SHORTEST PATHS IN SYNCHRONIZED TRAFFIC-LIGHT NETWORKS

Mohammad Khanjary [*,1], *Karim Faez* [**,2], *Mohammad Reza Meybodi* [**,3], *Masoud Sabaei* [**,4]

[*] Science and Research Branch, Islamic Azad University, Tehran, Iran
[**] Amirkabir University of Technology, Tehran, Iran
[1] khanjary@srbiau.ac.ir [2] kfaez@aut.ac.ir [3] mmeybodi@aut.ac.ir [4] sabaei@aut.ac.ir

ABSTRACT

The time-constrained shortest path problem is an important generalization of the shortest path problem. The basic feature in time-constrained shortest path problem is considering when a node in the network can be visited under some time constraints. In this paper, a label-setting shortest path algorithm will be proposed to use in the synchronized traffic-light networks which uses the waiting times for green light (node costs) as well as the required times to pass the streets (link costs) to calculate the optimal routes.

Index Terms—Time Constrained shortest path problem, traffic light networks, waiting time for green light, time window

1. INTRODUCTION

For a long time, finding shortest paths in graphs under different criteria and limitations has been one of the important problems which engaged many researchers both in academia and industry. Basically, the shortest path problem is concerned to find the path with minimum distance, time, or cost from an origin to a destination (or all other nodes) in a connected network e.g. [1]-[4]. Shortest path problem is an important and classic problem in the combinatorial optimization field due to its numerous applications especially in transportation and data communication. Some useful reviews on shortest path algorithms could be found in [5]-[6].

Recently, working on network problems with time constraints has been increased especially in different transportation applications such as shortest path problem (e.g. [7]), traveling salesman problem (e.g. [8]), vehicle routing problem (e.g. [9]), traffic network (e.g. [10]), and pickup and delivery problem (e.g. [11]). The time-constrained shortest path problem (TCSPP) is an especial generalization of the ordinary shortest path problem. The basic feature in TCSPP is considering when a node in the network can be visited under some time constraints. Time window is a common form of time constraints which defines the earliest and the latest time that a node is available (e.g. [12]).

In general, there are two types of time windows. The first is called *hard time window* where if one or more time window constraints are not satisfied, the route becomes infeasible (e.g. [13]). The second is called *soft time window* where a cost penalty is incurred if the arrival to a node is outside of its time window (e.g. [14]). When the *hard time window* is used, the goal is finding the least-cost path from the source node to the destination node while all intermediate nodes must be visited within their relevant time windows. When the *soft time window* is considered, the minimization of total cost is a goal too. In *soft time window*, the intermediate nodes may be visited outside their relevant time windows and the penalty of time window violation is an additional cost which will be considered in calculations [15]-[16].

In this paper, we are going to present a shortest path algorithm for synchronized traffic-light networks. Although, traditional *soft time window* appears to be a promising alternative for our research, two characteristics of traffic-light networks cannot be completely described by using the traditional *soft time window*: 1) The traffic light timings usually contain a repetitive ordered sequence of time windows with designated durations but common *soft time windows* only contain single time window. In other word, in traffic-light networks, every cycle of timing may contains several red, yellow and green light durations not only one duration. 2) In traditional soft time windows, one can pass the node if its arrival time falls into the range of time windows but in the traffic-light networks, each time window only allows vehicles in some particular directions to pass the intersection [15]. Therefore, another type of time windows must be used for traffic-light networks.

There are only two algorithms for considering the waiting times for green light (WTGL) in calculation of optimal routes in traffic-light networks. In first one, Chen and Yang [15] proposed an algorithm with time complexity of $O(m \log m + mn \log r + rn^3) \approx O(rn^3)$ and space complexity of $O(n^2 + rn^3) \approx O(rn^3)$, where m denotes the number of arcs in the network, n denotes the number of nodes in the network and r denotes the maximum number of time windows in a node. In this paper, firstly a suitable data structure has been designed to find the WTGL in different time windows and then an algorithm has been proposed

CCECE 2011 Niagara Falls, Canada
978-1-4244-9789-8/11/$26 ©2011 IEEE

based on it. If the time windows of all nodes (intersections) are available as well as costs of links (streets), the algorithm will find the shortest path from source node to destination node. Afterwards, the authors of this paper expanded their algorithm to solve the same problem with other limitations and criteria [16]-[20].

