Trade-offs for Fully Dynamic Transitive Closure on DAGs: Breaking Through the $O(n^2)$ Barrier *

Camil Demetrescu †
Dipartimento di Informatica e Sistemistica
Università di Roma "La Sapienza", Roma, Italy

Giuseppe F. Italiano ‡
Dipartimento di Informatica, Sistemi e Produzione
Università di Roma "Tor Vergata", Roma, Italy

Abstract

We present an algorithm for directed acyclic graphs that breaks through the $O(n^2)$ barrier on the single-operation complexity of fully dynamic transitive closure, where n is the number of edges in the graph. We can answer queries in $O(n^\epsilon)$ worst-case time and perform updates in $O(n^{\omega(1,\epsilon,1)-\epsilon} + n^{1+\epsilon})$ worst-case time, for any $\epsilon \in [0,1]$, where $\omega(1,\epsilon,1)$ is the exponent of the multiplication of an $n \times n^\epsilon$ matrix by an $n^\epsilon \times n$ matrix. The current best bounds on $\omega(1,\epsilon,1)$ imply an $O(n^{0.575})$ query time and an $O(n^{1.575})$ update time in the worst case. Our subquadratic algorithm is randomized, and has one-sided error. As an application of this result, we show how to solve single-source reachability in $O(n^{1.575})$ time per update and constant time per query.

*This work has been partially supported by MIUR, the Italian Ministry of Education, University and Research, under Project ALGO-NEXT (Algorithms for the Next Generation Internet and Web: Methodologies, Design and Experiments) and by the Sixth Framework Programme of the EU under contract no. 507613 (Network of Excellence "EuroNGI: Designing and Engineering of the Next Generation Internet"). Portions of this work have been presented at the *41st Annual Symp. on Foundations of Computer Science* (FOCS 2000) [3].

†Email: `demetres@dis.uniroma1.it`. URL: `http://www.dis.uniroma1.it/~demetres`. Part of this work has been done while visiting AT&T Shannon Laboratory, Florham Park, NJ.

‡Email: `italiano@info.uniroma2.it`. URL: `http://www.info.uniroma2.it/~italiano`. Part of this work has been done while visiting Columbia University, New York, NY.

1 Introduction

In this paper we present fully dynamic algorithms for maintaining information about reachability in a directed acyclic graph. Throughout the paper, we denote by m and by n the number of edges and vertices in the graph, respectively. A dynamic graph algorithm maintains a given property on a graph subject to dynamic changes, such as edge insertions and edge deletions. We say that an algorithm is *fully dynamic* if it can handle both edge insertions and edge deletions. A *partially dynamic* algorithm can handle either edge insertions or edge deletions, but not both: we say that it is *incremental* if it supports insertions only, and *decremental* if it supports deletions only.

In this paper we consider three variants of fully dynamic reachability problems, according to the kind of queries supported. In the *fully dynamic transitive closure problem* we wish to maintain a directed graph under an intermixed sequence of edge insertions, edge deletions, and reachability queries of the form: "Is vertex y reachable from vertex x?" In the *fully dynamic single-source reachability problem* we are given a source vertex s and the query becomes: "Is vertex v reachable from source vertex s?" In the *fully dynamic st-reachability problem*, we are given two specified vertices s and t, and the query becomes: "Is vertex t reachable from vertex s?"

Research on dynamic reachability spans over two decades. Before describing the results known, we list the bounds obtainable with simple-minded methods. If we do nothing during each update, then we have to explore the whole graph in order to answer reachability queries: this gives $O(m+n)$ time per query and $O(1)$ time per update in the worst case, where n is the number of vertices and m is the number of edges in the graph. We note that $O(m+n) = O(n^2)$ in the case of dense graphs.

On the other extreme, we could recompute the transitive closure from scratch after each update; as this task can be accomplished via matrix multiplication [1, 15], this approach yields $O(1)$ time per query and $O(n^\omega)$ time per update in the worst case, where ω is the best known exponent for matrix multiplication (currently $\omega < 2.376$ [2]). For single-source reachability or *st*-reachability, we can do better by just running a search starting from s after each update: in the worst case this takes $O(m+n)$ time. Again, $O(m+n) = O(n^2)$ in the case of dense graphs.

