[27] H. Rohnert, A dynamization of the all pairs least cost path problem, *Proc. 2nd Annual Symp. on Theoretical Aspects of Computer Science Lecture Notes in Computer Science*, vol. 182, Springer-Verlag, Berlin, 1985, 279–286.

[28] D. D. Sleator, and R. E. Tarjan, A data structure for dynamic trees, *J. Comput. System Sci.* 24 (1983), 362–381.

[29] P. M. Spira and A. Pan, On finding and updating spanning trees and shortest paths, *SIAM J. Comput.* 4 (1975), 375–380.

[30] R. Tamassia, A dynamic data structure for planar graph embedding, *Proc. 15th Int. Coll. on Automata, Languages and Programming*, *Lecture Notes in Computer Science*, vol. 317, Springer-Verlag, Berlin, 1988, 576–590.

[31] R. E. Tarjan, *Data structures and network algorithms*, CBMS-NSF Regional Conference Series in Applied Mathematics, vol. 44, SIAM, 1983.

[32] R. E. Tarjan, Amortized computational complexity, *SIAM J. Alg. Disc. Meth.* 6 (1985), 306–318.

[33] J. Westbrook, "Algorithms and data structures for dynamic graph problems", Ph. D. Dissertation, Tech. Rep. CS-TR-229-89, Department of Computer Science, Princeton University, 1989.

[34] D. M. Yellin, A dynamic transitive closure algorithm, Research Report, IBM Research Division, T. J. Watson Research Center, 1988.

[35] D. M. Yellin, and R. Strom, INC: a language for incremental computations, *Proc. ACM SIGPLAN '88 Conf. on Programming Language Design and Implementation*, 1988, 115–124.

[13] G. N. Frederickson, Data structures for on-line updating of minimum spanning trees, *SIAM J. Comput.* 14 (1985), 781–798.

[14] G. N. Frederickson and M. A. Srinivas, On-line updating of degree-constrained minimum spanning trees, *Proc. 22nd Annual Allerton Conference on Communication, Control and Computing*, 1984.

[15] M. L. Fredman, and R. E. Tarjan, Fibonacci heaps and their uses in improved network optimization algorithms, *J. Assoc. Comput. Mach.* 34 (1987), 596–615.

[16] M. R. Garey, D. S. Johnson, *Computers and intractability: a guide to the theory of NP-Completeness*, W. H. Freeman, 1979.

[17] D. Harel, On-line maintenance of the connected components of dynamic graphs, Unpublished manuscript, 1982.

[18] N. Horspool, Incremental generation of LR parsers, Technical Report, Department of Computer Science, University of Victoria, 1988.

[19] T. Ibaraki, and N. Katoh, On-line computation of transitive closure for graphs, *Inform. Process. Lett.* 16 (1983), 95–97.

[20] G. F. Italiano, Amortized efficiency of a path retrieval data structure, *Theoret. Comput. Sci.* 48 (1986), 273–281.

[21] G. F. Italiano, Finding paths and deleting edges in directed acyclic graphs, *Inform. Process. Lett.* 28 (1988), 5–11.

[22] H. V. Jagadish, A compression technique to materialize transitive closure, *ACM Trans. on Database Systems*, to appear.

[23] J. A. La Poutré, and J. van Leeuwen, Maintenance of transitive closure and transitive reduction of graphs, *Proc. Workshop on Graph-Theoretic Concepts in Computer Science*, 1988, 106–120.

[24] E. L. Lawler, *Combinatorial optimization: networks and matroids*, Holt, Rinehart and Winston, 1976.

[25] F. P. Preparata, and R. Tamassia, Fully dynamic techniques for point location and transitive closure in planar structures, *Proc. 29th Annual Symp. on Foundations of Computer Science*, 1988, 558–567.

[26] J. H. Reif, A topological approach to dynamic graph connectivity, *Inform. Process. Lett.* 25 (1987), 65–70.

References

[1] R. Agrawal, A. Borgida, and H. V. Jagadish, Efficient management of transitive relationships in large data and knowledge bases, *Proc. ACM-SIGMOD Int. Conf. on Management of Data*, 1989.

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

[3] G. Ausiello, G. F. Italiano, A. Marchetti Spaccamela, U. Nanni, Incremental minimal length paths, *Proc. 1st Annual ACM-SIAM Symp. on Discrete Algorithms*, 1990, 12–21.

[4] G. Ausiello, A. Marchetti Spaccamela, and U. Nanni, Dynamic maintenance of paths and path expressions in graphs, *Proc. 1st Int. Joint Conf. ISSAC 88 (Int. Symp. on Symbolic and Algebraic Computation) and AAECC 6 (6th Int. Conf. on Applied Algebra, Algebraic Algorithms and Error Correcting Codes)*, *Lecture Notes in Computer Science* 358, Springer-Verlag, Berlin, 1989, 1–12.

[5] A. L. Buchsbaum, P. C. Kanellakis, and J. S. Vitter, A data structure for arc insertion and regular path finding, *Proc. 1st Annual ACM-SIAM Symp. on Discrete Algorithms*, 1990, 22–31.

[6] M. Burke, and B. G. Ryder, Incremental iterative data flow analysis algorithms, Technical Report LCSR-TR-96, Department of Computer Science, Rutgers University, 1987.

[7] M. D. Carrol and B. G. Ryder, Incremental data flow analysis via dominator and attribute updates, *Proc. 15th Annual ACM SIGACT-SIGPLAN Symp. on Principles of Programming Languages*, 1988, 274–284.

[8] F. Chin and D. Houk, Algorithms for updating minimum spanning trees, *J. Comput. System. Sci.* 16 (1978), 333-344.

[9] G. Di Battista and R. Tamassia, Incremental planarity testing, *Proc. 30th Annual Symp. on Foundations of Computer Science*, 1989, 436–441.

[10] D. Eppstein, G. F. Italiano, R. Tamassia, R. E. Tarjan, J. Westbrook, M. Yung, Maintenance of a minimum spanning forest in a dynamic planar graph, *Proc. 1st Annual ACM-SIAM Symp. on Discrete Algorithms*, 1990, 1–11.

[11] S. Even, and H. Gazit, Updating distances in dynamic graphs, *Methods of Operations Research* 49 (1985), 371–387.

[12] S. Even, and Y. Shiloach, An on-line edge deletion problem, *J. Assoc. Comput. Mach.* 28 (1981), 1–4.

We conclude this section by mentioning that also theorem 3 can be extended to longest simple paths on directed acyclic graphs.

7. Concluding Remarks

In this paper we have described fast algorithms for maintaining a solution to the All Pairs Minimal Length Paths Problem during insertions of edges. The total time required to perform a sequence of at most $O(n^2)$ edge insertions is $O(n^3 \log n)$, and the space complexity is $O(n^2)$. We gave also several generalization of these results to shortest paths with integer edge costs in $[1 \ldots C]$, and to maximal length paths in directed acyclic graphs.

There are several related problems which seem worth further investigation. First, in our approach questions about the solution are answered quickly while updating the data structure may be expensive. Is there any trade-off between these two kinds of operations? Moreover, given non-integer edge costs, is it possible to do better than previously known algorithms? Finally, the impact of deletions on this problem deserves further investigation.

