World Academy of Science, Engineering and Technology
International Journal of Computer and Information Engineering
Vol:3, No:2, 2009

All-pairs shortest-paths problem for unweighted graphs in $O(n^2 \log n)$ time

Udaya Kumar Reddy K. R, and K. Viswanathan Iyer

Abstract—Given a simple connected unweighted undirected graph $G = (V(G), E(G))$ with $|V(G)| = n$ and $|E(G)| = m$, we present a new algorithm for the $all-pairs\ shortest-path$ (APSP) problem. The running time of our algorithm is in $O(n^2 \log n)$. This bound is an improvement over previous best known $O(n^{2.376})$ time bound of Raimund Seidel (1995) for general graphs. The algorithm presented does not rely on fast matrix multiplication. Our algorithm with slight modifications, enables us to compute the APSP problem for unweighted directed graph in time $O(n^2 \log n)$, improving a previous best known $O(n^{2.575})$ time bound of Uri Zwick (2002).

Keywords—Distance in graphs, Dynamic programming, Graph algorithms, Shortest paths.

Open Science Index, Computer and Information Engineering Vol:3, No:2, 2009 publications.waset.org/8870.pdf

I. Introduction

LET $G = (V(G), E(G))$ be a finite connected unweighted undirected graph without self-loops and multiple edges. Let $|V(G)| = n$ and $|E(G)| = m$. For $u, v \in V(G)$, let $d_G(u, v)$ be the $distance$ (the minimum number of edges on a shortest path between u and v) between u, v (we take $d_G(u, v) = \infty$ if no such path exist). Given G, the all-pairs shortest-path (APSP) problem asks to compute $d_G(u, v)$ between every pair of vertices $u, v \in V(G)$. A traditional method to solve the APSP problem of a graph G is to run breadth-first-search (BFS), once from every vertex of G. This takes time $O(mn)$ which can be cubic in n for dense graphs. Many researchers have improved for long time and various algorithms have been proposed via fast matrix multiplication algorithms (see for example, Galil and Margalit [5], [6] and Seidel [7]), and algorithms that achieve logarithmic speeds (see for example, Feder and Motwani [4] and Chan [2]). For general graphs, the best result is due to Seidel [7] for the unweighted undirected case, using Coppersmith and Winograd's matrix multiplication algorithm. As mentioned in Dragan [3], fast matrix multiplication algorithms are far from being practical and suffer from large hidden constants in the running time bound and many researchers started considering the all-pairs almost shortest-paths (APASP) problem for general graphs or for certain class of graphs designing an optimal time $O(n^2)$ algorithms. For more details of APSP problem on different graph classes, APASP problem and approximation results see Dragan [3] and for general graphs see Chan [2]. Many of the graph classes such as interval graphs, chordal graphs, circular arc graphs, permutation graphs, etc., are generally not sparse and can contain as many as $\Theta(n^2)$ edges. Therefore there is an interest in obtaining optimal algorithms for the APSP problem for dense graphs in general. For sparse graphs, many efforts have been made during the years to beat the naive $O(mn)$ time bound. Throughout the years slight improvements were obtained only for dense graphs. Recently, Chan [2] succeeded to obtain the $o(mn)$ result for unweighted undirected graphs for all $m \ll n^{1.376}$. There are also results for APSP problem of unweighted directed graphs. In this case, Alon, et. al, [1] obtained a $O(n^{2.688})$ algorithm. Later Zwick [8] has improved the result to obtain a running time of $O(n^{2.575})$ which is currently the best time bound. Also in this case fast matrix multiplication has been applied to solve the APSP problem.

In this paper we propose a new algorithm for the APSP problem for unweighted (undirected or directed) graphs in time $O(n^2 \log n)$. This bound beats the previous best known $O(n^{2.376})$ time bound on APSP of Seidel [7] for unweighted undirected case, and $O(n^{2.575})$ time bound for unweighted directed case of Zwick [8]. Our approach is simpler and practical than the previous approaches using fast matrix multiplication and word-packing tricks.

The rest of the paper is organized as follows. In Section II, starting with basic definitions and notions of the shortest paths, we show how to compute the APSP problem for an unweighted undirected graph G using dynamic programming technique and also shows that the APSP problem can be solved for unweighted directed graphs.

II. Computation of all-pairs shortest-path problem

In this section, we give a dynamic programming solution to solve the all-pairs shortest-path problem on an unweighted (undirected or directed) graph G. We assume that the vertices of G are numbered 1, 2, ... , n. One way to compute APSP problem on an unweighted graph G in time $\Theta(n^3)$ is to assign a weight of 1 to each edge in $E(G)$ and run the Floyd-Warshall algorithm. If there is a path from vertex i to vertex j, we get $d_{ij} < n$. Otherwise, we get $d_{ij} = \infty$ (for the case of directed graphs). Let A denote the n-by-n (boolean matrix) adjacency matrix in which the element in the ith row and the jth column is equal to 1 if there is an edge from vertex i to vertex j and equal to 0 if there is no such edge. Let D denote n-by-n distance matrix. Let $D^{(0)}$, $D^{(1)}$, ... , $D^{(x)}$ be the sequence of distance matrices generated by our algorithm for the given graph G, where $x \leq \log n$. For each $k \in \{0, 1, \ldots, x\}$, let $D^{(k)} = (d_{ij}^{(k)})$ for all $i, j \in V(G)$, where d_{ij} denote the distance

Udaya Kumar Reddy K.R is with the Department of Computer Science and Engineering, National Institute of Technology, Tiruchirappalli–620 015, India (Tel.: +91-9003301758; Fax: +91-431-2500133; email: krudaykumar@yahoo.com).

K. Viswanathan Iyer is with the Department of Computer Science and Engineering, National Institute of Technology, Tiruchirappalli–620 015, India (email: kvi@nitt.edu).

TABLE I
THE ARRAYS $a^{(k)}$ AND $b^{(k)}$ FROM $D^{(0)}$ AND $D^{(1)}$ IN FIG. 1.