Previous Work. To the best of our knowledge, for dynamic single-source reachability and dynamic *st*-reachability there are no better bounds than the simple-minded methods. Most of the attention has been focused on fully dynamic transitive closure. For the *incremental* version of this problem, the first algorithm was proposed by Ibaraki and Katoh [8] in 1983: its running time was $O(n^3)$ over any sequence of insertions. This bound was later improved to $O(n)$ amortized time per insertion by Italiano [9] and also by La Poutré and van Leeuwen [14]. Yellin [17] gave an $O(m^*\delta_{max})$ algorithm for m edge insertions, where m^* is the number of edges in the final transitive closure and δ_{max} is the maximum out-degree of the final graph. All these algorithms maintain explicitly the transitive closure, and so their query time is $O(1)$.

The first *decremental* algorithm was again given by Ibaraki and Katoh [8], with a

running time of $O(n^2)$ per deletion. This was improved to $O(m)$ per deletion by La Poutré and van Leeuwen [14]. Italiano [10] presented an algorithm that achieves $O(n)$ amortized time per deletion on directed acyclic graphs. Yellin [17] gave an $O(m^*\delta_{max})$ algorithm for m edge deletions, where m^* is the initial number of edges in the transitive closure and δ_{max} is the maximum out-degree of the initial graph. Again, the query time of all these algorithms is $O(1)$. More recently, Henzinger and King [5] gave a randomized decremental transitive closure algorithm for general directed graphs with a query time of $O(n/\log n)$ and an amortized update time of $O(n \log^2 n)$.

The first *fully dynamic* transitive closure algorithm was devised by Henzinger and King [5] in 1995: they gave a randomized Monte Carlo algorithm with one-sided error supporting a query time of $O(n/\log n)$ and an amortized update time of $O(n\hat{m}^{0.575} \log^2 n)$, where $\hat{m}$ is the average number of edges in the graph throughout the whole update sequence. Since $\hat{m}$ can be as high as $O(n^2)$, their update time is $O(n^{2.16} \log^2 n)$. Khanna, Motwani and Wilson [11] proved that, when a lookahead of $\Theta(n^{0.18})$ in the updates is permitted, a deterministic update bound of $O(n^{2.18})$ can be achieved. Very recently, King and Sagert [13] showed how to support queries in $O(1)$ time and updates in $O(n^{2.26})$ time for general directed graphs; their algorithm is randomized with one-sided error. The bounds of King and Sagert were further improved by King [12], who exhibited a deterministic algorithm on general digraphs with $O(1)$ query time and $O(n^2 \log n)$ amortized time per update operations. For the special case of acyclic graphis, King and Sagert [13] showed how to perform updates in $O(n^2)$ time; again, their algorithm is randomized with one-sided error.

We observe that fully dynamic transitive closure algorithms with $O(1)$ query time maintain explicitly the transitive closure of the input graph, in order to answer each query with exactly one lookup (on its adjacency matrix). Since an update may change as many as $\Omega(n^2)$ entries of this matrix, $O(n^2)$ seems to be the best update bound that one could hope for this class of algorithms. In the companion paper [4], we show how to realize an $O(n^2)$ update bound for fully dynamic transitive closure on general directed graphs while maintaining one lookup per query. Table 1 summarizes the best known bounds for the problems considered in this paper.

Problem	**Fast Query**		**Fast Update**	
	Query	**Update**	**Query**	**Update**
Fully dynamic transitive closure	$O(1)$	$O(n^2)$	$O(n^2)$	$O(1)$
Fully dynamic single-source reachability	$O(1)$	$O(n^2)$	$O(n^2)$	$O(1)$
Fully dynamic *st*-reachability	$O(1)$	$O(n^2)$	$O(n^2)$	$O(1)$

Table 1: Previous bounds for fully dynamic reachability problems.

It seems thus quite natural to ask whether the $O(n^2)$ barrier for the single-operation time complexity of fully dynamic reachability can be broken. We remark that this has been an elusive goal for many years and indeed dynamic *st*-reachability has been posed as an

open problem by Holm et al. [6]. Note that, in case of fully dynamic transitive closure, to achieve subquadratic update bounds one should expect to spend more time for queries.

