Applied Mathematics and Computation 217 (2011) 6401–6408

Contents lists available at ScienceDirect

Applied Mathematics and Computation

journal homepage: www.elsevier.com/locate/amc

Finding the shortest paths by node combination

Xin Lu [a,b,c,*], Martin Camitz [d]

[a] College of Information Systems and Management, National University of Defense Technology, Changsha, China
[b] Department of Sociology, Stockholm University, Stockholm, Sweden
[c] Department of Public Health Sciences, Karolinska Institute, Stockholm, Sweden
[d] Department of Epidemiology and Biostatistics, Karolinska Institute, Stockholm, Sweden

ARTICLE INFO

Keywords:
Shortest path
Node combination
Node combination algorithm
Dijkstra's algorithm
Weight matrix

ABSTRACT

By repeatedly combining the source node's nearest neighbor, we propose a node combination (NC) method to implement the Dijkstra's algorithm. The NC algorithm finds the shortest paths with three simple iterative steps: find the nearest neighbor of the source node, combine that node with the source node, and modify the weights on edges that connect to the nearest neighbor. The NC algorithm is more comprehensible and convenient for programming as there is no need to maintain a set with the nodes' distances. Experimental evaluations on various networks reveal that the NC algorithm is as efficient as Dijkstra's algorithm. As the whole process of the NC algorithm can be implemented with vectors, we also show how to find the shortest paths on a weight matrix.

0. Introduction

Let $G = (V,E,W)$ represent a network containing N nodes (vertices), where $V = \{v_1, v_2, \ldots, v_n\}$ is the set of nodes, $E = \{e_{ij} \mid$ if there is a link from v_i to $v_j\}$ is the set of edges and $W = \{w_{ij} \mid 1 \leqslant i,j \leqslant N\}$ is the weight matrix for E. Given two nodes v_s, v_t of G, the shortest path problem can be defined as how to find a path with the minimum sum of the weights on the edges in a v_s, v_t-path. Generally, v_s and v_t are called source node and sink node, respectively.

The shortest path problem is one of the most fundamental network optimization problems with widespread applications [1–4]. Among the various shortest path algorithms developed [5–12], Dijkstra's algorithm is probably the most well-known. It maintains a set S of solved nodes, comprising those nodes whose final shortest path distance from the source v_s has determined, and labels $d(i)$, storing the upper bound of the shortest path distance from v_s to v_i. The algorithm repeatedly selects the node $v_k \in V \backslash S$ with the minimum $d(i)$, adds v_k to S, and updates $d(i)$ for nodes that are incident to v_k (relaxation) [2,13]:

```
Step 0. Set $d(v_s) = 0$, for other nodes, $d(v_j) = w_{sj}$, $S = \{v_s\}$, $Q = V\backslash S$.
Step 1. Select a node $v_k$ from $Q$ such that $d(v_k) = \min_{v_j \in Q} d(v_j)$, if $d(v_k) = \infty$, stop, otherwise go to Step 2.
Step 2. Set $S = S \cup \{v_k\}$, $Q = Q\backslash\{v_k\}$. If $Q = \emptyset$, stop, otherwise go to Step 3.
Step 3. for every $v_j \in Q$, update $d(v_j) = \min\{d(v_j), d(v_k) + w_{kj}\}$. Go to Step 1.
```

In practice, Dijkstra's algorithm relies heavily on the strategies used to select the next minimum labeled node (Step 1) and the data structure utilized to maintain the set S. Readers can refer to [1,14] for more detailed discussions on these topics. When Q is implemented as an ordinary linked list or vector, the algorithm runs in $O(|V|^2)$ time. It is $O(|E|\lg|V|)$ if Q is implemented as a binary heap and $O(|E| + |V|\lg|V|)$ when as a Fibonacci heap [13,15].

* Corresponding author at: Department of Sociology, Stockholm University, SE-106 91 Stockholm, Sweden.
E-mail address: lu.xin@sociology.su.se (X. Lu).

