{f_{j-1},f_j}$. Multiplying the two matrices requires $O(E^{M(1-d_i,1-d_r,1-d_j)})$. Thus, $P_{i,j}\le\max\{P_{i,r},P_{r,j},M(1-d_i,1-d_r,1-d_j)\}$. In particular, $P_{i,j}\le\min_{i<r<j}\ \max\{P_{i,r},P_{r,j},M(1-d_i,1-d_r,1-d_j)\}$. (Notice that if we have decided to use a non-sparse matrix representation we can always switch back to the sparse method since the running time of this conversion is at most the number of entries of the matrix, which is less than the time it took us to create it.) $\square$

Lemma 4.2

$$C_k(d_0,d_1,\ldots,d_{k-1}) \;=\; \min\{\ \min_{0\le i<j\le k-1}\ \max\{\,P_{i,j}\ ,\ P_{j,i}\,\}\quad ,\quad \min_{0\le i\le k-1}\ 2-d_i\,\}\ .$$

Proof: Given sparse representations of $A_{f_i,f_{i+1}}\cdots A_{f_{j-1},f_j}$ and $A_{f_j,f_{j+1}}\cdots A_{f_{i-1},f_i}$ we can use radix sort to sort the first representation in linear time and the transpose of the second representation in linear time. We can then find whether $A_{f_i,f_{i+1}}\cdots A_{f_{j-1},f_j}\wedge(A_{f_j,f_{j+1}}\cdots A_{f_{i-1},f_i})^T$, contains a 1 anywhere in linear time. Thus, $C_k(d_0,d_1,\ldots,d_{k-1})\le\min_{0\le i<j\le k-1}\ \max\{\,P_{i,j}\ ,\ P_{j,i}\,\}$. There are $O(E^{1-d_i})$ vertices in V_{f_i}. For each vertex, we can check in $O(E)$ time whether it is on some C_k. Thus, $C_k(d_0,d_1,\ldots,d_{k-1})\le 2-d_i$. $\square$

Given $d_0, d_1, \ldots, d_{k-1}$, the algorithm solves this small dynamic programming problem, in constant time, and determines the optimal way of finding cycles of the current class. This is repeated for all cycle classes.

Clearly, the overall complexity of the algorithm is $\tilde{O}(E^{c_k})$, where

$$c_k \;=\; \max_{0 \le d_0, d_1, \ldots, d_{k-1} \le 1} C_k(d_0, d_1, \ldots, d_{k-1}) \;.$$

To obtain the asymptotic complexity of the algorithm, we need to determine the constant c_k, as a function of ω, the exponent of square matrix multiplication. This, in general, is not an easy task! It is not difficult to reduce the problem of computing c_k to a finite (though large) set of linear programs, of constant (though large) size each. The preceding two sections imply that $c_3 \le 2\omega/(\omega+1)$ and that $c_4 \le (4\omega-1)/(2\omega+1)$. To see that equality holds, notice that for $k=3$ we have $C_3(\frac{\omega-1}{\omega+1}, \frac{\omega-1}{\omega+1}, \frac{\omega-1}{\omega+1}) = 2\omega/(\omega+1)$. For $k=4$ we have $C_4(\frac{2\omega-2}{2\omega+1}, \frac{2\omega-2}{2\omega+1}, \frac{\omega-1}{2\omega+1}, \frac{\omega-1}{2\omega+1}) = (4\omega-1)/(2\omega+1)$. In the next section we show that $c_5 = 3\omega/(\omega+2)$.

5 Another concrete case: Finding pentagons (C_5's)

In this section we show that $c_5 = 3\omega/(\omega+2)$, thereby getting an $\tilde{O}(E^{3\omega/(\omega+2)})$ algorithm for detecting C_5's and proving Theorem 1.2. First, notice that $C(\frac{\omega-1}{2+\omega}, \frac{\omega-1}{2+\omega}, \frac{\omega-1}{2+\omega}, \frac{\omega-1}{2+\omega}, \frac{\omega-1}{2+\omega})$ already shows that $c_5 \ge 3\omega/(\omega+2)$ (this means that regular graphs already require a running time of $\tilde{O}(E^{3\omega/(\omega+2)})$. To see the upper bound, it would not be convenient to work directly with the dynamic programming problem which defines c_5. As in the case of the C_4 algorithm, it is more convenient to present a more concise algorithm which handles many types of cycle classes simultaneously.