Index i	1	2	3	4	5	6	7	8
$a^{(0)}[i]$	2	1	4	3	1	2	3	4
$b^{(0)}[i]$	5	6	6	7	NIL	3	4	7

Index i	1	2	3	4	5	6	7	8
$a^{(1)}[i]$	6	3	7	8	2	7	6	3
$b^{(1)}[i]$	3	5	2	6	6	1	8	2

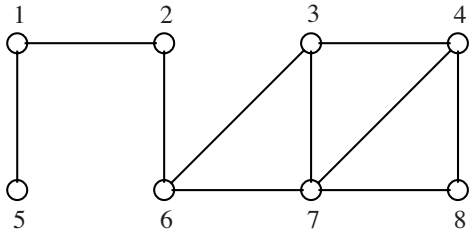
of a shortest path from i to j. For unweighted undirected graph G, for $1 \leq i \leq n$, let $a^{(k)}[i]$ denote some vertex $u_k = t$ such that $p = min(d_{it}^{(k)})$ for some p with $t \neq i$ and $t \neq (a^{(k-1)}[i] \ldots a^{(0)}[i]$ and $b^{(k-1)}[i] \ldots b^{(0)}[i])$, for $1 \leq t \leq n$, and for $1 \leq i \leq n$, let $b^{(k)}[i]$ denote some vertex $v_k = w$ such that $q = min(d_{iw}^{(k)})$ for some q with $w \neq i$ and $w \neq (a^{(k)}[i] \ldots a^{(0)}[i]$ and $b^{(k-1)}[i] \ldots b^{(0)}[i])$, for $1 \leq w \leq n$. Intuitively, for each k, and for all i, $j \in V(G)$, $a^{(k)}[i]$ is an array that holds an index u_k ($u_k \in \{1, \ldots, n\}$) of minimum distance in the ith row and jth column of the matrix $D^{(k)}$, and $b^{(k)}[i]$ is an array that holds an index v_k with $v_k \neq u_k$ ($v_k \in \{1, \ldots, n\}$) of next minimum distance in the ith row and jth column of the matrix $D^{(k)}$. For unweighted directed graph, for every i, let $a^{(k)}[i]$ denote some vertex $u_k = t$ such that $d_{it}^{(k)}$ is minimum with $t \neq i$ and $t \neq (a^{(k-1)}[i] \ldots a^{(0)}[i])$, for $1 \leq t \leq n$, and for every j, let $b^{(k)}[j]$ denote some vertex $v_k = w$ such that $d_{jw}^{(k)}$ is minimum with $w \neq j$ and $w \neq (b^{(k-1)}[j] \ldots b^{(0)}[j])$, for $1 \leq w \leq n$. Intuitively, for each k, and for all i, $j \in V(G)$, $a^{(k)}[i]$ is an array that holds an index u_k ($u_k \in \{1, \ldots, n\}$) of minimum distance in the ith row and jth column (row-wise) of the matrix $D^{(k)}$, and $b^{(k)}[i]$ is an array that holds an index v_k ($v_k \in \{1, \ldots, n\}$) of minimum distance in the jth column and ith row (column-wise) of the matrix $D^{(k)}$.

A. Some notions of the shortest paths

We begin with a lemma which shows that how to find $a^{(k)}$ and $b^{(k)}$ for $k = 0, 1, \ldots, x$.

Lemma 1:
Let $D^{(0)}, D^{(1)}, \ldots, D^{(x)}$ be the sequence of distance matrices for $k = 0, 1, 2, \ldots, x$ for some x. For all $i, j \in V(G)$, there exists some vertex u_k or some vertex v_k or both such that $u_k = j$ such that $d_{ij}^{(k)}$ is minimum and $v_k = j'$ such that $d_{ij'}^{(k)}$ is minimum, $j' \neq j$, with $u_k \neq (u_{k-1\ldots0}$ and $v_{k-1\ldots0})$ and $v_k \neq (u_{k\ldots0}$ and $v_{k-1\ldots0})$.

Proof. Assume that for all i, j, and for $k = 0$, $D^{(0)}$ contains $d_{ij}^{(0)} = 1$ if $(i, j) \in E(G)$, $d_{ij}^{(0)} = \infty$ if $(i, j) \notin E(G)$ and $d_{ii}^{(0)} = 0$. Assume that $j = 1, 2, \ldots, n$ in that order. First, for all i, j, and for $k = 1$, to compute $D^{(1)}$: since G is connected, in $D^{(0)}$ we find that each row i has at least one $d_{ij}^{(0)} = 1$. In $D^{(0)}$, since every row i has only distances $d_{ij}^{(0)} = 1$ (without considering $d_{ii}^{(0)} = 0$ and $d_{ij}^{(0)} = \infty$), and $d_{ij}^{(0)} \leq d_{ij+1}^{(0)} \leq \ldots \leq d_{in}^{(0)}$, we have $u_{k-1} = j$ and $v_{k-1} = j+1$. Now, for some i, and for $j \in \{1, 2, \ldots, n\}$, if there exists only one value $d_{ij}^{(0)} = 1$, then we have $u_{k-1} = j$ and v_{k-1} = NIL. Next, for all i, j, and for $k = 2$, to compute $D^{(2)}$: we may have distances $d_{ij}^{(k-1)} \geq 1$. Now we choose u_{k-1} (u_1) $= j$ such that $d_{ij}^{(k-1)}$ is minimum with $u_{k-1} \neq (u_{k-2}, v_{k-2})$ i.e., $u_1 \neq (u_0, v_0)$, and $v_{k-1} = j'$ such that $d_{ij'}^{(k-1)}$ is minimum with $v_{k-1} \neq (u_{k-2}, v_{k-2}, u_{k-1})$ i.e., $v_1 \neq (u_0, v_0, u_1)$. As for $k = 2$, a similar argument follows for $k = 3, 4, \ldots, x$, where $x \leq \log n$. Hence the lemma. ∎



$$D^{(0)} = \begin{pmatrix} 0 & 1 & \infty & \infty & 1 & \infty & \infty & \infty \\ 1 & 0 & \infty & \infty & \infty & 1 & \infty & \infty \\ \infty & \infty & 0 & 1 & \infty & 1 & 1 & \infty \\ \infty & \infty & 1 & 0 & \infty & \infty & 1 & 1 \\ 1 & \infty & \infty & \infty & 0 & \infty & \infty & \infty \\ \infty & 1 & 1 & \infty & \infty & 0 & 1 & \infty \\ \infty & \infty & 1 & 1 & \infty & 1 & 0 & 1 \\ \infty & \infty & \infty & 1 & \infty & \infty & 1 & 0 \end{pmatrix}$$