Our Results. In this paper, we affirmatively answer this question for acyclic graphs. Building on previous work by King and Sagert [13], we show how to trade off query times for updates in case of fully dynamic transitive closure on directed acyclic graphs: each query can be answered in time $O(n^\epsilon)$ and each update can be performed in time $O(n^{\omega(1,\epsilon,1)-\epsilon} + n^{1+\epsilon})$, for any $\epsilon \in [0,1]$, where $\omega(1,\epsilon,1)$ is the exponent of the multiplication of an $n \times n^\epsilon$ matrix by an $n^\epsilon \times n$ matrix. Balancing the two terms in the update bound yields that ϵ must satisfy the equation $\omega(1,\epsilon,1) = 1 + 2\epsilon$. The current best bounds on $\omega(1,\epsilon,1)$ [2, 7] imply that $\epsilon < 0.575$ [18]. Thus, the smallest update time is $O(n^{1.575})$, which gives a query time of $O(n^{0.575})$. Our subquadratic algorithm is randomized and has one-sided error. Next, we show how to solve dynamic single-source and *st*-reachability in $O(n^{1.575})$ time per update and constant time per query. Our results are summarized in Table 2.

Problem	**Query**	**Update**
Fully dynamic transitive closure	$O(n^{0.575})$	$O(n^{1.575})$
Fully dynamic single-source reachability	$O(1)$	$O(n^{1.575})$
Fully dynamic *st*-reachability	$O(1)$	$O(n^{1.575})$

Table 2: New bounds for fully dynamic reachability problems on directed acyclic graphs.

The remainder of this paper is organized as follows. In Section 2 we introduce one problem on dynamic matrices, and show how to solve it efficiently. Next, we show how to exploit this problem for the design of fully dynamic algorithms for transitive closure in Section 3. In Section 4 we consider dynamic single-source and *st*-reachability. Finally, in Section 5 we list some concluding remarks.

2 Dynamic Matrices

In this section we investigate a problem on dynamic matrices that will be central to designing our fully dynamic transitive closure algorithm.

We define the problem as follows. Let M be an $n \times n$ integer matrix. We consider the problem of performing an intermixed sequence of operations on M of the following kind:

- `Init`(X): perform the initialization $M \leftarrow X$, where X is an $n \times n$ integer matrix.
- `Update`(J, I): perform the update operation $M \leftarrow M + J \cdot I$, where J is an $n \times 1$ column integer vector, and I is a $1 \times n$ row integer vector. The product $J \cdot I$ is an $n \times n$ matrix defined for any $1 \leq x, y \leq n$ as:

$$(J \cdot I)[x, y] = J[x] \cdot I[y]$$

- `Lookup`(x, y): return the integer value $M[x, y]$.

It is straightforward to observe that `Lookup` can be supported in unit time and operations `Init` and `Update` in $O(n^2)$ worst-case time by explicitly performing the algebraic operations specified in the previous definition.

In the following we show that, if one is willing to give up unit time for `Lookup` operations, it is possible to support `Update` in $O(n^{\omega(1,\epsilon,1)-\epsilon})$ worst-case time for each update operation, for any ϵ, $0 \leq \epsilon \leq 1$, where $\omega(1, \epsilon, 1)$ is the exponent of the multiplication of an $n \times n^\epsilon$ matrix by an $n^\epsilon \times n$ matrix. Queries on individual entries of M are answered in $O(n^\epsilon)$ worst-case time via `Lookup` operations and `Init` still takes $O(n^2)$ worst-case time.

We now sketch the main ideas behind the algorithm. We follow a simple lazy approach: we log at most n^ϵ update operations without explicitly computing them and we perform a global reconstruction of the matrix every n^ϵ updates. The reconstruction is done through fast rectangular matrix multiplication. This yields an implicit representation for M which requires us to run through logged updates in order to answer queries about entries of M.

We maintain the following elementary data structures with $O(n^2)$ space:

- an $n \times n$ integer matrix $Lazy$ which maintains a lazy representation of M;
- an $n \times n^\epsilon$ integer matrix Buf_J in which we buffer update column vectors J;
- an $n^\epsilon \times n$ integer matrix Buf_I in which we buffer update row vectors I;
- a counter t of the number of performed `Update` operations since the last `Init`, modulo n^ϵ.