0096-3003/$ - see front matter © 2011 Elsevier Inc. All rights reserved.
doi:10.1016/j.amc.2011.01.019

Though the efficiency and various applications of Dijkstra's algorithm have been widely studied [16–20], Dijkstra's algorithm may not be easily understood, especially when implementing the labeling method [1,16,17,19,21]. In this paper, we introduce another way to implement Dijkstra's algorithm, called the Node Combination (NC) algorithm, with which the source node iteratively combines nodes into a new source node and updates the edge weights of the remaining node. When all of the nodes in the connected component of the source node are finally combined into a single node, the shortest paths from the source node to all other nodes are known. With the method of node combination, the process of finding shortest paths is comparatively simple and much more vivid than with Dijkstra's algorithm. Though there are many ways in which the NC algorithm can be improved on in terms of efficiency, the focus of this paper is on simplicity and comprehensibility. NC also compares favorably to Dijkstra's in terms of memory efficiency.

Much like Dijkstra's algorithm, a slight variation in the NC algorithm also produces the actual shortest paths, not just the lengths of the shortest paths.

This paper is organized as follows: in Section 1, we define what is meant by the combination of nodes. In Section 2, we introduce and illustrate by example the fundamental idea and explicit description of NC algorithm. Section 3 discusses the time complexity, and evaluates the performance of the NC algorithm compared with Dijkstra's algorithm. In Section 4, we show how to find the shortest paths using the NC algorithm and demonstrate implementation on a weight matrix. Finally, we summarize our results and draw conclusions.

1. Node combination

Definition. Let v_i, v_j be two connected nodes of graph $G = (V,E,W)$, the combination of v_i and v_j is the replacement of v_i and v_j with a new node whose incident edges are the edges incident to v_i or v_j. The resulting graph is denoted as $G(v_i \bullet v_j)$ (see Fig. 1).

We can see from Fig. 1 that the new node maintains connections with nodes 5 and 7, which were connected with node 3 before combination.

After a combination, the number of edges incident to the start node will increase by the number of edges incident to the combined node, less one. Node combination may lead to multiple edges, for example, if there were an edge between nodes 1 and 7 in Fig. 1(a), there would be two edges between the new node and node 7 after combination. Node combination is in many ways similar to edge contraction in graph theory except for the appearance of multiple edges.

Without loss of generality, in this paper we consider only undirected networks with nonnegative edge weights and without self-loops.

2. Node combination algorithm

2.1. Fundamental idea

The fundamental idea of the NC algorithm is to combine nodes instead of maintaining the labeling sets in Dijkstra's algorithm. Suppose that all nodes in the network are connected by ropes. The source node is placed in a pool, and other nodes are successively dragged into the pool one by one. Over time, there will be fewer and fewer nodes outside, and finally all nodes will have been dragged into the pool.

The combined nodes correspond to the set of solved nodes whose distances have been established in Dijkstra's algorithm. The adjacent neighbors of the combined node correspond to the set of potential nodes from which the closest one is picked. In the meantime, we can update the edge weights to store the distance labels from the source node, instead of maintaining a vector of distances, making the procedure more comprehensible.

2.2. Algorithm and proofs

Given a nonnegative weighted network $G = (V,E,W)$ with N nodes, let $W_{N\times N}$ be the weight matrix, v_s be the source node, d be the vector whose element $d(v_j)$ is to save the distance between v_s and v_j, then iterations of NC algorithm can be described as follows:

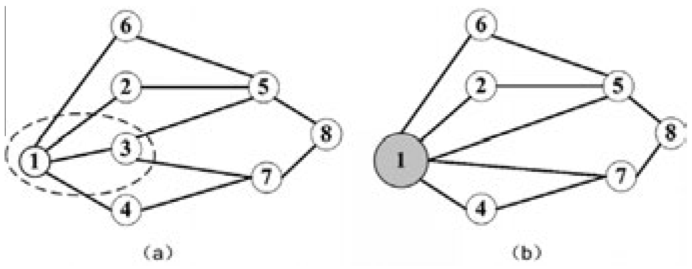

Fig. 1. Combine nodes 1 and 3. (a) before combination, (b) after combination.