$$D^{(1)} = \begin{pmatrix} 0 & 1 & 3 & 4 & 1 & \mathbf{2} & 4 & 5 \\ 1 & 0 & \mathbf{2} & 3 & \mathbf{2} & 1 & \mathbf{2} & 3 \\ \mathbf{3} & 2 & 0 & 1 & 4 & 1 & 1 & \mathbf{2} \\ \mathbf{4} & \mathbf{3} & 1 & 0 & 5 & \mathbf{2} & 1 & 1 \\ 1 & 2 & \mathbf{4} & \mathbf{5} & 0 & \mathbf{3} & 5 & 6 \\ 2 & 1 & 1 & 2 & 3 & 0 & 1 & 2 \\ \mathbf{4} & 2 & 1 & 1 & \mathbf{5} & 1 & 0 & 1 \\ \mathbf{5} & \mathbf{3} & 2 & 1 & \mathbf{6} & \mathbf{2} & 1 & 0 \end{pmatrix}$$

$$D^{(2)} = \begin{pmatrix} 0 & 1 & 3 & 4 & 1 & 2 & 3 & 4 \\ 1 & 0 & 2 & 3 & 2 & 1 & 2 & 3 \\ 3 & 2 & 0 & 1 & 4 & 1 & 1 & 2 \\ 4 & 3 & 1 & 0 & 5 & 2 & 1 & 1 \\ 1 & 2 & 4 & 5 & 0 & 3 & 4 & 5 \\ 2 & 1 & 1 & 2 & 3 & 0 & 1 & 2 \\ 3 & 2 & 1 & 1 & 4 & 1 & 0 & 1 \\ 4 & 3 & 2 & 1 & 5 & 2 & 1 & 0 \end{pmatrix}$$

Fig. 1. An undirected graph and the sequence of matrices $D^{(k)}$ computed by ALL-PAIRS-SHORTEST-PATHS (Fig. 2) for $0 \leq k \leq 2$.

Let P be a shortest path from i to j. In the following lemma we show that the subpaths P_1 and P_2 of P are shortest paths.

Lemma 2:
Let P be any shortest path from i to j, $i \neq j$. Let z be an intermediate vertex in P. Let P_1 be a subpath of P from $i \ldots z$ and P_2 be a subpath of P from $z \ldots j$. Then the subpaths P_1 and P_2 are always shortest paths.

Proof. Let $k = 1, 2, \ldots, x$, where $x \leq \log n$. By Lemma 1, the vertices u_{k-1}, v_{k-1} are chosen such that the distance $d_{iu_{k-1}}^{(k-1)}$ and $d_{iv_{k-1}}^{(k-1)}$ are shortest-path distances by themselves. So the vertex u_{k-1} or vertex v_{k-1} is an intermediate vertex in the path P. In finding P, choosing z from $\{u_{k-1}, v_{k-1}\}$ is of crucial importance.
First, for all i, j, and for $k = 1$, to compute $D^{(1)}$: Here vertex

Open Science Index, Computer and Information Engineering Vol:3, No:2, 2009 publications.waset.org/8870.pdf

Open Science Index, Computer and Information Engineering Vol:3, No:2, 2009 publications.waset.org/8870.pdf

z is chosen based on the subpath P_2. Consider the subpath P_1: Here $d_{iu_{k-1}}^{(k-1)} = d_{iv_{k-1}}^{(k-1)} = 1$. That is, the subpath from i to u_{k-1} or i to v_{k-1} are shortest-path distances by themselves since $(i, u_{k-1}) \in E(G)$ or $(i, v_{k-1}) \in E(G)$. Now, consider the subpath P_2:

$Case$ 1: for some i, in computing $d_{ij}^{(k)}$, u_{k-1} or $v_{k-1} > i$. Here $d_{u_{k-1}j}^{(k-1)} = d_{v_{k-1}j}^{(k-1)} = 1$ since $(u_{k-1}, j) \in E(G)$ and $(v_{k-1}, j) \in E(G)$. So $z = u_{k-1}$ or v_{k-1}.

$Case$ 2: for some i, in computing $d_{ij}^{(k)}$, u_{k-1} or $v_{k-1} < i$. Here $d_{u_{k-1}j}^{(k-1)} \geq 1$ and $d_{v_{k-1}j}^{(k-1)} \geq 1$. That is, the values of $d_{u_{k-1}j}^{(k-1)}$ and $d_{v_{k-1}j}^{(k-1)}$ are already computed as shortest-path distances in $D^{(k)} = d_{u_{k-1}j}^{(k)}$ or $D^{(k)} = d_{v_{k-1}j}^{(k)}$. In this case $z = u_{k-1}$ if $d_{u_{k-1}j}^{(k-1)} < d_{v_{k-1}j}^{(k-1)}$ and $z = v_{k-1}$ if $d_{u_{k-1}j}^{(k-1)} > d_{v_{k-1}j}^{(k-1)}$. Suppose if $d_{u_{k-1}j}^{(k-1)} = d_{v_{k-1}j}^{(k-1)}$, then $z = u_{k-1}$ or u_{k-1}. In general, for $k = 1$, $z = u_{k-1}$ if $d_{u_{k-1}j}^{(k-1)} \leq d_{v_{k-1}j}^{(k-1)}$ and $z = v_{k-1}$ otherwise. Thus, the subpath P_1 from i to z is a shortest path with no intermediate vertices in the set $\{1, 2, \ldots, n\}$ and the subpath P_2 from z to j is a shortest path with all intermediate vertices in the set $\{1, 2, \ldots, n\}$.

Next, for all i, j, and for $k = 2$, to compute $D^{(2)}$: Here vertex z is chosen based on P_1 or P_2. Consider the subpath P_1: Now, $d_{iu_{k-1}}^{(k-1)} \geq 1$ and $d_{iv_{k-1}}^{(k-1)} \geq 1$. That is, the subpath from i to u_{k-1} or i to v_{k-1} are shortest-path distances by themselves since the values of $d_{iu_{k-1}}^{(k-1)}$ and $d_{iv_{k-1}}^{(k-1)}$ are already computed in $D^{(k-1)}$ for all $i, j \in V(G)$. Now, consider the subpath P_2: Here $d_{u_{k-1}j}^{(k-1)} \geq 1$ and $d_{v_{k-1}j}^{(k-1)} \geq 1$. Again the values of $d_{u_{k-1}j}^{(k-1)}$ and $d_{v_{k-1}j}^{(k-1)}$ are already computed as shortest-path distances in $D^{(k-1)}$ or in $D^{(k)}$ if $Case$ 2 above satisfies. In this case $z = u_{k-1}$ if $d_{u_{k-1}j}^{(k-1)} < d_{v_{k-1}j}^{(k-1)}$ and $z = v_{k-1}$ if $d_{u_{k-1}j}^{(k-1)} > d_{v_{k-1}j}^{(k-1)}$. Suppose if $d_{u_{k-1}j}^{(k-1)} = d_{v_{k-1}j}^{(k-1)}$, then $z = u_{k-1}$ if $d_{iu_{k-1}}^{(k-1)} < d_{iv_{k-1}}^{(k-1)}$ and $z = v_{k-1}$ otherwise. Thus, P_1 is a shortest path from i to z with all intermediate vertices in the set $\{1, 2, \ldots, n\}$ and P_2 is a shortest path from z to j with all intermediate vertices in the set $\{1, 2, \ldots, n\}$. As for $k = 2$, a similar argument follows for $k = 3, 4, \ldots, x$, where $x \leq \log n$.