Before showing how to implement operations, we discuss a simple invariant property maintained in our data structure, which guarantees the correctness of our approach. We use the following notation:

Definition 1 *We denote by $Buf_J\langle j\rangle$ the $n \times j$ matrix obtained by considering only the first j columns of Buf_J. Similarly, we denote by $Buf_I\langle i\rangle$ the $i \times n$ matrix obtained by considering only the first i rows of Buf_I.*

Invariant 1 *At any time in the sequence of operations σ, the following invariant is maintained:*

$$M = Lazy + Buf_J\langle t\rangle \cdot Buf_I\langle t\rangle.$$

`Update`

```
     procedure Update(J, I)
1.   begin
2.       t ← t + 1
3.       if t ≤ n^ε then
4.           Buf_J[·, t] ← J
5.           Buf_I[t, ·] ← I
6.       else
7.           t ← 0
8.           Lazy ← Lazy + Buf_J · Buf_I
9.   end
```

`Update` first increases t and, if $t \leq n^\epsilon$, it copies column vector J onto the t-th column of Buf_J (line 4) and row vector I onto the t-th row of Buf_I (line 5). If $t > n^\epsilon$, there is no more room in Buf_J and Buf_I for buffering updates. Then the counter t is reset in line 7 and the reconstruction operation in line 8 synchronizes $Lazy$ with M via rectangular matrix multiplication of the $n \times n^\epsilon$ matrix Buf_J by the $n^\epsilon \times n$ matrix Buf_I.

`Lookup`

```
     procedure Lookup(x, y)
1.   begin
2.       return Lazy[x, y] + Σ_{j=1}^{t} Buf_J[x, j] · Buf_I[j, y]
3.   end
```

`Lookup` runs through the first t columns and rows of buffers Buf_J and Buf_I, respectively, and returns the value of $Lazy$ corrected with the inner product of the x-th row of $Buf_J\langle t\rangle$ by the y-th column of $Buf_I\langle t\rangle$.

`Init`

```
     procedure Init(X)
1.   begin
2.       Lazy ← X
3.       t ← 0
4.   end
```

`Init` simply sets the value of $Lazy$ and empties the buffers by resetting t.

The following theorem discusses the time and space requirements of operations `Update`, `Lookup`, and `Init`. As already stated, the correctness easily follows from the fact that Invariant 1 is maintained throughout any sequence of operations.

Theorem 1 *Each* `Update` *operation can be supported in* $O(n^{\omega(1,\epsilon,1)-\epsilon})$ *worst-case time and each* `Lookup` *in* $O(n^\epsilon)$ *worst-case time, where* $0 \leq \epsilon \leq 1$ *and* $\omega(1, \epsilon, 1)$ *is the exponent for rectangular matrix multiplication.* `Init` *requires* $O(n^2)$ *time in the worst case. The space required is* $O(n^2)$.

Proof. An amortized update bound follows trivially from amortizing the cost of the rectangular matrix multiplication $Buf_J \cdot Buf_I$ against n^ϵ update operations. This bound can be made worst-case by standard techniques, i.e., by keeping two copies of the data structures: one is used for queries and the other is updated by performing matrix multiplication in the background.

As far as `Lookup` is concerned, it answers queries on the value of $M[x, y]$ in $\Theta(t)$ worst-case time, where $t \leq n^\epsilon$. □

Corollary 1 *If $O(n^\omega)$ is the time required for multiplying two $n \times n$ matrices, then we can support* `Update` *in $O(n^{2-(3-\omega)\epsilon})$ worst-case time and* `Lookup` *in $O(n^\epsilon)$ worst-case time. Choosing $\epsilon = 1$, the best known bound for matrix multiplication ($\omega < 2.376$) implies an $O(n^{1.376})$* `Update` *time and an $O(n)$* `Lookup` *time.*

Proof. A rectangular matrix multiplication between a $n \times n^\epsilon$ matrix by a $n^\epsilon \times n$ matrix can be performed by computing $O((n^{1-\epsilon})^2)$ multiplications between $n^\epsilon \times n^\epsilon$ matrices. This is done in $O\left((n^{1-\epsilon})^2 \cdot (n^\epsilon)^\omega\right)$. The amortized time of the reconstruction operation $Lazy \leftarrow Lazy + Buf_J \cdot Buf_I$ is thus $O\left(\frac{(n^{1-\epsilon})^2 \cdot (n^\epsilon)^\omega + n^2}{n^\epsilon}\right) = O(n^{2-(3-\omega)\epsilon})$. The rest of the claim follows from Theorem 1. □

3 Breaking Through the $O(n^2)$ Barrier

In this section we present the first algorithm that breaks through the $O(n^2)$ barrier on the single-operation complexity of fully dynamic transitive closure in the case of directed acyclic graphs. The problem is to maintain a directed graph $G = (V, E)$ under an intermixed sequence of the following operations:

TC_Insert(x, y): insert an edge from x to y in G;

TC_Delete(x, y): delete the edge from x to y in G;

TC_Query(x, y): report *yes* if there is a path from x to y in G, and *no* otherwise.