Step 0. *Initialization*. Set $d(v_s) = 0$.
Step 1. *Find the nearest neighbor*. Select v_k from the neighbors of v_s, which makes $w_{sk} = \min\{w_{sj}\}$, let $d(v_k) = w_{sk}$. If there are no adjacent nodes to v_s, stop.
Step 2. *Combine node*. Delete v_k, $V = V - v_k$. If $V = \emptyset$, stop.
Step 3. *Modify edge weights*. For each edge e_{kj}, Update $w_{sj} = \min\{w_{sj}, w_{sk} + w_{kj}\}$. Go to Step 1.

The correctness of this algorithm can be proved by the following theorems:

Theorem 2.1. *NC algorithm solves the Single-Source Shortest Path (SSSP) problem in an increasing order of $d(v_k)$.*

Proof. Let w_{sj} and $w_{sj}^{(k)}$ be the weights of edge e_{sj} before and after combination of node v_k, respectively. Because $d(v_k) = \min\{w_{sj}\} = w_{sk}$, we only need to prove $\min\left\{w_{sj}^{(k)}\right\} \geqslant w_{sk}$ for $\forall v_j \in V - v_k$ after node combination. According to Step 3, the above inequality can be written as $\min\left\{w_{sj}^{(k)}\right\} = \min\left\{\min\left\{w_{sj}, w_{sk} + w_{kj}\right\}\right\} \geqslant w_{sk}$ for $\forall v_j \in V - v_k$, which is true. □

Theorem 2.2. *Given a network $G = (V, E, W)$ with nonnegative edge weights and a source node $v_s \in V$, NC algorithm computes $d(v_k)$ for every $v_k \in V$.*

Proof. Note that the edge weights are non-increasing during the algorithm. Let S be the set of solved nodes and $w_{sj}^{(k)}$ be the latest weight of edge e_{sj} when $|S| = k$.

(1) When $|S| = 0$ or $|S| = 1$, it is evident that $d(v_s) = 0$ and e_{sh} is the shortest path from v_s to its nearest neighbor v_h. This means that $w_{sh}^{(1)} = \min\left\{w_{sj}^{(1)}\right\} = d(v_h)$.
(2) Suppose that v_f is the first node that NC fails to find the shortest path for, i.e., $w_{sf}^{(k)} = \min_{j \in V-S}\left\{w_{sj}^{(k)}\right\} > d(v_f)$. (We have assumed $|S| = k$ when combining v_f.)
  (a) We know that the shortest path cannot be e_{sf}, since then $w_{sf}^{(0)} = d(v_f) < w_{sf}^{(k)}$, a possibility rules out by the NC algorithm.
  (b) Let v_m be the last node rather than v_f in the shortest path from s to f. We then have $v_m \notin S$ for $|S| = k$. For $|S| \neq k$ when combining v_m, $w_{sf}^{(i+1)} = \min\left\{w_{sf}^{(i)}, w_{sm}^{(i)} + w_{mf}^{(0)}\right\} = d(v_m) + w_{mf}^{(0)} = d(v_f)$ which contradicts our hypothesis.

So we can assume the shortest path is $v_s \to \cdots \to v_x \to v_y \to \cdots \to v_m \to v_f$, in which v_y is the first node not combined. When v_x was combined, $d(v_y) = w_{sy}^{(i'+1)} \leqslant w_{sf}^{(i'+1)} \leqslant d(v_f)$. However, now u_f is selected to be combined, so $w_{sy}^{(k)} \geqslant w_{sf}^{(k)}$, since the edge weights are non-increasing, the two inequalities must be equalities: $d(v_y) = w_{sy}^{(i'+1)} = w_{sf}^{(k)} = d(v_f)$, which again contradicts our hypothesis. Thus when each node was combined: $w_{sk} = \min\{w_{sj}\} = d(v_k)$. □

2.3. An example

Fig. 2 illustrates the execution of NC algorithm to find the distances between node 1 and all other nodes in the network, where $d(i)$ is the distance from node 1 to node i.