Now, suppose that for some i, j, if we choose a vertex r in $V(G) - \{u_{k-1}, v_{k-1}\}$ as intermediate vertex, then the subpath P_2 from r to j is $d_{rj}^{(k-1)} \geq d_{zj}^{(k-1)}$. Thus the subpath from z to j is a shortest path with all intermediate vertices in the set $\{1, 2, \ldots, n\}$ and in turn from i to z is also a shortest path with all intermediate vertices in the set $\{1, 2, \ldots, n\}$. Thus, if the path P from i to j goes through the intermediate vertex z, then the subpaths of that path from i to z and from z to j are themselves shortest paths. Hence the lemma. ∎

Example A.1: Consider the shortest path P from vertex i to vertex j for $i = 8$ and $j = 1$ in the graph of Fig. 1. In Fig. 1. we see that the shortest path from vertex 8 to vertex 1 is computed in $D^{(2)}$. Let $(p \longrightarrow q) = r$ denote the distance from p to q. For vertex i, the vertices $u_{k-1} = 2$, and $v_{k-1} = 3$, and the distances from u_{k-1} to j and v_{k-1} to j are: $(2 \longrightarrow 1) = 1$ and $(3 \longrightarrow 1) = 3$. So we choose $z = 2$ as intermediate vertex since $(2 \longrightarrow 1) < (3 \longrightarrow 1)$. The distances from vertex i to u_{k-1} and i to v_{k-1} are: $(8 \longrightarrow 2) = 3$ and $(8 \longrightarrow 3) = 2$. Though $(8 \longrightarrow 3) < (8 \longrightarrow 2)$, based on vertex z (or subpath $P_2 = z$ to j) we conclude that the subpaths $P_1 = (8 \longrightarrow 2)$ and $P_2 = (2 \longrightarrow 1)$ are shortest paths from 8 to 1. This illustrates Lemma 2. The shortest path from 8 to 1 is computed in the sequence as shown in Table II. At each step in Table II one can verify that the subpaths from i to z and from z to j are shortest-path distances by themselves. ∎

The next lemma shows that any shortest path P can be categorized into two parts.

Lemma 3:
From among all the paths from i to j, let P be a shortest path from i to j, $i \neq j$. Let the path P be simple. Let z be an intermediate vertex in P. Then all such paths of P can be partitioned into two categories: those that do not include the intermediate vertex z and those that do.

Proof. If the path P does not include z, then the shortest path from i to j with all intermediate vertices in the set $\{1, 2, \ldots, n\}$ in $D^{(k-1)}$ is also a shortest path from i to j with all intermediate vertices in the set $\{1, 2, \ldots, n\}$ in $D^{(k)}$. If P contains z, then we can split the path P into P_1 and P_2, where P_1 is a path from i to z and P_2 is a path from z to j. By Lemma 2, we know that the subpaths P_1 and P_2 of P are shortest paths with all intermediate vertices in the set $\{1, 2, \ldots, n\}$. Hence the lemma. ∎

B. The structure of a shortest-path

The first step in the dynamic programming paradigm is to find the optimal substructure and then use it to construct an optimal solution to the problem from optimal solutions to subproblems. The structure of a shortest path of our approach looks similar to that of Floyd-Warshall algorithm. That is, instead of drawing the intermediate vertices from the set $\{1, 2, \ldots, k\}$ for $k \in \{1, 2, \ldots, n\}$, we draw the intermediate vertices from the set $\{1, 2, \ldots, n\}$ using $\{u_k, v_k\}$, for some $u_k, v_k \in \{1, 2, \ldots, n\}$ for each $k \in \{1, 2, \ldots, x\}$, where $x \leq \log n$. Based on the above Lemmas 1−3, we can formulate the structure of a shortest-path as follows. For every pair of vertices $i, j \in V(G)$, let P be the minimum-distance path from among all paths from vertex i to vertex j and the path P is simple. We can partition all such paths into two categories: those that do not use the vertex z as intermediate and those that do.

- If vertex z is not in P, then the shortest-path from vertex i to vertex j is of length $d_{ij}^{(k-1)}$.
- If vertex z is in P, then we break down P into P_1 and P_2, where,
  - P_1 is a shortest-path from vertex i to vertex z of length $d_{iz}^{(k-1)}$, where $z = (a^{(k-1)}[i]$ or $b^{(k-1)}[i])$.
  - P_2 is a shortest-path from vertex z to vertex j of length $d_{zj}^{(k-1)}$.

TABLE II
SHORTEST PATH FROM VERTEX 8 TO VERTEX 1

Edge (i, j) or (j, i)	Path $(i \longrightarrow j)$	$\{u_{k-1}, v_{k-1}\}$	Vertex z	$D^{(k)}$	Distance
(2, 6)	(2 ⟶ 6)	NIL	NIL	$D^{(0)}$	1
(2, 7)	(2 ⟶ 6 ⟶ 7)	{1, 6}	6	$D^{(1)}$	1 + 1 = 2
(8, 2)	(8 ⟶ 7 ⟶ 2)	{4, 7}	7	$D^{(1)}$	1 + 2 = 3
(8, 1)	(8 ⟶ 2 ⟶ 1)	{2, 3}	2	$D^{(2)}$	3 + 1 = 4