Following King and Sagert [13], we keep a count of the number of distinct paths between any pair of vertices. These counters may be as large as 2^n: as shown in [13], the wordsize can be reduced from n to $2c \lg n$ for any $c \geq 5$ by performing arithmetic operations modulo a random prime number. This yields $O(1)$ for reachability queries and $O(n^2)$ for updates on directed acyclic graphs: "yes" answers on reachability queries are always correct, while "no" answers are wrong with probability $O(\frac{1}{n^c})$.

To improve those bounds, we use a lazy path counting technique based on the implicit matrix representation of Section 2. Surprisingly, these simple ideas solve a problem that has been open for many years.

With our technique, we obtain the following bounds: queries are answered in $O(n^\epsilon)$ time and updates in $O(n^{\omega(1,\epsilon,1)-\epsilon} + n^{1+\epsilon})$ time, for any $0 \leq \epsilon \leq 1$, where $\omega(1, \epsilon, 1)$ is the

exponent of the multiplication of an $n \times n^\epsilon$ matrix by an $n^\epsilon \times n$ matrix. According to the current best bounds on $\omega(1, \epsilon, 1)$, this yields $O(n^{0.575})$ time per query and $O(n^{1.575})$ time per update.

3.1 Lazy Path Counting

We keep a count of the number of distinct paths between any pair of vertices in graph G by means of an instance M of the dynamic matrix data structure described in Section 2. We assume that $M[x, y]$ is the number of distinct paths between node x and node y in graph G. Since G is acyclic, this number is well-defined.

We now show how to implement `TC_Insert`, `TC_Delete` and `TC_Query` using the primitives `Update` and `Lookup` described in Section 2. We assume all arithmetic operations are performed in constant time.

`TC_Insert`

```
     procedure TC_Insert(x, y)
1.   begin
2.       E ← E ∪ {(x, y)}
3.       for z = 1 to n do
4.           J[z] ← M.Lookup(z, x)
5.           I[z] ← M.Lookup(y, z)
6.       M.Update(J, I)
7.   end
```

`TC_Insert` first puts edge (x, y) in the graph and then, after querying matrix `M`, computes two vectors J and I such that $J[z]$ is the number of distinct paths $z \rightsquigarrow x$ in G and $I[z]$ is the number of distinct paths $y \rightsquigarrow z$ in G (lines 3–5). Finally, it updates `M` in line 6. The operation performed on M is $M \leftarrow M + J \cdot I$: this means that the number $M[u, v]$ of distinct paths between any two nodes (u, v) is increased by the number $J[u]$ of distinct paths $u \rightsquigarrow x$ times the number $I[v]$ of distinct paths $y \rightsquigarrow v$, i.e., $M[u, v] \leftarrow M[u, v] + J[u] \cdot I[v]$.

`TC_Delete`

```
     procedure TC_Delete(x, y)
1.   begin
2.       E ← E − {(x, y)}
3.       for z = 1 to n do
4.           J[z] ← M.Lookup(z, x)
5.           I[z] ← M.Lookup(y, z)
6.       M.Update(−J, I)
7.   end
```

`TC_Delete` is identical to `TC_Insert`, except for the fact that it removes the edge (x, y) from the graph and performs the update of `M` in line 6 with $-J$ instead of J. The operation performed on M is $M \leftarrow M - J \cdot I$: this means that the number $M[u, v]$ of distinct paths

between any two nodes (u, v) is decreased by the number $J[u]$ of distinct paths $u \rightsquigarrow x$ times the number $I[v]$ of distinct paths $y \rightsquigarrow v$, i.e., $M[u, v] \leftarrow M[u, v] - J[u] \cdot I[v]$.

TC_Query

```
    procedure TC_Query(x, y)
1.  begin
2.      if M.Lookup(x, y) > 0 then return 1
3.      else return 0
4.  end
```

`TC_Query` simply looks up the value of $M[x, y]$ and returns 1 if the current number of distinct paths between x and y is positive, and zero otherwise.

◁ ◇ ▷

We are now ready to discuss the running time of our implementation of operations `TC_Insert`, `TC_Delete`, and `TC_Query`.