Let $\delta = (\omega-1)/(2+\omega)$. We show that $C(d_0, d_1, d_2, d_3, d_4) \le 3\omega/(2+\omega)$ for each cycle type $(d_0, d_1, d_2, d_3, d_4)$, namely, cycles $(v_0, v_1, v_2, v_3, v_4)$ with $v_i \in V_{f_i}$.

Cycles of type $(d_0, d_1, d_2, d_3, d_4)$ where each $d_i > \delta$: Clearly, $P_{4,0} = 1$. On the other hand, $P_{0,4} \le \max\{P_{0,2}, P_{2,4}, M(1-d_0, 1-d_2, 1-d_4)\}$. Now, $P_{0,2} \le \omega(1-\delta)$, $P_{2,4} \le \omega(1-\delta)$, and $M(1-d_0, 1-d_2, 1-d_4) \le \omega(1-\delta)$. Thus, $P_{0,4} \le \omega(1-\delta)$. Thus, $C(d_0, d_1, d_2, d_3, d_4) \le \omega(1-\delta) = 3\omega/(2+\omega)$.

Cycles of type $(d_0, d_1, d_2, d_3, d_4)$ where $d_0, d_1, d_2, d_3 > \delta$ and $d_4 \le \delta$: As in the previous case, $P_{0,3} \le \omega(1-\delta)$. On the other hand, $P_{3,0} \le P_{3,4} + d_4 \le 1 + \delta$. Thus, $C(d_0, d_1, d_2, d_3, d_4) \le \max\{\omega(1-\delta), 1+\delta\} = 3\omega/(2+\omega)$.

Cycles of type $(d_0, d_1, d_2, d_3, d_4)$ where $d_0, d_1, d_2 > \delta$ and $d_3, d_4 \le \delta$: As in the previous cases, $P_{0,2} \le \omega(1-\delta)$. On the other hand, $P_{2,0} \le P_{2,4} + d_4 \le P_{2,3} + d_3 + d_4 \le 1 + 2\delta$. Thus, $C(d_0, d_1, d_2, d_3, d_4) \le \max\{\omega(1-\delta), 1+2\delta\} = 3\omega/(2+\omega)$.

Cycles of type $(d_0, d_1, d_2, d_3, d_4)$ **where** $d_1, d_3, d_4 \le \delta$**:** $P_{0,2} \le P_{0,1}+d_1 \le 1+\delta$. On the other hand, $P_{2,0} \le P_{2,4}+d_4 \le P_{2,3}+d_3+d_4 \le 1+2\delta$. Thus, $C(d_0, d_1, d_2, d_3, d_4) \le \max\{1+\delta, 1+2\delta\} = 3\omega/(2+\omega)$.

Cycles of type $(d_0, d_1, d_2, d_3, d_4)$ **where** $d_0, d_1 > \delta$**,** $d_2, d_3, d_4 \le \delta$**:** Let $\epsilon = \delta - d_2$. We have three subcases. If $d_0 > \delta+\epsilon$ and $d_1 > \delta+\epsilon$ then consider $P_{0,2}$ and $P_{2,0}$. Clearly, $P_{2,0} \le P_{2,4}+d_4 \le P_{2,3}+d_3+d_4 \le 1+2\delta$. On the other hand, $P_{0,2} \le M(1-d_0, 1-d_1, 1-d_2) \le M(1-\delta-\epsilon, 1-\delta-\epsilon, 1-\delta+\epsilon) \le M(1-\delta, 1-\delta, 1-\delta) = \omega(1-\delta)$. Thus, $C(d_0, d_1, d_2, d_3, d_4) \le \max\{1+2\delta, \omega(1-\delta)\} = 3\omega/(2+\omega)$. Otherwise, if $d_0 \le \delta+\epsilon$ then consider $P_{1,4}$ and $P_{4,1}$. Clearly $P_{1,4} \le P_{1,3}+d_3 \le P_{1,2}+d_2+d_3 \le 1+2\delta$. On the other hand, $P_{4,1} \le P_{4,0}+d_0 \le 1+\delta+\epsilon$. Thus, $C(d_0, d_1, d_2, d_3, d_4) \le \max\{1+2\delta, 1+\delta+\epsilon\} = 3\omega/(2+\omega)$. Otherwise, if $d_1 \le \delta+\epsilon$ then consider $P_{0,2}$ and $P_{2,0}$. Clearly, $P_{0,2} \le P_{0,1}+d_1 \le 1+\delta+\epsilon$. On the other hand, $P_{2,0} \le P_{2,4}+d_4 \le P_{2,3}+d_3+d_4 \le 1+2\delta$. Thus, $C(d_0, d_1, d_2, d_3, d_4) \le \max\{1+2\delta, 1+\delta+\epsilon\} = 3\omega/(2+\omega)$.