In second algorithm [21], Miller-Hooks and Yang proposed a label-correcting algorithm to consider WTGL in calculation of optimal routes when both street travel times and traffic lights timings vary over time and are known only probabilistically (by using some probability functions). It has been proved that the algorithm has time complexity of $O(n^4T^2K)$ where n is the number of nodes, T is the number of departure time intervals and K is the number of possible travel times for each departure time. This algorithm is based on probability functions and does not use the time windows, therefore, when link costs and node timings are available it does not work. Since, we are going to present a general shortest path algorithm for synchronized traffic-light networks based on time windows, the Chen and Yang algorithm [15] will be considered to compare with our algorithm.

Our algorithm is a label-setting shortest path algorithm to be used in synchronized traffic-light networks. We model the different traffic light timings as repetitive ordered sequence of time windows and classify them into limited number of classes. Then, we assume that all traffic lights with similar class of timing are synchronized and we will use the timers to simulate each class. By knowing the class of timing of different routes in each intersection, the current time of timers will be used to determine the WTGL for the time when a vehicle reaches to an intersection. We will prove the algorithm and show that the proposed algorithm has lower time and space complexity as compared to Chen and Yang algorithm.

The rest of this paper is organized as follows. Section 2 presents the proposed algorithm and section 4 concludes the paper.

2. THE ALGORITHM

Let N = (V, A, TLT, $class$, $time$, s, d) denotes a synchronized traffic-light network, where V denotes the nodes (intersections) set, A denotes the arcs (streets) set without multiple arcs and self-loops, TLT denotes the set of classes of different traffic light timings which are used in the network, $class[i][j][k]$ denotes the assigned class of TLT for when a vehicle arrives to node j from node i and the next node is k, $time(u, v)$ denotes the required time to pass the $arc(u, v) \in A$ and the goal is finding the shortest path from node s to node d.

In fact, every TLT includes a repetitive ordered sequence of time windows which will be used to control the traffic lights. Therefore class i of TLT is defined as follows: $TLT[i] = \{GLD[1], RLD[1], GLD[2], RLD[2], ..., GLD[r], RLD[r], TD, ST, CT\}$ where $GLD[k]$ represents the kth green light duration (the time length of kth green light), $RLD[k]$ represents the kth red light duration (the time length of kth red light), TD represents the total duration of a cycle ($TLT[i].TD == TLT[i].GLD[1] + TLT[i].RLD[1] + TLT[i].GLD[2] + TLT[i].RLD[2] + ... + TLT[i].GLD[r] + TLT[i].RLD[r]$), ST represents the start time of this class and CT represents the current time of timer which has been assigned to this class. In this definition, r is the maximum number of green and red light time windows in a single cycle among all traffic light timings which are used in the traffic network. Notice that 1) when total duration of a class is passed (after $TLT[i].RLD[r]$), the sequence of time windows will be restarted from first time window ($TLT[i].GLD[1]$), 2) if number of used green and red durations was less than r in a timing, the unused durations will be set to zero.

All different traffic light timings of all over the traffic network must be distinguished and classified into limited number of classes. Afterward, a relevant class of TLT will be assigned to every possible route in intersections. To see what means "every possible route in intersections", consider Fig. 1 which shows a 4-way intersection.

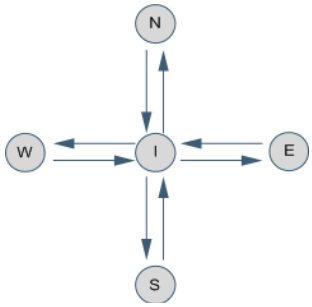

Fig. 1. Possible routes at a 4-way intersection.

According to Fig. 1, when a vehicle comes to node I from node S can go to node N, node E, node W and also can return to the node S. Notice that if the traffic rules forbids turning to some directions in an intersection, the red light duration of those routes will be set to infinity. Therefore, when a one vehicle comes to node I from node S, if we assume that turning back to the node S is forbidden, the possible routes will be <S, I, W>, <S, I, N> and <S, I, E>. Similarly, this could be considered for when a vehicle comes from N, E or W to node I. As an example, according to definition of the routes $class[S][I][N] = j$ means the ordered sequence of times windows according to $TLT[j]$ will be used for traffic light timing when a vehicle comes to node I from node S and next node is N.

Notice that since these time windows are repetitive, by assuming $GLD[0] == GLD[r]$ and $RLD[0] == RLD[r]$, we have the relationship that $GLD[(k \times r) + i] == GLD[i]$ and $RLD[(k \times r) + i] == RLD[i]$ for any nonnegative integers k and i, where $i \leq r$. According to the definition of TLT, r is the maximum number of time windows.