(1) Combine nodes 1 and 2. The source node 1 first finds its nearest neighbor, node 2, and immediately gains the distance from 1 to 2, which is $d(2) = 5$. Then node combination is carried out, modifying $w_{1,5} = \min\{w_{1,5}, d(2) + w_{2,5}\} = \min\{\infty, 5 + 10\} = 15$.
(2) Combine 1, 3. The nearest neighbor of the new starting node 1 is node 3, $d(3) = 8$. Then the node combination modifies $w_{1,5} = \min\{w_{1,5}, d(3) + w_{3,5}\} = \min\{15, 8 + 10\} = 15$, and $w_{1,7} = 18$.
(3) Combine 1, 6. $d(6) = 8$, modify $w_{1,5} = 13$.
(4) Combine 1, 4. $d(4) = 10$, modify $w_{1,7} = 17$.
(5) Combine 1, 5. $d(5) = 13$, modify $w_{1,8} = 19$.
(6) Combine 1, 7. $d(7) = 17$, modify $w_{1,8} = 19$.
(7) Combine 1, 8. $d(8) = 19$.
(8) If all nodes have been combined together, terminate.

From the above iterations, we can see that the NC algorithm can solve the SSSP problem by $N - 1$ times of node combination for a connected network with N nodes. The distance of shortest paths from node 1 to other nodes (2 to 8) are: 5, 8, 10, 13, 8, 17, and 19.