Procedure ALL-PAIRS-SHORTEST-PATHS(G)
1) Initialize $D[i, j] \leftarrow \infty$ for every i and j, and for every i, $D[i, i] \leftarrow 0$
2) Modify $D[i, j] \leftarrow 1$ whenever $A[i, j] = 1$, where A is the adjacency matrix of G.
3) **repeat**
4) **for** $i \leftarrow 1$ **to** n **do** $amin \leftarrow \infty$; $bmin \leftarrow \infty$
5) **for** $j \leftarrow 1$ **to** n
6) **do if** $D[i, j] < amin$ **and** $j \neq i$ **and** $BR[i, j] = 0$
7) **then** $amin \leftarrow D[i, j]$; $BR[i, j] \leftarrow 1$; $a[i] \leftarrow j$; $s_1 \leftarrow j$
8) **if** $D[i, j] < D[i, s_1]$ **then** $BR[i, s_1] \leftarrow 0$
9) **for** $j \leftarrow 1$ **to** n
10) **do if** $D[i, j] < bmin$ **and** $j \neq i$ **and** $j \neq a[i]$ **and** $BR[i, j] = 0$
11) **then** $bmin \leftarrow D[i, j]$; $BR[i, j] \leftarrow 1$; $b[i] \leftarrow j$; $s_2 \leftarrow j$
12) **if** $D[i, j] < D[i, s_2]$ **then** $BR[i, s_2] \leftarrow 0$
13) **for** $i \leftarrow 1$ **to** n
14) **do for** $j \leftarrow 1$ **to** n
15) **do** $t_1 \leftarrow D[i, a[i]] + D[a[i], j]$; $t_2 \leftarrow D[i, b[i]] + D[b[i], j]$
16) $D[j, i] \leftarrow D[i, j] \leftarrow$ MIN$\Big\{D[i, j],$ MIN$\{t_1, t_2\}\Big\}$
17) $flag \leftarrow$ **true**
18) **for** $i \leftarrow 1$ **to** n
19) **do for** $j \leftarrow 1$ **to** n **do** $temp[i, j] \leftarrow D[i, j]$
20) **if** $D[i, j] \neq temp[i, j]$ **then** $flag \leftarrow$ **false**
21) **until** $flag$ = **false**

Fig. 2. Computing APSP for unweighted undirected graph G.

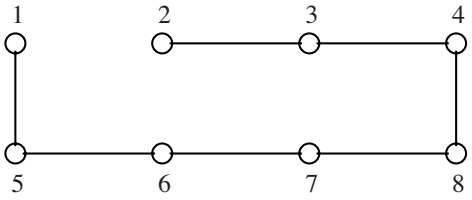


Fig. 3. An undirected graph having a single path from 1 to 2.

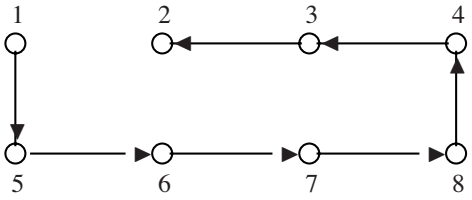


Fig. 4. A directed graph having a single path from 1 to 2.

C. A recursive solution

The second step in the dynamic programming paradigm is to define the cost of an optimal solution recursively in terms of the optimal solutions to subproblems. For each $k \in \{0, 1, \ldots, x\}$, and for all $i, j \in V(G)$, we define $d_{ij}^{(k)}$ recursively as follows. Initially, for $k = 0$, the problem is trivial and the matrix $D^{(0)}$ is defined as follows:

$$d_{ij}^{(0)} = \begin{cases} 0 & \text{if } i = j \\ 1 & \text{if } i \neq j \text{ and } (i,j) \in E(G) \\ \infty & \text{if } i \neq j \text{ and } (i,j) \notin E(G) \end{cases} \quad (1)$$

To compute $d_{ij}^{(k)}$ for $k \geq 1$, we take the advantage of the structure of an optimal shortest path. Thus, we obtain,

$$d_{ij}^{(k-1)} = d_{ij}^{(k)} = min\Big\{d_{ij}^{(k-1)},\ min\{t_1, t_2\}\Big\} \quad (2)$$

where, $t_1 = d_{ip}^{(k-1)} + d_{pj}^{(k-1)}$ and $t_2 = d_{iq}^{(k-1)} + d_{qj}^{(k-1)}$. Also, $p = a^{(k-1)}[i]$ (u_{k-1}) and $q = b^{(k-1)}[i]$ (v_{k-1}). Eq. (2) computes the sequence of matrices: $D^{(1)}, \ldots, D^{(x)}$. Since the computation of $D^{(k)}$ uses only $D^{(k-1)}$, we do not have to save the earlier matrices. Let $\delta(i, j) = d_G(i, j)$ from vertex i to vertex j. The final matrix in the sequence, $D^{(x)} = (d_{ij}^{(x)}) = \delta(i, j)$ for all $i, j \in V(G)$ and hence is nothing but the distance matrix being sought.

Note C.1: In (2), we observe that the value of $d_{ij}^{(k)}$ after computing is assigned to $d_{ij}^{(k-1)}$. This is because, if for

Open Science Index, Computer and Information Engineering Vol:3, No:2, 2009 publications.waset.org/8870.pdf

Open Science Index, Computer and Information Engineering Vol:3, No:2, 2009 publications.waset.org/8870.pdf

some i, $d_{ij}^{(k)}$ has $\{u_{k-1}, v_{k-1}\}$ and u_{k-1} or $v_{k-1} < i$, then the shortest-path distances that is already computed from u_{k-1} to j or v_{k-1} to j can be used to compute $d_{ij}^{(k)}$. For example, in Fig. 1 to compute $d_{ij}^{(1)}$ from vertex i = 5 to j (j = 1, 2, ... , n), we have $z = u_{k-1} = 1$ and $1 < 5$, so the shortest-path distances computed from vertex 1 to n is used in computing from vertex 5 to n. That is, as we move down on each i starting from i = 1 in the increasing order, for each k, we can compute more number of distances since $d_{ij}^{(k)}$ draws the intermediate vertices from the set $\{1, 2, \ldots, n\}$.