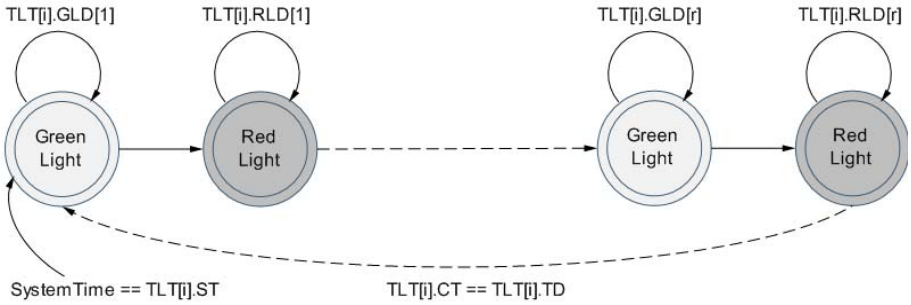

Fig. 2. State transition diagram for TLT[*i*].

2.1. Timers and State Transition Diagram

To execute the algorithm in a distributed manner, every class of TLT will be simulated by a timer (CT) and a state transition diagram. In the first running, the timer will be started according to the start time (*ST*) of the class which belongs to and will be increased one-by-one every second. The maximum value of a timer is the total duration (*TD*) of the class and timer restarts from zero when it arrives to its *TD*. To process a timer for finding the waiting times, a state transition diagram will be used. The state transition diagram for class *i* of TLT has been shown in Fig. 2.

According to Fig. 2, when the current time of the system is equal to start time of class i of TLTs (*SystemTime* == *TLT[i].ST*), the state transition diagram will be started with first green light duration. The timer will be stayed in first green light for *TLT[i].GLD[1]* seconds and then moves to the first red light duration. Again, it will be stayed in this state for *TLT[i].RLD[i]* seconds and then moves to the second green light. This movement will be continued till when timer enters to the last red light duration. It will stay in that state for *TLT[i].RLD[r]* seconds and afterward returns to the first state (first green light) and, this cycle will be repeated. Notice that for those classes of timing which do not need to use all *r* states e.g. 2-states timings which use only one green light duration and one red light duration, duration of unused green and red light states will be set to 0.

2.2. Pseudo Code

In ordinary shortest path algorithms which only arcs have cost, in every step the node with smallest value of path length will be selected, removed from unvisited (temporary) nodes and labeled as a visited (permanent) node and then, this node will be used to extract the current found optimal paths. But when nodes have cost too, such approach does not work. Because in different times, nodes have different costs and all possible combination of arcs must be considered to find the real shortest path from source node to other nodes.

Therefore, we assign the length (cost) of paths to the arcs instead of nodes. In our algorithm, *path_length[v][w]* means the length of path from source node to the node *w* through the *arc(u, w)*. This means that it is possible to reach a node with different length when last arcs are different. In every step, the TEMPORARY arc with minimum *path_length* will be selected and labeled PERMANENT. Then, the algorithm will extend this arc if the paths which will be created based on this arc are shorter than previous ones. This will be continued till the selected TEMPORARY arc with minimum path length ends to the destination node. The TCSPP function has been defined between line 1 and 17 of the following algorithm.

```
1  function TCSPP(Graph G, Vertex source, Vertex destination)
2    for each vertex i and j in Graph G
3      path_length[i][j] = INFINITY
4      predecessor[i][j] = UNDEFINED
5      label[i][j] = TENTATIVE
6    path_length[0][source] = 0
7    while any arc with TENTATIVE label exists in G
8      choose arc(v,w) with minimum path_length
       and TENTATIVE label
9      label[v][w] = PERMANENT
10     if (w == destination)
11       return predecessor[v][w]
12     for each arc[w][x]
13       temp = path_length[v][w]
                  + WTGL(v, w, x, path_length[v][w]) + time[w][x]
14       if temp < path_length[w][x]
15         path_length[w][x] = temp
16         predecessors[w][x] = (v,w)
17   return "There is no route from source to destination"
18
19 function WTGL(Vertex i, Vertex j, Vertex k, Time delay)
20   index = class[i][j][k]
21   marker = (delay + TLT[index].CT) MOD TLT[index].TD
22   sum = 0
23   for (k from 1 to r)
24     if marker <= (sum + TLT[index].GLD[k])
25       return 0
26     else
27       if marker <= (sum + TLT[index].GLD[k] +
                TLT[index].RLD[k])
28         return (sum + TLT[index].GLD[k] +
                TLT[index].RLD[k]) - marker
29     sum = sum + TLT[index].GLD[k] + TLT[index].RLD[k]
```

As seen in line 13, the waiting time for green lights will be considered to calculate the path lengths by using WTGL function and then the comparison between current path and pervious path will be done. Indeed, the WTGL function processes the state transition diagram of Fig. 2. In WTGL function, *i*, *j* and *k* parameters represent the pervious, current and next node respectively and *delay* parameter is the time that the vehicle will reach to node *j*. In other word, *delay* is total time which is expected to take from the source node to the current node (*j*).