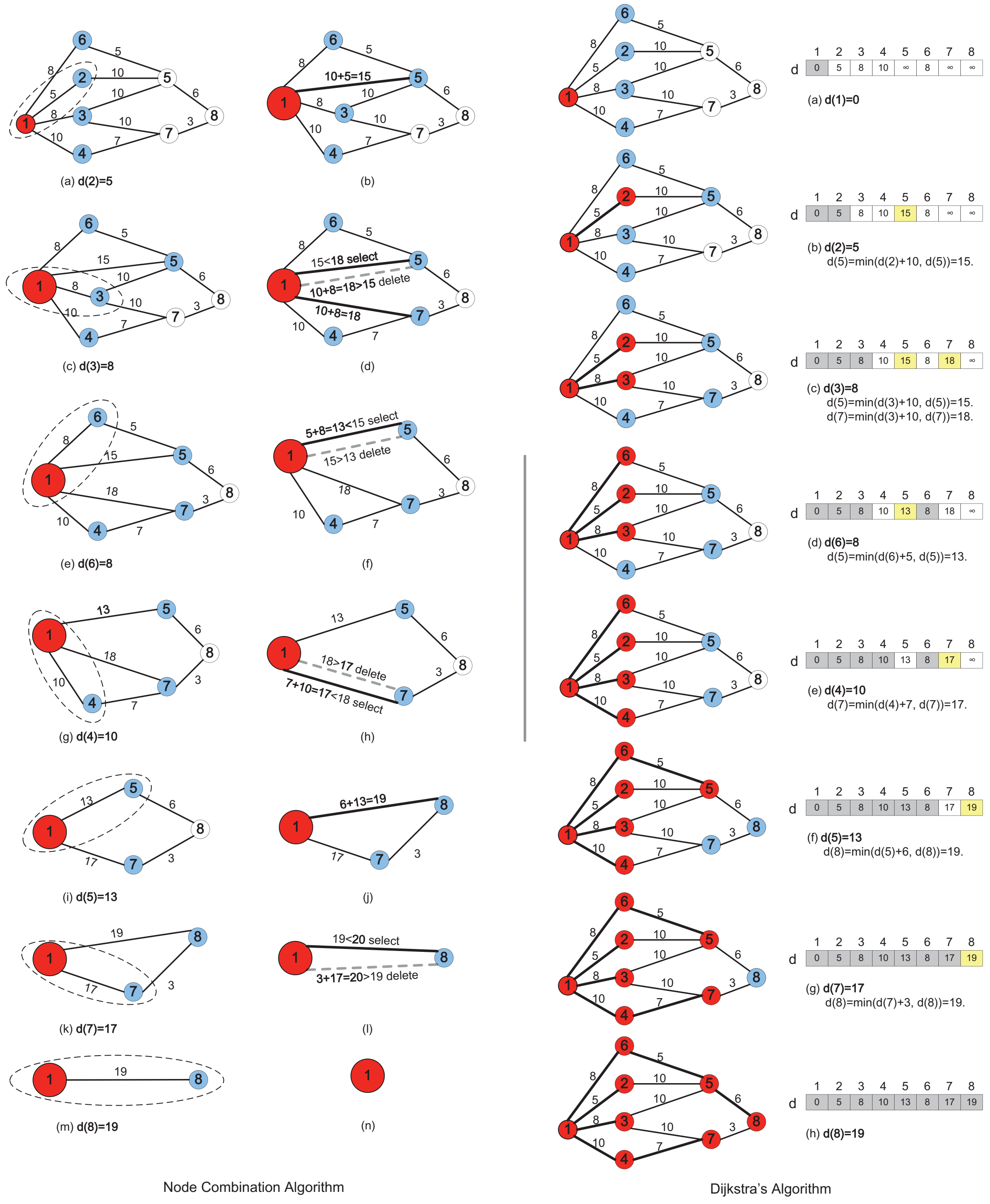

Fig. 2. Comparison of the NC algorithm and Dijkstra's algorithm.

3. Complexity and experimental evaluation

A naive approach to implement the node combination is to set the incident edges' weights of the combined node v_k to infinity: $w_{ik} = \infty$ for $v_i \in V$ or the maximum value depending on the language. This ensures proper ordering of the remaining node. To save computing time it is smarter to maintain a status vector V to identify the nodes that have been combined. The practical implementation reintroduces a labeling set, as in Dijkstra's algorithm, that was removed for the conceptual outline. Suppose the network is stored as an adjacent matrix, the pseudocode using a status vector V can be written as Algorithm 1:

Algorithm 1: Node_Combination(G, s)

```
1 $W[s,u] := 0,\ v_u := v_s,\ V := V - \{s\}$    /*Initialization*/
2 while $W[s,u] < \infty$ and $|V| > 0$
3   $V := V - \{u\}$    /*Node Combination*/
4   for each $j$ in $V$
5     $W[s,j] := \min\{W[s,j], W[s,u] + W[u,j]\}$    /*updating edge weights*/
6   $v_u :=$ the nearest neighbor of $s$ in $V$    /*finding the nearest neighbor*/
/*at the end of the algorithm, the sth row in W contains the corresponding distances*/
```

For comparison, we provide an implementation of Dijkstra's algorithm.

Algorithm 2: Dijkstra(G, s)

```
1 $d[s] := 0,\ v_u := v_s,\ V := V - \{s\}$    /*Initialization*/
2 while $d[u] < \infty$ and $|V| > 0$
3   $V := V - \{u\}$    /*mark u as visited*/
4   for each $j$ in $V$
5     $d[j] := \min\{d[j], d[u] + W[u,j]\}$    /*relaxation*/
6   $d[u] :=$ the smallest value in $d$ for nodes in $V$
/*at the end of the algorithm, vector d contains the corresponding distances*/
```

From the above pseudocode, we can see that the potential distances are now recorded as updated edge weights so that no additional memory is required. As the times of comparison and addition in Steps 2 and 5 are the same in both algorithms, the NC algorithm computes the SSSP problem with complexity $O(N^2)$ when the network is stored as an adjacency matrix. This provided of course that no improved searching strategy is implemented for finding the nearest neighbor.

To show the validity of this algorithm, simulations are performed for Algorithms 1 and 2. As the focus of this paper is simplicity and memory efficiency, rather than run time efficiency, we refrain from comparing the two algorithms using alternative data structures. Both algorithms are coded with the same data structures. Algorithms are coded in C and run on an Intel Core Duo CPU with a 2.4 GHz processor, 3 Mb of cache and 3 Gb of RAM.