Theorem 2 *Any* `TC_Insert` *and any* `TC_Delete` *operation can be performed in* $O(n^{\omega(1,\epsilon,1)-\epsilon} + n^{1+\epsilon})$ *worst-case time, for any* $0 \leq \epsilon \leq 1$*, where* $\omega(1, \epsilon, 1)$ *is the exponent of the multiplication of an* $n \times n^{\epsilon}$ *matrix by an* $n^{\epsilon} \times n$ *matrix. Any* `TC_Query` *takes* $O(n^{\epsilon})$ *in the worst case. The space required is* $O(n^2)$*.*

Proof. We recall that, by Theorem 1, each entry of M can be queried in $O(n^{\epsilon})$ worst-case time, and each `Update` operation can be performed in $O(n^{\omega(1,\epsilon,1)-\epsilon})$ worst-case time. Since I and J can be computed in $O(n^{1+\epsilon})$ worst-case time by means of n queries on M, we can support both insertions and deletions in $O(n^{\omega(1,\epsilon,1)-\epsilon} + n^{1+\epsilon})$ worst-case time, while a reachability query for any pair of vertices (x, y) can be answered in $O(n^{\epsilon})$ worst-case time by simply querying the value of $M[x, y]$. □

Corollary 2 *Any* `TC_Insert` *and any* `TC_Delete` *operation requires* $O(n^{1.575})$ *worst-case time, and any* `TC_Query` *requires* $O(n^{0.575})$ *worst-case time. The space required is* $O(n^2)$*.*

Proof. Balancing the two terms in the update bound $O(n^{\omega(1,\epsilon,1)-\epsilon} + n^{1+\epsilon})$ yields that ϵ must satisfy the equation $\omega(1, \epsilon, 1) = 1 + 2\epsilon$. The current best bounds on $\omega(1, \epsilon, 1)$ [2, 7] imply that $\epsilon < 0.575$ [18]. Thus, the smallest update time is $O(n^{1.575})$, which gives a query time of $O(n^{0.575})$. □

The algorithm we presented is deterministic. However, as the numbers involved may be as large as 2^n, performing arithmetic operations in constant time requires wordsize $O(n)$. To reduce wordsize to $O(\log n)$ while maintaining the same subquadratic bounds ($O(n^{1.575})$ per update and $O(n^{0.575})$ per query) we perform all arithmetic operations modulo some random prime number as explained in [13]. Again, this produces a randomized Monte Carlo algorithm, where "yes" answers on reachability queries are always correct, while "no" answers are wrong with probability $O(\frac{1}{n^c})$ for any constant $c \geq 5$.

It is also not difficult to extend our subquadratic algorithm to deal with insertions/deletions of more than one edge at a time. In particular, we can support any insertion/deletion of up to $O(n^{1-\eta})$ edges incident to a common vertex in $O(n^{\omega(1,\epsilon,1)-\epsilon} + n^{2-(\eta-\epsilon)})$ worst-case time. We emphasize that this is still $o(n^2)$ for any $1 > \eta > \epsilon > 0$.

4 Dynamic Single-Source and *st*-Reachability

In this section we show how to use the data structure presented in Section 3 to solve dynamic single-source reachability problems. Given a directed graph $G = (V, E)$ and specified source vertex $s \in V$, the problem is to maintain G under an intermixed sequence of the following operations:

s_Insert(x, y): insert an edge from x to y in G;

s_Delete(x, y): delete the edge from x to y in G;

s_Query(v): report *yes* if there is a path from s to v in G, and *no* otherwise.

We keep the data structure of Section 3, plus an array R of integers such that $R[v] > 0$ if and only if vertex v is reachable from source s. The operations can be simply implemented as follows:

s_Insert

```
    procedure s_Insert(x, y)
1.  begin
2.      TC_Insert(x, y)
3.      for z = 1 to n do
4.          R[z] ←TC_Query(s,z)
5.  end
```

s_Delete

```
    procedure s_Delete(x, y)
1.  begin
2.      TC_Delete(x, y)
3.      for z = 1 to n do
4.          R[z] ←TC_Query(s,z)
5.  end
```

s_Query

```
    procedure s_Query(v)
1.  begin
2.      return R[v]
3.  end
```

In the following theorem we discuss the running time of the operations.