D. Design of the algorithm

Based on the above observations, we list our procedure ALL-PAIRS-SHORTEST-PATHS(G) which is shown in the Fig. 2 for computing the APSP on unweighted undirected graph G. In Fig. 2, the function MIN(a, b) returns the minimum value of two items. For every i, the arrays $a[i]$ and $b[i]$ contain, respectively, an index u_{k-1} (u_{k-1} = 1, ... , n) of minimum distance in the ith row of matrix $D^{(k)}$ and an index v_{k-1} (v_{k-1} = 1, ... , n) of next minimum distance in the ith row of matrix $D^{(k)}$ (lines 7 and 11). Let BR denote an n-by-n boolean matrix. In lines 7 and 11, the entry in the ith row and the jth column of BR, marks as 1 for each $a[i]$ and $b[i]$ of $D^{(k-1)}$. Otherwise, the entry remains 0. Now, based on the values of $a[i]$ and $b[i]$ we can compute $d_{ij}^{(k)}$ for all i, $j \in V(G)$ (line 16). For every i and j, if $D^{(k)}$ is equal to $D^{(k+1)}$ (k = 1, 2, ... , x) (lines 18-20), then we terminate the algorithm and the matrix $D^{(x)}$ contains the actual shortest-path distances for all u, $v \in V(G)$. Thus Fig. 2 implements Eq. (1) and (2). We first give the algorithm, then prove its correctness and show that it requires $O(n^2 \log n)$ time, if G has n vertices.

Algorithm C.1: Computing all-pairs-shortest-path distances.
Input. A connected, unweighted undirected graph $G = (V(G), E(G))$.
Output. An $n \times n$ distance matrix $D^{(x)}$ of the shortest-path distances.
Method.

1) Initially, for every i and j, $BR[i, j]$ and $temp[i, j]$ are initialized to 0. Then ALL-PAIRS-SHORTEST-PATHS(G) (Fig. 2) is called to compute shortest-path distances between all pairs of vertices u, $v \in V(G)$. ■

Example C.1: Computation of distance matrix

Consider a connected undirected graph G in Fig. 1 and for each i, $a^{(0)}[i]$, $b^{(0)}[i]$, $a^{(1)}[i]$ and $b^{(1)}[i]$ in Table I. In Fig. 1, starting from the initialized matrix $D^{(0)}$ the procedure ALL-PAIRS-SHORTEST-PATHS (Fig. 2) computes the matrices $D^{(1)}$ and $D^{(2)}$ using (2).

- Computation of matrix $D^{(1)}$: Computation of $D^{(1)}$ uses $D^{(0)}$.
  Using (2), for $k = i = 1$, for instance we can compute distances $d_{ij}^{(k)}$ = 2, for j = 6 based on $a^{(0)}[i]$ = 2. That is, to compute $d_{16}^{(1)}$, we have, $t_1 = d_{12}^{(0)} + d_{26}^{(0)} = 1 + 1 = 2$. Similarly, for i = 2, and j = 5 we can compute $d_{ij}^{(k)}$ based on $a^{(0)}[i] = 1$. That is, to compute $d_{25}^{(1)}$, we have, $t_1 = d_{21}^{(0)} + d_{15}^{(0)} = 1 + 1 = 2$. Next for i = 2, and j = 3, 7, based on $b^{(0)}[i] = 6$ we can compute $d_{23}^{(1)} = d_{27}^{(1)} = 2$. Next for i = 3, and j = 1, we can compute based on $b^{(0)}[i] = 6$. That is, $d_{31}^{(1)} = d_{36}^{(0)} + d_{61}^{(0)} = 1 + 2 = 3$. Note that $d_{16}^{(1)}$ is already computed for i = 1. Since the graph is undirected $d_{61}^{(1)} = d_{16}^{(1)} = 2$. Thus, it is similar for every i and j and the result of $D^{(1)}$ matrix is shown in Fig. 1. Hence, the shortest-path distances that is computed for each i is represented as bold-font distances in $D^{(1)}$ of Fig. 1 and since the graph is undirected the rest of the distances are computed from i to j or j to i. Also, in $D^{(1)}$, we find that some of the $d_{ij}^{(1)}$ entries may not be the shortest-path distance (for example, (1, 7), (1, 8), (5, 7) and (5, 8) are not shortest-path distances) since it depends on some path which is not a shortest path from i to j. These intermediate results will be computed in the next higher matrix $D^{(2)}$ in which there exists a shortest path from i to j.
- Computation of matrix $D^{(2)}$: Computation of $D^{(2)}$ uses only $D^{(1)}$.
  Now based on $a^{(1)}[i]$ and $b^{(1)}[i]$ we can find $D^{(2)}$ matrix using (2). Again, for i = 1, and j = 7, 8, based on $a^{(1)}[1] = 6$ we can compute $d_{17}^{(2)} = d_{16}^{(1)} + d_{67}^{(1)} = 2 + 1 = 3$ and $d_{18}^{(2)} = d_{16}^{(1)} + d_{68}^{(1)} = 2 + 2 = 4$. Next, for i = 5, and j = 7, 8, based on $b^{(1)}[1] = 6$ we can compute $d_{57}^{(2)} = d_{56}^{(1)} + d_{67}^{(1)} = 3 + 1 = 4$ and $d_{58}^{(2)} = d_{56}^{(1)} + d_{68}^{(1)} = 3 + 2 = 5$. Note that except for (1, 7) or (7, 1), (1, 8) or (8, 1), (5, 7) or (7, 5) and (5, 8) or (8, 5), for remaining i, j we find $d_{ij}^{(2)} = d_{ij}^{(1)}$. The result of $D^{(2)}$ is shown in Fig. 1. Thus, in Fig. 1, the matrix $D^{(x)} = D^{(2)}$, for x = 2, gives the final answer: $d_{ij}^{(2)} = \delta(i, j)$ for all i, $j \in V(G)$. ■

It is clear that Fig. 1 gives a typical example of an undirected graph G in which all cases occurring in (2) are put to use in the algorithm. Fig. 3 gives an extreme example of an undirected graph in which there exists only one shortest path between vertices 1 and 2 and the path is $\{1, 5, 6, 7, 8, 4, 3, 2\}$. In this example the length of the shortest path from 1 to 2 is a $diameter$ which is the largest of all shortest-path distances. Fig. 4 gives an extreme example of a directed graph in which there exists only one shortest path between vertices 1 and 2.

Theorem 1:
Algorithm C.1 correctly finds the all-pairs shortest-path distance problem of unwieghted undirected graph G.

Proof. Suppose that the procedure ALL-PAIRS-SHORTEST-PATHS (Fig. 2) is run on G from every vertex $s \in V(G)$ to j, for all $j \in V(G) - \{s\}$. Let $D^{(0)}$, $D^{(1)}$, ... , $D^{(x)}$ be the sequence of distance matrices generated by ALL-PAIRS-SHORTEST-PATHS for the graph G, where $x \leq \log n$. Let $\delta(i, j)$ be the shortest-path distance from i to j. Then, ALL-PAIRS-SHORTEST-PATHS discovers the shortest-path distance over all possible vertex pairs i, j in G, and upon termination, we have $D^{(x)} = \delta(i, j)$ for all i, $j \in V(G)$. To prove correctness it suffices to prove, by induction on the

Open Science Index, Computer and Information Engineering Vol:3, No:2, 2009 publications.waset.org/8870.pdf

number k (k = 1, ... , x) of distance matrices (that **"repeat"** loop in Fig. 2 has been executed) of G. The inductive hypothesis is that $D^{(k)} = \delta(i, j)$ for all $i, j \in V(G)$ produced in line 16 (Fig. 2) are shortest-path distances.