Acknowledgments

We are indebted to Dany Breslauer, Kunsoo Park, Neil Sarnak and Bill Schilit who provided many valuable comments in reading earlier versions of this paper. The second author is also grateful to Zvi Galil and Moti Yung for stimulating discussions on this topic.

$$\sum_{j=2}^{n/3} \left(\frac{n}{3}+1\right)\left(\frac{2}{3}n-j+1\right)C,$$

which is $\Omega(Cn^3)$.

We notice that the worst-case per operation time complexity of our data structure remains $O(n^2)$. As the above example shows, this is also the total number of updates that may be required in the matrix D during either the insertion of a new edge or an edge cost decrease. For the special case of planar graphs, we achieve a $\Theta(Cn^3)$ bound for our problem.

6. On-Line maintenance of longest paths in directed acyclic graphs

A modification of our algorithms is able to perform *maxpath* operations (i.e., queries about longest simple paths) on directed acyclic graphs where the insertions of new edges do not introduce cycles. This is not a significant restriction, since the longest path problem is known to be NP-complete for arbitrary graphs [16], while a polynomial time algorithm exists for directed acyclic graphs [24].

We now sketch how to handle longest paths in directed acyclic graphs with unit edge costs. Following the ideas outlined in the previous sections, we maintain the descendants and ancestors of each vertex as trees of forward and backward *maximal* length paths. The operation *length* can be performed as before, while *maxpath* can be performed as the *minpath* operation given above. In addition, simultaneous queries about minimal and maximal length paths can be supported by maintaining both the trees of minimal and maximal length, thus only at the price of doubling the space required per vertex.

In case of maximal length simple path in directed acyclic graphs, equalities analogous to (1), (2) and (3) hold. In fact denoting by $\delta(x,y)$ the length of a maximal path from x to y ($\delta(x,y) = -\infty$ if there is no path from x to y), and by $\delta_{new}(x,y)$ the length of a maximal path from x to y after the insertion of edge (i,j), then the following equalities hold.

$$\delta(x,x) = 0, \forall x,y \in V; \tag{15}$$

$$\delta(x,y) = \max_{u \in V}\{\delta(x,u) + \delta(u,y)\}, \forall x,y \in V; \tag{16}$$

$$\delta_{new}(x,y) = \max\{\delta(x,y), \delta(x,i) + 1 + \delta(j,y)\}, \forall x,y \in V; \tag{17}$$

It is important to underline that equality (16) does not hold for cyclic graphs. Therefore results analogous to the ones shown in the previous sections are extendable only to directed acyclic graphs. In particular, Theorem 2 can be extended as follows.

Theorem 4 *Given a directed acyclic graph G, provided that new edge insertions do not introduce any cycles, there exists a data structure which supports each* length *operation in $O(1)$ time and each* maxpath *operation in $O(k)$ time, where k is the length of the traced (maximal) path. The total time required to maintain the data structure during the insertion of at most $O(n^2)$ edges is $O(n^3 \log n)$, where n is the number of vertices in the graph. The space complexity of the data structure is $O(n^2)$.* □

In order to show that the credits given are enough to pay for the work done for malignant pairs, let us consider any update operation σ which either inserts an edge (i,j) or decreases the cost of an edge (i,j). Let $\langle x,y\rangle$ be any malignant pair during σ. Denote by π_x the path in $ANC(i)$ from vertex x to vertex i and by π_y the path in $DESC(j)$ from j to y. Let $d_x \geq 0$ and $d_y \geq 0$ be respectively the number of edges of π_x and π_y. Note that $d(x,y) \leq C(d_x+d_y+1)$. Because of lemma 4, for any vertex $u \neq x$ in π_x and $v \neq y$ in π_y the pairs $\langle u,y\rangle$ and $\langle x,v\rangle$ are benign during σ. The total number of credits given to vertex x during operation σ because of vertices in π_y is

$$4 \sum_{s\in(\pi_y-\{y\})} \left\lceil \frac{Cn}{d(x,s)} \right\rceil \geq 4 \sum_{h=d_x+1}^{d_x+d_y} \left\lceil \frac{n}{h} \right\rceil \geq 4n \log_e \left(\frac{d_x+d_y+1}{d_x+1} \right).$$

Similarly the total number of credits given to vertex y during operation σ because of vertices in π_x is

$$4 \sum_{s\in(\pi_x-\{x\})} \left\lceil \frac{Cn}{d(s,y)} \right\rceil \geq 4 \sum_{h=d_y+1}^{d_x+d_y} \left\lceil \frac{n}{h} \right\rceil \geq 4n \log_e \left(\frac{d_x+d_y+1}{d_y+1} \right).$$

By applying the same argument in the proof of theorem 2, at least $2n$ credits are given to both x and y during operation σ. These credits are sufficient to pay for the work of all malignant pairs during σ of the form $\langle x,*\rangle$ and $\langle *,y\rangle$. This gives the theorem. $\Box$

The bound given in theorem 3 is only a factor of $\log(nC)$ away from the best possible bound for the problem of maintaining on-line a solution to the All Pairs Shortest Paths Problem with integer edge costs in $[1 \ldots C]$. In fact we show now that there exist sequences of *add* and *decrease* operations that force as many as $\Omega(Cn^3)$ changes in the distance matrix D. For the sake of simplicity assume that C is an integer (otherwise replace it with $\lfloor C \rfloor$). Consider once again the graph $G=(V,E)$ given in figure 2, with vertex set

$$V = \{s_1, s_2, \ldots, s_{\frac{n}{3}}, x_1, x_2, \ldots, x_{\frac{n}{3}}, t_1, t_2, \ldots, t_{\frac{n}{3}}\},$$

where n is a multiple of 3, and edges (s_i, x_1), $i = 1, \ldots, \frac{n}{3}$, (x_i, x_{i+1}), $i = 2, \ldots, \frac{n}{3}-1$, and $(x_{\frac{n}{3}}, t_i)$, $i = 1,2,\ldots,\frac{n}{3}$, each of which has cost C. For $1 \leq j \leq \frac{n}{3}$ denote by μ_j operation $add(x_1, x_j, C)$. Similarly, for $1 \leq j \leq \frac{n}{3}$ denote by ν_j operation $decrease(x_1, x_j, 1)$. Define now a packet of operations λ_j, $1 \leq j \leq \frac{n}{3}$, in terms of the two operations μ_j and ν_j as follows:

$$\lambda_j = \mu_j \underbrace{\nu_j \nu_j \ldots \nu_j}_{C-1}.$$

Perform the sequence of operations $\Lambda = \lambda_2, \lambda_3, \ldots, \lambda_{\frac{n}{3}}$. Notice that the total number of operations in Λ is $O(Cn)$. Each packet λ_j forces $(n/3+1)(2n/3-j+1)C$ changes in the matrix D. After performing all the operations in the sequence Λ, the total number of updates required in D is therefore

$$d_{new}(x,y) = \min\{d_{old}(x,y), d_{old}(x,i) + w(i,j) + d_{old}(j,y)\}, \qquad \forall x,y \in V. \tag{14}$$

The correctness is now an extension of theorem 1. The total time required to maintain the data structure during any sequence $\mathcal{S}$ of (at most $O(Cn^2)$) edge insertions and edge cost decreases is $O(Cn^3 \log(nC))$, as the following theorem shows.