Cycles of type $(d_0, d_1, d_2, d_3, d_4)$ **where** $d_0, d_1, d_3 > \delta$**,** $d_2, d_4 \le \delta$**:** We may assume $d_4 < d_2$ since the other case is symmetric. Let $\epsilon = \delta - d_2$. We have four subcases. If $d_0 \le \delta+\epsilon$ then consider $P_{1,3}$ and $P_{3,1}$. Clearly, $P_{1,3} \le 1+d_2 \le 1+\delta$, and $P_{3,1} \le 1+d_4+d_0 \le 1+(\delta-\epsilon)+(\delta+\epsilon) = 1+2\delta$. Thus, $C(d_0, d_1, d_2, d_3, d_4) \le \max\{1+\delta, 1+2\delta\} = 3\omega/(2+\omega)$. Otherwise, if $d_1 \le \delta+\epsilon$ then consider $P_{0,3}$ and $P_{3,0}$. Clearly, $P_{0,3} \le 1+d_1+d_2 \le 1+(\delta+\epsilon)+(\delta-\epsilon) = 1+2\delta$. On the other hand, $P_{3,0} \le 1+d_4 \le 1+\delta$. Thus, $C(d_0, d_1, d_2, d_3, d_4) \le \max\{1+2\delta, 1+\delta\} = 3\omega/(2+\omega)$. Otherwise, if $d_3 \le \delta+\epsilon$ then consider $P_{0,2}$ and $P_{2,0}$. Clearly, $P_{0,2} \le M(1-d_0, 1-d_1, 1-d_2) \le M(1-\delta-\epsilon, 1-\delta-\epsilon, 1-\delta+\epsilon) \le M(1-\delta, 1-\delta, 1-\delta) \le \omega(1-\delta)$. On the other hand, $P_{2,0} \le 1+d_3+d_4 \le 1+(\delta+\epsilon)+(\delta-\epsilon) = 1+2\delta$. Thus, $C(d_0, d_1, d_2, d_3, d_4) \le \max\{\omega(1-\delta), 1+2\delta\} = 3\omega/(2+\omega)$. Otherwise, d_0, d_1, d_3 are all greater than $\delta+\epsilon$ so consider $P_{0,3}$ and $P_{3,0}$. Clearly, $P_{0,3} \le \max\{P_{0,1}, P_{1,3}, M(1-d_0, 1-d_1, 1-d_3)\}$. Now, $P_{0,1} = 1$, $P_{1,3} \le 1+d_2 \le 1+\delta$ and $M(1-d_0, 1-d_1, 1-d_3) \le M(1-\delta, 1-\delta, 1-\delta) = \omega(1-\delta)$. Thus, $P_{0,3} \le \omega(1-\delta)$. On the other hand, $P_{3,0} \le 1+d_4 \le 1+\delta$. Thus, $C(d_0, d_1, d_2, d_3, d_4) \le \max\{\omega(1-\delta), 1+\delta\} = 3\omega/(2+\omega)$.