Theorem 3 *Any* `s_Insert` *and any* `s_Delete` *operation can be performed in* $O(n^{1.575})$ *worst-case time, and any* `s_Query` *can be answered in constant time. The space required is* $O(n^2)$.

Proof. Each single-source update is implemented by one transitive closure update, followed by n transitive closure queries. The bounds follow immediately from Corollary 2. □

5 Conclusions

In this paper we have shown that a surprisingly simple technique for maintaining dynamic matrices of integers, combined with a previous idea of counting paths [13] yields algorithms that, for the first time in the study of fully dynamic transitive closure, break through the $O(n^2)$ barrier on the single-operation complexity of these problems. Our algorithms work on directed acyclic digraphs, and are randomized with one-sided error. After our work, Sankowski [16] has presented subquadratic algorithms for fully dynamic transitive closure on general graphs.

Acknowledgements

We are indebted to Garry Sagert and Mikkel Thorup for enlightening discussions, and to Valerie King and the anonymous referees for many useful comments on this work.

References

[1] A.V. Aho, J.E. Hopcroft, and J.D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison Wesley, 1974.

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

[3] C. Demetrescu and G.F. Italiano. Fully dynamic transitive closure: Breaking through the $O(n^2)$ barrier. In *Proc. of the 41st IEEE Annual Symposium on Foundations of Computer Science (FOCS'00)*, pages 381–389, 2000.

[4] C. Demetrescu and G.F. Italiano. Maintaining dynamic matrices for fully dynamic transitive closure. Manuscript, 2002.

[5] M. Henzinger and V. King. Fully dynamic biconnectivity and transitive closure. In *Proc. 36th IEEE Symposium on Foundations of Computer Science (FOCS'95)*, pages 664–672, 1995.

[6] J. Holm, K. de Lichtenberg, and M. Thorup. Poly-logarithmic deterministic fully-dynamic algorithms for connectivity, minimum spanning tree, 2-edge, and biconnectivity. *Journal of the ACM (JACM)*, 48(4):723–760, July 2001.

[7] X. Huang and V.Y. Pan. Fast rectangular matrix multiplication and applications. *Journal of Complexity*, 14(2):257–299, June 1998.

[8] T. Ibaraki and N. Katoh. On-line computation of transitive closure for graphs. *Information Processing Letters*, 16:95–97, 1983.

[9] G. F. Italiano. Amortized efficiency of a path retrieval data structure. *Theoretical Computer Science*, 48(2–3):273–281, 1986.

[10] G. F. Italiano. Finding paths and deleting edges in directed acyclic graphs. *Information Processing Letters*, 28:5–11, 1988.

[11] S. Khanna, R. Motwani, and R. H. Wilson. On certificates and lookahead on dynamic graph problems. In *Proc. 7th ACM-SIAM Symp. Discrete Algorithms*, pages 222–231, 1996.

[12] V. King. Fully dynamic algorithms for maintaining all-pairs shortest paths and transitive closure in digraphs. In *Proc. 40th IEEE Symposium on Foundations of Computer Science (FOCS'99)*, pages 81–91, 1999.

[13] V. King and G. Sagert. A fully dynamic algorithm for maintaining the transitive closure. In *Proc. 31st ACM Symposium on Theory of Computing (STOC'99)*, pages 492–498, 1999.

[14] J. A. La Poutré and J. van Leeuwen. Maintenance of transitive closure and transitive reduction of graphs. In *Proc. Workshop on Graph-Theoretic Concepts in Computer Science*, pages 106–120. Lecture Notes in Computer Science 314, Springer-Verlag, Berlin, 1988.

[15] I. Munro. Efficient determination of the transitive closure of a directed graph. *Information Processing Letters*, 1(2):56–58, 1971.

[16] P. Sankowski. Dynamic transitive closure via dynamic matrix inverse. In *Proc. of the 45th IEEE Annual Symposium on Foundations of Computer Science (FOCS'04)*, pages 509–517, Rome, Italy, October 17–19 2004.

[17] D. M. Yellin. Speeding up dynamic transitive closure for bounded degree graphs. *Acta Informatica*, 30:369–384, 1993.

[18] U. Zwick. All pairs shortest paths in weighted directed graphs - exact and almost exact algorithms. In *Proc. of the 39th IEEE Annual Symposium on Foundations of Computer Science (FOCS'98)*, pages 310–319, Los Alamitos, CA, November 8–11 1998.