Theorem 3 *There exists a data structure which supports each* length *operation in $O(1)$ time and each* minpath *operation in $O(k)$ time, where k is the length of the traced (minimal) path. The total time required to maintain the data structure during the insertion of at most $O(n^2)$ edges and during at most $O(Cn^2)$* decrease *operations is $O(Cn^3 \log(nC))$, where n is the number of vertices in the graph. The space complexity of the data structure is $O(n^2)$.*

Proof : We proceed along the lines of the proof of theorem 2. We notice that during any update operation σ (either an edge insertion or an edge cost decrease) it is still possible to define benign and malignant pairs. There is only a change in terminology, due to the fact that now the cost of the same edge (i,j) can be decreased many times by different *decrease* operations. Therefore, while for unit edge costs only one operation in a given sequence could deal with a given edge (i,j), now there can be many operations in a given sequence dealing with edge (i,j) (i.e., either inserting or decreasing the cost of edge (i,j)). As a consequence, we now define benign and malignant pairs *during the update operation σ*. In a similar vein, it is possible to extend fact 1 and lemma 4 to the new setting of operations. Because of fact 1, the total time required to maintain the data structure during any sequence $\mathcal{S}$ of operations of the extended repertoire is bounded by the total number of benign pairs plus the total number of malignant pairs during the edge insertions and edge cost decreases of $\mathcal{S}$.

We start by bounding the total time spent because of benign pairs. We observe that for each pair $\langle x,y \rangle$ that is benign during an update operation σ, the entry (x,y) of the distance matrix D is decreased at least by 1 during σ. Since for each edge (i,j) we can have one edge insertion and at most C edge cost decreases, there can be at most $O(Cn)$ updates of each entry (x,y) of the matrix D during any sequence of operations. As a result, the total time spent because of benign pairs is $O(Cn^3)$. In order to bound the total time spent because of malignant pairs, we use the following credit policy:

When performing an operation σ that either inserts or decreases the cost of an edge (i,j), for each benign pair $\langle x,y \rangle$ during σ give $4\left\lceil \frac{Cn}{d(x,y)} \right\rceil$ credits both to x and y, where $d(x,y) \geq 1$ is the minimum distance from x to y after operation σ.

Since we give credits to a pair of vertices if and only if it is a benign pair, their distance strictly decreases by at least 1. Furthermore, the distance between any two vertices for which there is a path is bounded above by Cn. As a result, the total number of credits given during any sequence $\mathcal{S}$ of edge insertions and edge cost decreases is

$$O\left(\sum_{x \in V} \sum_{y \in V} \sum_{d=1}^{Cn} \left\lceil \frac{Cn}{d} \right\rceil\right) \leq O(Cn^3 \log(nC)).$$

where n is a multiple of 3, and edges (s_i, x_1), $i = 1,2,\ldots,\frac{n}{3}$, (x_i, x_{i+1}), $i = 2,\ldots,\frac{n}{3}-1$, and $(x_{\frac{n}{3}}, t_i)$, $i = 1,2,\ldots,\frac{n}{3}$. If now the $(n/3-1)$ edges

$$(x_1, x_i), \quad i = 2,\ldots,\frac{n}{3},$$

are inserted into G by increasing i, the changes in the distance matrix D can be computed as follows. The insertion of edge (x_1, x_i), $2 \leq i \leq \frac{n}{3}$, creates new shorter paths from vertex s_j, $1 \leq j \leq \frac{n}{3}$, and from vertex x_1 to vertices x_h, $i \leq h \leq \frac{n}{3}$, and t_k, $1 \leq k \leq \frac{n}{3}$. This gives a total of $(n/3+1)(2n/3-i+1)$ changes required in the distance matrix D. After inserting all the edges, the total number of updates required in D is therefore

$$\sum_{i=2}^{n/3} \left(\frac{n}{3}+1\right)\left(\frac{2}{3}n-i+1\right),$$

which is $\Omega(n^3)$.

We notice that the worst-case per operation time complexity of our data structure is $O(n^2)$. As the above example shows, this is also the total number of updates that may be required in the matrix D during the insertion of a new edge. For the special case of planar graphs, we achieve a $\Theta(n^3)$ bound for our problem.

5. Integer edge costs

The data structure presented in section 3 can be adapted to deal with integer edge costs in the range $[1 \ldots C]$. The total time required to maintain on-line a solution to the All Pairs Shortest Paths Problem during (at most $O(n^2)$) insertions of edges and (at most $O(Cn^2)$) edge cost decreases is $O(Cn^3)\log(nC))$ in the worst-case. The space required by the data structure is $O(n^2)$. In this case too, there are sequences of edge insertions that force $\Omega(Cn^3)$ changes in the distance matrix D. Once again, our algorithm is only a logarithmic factor away from the best possible bound.

Denoting by $w(i,j)$ the cost of edge (i,j), we extend the repertoire of allowable operations as follows. Operations *length* and *minpath* are unchanged. Operation *add*(i,j,w) inserts the edge (i,j) with integer cost w, $1 \leq w \leq C$. We consider also a *decrease*(i,j,Δ) operation that decreases the cost $w(i,j)$ of edge (i,j) by the positive integer Δ $(0 < \Delta < w(i,j))$. Notice that *add*(i,j,w) is a special case of a *decrease* operation; indeed, *add*(i,j,w) can be thought of as decreasing the cost of edge (i,j) from $+\infty$ to w. As a result, if we take care of edge costs this extended set of operations can be supported by means of the same data structures and procedures presented in section 3. The only difference is that now trees of forward and backward shortest paths are maintained throughout any sequence of operations of the extended repertoire. Obviously we must modify the lines 4. and 9. of the procedure UpdateForward by substituting $w(i,j)$ in place of 1, where $w(i,j)$ is the new cost of edge (i,j) after either the insertion of (i,j) or a *decrease* operation on (i,j). In order to prove the correctness we observe that equalities (1) and (2) still hold and that equality (3) must be replaced by the following, which takes care of the new cost $w(i,j)$:

Figure 2:

$$4 \sum_{h=d_x+1}^{d_x+d_y} \left\lceil \frac{n}{h} \right\rceil \geq 4n \log_e \left(\frac{d_x + d_y + 1}{d_x + 1} \right).$$

Similarly, the total number of credits given to vertex y because of vertices in π_x is

$$4 \sum_{h=d_y+1}^{d_x+d_y} \left\lceil \frac{n}{h} \right\rceil \geq 4n \log_e \left(\frac{d_x + d_y + 1}{d_y + 1} \right).$$

As a consequence, the number of credits given to both x and y during the insertion of edge (i, j) is bounded below by

$$4n \log_e \frac{(d_x + d_y + 1)^2}{(d_x + 1)(d_y + 1)}.$$

Since for d_x and d_y positive integers such that $d_x + d_y > 0$ we have that

$$\frac{(d_x + d_y + 1)^2}{(d_x + 1)(d_y + 1)} \geq 2$$

then the credits given to x and y are bounded below by $4n \log_e 2 > 2n$. The credits given to both x and y can henceforth pay for the work due to all the malignant pairs of the form $\langle x, * \rangle$ and $\langle *, y \rangle$ which are at most $2n$. Since this argument can be repeated for any malignant pair, it proves that $\Phi_{i,j} \geq |M_{i,j}|$. Namely, all the credits given during operation $add(i, j)$ are able to pay for the overall work due to malignant pairs during the same operation.