A cyclicity argument shows that the above 6 cases handle all possible cycle types. □

6 Conjectures for $k \ge 6$

Based on extensive numerical experimentations, we conjecture that

Conjecture 6.1

$$c_6 = \begin{cases} \frac{10\omega-3}{4\omega+4} & \text{if } 2 \le \omega \le \frac{13}{6} \\ \frac{22-4\omega}{17-4\omega} & \text{if } \frac{13}{6} \le \omega \le \frac{9}{4} \\ \frac{11\omega-2}{4\omega+5} & \text{if } \frac{9}{4} \le \omega \le \frac{16}{7} \\ \frac{10-\omega}{7-\omega} & \text{if } \frac{16}{7} \le \omega \le \frac{5}{2} \\ \frac{5}{3} & \text{if } \frac{5}{2} \le \omega \le 3 \end{cases}$$

Our numerical experiments lead us to conjecture that for all odd k, the hardest case is that of regular graphs. It is an easy exercise to maximize $C(\delta, \delta, \ldots, \delta)$. It is maximized when $\delta = (\omega - 1)/(\omega + (k-1)/2)$ and the value is then $(k+1)\omega/(2\omega + k - 1)$. We therefore have:

Conjecture 6.2 *For all odd $k \geq 3$, $c_k = (k+1)\omega/(2\omega + k - 1)$. In particular, the algorithm from Section 4 finds a C_k in $\tilde{O}(E^{(k+1)\omega/(2\omega+k-1)})$ time.*

Sections 2 and 5 show that Conjecture 6.2 holds for $k = 3, 5$. A combinatorial $O(E^{2-2/(k+1)})$ time algorithm for detecting odd cycles of length k is presented in [AYZ97]. It is easy to see that if Conjecture 6.1 is true, then our algorithm improves on this algorithm if $\omega < 2k/(k-1)$. (Many believe that $\omega = 2 + o(1)$.)

For even k, our numerical experiments show that regular graphs are *not* the worst case. This is provably so for $k = 4, 6, 8$. In fact, we believe, as expressed in Conjecture 6.1, that c_k is expressed as different functions of ω for different values of ω for all even $k \geq 6$. A combinatorial $O(E^{2-2/k})$ time algorithm for detecting even cycles of length k is presented in [AYZ97]. We conjecture that, for every even k, we have $c_k < 2 - 2/k$, if ω is sufficiently close to 2.

7 Concluding remarks

Determining the values of c_k, for $k \geq 6$ is an interesting open problem. As we said, we suspect the answer for even values of k to be quite complicated. It would be nice, at first, to obtain a proof of Conjecture 6.1.

Fast rectangular matrix multiplication is used for speeding up graph algorithms also by Zwick [Zwi02], Demetrescu and Italiano [DI00], and by Kratsch and Spinrad [KS03]. Finding other applications of rectangular matrix multiplications, or of the techniques developed here, is also an interesting problem.

References

[AYZ95] N. Alon, R. Yuster, and U. Zwick. Color-coding. *Journal of the ACM*, 42:844–856, 1995.

[AYZ97] N. Alon, R. Yuster, and U. Zwick. Finding and counting given length cycles. *Algorithmica*, 17:209–223, 1997.

[CLRS01] T.H. Cormen, C.E. Leiserson, R.L. Rivest, and C. Stein. *Introduction to algorithms.* The MIT Press, second edition, 2001.

[Cop97] D. Coppersmith. Rectangular matrix multiplication revisited. *Journal of Complexity*, 13:42–49, 1997.

[CW90] D. Coppersmith and S. Winograd. Matrix multiplication via arithmetic progressions. *Journal of Symbolic Computation*, 9:251–280, 1990.

[DI00] C. Demetrescu and G. F. Italiano. Fully dynamic transitive closure: breaking through the $O(n^2)$ barrier. In *Proc. of 41st FOCS*, pages 381–389, 2000.

[EG03] F. Eisenbrand and F. Grandoni. Detecting directed 4-cycles still faster. *Information Processing Letters*, 87(1):13–15, 2003.

[HP98] X. Huang and V.Y. Pan. Fast rectangular matrix multiplications and applications. *Journal of Complexity*, 14:257–299, 1998.

[KS03] D. Kratsch and J. Spinrad. Between $O(nm)$ and $O(n^\alpha)$. In *Proc. of 14th SODA*, pages 709–716, 2003.

[Zwi02] U. Zwick. All-pairs shortest paths using bridging sets and rectangular matrix multiplication. *Journal of the ACM*, 49:289–317, 2002.