The basis, k = 0, is trivial since $D^{(0)}$ is obtained from the adjacency matrix A by considering the 1 entries and noting the corresponding i and j values (**"repeat"** loop has not yet been executed).

Now, assume that the inductive hypothesis is true for k = 1, 2, ... , $x-1$ ($D^{(1)}$, $D^{(2)}$, ... , $D^{(x-1)}$) (**"repeat"** loop has been executed x-1 times). Now to carry out inductive step for $k = x$, based on the distances stored in the matrix $D^{(x-1)}$ and for every i, based on $u_{k-1} = a^{(k-1)}[i]$ and $v_{k-1} = b^{(k-1)}[i]$ with all intermediate vertices drawn in the set $\{1, 2, \ldots, n\}$, we can compute $D^{(x)}$ using (2) correctly. That is, by Lemma 1, for some i, we have $u_{k-1} = j$ and $v_{k-1} = j'$ with $j' \neq j$, $u_{k-1} \neq (u_{k-2\ldots0}$ and $v_{k-2\ldots0})$ and $v_{k-1} \neq (u_{k-1\ldots0}$ and $v_{k-2\ldots0})$. Consider $d_1 = d^{(k-1)}_{iu_{k-1}} + d^{(k-1)}_{u_{k-1}j}$ and $d_2 = d^{(k-1)}_{iv_{k-1}} + d^{(k-1)}_{v_{k-1}j}$ for some i, j. By Lemma 2, we have $z = u_{k-1}$ if $d_1 < d_2$ and $z = v_{k-1}$ otherwise. Let P be a shortest path from i to j. By Lemma 3, we have $d^{(k)}_{ij} = d^{(k-1)}_{ij}$ if z is not in P and $d^{(k)}_{ij} = d^{(k-1)}_{iz} + d^{(k-1)}_{zj}$ if z is in P. A similar argument follows for all i, $j \in V(G)$. Therefore the execution of Eq. (2) makes $D^{(k)} = (d^{(k)}_{ij})$ *true* for all $i, j \in V(G)$. ■

The following corollary is a consequence of Theorem 1.

Corollary 1:
Algorithm C.1 correctly finds the all-pairs shortest-path distance problem of unwieghted directed graph G.

Proof. Clearly, recurrence (2) can be used for unweighted directed graphs with the modified values of $a^{(k)}$ and $b^{(k)}$ for k = 0, 1, ... , x, as given in Section II. Now the result for directed graphs follows immediately from Theorem 1. ■

1) Complexity analysis: In Section II-D we have outlined an algorithm for computing APSP for unwieghted undirected graph via the procedure ALL-PAIRS-SHORTEST-PATHS(G). We will show in this section that the Algorithm C.1 can be implemented to run in $O(n^2 \log n)$ time. First we shall analyze for unweighted directed graphs since the procedure in Fig. 2 can be used for directed graphs with slight modifications for $a^{(k)}$ and $b^{(k)}$ (lines 4−12) as defined in Section II and we also modify line 16 (i.e., $D[i, j] \leftarrow$ MIN$\{D[i, j],$ MIN$\{t_1, t_2\}\}$) of Fig. 2 and we can show that $O(n^2 \log n)$ time bound is tight for directed graphs in the worst case. We shall then conclude the analysis for unweighted undirected graph G.

The worst case of our algorithm of Fig. 2 happens when the length of the shortest path is a *diameter* (maximum of all shortest-path distances) denoted by Δ_G of G or if the given graph G has a single path from vertex i to vertex j. We first examine the code of lines 1−21 except **"repeat .. until"** loop (lines 3 and 21). We can easily see that the cost of lines 1−2, 4−12, 13−16, and 18−20 each is $O(n^2)$. We now examine the code of **"repeat"** loop from lines 4−20. Consider the graph G that has a single path from i to j. For example, the graph in Fig. 4 has a single path from vertex 1 to vertex 2 that has a diameter Δ_G = 7 for n = 8 vertices. For a directed graph, solving the APSP problem of a graph G that has a single path from i to j computes in the following sequence: starting from the initialized matrix $D^{(0)}$ in line 2 of Fig. 2, our algorithm computes the matrices $D^{(1)}$, $D^{(2)}$, ... , $D^{(x)}$, for k = 1, 2, ... , x, where $x \leq \log n$ as follows:

- During computation of $D^{(1)}$, $d^{(1)}_{ij}$ computes all distances having $2 \leq d^{(1)}_{ij} \leq 2$, for all $i, j \in V(G)$ using $D^{(0)}$.
- During computation of $D^{(2)}$, $d^{(2)}_{ij}$ computes all distances having $3 \leq d^{(2)}_{ij} \leq 4$, for all $i, j \in V(G)$ using $D^{(1)}$.
- During computation of $D^{(3)}$, $d^{(3)}_{ij}$ computes all distances having $5 \leq d^{(3)}_{ij} \leq 8$, for all $i, j \in V(G)$ using $D^{(2)}$.
- During computation of $D^{(x-1)}$, $d^{(x-1)}_{ij}$ computes all distances having $\lceil \Delta_G/2^2 \rceil$+1 $\leq d^{(x-1)}_{ij} \leq \lceil \Delta_G/2^1 \rceil$, for all $i, j \in V(G)$ using $D^{(x-2)}$ and
- Finally, during computation of $D^{(x)}$, $d^{(x)}_{ij}$ computes all distances having $\lceil \Delta_G/2^1 \rceil$+1 $\leq d^{(x)}_{ij} \leq \lceil \Delta_G/2^0 \rceil$, for all $i, j \in V(G)$ using $D^{(x-1)}$.