To conclude our proof, we notice that because of equations (11), (12) and (13), the total time spent while updating the data structure during any sequence $\mathcal{S}$ of edge insertions is

$$\begin{aligned} O\left(\sum_{(i,j) \in \mathcal{S}} (|B_{i,j}| + |M_{i,j}|) \right) &= O\left(\sum_{(i,j) \in \mathcal{S}} |B_{i,j}| \right) + O\left(\sum_{(i,j) \in \mathcal{S}} |M_{i,j}| \right) \\ &\leq O(n^3) + O\left(\sum_{(i,j) \in \mathcal{S}} \Phi_{i,j} \right) \\ &\leq O(n^3 \log n). \ \square \end{aligned}$$

The bound given in theorem 2 is only a factor of $\log n$ away from the best possible bound for the problem of maintaining on-line a solution to the All Pairs Minimal Length Paths Problem. In fact we show now that there exist sequences of *add* operations that force as many as $\Omega(n^3)$ changes in the distance matrix D. Consider the graph $G = (V, E)$ given in figure 2, with vertex set

$$V = \{s_1, s_2, \ldots, s_{\frac{n}{3}}, x_1, x_2, \ldots, x_{\frac{n}{3}}, t_1, t_2, \ldots, t_{\frac{n}{3}}\},$$

The space required is $O(n^2)$ since three $n \times n$ matrices and $2n$ trees, each of size at most n, are used. The time required by procedure *length*(x,y) is $O(1)$ since it accesses entry (x,y) of the distance matrix D. As previously noticed, procedure *minpath*(x,y) takes $O(k)$ to return a minimal path from x to y (if one exists), where k is the length of the achieved minimal path.

We have now to compute the total time required to maintain the data structure during any sequence $\mathcal{S}$ of edge insertions. Because of fact 1, this time is bounded by the total number of benign pairs plus the total number of malignant pairs during the insertions of edges:

$$O\left(\sum_{(i,j)\in\mathcal{S}} (|B_{i,j}| + |M_{i,j}|)\right). \tag{11}$$

During the insertion of edge (i,j), for each pair $\langle x,y\rangle$ in $B_{i,j}$ the entry (x,y) of the distance matrix D is either changed from $+\infty$ to a positive integer value $\ell \leq n-1$ or decreased at least by 1. Since each entry of D can hold only the values $1,2,\ldots n-1,+\infty$ and there are at most $O(n^2)$ different entries, the total time spent because of benign pairs is

$$\sum_{(i,j)\in\mathcal{S}} |B_{i,j}| \leq O(n^3). \tag{12}$$

In order to bound the total time spent because of malignant pairs, we use an amortization argument based upon the credit technique by Tarjan [32]. Our credit policy is the following:

When inserting edge (i,j), for each benign pair $\langle x,y\rangle$ in $B_{i,j}$ give $4\left\lceil \frac{n}{d(x,y)} \right\rceil$ credits both to x and y, where $d(x,y) \geq 1$ is the minimum distance from x to y after the insertion of (i,j).

Denote by $\Phi_{i,j}$ the number of credits given during the insertion of edge (i,j). Since we give credits to a pair of vertices if and only if it is a benign pair, their distance strictly decreases. Therefore, the total number of credits given during any sequence $\mathcal{S}$ of edge insertions is

$$\sum_{(i,j)\in\mathcal{S}} \Phi_{i,j} \leq O\left(\sum_{x\in V}\sum_{y\in V}\sum_{d=1}^{n-1}\left\lceil\frac{n}{d}\right\rceil\right) \leq O(n^3 \log n). \tag{13}$$

To complete our proof, we have to show that the credits given are enough to pay for the work done because of malignant pairs. We will show something stronger, namely that all the credits $\Phi_{i,j}$ given to benign pairs while inserting edge (i,j) are enough to pay for the work done because of malignant pairs during the insertion of the same edge (i,j).

Consider any malignant pair $\langle x,y\rangle$ in $M_{i,j}$. Denote by π_x the path in $ANC(i)$ from x to i and by π_y the path in $DESC(j)$ from j to y. Let $d_x \geq 0$ and $d_y \geq 0$ be respectively the lengths of π_x and π_y. Notice that the case $d_x = d_y = 0$ cannot happen, since this would imply $\langle x,y\rangle = \langle i,j\rangle$ and therefore $\langle x,y\rangle \notin M_{i,j}$. As a result, $d_x + d_y > 0$. Because of lemma 4, for any vertex $u \neq x$ in π_x and $v \neq y$ in π_y, the pairs $\langle u,y\rangle$ and $\langle x,v\rangle$ are benign for (i,j). The total number of credits given to vertex x because of vertices in π_y is

Before analyzing the time and space complexity of the data structure, we need some new terminology. During the insertion of an edge (i,j) in the graph G, for any x ancestor of i let us denote by $T(x)$ the set of vertices in *DESC*(j) visited by *UpdateForward* while updating the tree *DESC*(x). Note that as a special case, $T(x)$ might be empty. We say that a pair of vertices $\langle x,y\rangle$ is *benign for edge* (i,j) if x is an ancestor of i, y is in $T(x)$, and before the insertion of (i,j) $D[x,y] > D[x,i]+1+D[j,y]$ (i.e., (i,j) introduces a new shorter path from x to y). Furthermore, we say that a pair of vertices $\langle x,y\rangle$ is *malignant for edge* (i,j) if x is an ancestor of i, y is in $T(x)$, y is visited while updating *DESC*(x) and before the insertion of (i,j) $D[x,y] \leq D[x,i]+1+D[j,y]$ (i.e., (i,j) introduces no new shorter path from x to y). Denote by $B_{i,j}$ the set of pairs which are benign for (i,j) and by $M_{i,j}$ the set of pairs which are malignant for (i,j). Since edge (i,j) can be inserted at most once in the graph, we have by definition that $\langle i,j\rangle \notin M_{i,j}$, and $\langle i,j\rangle \in B_{i,j}$. The following fact is a consequence of the definition of benign and malignant pairs:

Fact 1 *The total time required by procedure* add *to insert an edge* (i,j) *is* $O(|B_{i,j}|+|M_{i,j}|)$.

We now characterize a property of benign and malignant pairs.

Lemma 4 *For any pair* $\langle x,y\rangle$ *in* $M_{i,j}$*, denote by* π_x *the path from* x *to* i *in* ANC(i) *and by* π_y *the path from* j *to* y *in* DESC(j)*. For any vertex* $u \neq x$ *in* π_x *and for any vertex* $v \neq y$ *in* π_y *then the pairs* $\langle u,y\rangle$ *and* $\langle x,v\rangle$ *are benign for* (i,j)*.*

Proof : We proceed by contradiction. Consider first a malignant pair $\langle x,y\rangle$ and assume that there exists a vertex $u \neq x$ in π_x such that also the pair $\langle u,y\rangle$ is malignant. This means that vertices in the subtree rooted at y in *DESC*(j) are not included in the pruned tree N given to the ancestors of u on line 11. of procedure *UpdateForward*. Since $T(x) \subseteq N$ because of line 11. in procedure *UpdateForward* then $y \notin T(x)$, which is clearly a contradiction. To prove the second part of the lemma, consider a malignant pair $\langle x,y\rangle$ and assume that there exists a vertex $v \neq y$ in π_y such that also the pair $\langle x,v\rangle$ is malignant. This means that the vertices in the subtree rooted at v in *DESC*(j) are not visited while updating *DESC*(x), because of line 4. in procedure *UpdateForward*. Therefore $y \notin T(x)$, again a contradiction. □

The following theorem proves the correctness and analyze the complexity of our approach.