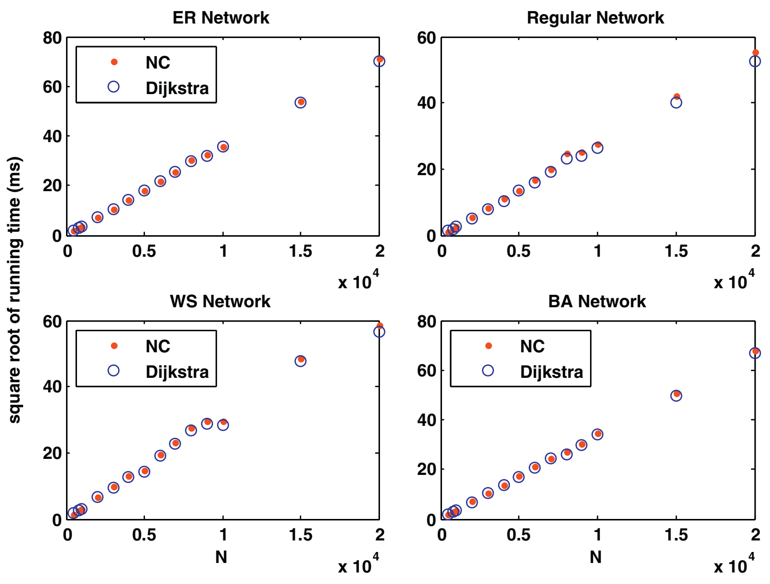

Fig. 3. Average running times of the NC algorithm and Dijkstra's algorithm. Average degrees are 10, 6, 6, 6, respectively. All networks are connected, and the edge weights are random numbers between 1 and 10. Each simulation calculates the shortest paths from node 0 to all others. Running times are the averaged values of 100 simulations.

Four kinds of networks are tested: the ER random networks, [22] regular networks, WS small-world networks [23], and the BA scale-free networks [24,25]. Detailed descriptions of these networks can be found in [26,27]. The square roots of the running time in milliseconds are displayed in Fig. 3. The NC algorithm uses less memory than Dijkstra's algorithm but, as we can see from the figures, it is as efficient for all types of networks.

4. Further discussion

4.1. Implementation of finding the shortest paths

The NC algorithm can be easily implemented to find the shortest paths, not just the distances. Let $P_{sj}(1 \leqslant j \leqslant N)$ be the shortest path from the source node v_s to node v_j, u_{sj} be the second last node on P_{sj}. To record u_{sj}, we can declare a vector P with length of N, and initialize all the elements as s. If w_{sj} is updated in Step 3 ($w_{sj} \leftarrow w_{sk} + w_{kj}$), set $P(j) = k$.

When the NC algorithm terminates, P records the information of shortest paths between v_i and all the other nodes. To find the shortest path between v_s and v_j, we can trace from $P(j)$: if $u_{sj} = P(j) = k$, than $u_{sk} = P(k), \ldots$, till $P(k) = s$. The shortest path is:

$$s, \ldots, P(P(\cdots P(j))), \ldots, P(P(j)), P(j), j$$

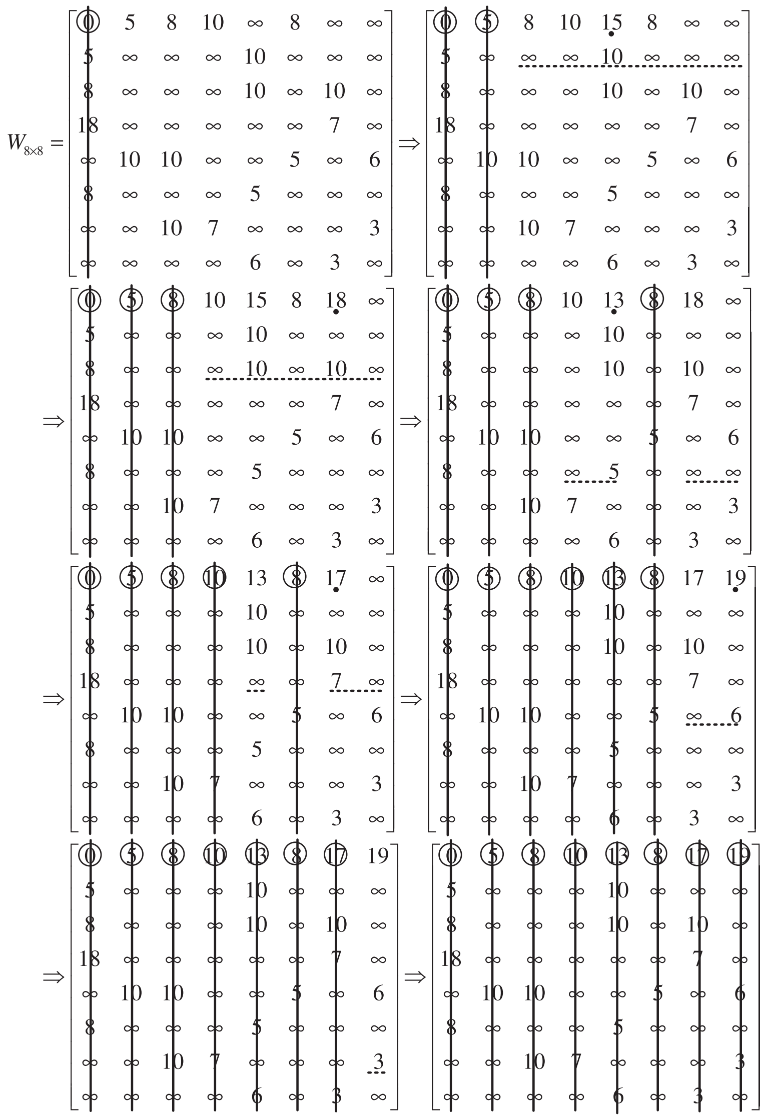

Fig. 4. Weight matrix implementation of NC algorithm.

4.2. Weight matrix implementation of NC algorithm

As has been done with Dijkstra's algorithm, we can implement the NC algorithm on the weight matrix to calculate distances from source node v_s to other nodes:

Step 0. Set $w_{ss} = 0$ and draw a circle on this element, cross out the sth column of the weight matrix $W_{N\times N}$.
Step 1. Circle the minimum element of the sth row. If all the elements in the sth row have been circled or the value of uncircled elements are ∞, stop.
Step 2. For each element w_{sj} which has not been circled in the sth row, update $w_{sj} = \min\{w_{sj}, w_{sk} + w_{kj}\}$, where w_{sk} is the minimum value found in Step 1. Cross out the kth column of $W_{N\times N}$. Go to Step 1.

When the iteration is finished, the values of elements in the sth row are the distances from v_s.

Using the network of Section 2.3, Fig. 4 illustrates how to calculate shortest paths on the weight matrix with the NC algorithm. Elements marked with a dot underneath are the edge weights modified by $w_{sj} = \min\{w_{sj}, w_{sk} + w_{kj}\}$ in Step 2, e.g., in the second matrix we have

$$w'_{1,5} = \min\{w_{1,5}, w_{1,2} + w_{2,5}\} = \min\{\infty, 5 + 10\} = 15.$$

Elements in row 1 of the last matrix are the distances of the shortest paths from node 1 to others (1 to 8): 0, 5, 8, 10, 13, 8, 17, and 19.

We can see from Fig. 4 that the operation of weight matrix using the NC algorithm to solve the SSSP problem is very simple; the elements to be modified are always in the row in which the source node stands (i.e., the sth row).

5. Conclusion

The NC algorithm finds the shortest path by node combination instead of by labeling operations. The difference between the NC algorithm and Dijkstra's algorithm is, first, the set of visited (solved) nodes whose distances have been established. In the NC algorithm, nodes are combined into the new source node, which means that we need not maintain this set. Second, the relaxation is done on the edge weight directly, which means that no additional memory or CPU-cycles are needed to record the temporary distances. Third, the NC algorithm is carried out by repeatedly finding the source node's nearest neighbor, which makes the process of finding shortest paths more comprehensible and vivid. Experimental evaluations reveal that with less memory cost, the NC algorithm finds the shortest paths in the same amount of time as Dijkstra's algorithm.

Implementation of the NC algorithm on weight matrices is also more convenient: both the minimum element to be found and the elements to be modified are in the same row. If the combination of nodes is implemented by setting edge weights to infinity, the NC algorithm can be conducted by vectors, which is suitable on mathematics platforms such as Matlab and other programming languages where the concept of infinity is implemented.

In conclusion, we have demonstrated an alternative way to understand Dijkstra's algorithm. Node combination makes the process of finding the shortest paths much more straightforward, comprehensible, and memory-sparing.

Acknowledgements

The authors thank B.J. Kim, F. Liljeros and S. Zhang for valuable discussions, and X. Lu acknowledges the support from the Innovation Foundation of National University of Defense Technology (Grant No. S070501).