Thus, as the matrices $D^{(1)}$, $D^{(2)}$, $D^{(3)}$, ... , $D^{(x)}$, are computed, the length of Δ_G reduces to $\lceil \Delta_G/2^1 \rceil$, $\lceil \Delta_G/2^2 \rceil$, $\lceil \Delta_G/2^3 \rceil$, ... , $\lceil \Delta_G/2^x \rceil$ respectively. Hence, the number of iterations performed by **"repeat .. until"** loop from lines 4−20 is $O(\log n)$. Consequently, the total running time of lines 4−20 is $O(n^2 \log n)$. We know that the cost of lines 1−2 is $O(n^2)$. Thus the total running time for unweighted directed graphs is $O(n^2 + n^2 \log n) = O(n^2 \log n)$. Therefore, we conclude that the total running time for unweighted undirected graphs is also in $O(n^2 \log n)$. That is, in our approach, if the running time in $O(n^2 \log n)$ is true for unweighted directed graphs then it is also true for unweighted undirected graphs in $O(n^2 \log n)$. Since the computation of $D^{(k)}$ uses only $D^{(k-1)}$, for each $k \in \{1, 2, \ldots, x\}$, we do not have to save the earlier matrices. So the space complexity dictated by the arrays $D[i, j]$, $BR[i, j]$ and $temp[i, j]$ in Fig. 2 is $O(n^2)$.

Theorem 2:
The all-pairs shortest-path distance problem for unwieghted (undirected or directed) graphs of Algorithm C.1 can be implemented to run in time $O(n^2 \log n)$ if G has n vertices.

E. Constructing a shortest-path

Let π_{ij} denote the predecessor of vertex j on a shortest path from vertex i. Let Π denote an $n \times n$ predecessor matrix. Suppose that $\Pi^{(0)}, \Pi^{(1)}, \ldots, \Pi^{(x)}$ are the sequence of matrices generated by our algorithm for the given graph G, where $x \leq \log n$. Suppose that the matrix $\Pi^{(k)} = \pi^{(k)}_{ij}$ (k = 1, ... , x) for all $i, j \in V(G)$. Let $\alpha(i, j)$ be the shortest-path from vertex i to vertex j. Then, for each possible vertex pairs i, j in G, we can compute the the matrix Π (shortest-path) from D matrix, and upon termination, we have $\Pi^{(x)} = \alpha(i, j)$ for all $i, j \in V(G)$. This can be implemented to run in time $O(n^2 \log n)$. Initially, for k = 0, the matrix $\Pi^{(0)}$ is defined as follows:

$$\pi^{(0)}_{ij} = \begin{cases} \text{NIL} & \text{if } i = j \text{ or } (i \neq j \text{ and } (i, j) \notin E(G)) \\ i & \text{if } i \neq j \text{ or } (i \neq j \text{ and } (i, j) \in E(G)) \end{cases} \quad (3)$$

For $k \geq 1$, following is the recursive definition of $\pi_{ij}^{(k)}$:

$$\pi_{ij}^{(k-1)} = \pi_{ij}^{(k)} = \begin{cases} \pi_{ij}^{(k-1)} & \text{if } d_{ij}^{(k-1)} \leq min\{t_1, t_2\} \\ \pi_{kj}^{(k-1)} & \text{if } d_{ij}^{(k-1)} > min\{t_1, t_2\} \end{cases} \quad (4)$$

Refer Section II-C for definition of t_1 and t_2. Equation (3) and (4) can be computed from (1) and (2) respectively. The following corollary is a consequence of Theorem 2.

Corollary 2:
The all-pairs shortest-path problem for unwieghted (undirected or directed) graphs can be solved in time $O(n^2 \log n)$ if G has n vertices.

Proof. Clearly, the matrix $\Pi^{(k)}$ (shortest-path) can be computed from the matrix $D^{(k)}$ for each k = 0, 1, ... , x and the result for unwieghted (undirected or directed) graphs follows immediately from Theorem 2. ■

REFERENCES

[1] N. Alon, Z. Galil, and O. Margalit, "On the exponent of the all-pairs shortest path problem," *J. Comput. Sys. Sci.*, vol. 54, pp. 255-262, 1997.
[2] T. M. Chan, "All-pairs shortest paths for Unweighted Undirected Graphs in $o(mn)$ Time," *In Proc. 17th annual ACM-SIAM Sympos. on Discrete Algorithms*, pp. 514-523, 2006.
[3] F. F. Dragan, "Estimating all pairs shortest paths in restricted graph families: a unified approach," *J. of Algorithms*, vol. 57, pp. 1-21, 2005.
[4] T. Feder and R. Motwani, "Clique patritions, graph compression and speeding-up algorithms," *J. Comput. Sys. Sci.*, vol. 51, pp. 261-272, 1995.
[5] Z. Galil and O. Margalit, "All pairs shortest distances for graphs with small integer length edges," *Inf. Comput.*, vol. 134, pp. 103-139, 1997.
[6] Z. Galil and O. Margalit, "All pairs shortest paths for graphs with small integer length edges," *J. Comput. Sys. Sci.*, vol. 54, pp. 243-254, 1997.
[7] R. Seidel, "On the all-pairs shortest path problem in unweighted undirected graphs," *J. Comput. Sys. Sci.*, vol. 51, pp. 400-403, 1995.
[8] U. Zwick, "All-pairs shortest paths using bridging sets and rectangular matrix multiplication," *J. ACM*, vol. 49, pp. 289-317, 2002.

Udaya Kumar Reddy K. R. completed his B.E in Computer Science and Engineering from Golden valley Institute of Technology, K.G.F, Bangalore University, India in 1998. In 2004 he completed his M.E in Computer Science and Engineering from University Visvesvaraya College of Engineering, Bangalore, India. Currently he is pursuing Ph.D in Computer Science and Engineering in National Institute of Technology, Trichy, India (formerly Regional Engineering College). His fields of interests are Algorithmic graph theory and Theory of computation.

K. Viswanathan Iyer completed his B.E in Electronics and Communication from the Indian Institute of Science, Bangalore, India in 1981. In 1984 he completed his M.Sc.(Engg.) from Indian Institute of Science, Bangalore, India. In 2007, he finished his Ph.D in Computer Science from National Institute of Technology, Trichy, India. After his masters, he worked in various industries for about 8 years. Since 1992 he has been with National Institute of Technology, Trichy, India (formerly Regional Engineering College) where he is now a Professor in the Department of Computer Science and Engineering. His fields of interests are Algorithms and Theory. He has 20 publications in journals and conference proceedings. He is listed in the 2009 edition of Marquis Who's Who in the World.

Open Science Index, Computer and Information Engineering Vol:3, No:2, 2009 publications.waset.org/8870.pdf