Theorem 2 *There exists a data structure that correctly supports each* length *operation in* $O(1)$ *time and each* minpath *operation in* $O(k)$ *time, where* k *is the length of the traced (minimal) path. The total time required to maintain the data structure during the insertion of at most* $O(n^2)$ *edges is* $O(n^3 \log n)$*, where* n *is the number of vertices in the graph. The space complexity of the data structure is* $O(n^2)$*.*

Proof : The correctness of *length* and *add* operations derives from theorem 1, while the data structure correctly supports *minpath* operations because of theorem 1 and lemma 1.

Suppose now there exists a pair of vertices $x, y \in V$ such that $D_{new}[x,y] > d_{new}(x,y)$. This can happen if and only if x is an ancestor of i, y is a descendant of j and y was not inserted in the queue Q during the execution of the procedure *UpdateForward*$(x,i,j,DESC(j))$. As a consequence, there exist two vertices u and v such that x is contained in the subtree of $ANC(i)$ rooted at u and y is contained in the subtree of $DESC(j)$ rooted at v, and for which procedure *UpdateForward*$(u,i,j,DESC(j))$ pruned at v the tree considered because the condition on line 4. was not satisfied. Hence, $D[u,v] < D[u,i] + 1 + D[j,v]$ or, what is the same for the induction step, $d(u,v) < d(u,i) + 1 + d(j,v)$. Using this inequality yields

$$\begin{aligned} d(x,y) &\leq d(x,u) + d(u,v) + d(v,y) \\ &< d(x,u) + d(u,i) + 1 + d(j,v) + d(v,y). \end{aligned}$$

Because of the induction step

$$d(j,v) + d(v,y) = D[j,v] + D[v,y]$$

and by lemma 2

$$D[j,v] + D[v,y] = D[j,y].$$

Applying again the induction step to the last equality, we derive

$$d(j,v) + d(v,y) = d(j,y).$$

Similarly, we obtain

$$d(x,u) + d(u,i) = d(x,i).$$

Therefore,

$$d(x,y) < d(x,i) + 1 + d(j,y).$$

Applying equality (3), we get

$$d_{new}(x,y) = \min\{d(x,y), d(x,i) + 1 + d(j,y)\} = d(x,y).$$

Thus,

$$D_{new}[x,y] > d_{new}(x,y) = d(x,y) = D[x,y].$$

This is clearly a contradiction since entries in the matrix D can only be decreased by procedure *UpdateForward*.

Since it cannot be either $D_{new}[x,y] < d_{new}(x,y)$ or $D_{new}[x,y] > d_{new}(x,y)$ for any pair of vertices, equality (10) must hold. □

Lemma 3 *Assume that* $D[x,y] = d(x,y)$ *for any pair of vertices* x *and* y *in the graph before inserting an edge from* i *to* j. *Then given any pair of vertices* u *and* v *such that* $d(u,v) \leq d(u,i) + 1 + d(j,v)$, *for any vertex* y *contained in the subtree of* DESC(j) *rooted at* v *and for any vertex* x *contained in the subtree of* ANC(i) *rooted at* u, *the following two inequalities hold.*

$$d(u,y) \leq d(u,i) + 1 + d(j,y) \tag{5}$$

$$d(x,v) \leq d(x,i) + 1 + d(j,v) \tag{6}$$

Proof : We prove only inequality (5). The proof for inequality (6) is completely analogous and therefore it has been omitted. Consider any vertex y in the subtree of $DESC(j)$ rooted at v. Due to equality (2) and the hypothesis

$$d(u,y) \leq d(u,v) + d(v,y) \leq d(u,i) + 1 + d(j,v) + d(v,y). \tag{7}$$

Since y is in the subtree of $DESC(j)$ rooted at v, then by lemma 2 $D[j,y] = D[j,v] + D[v,y]$. Since $D[x,y] = d(x,y)$ for any pair of vertices x and y in the graph, this is equivalent to say

$$d(j,y) = d(j,v) + d(v,y). \tag{8}$$

Combining (7) and (8) gives inequality (5). □

The correctness of our approach hinges on the following theorem.

Theorem 1 *After each operation, for the data structure the following equality holds*

$$D[x,y] = d(x,y) \qquad \forall x, y \in V. \tag{9}$$

That is, given any two vertices x *and* y, *the entry* (x,y) *of the matrix* D *correctly gives the length of the minimal path from* x *to* y.

Proof : We proceed by induction on the number of operations performed. Since *minpath* and *length* operations do not modify the data structure, we consider only *add* operations.

The basis of the induction is true since at the beginning the graph contains no edges, $D[x,y] = d(x,y) = +\infty$ for any two vertices x and y ($x \neq y$), and $D[x,x] = d(x,x) = 0$ for each vertex x.

Suppose equality (9) holds before the insertion of an edge (i,j). Let us denote by $D_{new}[x,y]$ and by $d_{new}(x,y)$ respectively the values of $D[x,y]$ and $d(x,y)$ after inserting the edge (i,j). We want to prove that given $D[x,y] = d(x,y)$

$$D_{new}[x,y] = d_{new}(x,y) \qquad \forall x, y \in V. \tag{10}$$

The case $D_{new}[x,y] < d_{new}(x,y)$ cannot happen since by lines 4. and 9. in procedure *UpdateForward* and by the induction step :

$$\begin{aligned} D_{new}[x,y] &\geq \min\{D[x,y], D[x,i] + 1 + D[j,y]\} \\ &= \min\{d(x,y), d(x,i) + 1 + d(j,y)\} \\ &= d_{new}(x,y). \end{aligned}$$

Lemma 1 *After each operation and for any pair of vertices x and y in the graph, $D[x,y]$ is equal to the depth of vertex y in the tree* DESC(x)*, if y is a descendant of x. Otherwise, y is not in* DESC(x) *and $D[x,y] = +\infty$.*

Proof : By induction on the number of operations performed. At the beginning, the lemma is true since $D[x,y] = +\infty$ for $x \neq y$, $D[x,x] = 0$ and $DESC(x)$= $\{x\}$, for any $x \in V$. Assume now that the lemma holds before the i-th operation. If the i-th operation is either a *length* or a *minpath* operation, then the lemma still holds afterwards because these two operations do not modify the data structure.

If the i-th operation is $add(i,j)$ then we show that, after performing this operation, for any vertex x the x-th row of matrix D still satisfies the lemma. We proceed now by using a second induction, this time on the number of elementary operations performed onto $DESC(x)$. The basis of the second induction is satisfied since if j is inserted into $DESC(x)$, it is inserted on line 6. of procedure *UpdateForward* as a child of i. This implies that j is correctly inserted at depth $D[x,j] = D[x,i] + 1$ in $DESC(x)$ because of the first induction step. Assume now that all the insertions in the tree $DESC(x)$ before inserting a new vertex $y \neq j$ in it satisfy the lemma. When y is to be inserted in $DESC(x)$ it is inserted as a child of v, with v being the parent of y in the tree $DESC(j)$ (line 7. of procedure UpdateForward). Furthermore, v must have been inserted in $DESC(x)$ by procedure *UpdateForward* (otherwise vertex y will not be considered for possible insertion in $DESC(x)$ because of line 4.). As a consequence of the second induction step $D[x,v] = D[x,i] + 1 + D[j,v]$ is equal to the depth of v in the tree $DESC(x)$. Since $D[x,y]$ will be set to $D[x,i] + 1 + D[j,y]$ (line 9.) and $D[j,y] = D[j,v] + 1$ because of the first induction step on tree $DESC(j)$, then $D[x,y] = D[x,i] + 1 + D[j,v] + 1$ will be equal to the depth of y in the tree $DESC(x)$. Since for vertices y not in $DESC(x)$ after the update the entry $D[x,y]$ is not changed, this concludes the two induction steps and gives the lemma. □

Lemma 1 shows that if the distance matrix D is correctly updated, then procedure *minpath* correctly returns a path with minimal number of edges. We recall that we denote by $d(x,y)$ the length of a minimal path between vertices x and y in the graph.