References

[1] G. Gallo, S. Pallottino, Shortest path algorithms, Annals of Operations Research 13 (1) (1988) 1–79.
[2] D.B. West, Introduction to Graph Theory, second ed., Prentice Hall, 2001.
[3] D.Z. Du, P.M. Pardalos (Eds.), Network Optimization Problems: Algorithms, Applications and Complexity, World Scientific, Series on Applied Mathematics, vol. 2, 1993.
[4] P.M. Pardalos, M.G.C. Resende (Eds.), Handbook of Applied Optimization, Oxford University Press, 2002.
[5] E.W. Dijkstra, A note on two problems in connection with graphs, Numerische Mathematik 1 (1959) 269–271.
[6] R.E. Bellman, On a routing problem, Quarterly of Applied Mathematics 16 (1958) 87–90.
[7] L.R. Ford, D.R. Fulkerson, Flows in Networks, Princeton University Press, Princeton, 1962.
[8] E.F. Moore, The shortest path through a maze, in: Proceedings of the International Symposium on the Theory of Switching, Harvard University Press, 1959, pp. 285–292.
[9] R.W. Floyd, Algorithm 97: shortest path, Communications of the ACM 5 (6) (1962) 345.
[10] P. Festa, Shortest path tree algorithms, in: Encyclopedia of Optimization, second ed., Springer, 2009, pp. 3507–3519.
[11] S. Warshall, A theorem on boolean matrices, Journal of the ACM 9 (1) (1962) 11–12.
[12] G.B. Dantzig, All shortest routes in a graph, in: Theory of Graphs, International Symposium, Gordon and Breach, New York, 1967, pp. 91–92.
[13] T.H. Cormen, C.E. Leiserson, R.L. Rivest, C. Stein, Introduction to Algorithms, third ed., MIT Press, 2009.
[14] R.K. Ahuja, T.L. Magnanti, J. Orlin, Network Flows: Theory, Algorithms, and Applications, Springer, 1993.
[15] M.L. Fredman, R.E. Tarjan, Fibonacci heaps and their uses in improved network optimization algorithms, Journal of the ACM 34 (3) (1987) 596–615.
[16] B.V. Cherkassky, A.V. Goldberg, T. Radzik, Shortest paths algorithms: theory and experimental evaluation, Mathematical Programming 73 (2) (1996) 129–174.

[17] F.B. Zhan, Three fastest shortest path algorithms on real road networks: data structures and procedures, Journal of Geographic Information and Decision Analysis 1 (1) (2001) 69–82.
[18] R.H. Möhring, H. Schilling, B. Schütz, D. Wagner, T. Willhalm, Partitioning graphs to speed up Dijkstra's algorithm, in: Proceedings 4th International Workshop on Experimental and Efficient Algorithms (WEA), 2005, pp. 189–202.
[19] A.K. Ziliaskopoulos, F.D. Mandanas, H.S. Mahmassani, An extension of labeling techniques for finding shortest path trees, European Journal of Operational Research 198 (1) (2009) 63–72.
[20] M.H. Xu, Y.Q. Liu, Q.L. Huang, Y.X. Zhang, G.F. Luan, An improved Dijkstra's shortest path algorithm for sparse network, Applied Mathematics and Computation 185 (1) (2007) 247–254.
[21] R.E. Tarjan, Data structures and network algorithms, SIAM, Philadelphia, PA, 1983.
[22] P. Erdös, A. Rényi, On random graphs. I, Publicationes Mathematicae Debrecen 6 (1959) 290–297.
[23] D.J. Watts, S.H. Strogatz, Collective dynamics of 'small-world' networks, Nature 393 (6684) (1998) 440–442.
[24] A.L. Barabasi, R. Albert, H. Jeong, Mean-field theory for scale-free random networks, Physica A 272 (1–2) (1999) 173–187.
[25] R. Albert, H. Jeong, A.L. Barabasi, The diameter of the world wide web, Nature 401 (1999) 130–131.
[26] S. Boccaletti, V. Latora, Y. Moreno, M. Chavez, D. Hwang, Complex networks: structure and dynamics, Physics Reports-Review Section of Physics Letters 424 (4–5) (2006) 175–308.
[27] M.E.J. Newman, The structure and function of complex networks, SIAM Review 45 (2) (2003) 167–256.