Lemma 2 *Assume that $D[x,y] = d(x,y)$ for any pair of vertices x and y in the graph. Then for any pair of vertices j and v such that v is reachable from j, and for any vertex y contained in the subtree of* DESC(j) *rooted at v, after each operation the following equality holds.*

$$D[j,y] = D[j,v] + D[v,y]. \tag{4}$$

Proof : We proceed by contradiction. If $D[j,y] > D[j,v] + D[v,y]$, then since $d(x,y) = D[x,y]$ for any pair of vertices x and y, we have $d(j,y) > d(j,v) + d(v,y)$ which contradicts equality (2). If $D[j,y] < D[j,v] + D[v,y]$, then $d(v,y) = D[v,y] > D[j,y] - D[j,v]$. Since by lemma 1 and by the hypothesis $D[x,y] = d(x,y)$ the path from v to y in $DESC(j)$ is a path in the graph of length $D[j,y] - D[j,v]$, this contradicts the fact that $d(v,y)$ is the length of a minimal path from v to y. Since it cannot be neither $D[j,y] > D[j,v] + D[v,y]$ nor $D[j,y] < D[j,v] + D[v,y]$, then $D[j,y] = D[j,v] + D[v,y]$. □

```
        procedure UpdateForward (vertices : x,i,j ; tree : T) ;
        vertices : y,w; queue : Q; tree : N;
        begin
1.        Q := {j}; N := ∅;
2.        while Q ≠ ∅ do begin
3.          remove an item y from the queue Q;
4.          if D[x,i] + 1 + D[j,y] < D[x,y] then begin
5.                if y = j then begin
6.                      insert j in DESC(x) as a child of i; N := {j};
                        update FORWARD[x,j] accordingly ;
                    end
                  else begin
7.                      insert y in DESC(x) with the same parent it has in DESC(j);
8.                      insert y in N with the same parent it has in DESC(j);
                        update FORWARD[x,y] accordingly ;
                    end
9.                D[x,y] := D[x,i] + 1 + D[j,y];
10.               for each w child of y in T
                    do insert w into the queue Q;
            end;
          end;
11.       if N ≠ ∅ then for each edge (x,u) in ANC(i)
            do UpdateForward(u,i,j,N) ;
        end;
```

The updates of the trees of minimal backward paths can be carried out in a completely analogous fashion. In particular, the procedure *UpdateBackward* that updates the trees $ANC(y)$ for each y descendant of vertex j, can be obtained by interchanging the roles of the trees *DESC* and *ANC* and of the matrices *FORWARD* and *BACKWARD* and by neglecting line 9. The insertion of a new edge (i,j) can be performed by executing first *UpdateForward*$(i,i,j,DESC(j))$ and then *UpdateBackward*$(j,i,j,DESC(j))$. This needs some care, however. In fact both *UpdateForward* and *UpdateBackward* are supposed to run on the data structure before it is updated because of the insertion of (i,j). This can be taken care of by simply executing the two procedures without updating explicitly the data structure; rather, all the updates to be performed are stored in a queue and carried out after the execution of *UpdateForward*$(i,i,j,DESC(j))$ and of *UpdateBackward*$(j,i,j,ANC(i))$. This yields the same time complexity as if the updates were to be performed immediately.

4. Correctness and Time Complexity

In this section, we analyze the time complexity of the data structure and prove its correctness. Before stating our results, we need few technical lemmas.

Equation (3) states that the insertion of an edge (i,j) can introduce new shorter paths only from ancestors x of vertex i (for which $d(x,i) < +\infty$) to descendants y of vertex j (for which $d(j,y) < +\infty$). In this case, the trees of minimal forward paths might have to be updated for vertices in $ANC(i)$ and the trees of minimal backward paths might have to be updated for vertices in $DESC(j)$. We now describe how to update trees of minimal forward paths because of the insertion of an edge (i,j). A similar argument can be applied to the trees of minimal backward paths. In the update of the trees of minimal forward paths, a crucial role is played by the two trees $ANC(i)$ and $DESC(j)$: in order to guarantee a correct update of the forward trees, it is sufficient that for each vertex x in $ANC(i)$ the tree $DESC(x)$ is updated using $DESC(j)$.

However, to improve the time required for updating the data structure we need a more sophisticated approach. Our algorithm can be described as follows. We start from vertex i and update both the i-th row of the matrix D and the tree $DESC(i)$ by visiting the tree $DESC(j)$. The details of this update will be given later on. At the end of the update of $DESC(i)$, a tree T contained in $DESC(j)$ is defined which includes only vertices v for which the distance from i to v has decreased after the insertion of edge (i,j). This tree T is given to the children of i in $ANC(i)$ which will perform the updates of their forward trees of minimal paths in a recursive fashion. In other words, each vertex x in $ANC(i)$ starting from vertex i receives a tree T that is a (possibly pruned) copy of $DESC(j)$. After receiving T, vertex x updates $DESC(x)$ using T. At the same time, x performs a further pruning of T and passes it to its children in $ANC(i)$ (i.e., vertices u for which there is an edge (u,x) in the graph).

The update of $DESC(x)$ for each vertex x in $ANC(i)$ is carried out as follows. Vertex x visits the tree T it receives, starting from the root of T. The subsequent update of $DESC(x)$ and the pruning of T are subject to the following two rules.

(i) When a vertex y in T for which $D[x,i]+1+D[j,y] < D[x,y]$ is reached, then a shorter path from x to y has been found. In this case, vertex y is inserted into $DESC(x)$ either as a child of i if $y = j$ or as a child of the same parent it has in $DESC(j)$ otherwise. By inserting a vertex y into a tree, we mean that any previous occurrence of y in the tree is implicitly deleted. With the help of the $FORWARD$ and $BACKWARD$ matrices this task can be accomplished in constant time and therefore will not affect the asymptotic time spent during the update. At the end of this step, all the children of y in T are examined in a recursive fashion.

(ii) When a vertex y in T for which $D[x,i] + 1 + D[j,y] \geq D[x,y]$ is reached, then no shorter path from x to y was introduced by the insertion of the edge (i,j). In this case no update is performed in $DESC(x)$ and the tree T is pruned by cutting the subtree rooted at y. In fact, for each vertex y' in such a subtree the new arc (i,j) does not decrease the distance from x to y'.

After vertex x finishes to update $DESC(x)$, the pruned copy of T is passed to the children of x in $ANC(i)$. This can be implemented as the following pseudo-code shows:

```
function length (vertices : x, y) : integer;
begin
   return(D[x, y])
end;
```

A minimal path from u to v can be returned by first examining the entry $FORWARD[u, v]$. If it is *null*, then there is no path from u to v. Otherwise $FORWARD[u, v]$ allows us to locate v in $DESC(u)$, the tree of forward minimal paths rooted at u. If for each vertex in such trees reverse pointers to the parent are maintained, a bottom-up traversal from v to the root u in $DESC(u)$ takes at most $O(k)$ time to return a minimal length path from u to v in G, where $k \leq n$ is the length of the achieved path. A path $e_1, e_2, \ldots, e_k$ is returned as the sorted list of vertices $t(e_1), t(e_2), \ldots, t(e_k)$; if there is no path from x to y then the empty list is returned.

```
function minpath (vertices : x, y) : list of vertices ;
pointer to vertices : p ;
list of vertices : π ;
begin
   π := ∅ ;
   p := FORWARD[x, y] ;
   while p ≠ null do begin
      insert the vertex pointed to by p at the beginning of π ;
      p := parent(p) ;
   end ;
   return(π) ;
end ;
```

We now show how to update the distance matrix D, the two matrices $FORWARD$ and $BACKWARD$, and the $2n$ trees $DESC(v)$ and $ANC(v)$, after the insertion of a new edge (i, j). Denote by $d(x, y)$ the length of a minimal path from vertex x to vertex y in the graph G (hereafter referred to also as minimal length function). The following properties hold:

$$d(x, x) = 0, \forall x \in V. \tag{1}$$

$$d(x, y) = \min_{u \in V} \{d(x, u) + d(u, y)\}, \forall x, y \in V. \tag{2}$$

Furthermore, if $d_{new}(x, y)$ denotes the minimal length function after the insertion of an edge (i, j) and $d_{old}(x, y)$ denotes the minimal length function before the insertion of (i, j), then the following property must be true:

$$d_{new}(x, y) = \min\{d_{old}(x, y), d_{old}(x, i) + 1 + d_{old}(j, y)\}, \forall x, y \in V. \tag{3}$$

3. The Data Structure

We now present a data structure that maintains on-line a solution to the All Pairs Minimal Length Paths Problem. Namely, we assume that all the edges have unit costs. For the sake of brevity, we denote by $add(i,j)$ the operation $add(i,j,1)$. Our data structure supports each *length* operation in $O(1)$ time, and each *minpath* operation in $O(k)$ time, where k is the length of the minimal length path traced. The overall time required to maintain the data structure during a sequence of (at most $O(n^2)$) *add* operations is $O(n^3 \log n)$. The space complexity is $O(n^2)$.

In addition to maintaining and updating the $n \times n$ distance matrix D, we associate with each vertex $v \in V$ the two sets of its descendants and ancestors, called *DESC(v)* and *ANC(v)*. In order to retrieve information about minimal length paths, $DESC(v)$ is organized as a tree of forward minimal paths rooted at v and $ANC(v)$ as a tree of backward minimal paths rooted at v. Moreover, we use two $n \times n$ matrices of pointers, named $FORWARD$ and $BACKWARD$ defined as follows. $FORWARD[u,v]$ contains a pointer to vertex v in the tree of minimal forward paths $DESC(u)$ if vertex v is a descendant of vertex u. Otherwise, $FORWARD[u,v]$ contains a special *null* value. Similarly, $BACKWARD[u,v]$ contains a pointer to vertex v in the tree of minimal backward paths $ANC(u)$ if vertex v is an ancestor of vertex u. Otherwise, $BACKWARD[u,v]$ contains a special *null* value. All of these data structures can be initialized in $O(n^2)$ time:

```
procedure initialize;
vertices : x,y;
begin
   for each x ∈ V do begin
      for each y ∈ V do begin
         FORWARD[x,y] := null;
         BACKWARD[x,y] := null;
         D[x,y] := +∞ ;
      end ;
      DESC(x) := {x} ;
      ANC(x) := {x} ;
      D[x,x] := 0 ;
      let FORWARD[x,x] point to the root of DESC(x) ;
      let BACKWARD[x,x] point to the root of ANC(x) ;
   end ;
end ;
```

With the help of these data structures, *length* operations can be performed in $O(1)$ time as the following pseudo-code shows:

Figure 1: A digraph G with the trees of forward and backward minimal and maximal paths length rooted at one of its vertices.

of *minimal length* (or in short *minimal*) if there is no other path from x to y that contains a smaller number of edges. A *cycle* is a nonempty path from a vertex to itself. A vertex v is said to be *reachable* from a vertex u if there is a path p from u to v. If v is reachable from u, then u is said to be an *ancestor* of v and v a *descendant* of u. If $u \neq v$, u is a *proper ancestor* of v and v is a *proper descendant* of u. If there is an edge from u to v, then u and v are said to be *adjacent*. If $G = (V, E)$ is a digraph, the digraph which has the same vertex set as G and has an edge from u to v if and only if v is reachable from u in G, is called the *transitive closure* of G. A digraph with no cycles is called a *directed acyclic graph* (in short *dag*). A *rooted tree* T is a dag satisfying the following three properties.

(i) There is only one vertex, referred to as the *root*, with no edges entering it.

(ii) Every vertex but the root has exactly one entering edge.

(iii) There is a (unique) path from the root to each vertex.

A *leaf* of the tree is a vertex with no outgoing edges. If there is an edge (u, v) in T, u is said to be a *parent* of v and v a child of u. The *depth* of a vertex v in T is the length of the path from the root to v. Given a digraph $G = (V, E)$, a *spanning tree* is a rooted tree $T = (V, S)$ such that $S \subseteq E$.

Given a digraph $G = (V, E)$ and a vertex $x \in V$, a *tree of forward paths rooted at* x is a tree $T_f(x) = (X, S)$ rooted at x such that

(i) X is the set of descendants of x in G; and

(ii) $S \subseteq E$.

Similarly, a *tree of backward paths* rooted at x is a tree $T_b(x) = (X, S)$ rooted at x such that

(i) X is the set of ancestors of x in G; and

(ii) $S \subseteq E^R$, where $E^R = \{(y, x) \mid (x, y) \in E\}$ is called the *set of reversed edges*.

A tree of forward (backward) paths $T = (X, S)$ rooted at x is said to be of *minimal length* (or equivalently it is said to be a *tree of forward (backward) minimal paths*) if for each vertex $v \in X$, the depth of v in T coincides with the length of the minimal path from x to v. In a similar vein, given a dag G a tree of forward (backward) paths $T = (X, S)$ rooted at x is said to be of *maximal length* (or equivalently it is said to be a *tree of forward (backward) maximal paths*) if for each vertex $v \in X$, the depth of v in T coincides with the length of the maximal path from x to v. Figure 1 shows a digraph G with the trees of forward and backward paths of minimal length rooted at one of its vertices.

The above terminology can be extended to a graph G with edge costs. Denote by $w(i, j)$ the cost of edge (i, j). The cost of a path p can be defined as $w(p) = \sum_{(i,j)\in p} w(i, j)$. A path p from x to y is said to be the *shortest path* from x to y if there is no path from x to y whose cost is less than $w(p)$. A *tree of forward (backward) shortest paths* T is a tree of forward (backward) paths such that each path in T is a shortest path in G.

shortest paths from scratch [15]. Rohnert [27] and recently Yellin [34] gave similar bounds. We remark that the bounds achieved by all these data structures cannot be amortized against sequences of operations; in particular, the total time they require to support a sequence of σ both edge insertions and edge cost decreases is $O(\sigma n^2)$.

In this paper, we consider the problem of maintaining on-line a solution to the All Pairs Shortest Path Problem on a directed graph under edge insertions and edge cost decreases, with integer edge costs in a fixed range $[1 \dots C]$. The maximum number of edge updates for this problem is $O(Cn^2)$, where n is the number of vertices in the graph, since each edge can be inserted at most once and its cost decreased at most C times. As a consequence, the algorithms in [11, 27, 34] solve this problem in a total of $O(Cn^4)$ worst-case time. In this paper, we present algorithms and data structures to solve this problem in a total of $O(Cn^3 \log(nC))$ worst-case time and $O(n^2)$ space. Since we show that there are sequences of edge insertions and edge cost decreases that force as many as $\Omega(Cn^3)$ changes in the distance matrix D, our algorithm is only a factor of $\log(nC)$ away from the best possible bound.

For the case of unit edge costs, previous algorithms [11, 27, 34] require $O(n^4)$ worst-case time over a sequence of at most $O(n^2)$ edge insertions. Using our approach, the overall time required to solve this problem becomes $O(n^3 \log n)$ in the worst case. In this case too, we show that there are sequences of edge insertions that require $\Omega(n^3)$ changes in the distance matrix D; therefore our algorithm is again only a logarithmic factor away from the best possible bound. In the remainder of this paper we refer to the All Pairs Shortest Paths Problem with unit edge costs as the All Pairs Minimal Length Paths Problem.

As a side result, we show also how to achieve similar bounds while maintaining longest paths in directed acyclic graphs, when *add* operations do not introduce cycles. This is not a restriction, since the maximal length path problem in cyclic graphs is known to be NP-complete [16].

The remainder of the paper consists of six sections. In section 2 we give preliminary definitions. A data structure for maintaining incrementally shortest paths with unit edge costs is introduced in section 3, and its time complexity is analyzed in section 4. Section 5 discusses the case of integer edge costs, while section 6 is devoted to longest paths in directed acyclic graphs. Section 7 contains concluding remarks.

2. Preliminaries

We assume that the reader is familiar with the standard graph theoretical terminology as contained for example in [2, 31]. In particular, a *directed graph* $G = (V, E)$ (also referred to as a *digraph*) consists of a finite set $V = \{1, 2, \dots, n\}$ of *vertices* and a finite set E of edges such that each edge e has a *head* $h(e) \in V$ and a *tail* $t(e) \in V$. We consider the edge $e = (t(e), h(e))$ as directed from $t(e)$ to $h(e)$ and we say that e leaves $t(e)$ and enters $h(e)$. A *path* $p = e_1, e_2, \dots e_k$ is a sequence of edges such that $h(e_i) = t(e_{i+1})$ for $1 \leq i \leq k-1$. The path p is from vertex $t(p) = t(e_1)$ to vertex $h(p) = h(e_k)$ and contains the edges $e_1, e_2, \dots, e_k$ and the vertices $t(e_1), t(e_2), \dots t(e_k), h(e_k)$. The path is *simple* if all its vertices are distinct. The *length* of a path is the number of edges it contains. A path p from x to y is said to be

1. Introduction

A dynamic data structure is able to update the solution of a problem after dynamic changes, rather than having to recompute it from scratch each time. Given their powerful versatility, it is not surprising that dynamic data structures are often more difficult to design than static ones. In the realm of graph problems, several dynamic data structures have been proposed to support insertions and deletions of edges and/or vertices in a graph, in addition to certain types of queries. In particular, much attention has been devoted to the dynamic maintenance of connectivity [10, 12, 13, 17, 26, 28, 33], transitive closure [4, 19, 20, 21, 23, 25, 34], planar graphs [9, 25, 30], shortest paths [11, 27, 29, 34] and minimum spanning trees [8, 10, 13, 14, 29]. Dynamic algorithms for graphs are of theoretical as well as practical interest in several application areas, such as communication networks, data base and knowledge base systems [1, 5, 22], incremental data flow analysis [6, 7], high level languages for incremental computations [35], and incremental generation of parsers [18].

An important problem in this area is the On-Line All Pairs Shortest Paths Problem. The problem consists of maintaining, during both edge insertions and edge cost decreases, a distance matrix D such that entry $D[i,j]$ contains the length of the shortest path from i to j, for any pair of vertices (i,j). Another way to look at this problem is to maintain an underlying directed graph under an arbitrary sequence of operations of the following four kinds:

- $add(x,y,w)$: insert an edge of cost w from vertex x to vertex y;
- $decrease(x,y,\Delta)$: decrease by $\Delta > 0$ the cost of edge (x,y);
- $length(x,y)$: return the length of the shortest path from x to y, if one exists; return $+\infty$ otherwise;
- $minpath(x,y)$: return the shortest path from x to y, if one exists.

The problem must be solved in an on-line fashion, i.e., each operation must be performed before the next one is known. Another constraint of the problem is that we would like to answer *length* queries in $O(1)$ time and perform *minpath* operations in time linear in the length of the traced path. We assume that *add* operations insert edges that are not already in the graph, since repeated insertions of the same edge can be easily taken care of.

This problem has been previously studied [11, 27, 29, 34] but the known solutions seem far from optimal. Even and Gazit [11] showed how to maintain a solution to the All Pairs Shortest Paths during edge insertions, edge deletions and edge cost updates. Their data structure requires $O(n^2)$ time to be updated after either an edge insertion or an edge cost decrease and $O(mn + n^2 \log n)$[1] time after either an edge deletion or an edge cost increase, where m is the current number of edges in the graph. Notice that the time required by an edge deletion or an edge cost increase is the same as the time required to recompute all pairs

[1] All the logarithms are assumed to be to the base 2 unless explicitly specified otherwise.

Incremental Algorithms for Minimal Length Paths [1]

Giorgio Ausiello [2]
Giuseppe F. Italiano [2,3]
Alberto Marchetti Spaccamela [4]
Umberto Nanni [2,4,5]

Technical Report Columbia University
CUCS-001-90
January 1990

Abstract

We consider the problem of maintaining on-line a solution to the All Pairs Shortest Paths Problem in a directed graph $G = (V, E)$ where edges may be dynamically inserted or have their cost decreased. For the case of integer edge costs in a given range $[1 \ldots C]$, we introduce a new data structure which is able to answer queries concerning the length of the shortest path between any two vertices in constant time, and to trace out the shortest path between any two vertices in time linear in the number of edges reported. The total time required to maintain the data structure under a sequence of at most $O(n^2)$ edge insertions and at most $O(Cn^2)$ edge cost decreases is $O(Cn^3 \log(nC))$ in the worst case, where n is the total number of vertices in G. For the case of unit edge costs, the total time required to maintain the data structure under a sequence of at most $O(n^2)$ insertions of edges becomes $O(n^3 \log n)$ in the worst case. Both data structures can be adapted to solve the problem of maintaining on-line maximal length paths in directed acyclic graphs. All our algorithms improve on previously known algorithms and are only a logarithmic factor away from the best possible bounds.

[1] Work partially supported by the ESPRIT II Basic Research Actions Program of the European Communities under contract No. 3075 (Project ALCOM) and by the Italian MPI National Project "Algoritmi e Strutture di Calcolo". A preliminary version of this paper appears in [3].

[2] Dipartimento di Informatica e Sistemistica, Università di Roma "La Sapienza", Roma, Italy.

[3] Department of Computer Science, Columbia University, New York, NY 10027. Partially supported by NSF Grants CCR-86-05353, CCR-88-14977 and by an IBM Graduate Fellowship.

[4] Dipartimento di Matematica Pura ed Applicata, Università di L'Aquila, L'Aquila, Italy.

[5] Partially supported by Selenia S.p.A., Italy.