To find the waiting times for green light when a vehicle comes to node *j* from node *i* and next node is *k* in time equal to *delay*, firstly the cycle times of timings must be eliminated from *delay*. To do that, the *delay* parameter will be divided to total duration of relevant class of TLT (*TLT[index].TD*) as seen in line 21 and the modulus (remainder) of this division will be assigned to *marker*. Then, in a loop with *r* iterations, the state transition diagram will be processed. If the *marker* is in green light state, zero second otherwise the remaining time to move from red light state to green light state will be returned. The WTGL() function has been defined between lines 19 and 29 of the algorithm.

2.3. Proof

Let N = (*V*, *A*, *TLT*, *class*, *time*, *s*, *d*) denotes a traffic-light network, where *V* denotes the node set (with time windows constraints), *A* denotes the arc set (without multiple arcs and self loops), *TLT* denotes set of traffic light timings classes where maximum number of them is *c* and maximum number of time windows in each *TLT* is *r*, *class* denotes the assigned class of *TLT* to different routes in intersections, *time(v, u)* denotes the required time to pass the *arc(v, u)* $\in A$, *s* denotes the source node and *d* denotes the destination node. Also, let assume that *arc(v, u)* is the link from node *v* to *u* and *path_length(v, u)* is the total time of the shortest path from source node to node *u* through the *arc(v, u)*. Then, our goal is finding a shortest path from node *s* to node *d* in N where the nodes are constrained by synchronized and classified traffic lights.

We prove the algorithm by induction. At each iteration, the algorithm partitions all arcs into two sets, called *O* and $\bar{O}$, where $\bar{O} = A - O$. Set $\bar{O}$ includes the arcs which have been labeled PERMANENT and consequently set *O* includes the arcs with label TENTATIVE. The induction hypotheses are found on two premises: 1) the *path_length(v, u)* of each *arc(v, u)* in $\bar{O}$ is optimal and 2) the *path_length(v, u)* of each *arc(v, u)* in *O* is the total time of the shortest path from *s* to *u* through *arc(v, u)* provided that each intermediate arc in the path lies in $\bar{O}$ [15].

To prove hypothesis 1, remind that at the beginning of iterations, the algorithm moves an *arc(v, u)* with the smallest *path_length* from set *O* to set $\bar{O}$. We must show that *path_length(v, u)* of *arc(v, u)* is optimum. Notice that by our induction hypothesis, *path_length(v, u)* is the total time of a shortest path to node *u* thought *arc(v, u)* among all paths that does not contain any intermediate arc in *O*. We now show that the total time of any path from node *s* to node *u* through *arc(v, u)* that contains some arcs in *O* as an intermediate arcs will be at least *path_length(v, u)*.

Consider any path *P* from the source node to node *u* through *arc(v, u)* which contains at least one arc in *O* as an intermediate arc. The path *P* can be decomposed into two segments P_1 and P_2, where P_1 does not contain any arc in *O* as an intermediate arc but the last arc, say *arc(h, k)*. According to the induction hypotheses, the total time of P_1 is at least *path_length(h, k)*. Moreover, since *arc(v, u)* has the smallest *path_length* in *O*, *path_length(h, k)* $\geq$ *path_length(v, u)*. Therefore, the path segment P_1 has total time equal to at least *path_length(v, u)*. Furthermore, since all arc times are nonnegative, the total time of the path segment P_2 is nonnegative. Consequently, the total time of path *P* is no less than *path_length(v, u)*. This result confirms that *path_length(v, u)* is the shortest path from the source node to the node *u* through the *arc(v, u)*. Therefore, the hypothesis 1 has been proved.

To prove the hypothesis 2, notice that after labeling an *arc(v, u)* PERMANENT, the *path_length* of some arcs in $O - \{arc(v, u)\}$ may decrease since *arc(v, u)* could become an intermediate arc in the temporary shortest paths to these arcs. After labeling an *arc(v, u)* PERMANENT, the algorithm examines all emanating *arc(u, w)* from node *u* as follows:

path_length(u, w) = path_length(v, u)
+ WTGL(v, u, w, path_length(v, u)) + time(u, w)
if
path_length(v, u) + WTGL(v, u, w, path_length(v, u))
+ time(u, w) < path_length(u, w)

Therefore, after the update operation, the *path_length(u, w)* of each *arc(u, w)* in $O - \{arc(v, u)\}$ is the total time of a shortest path from node *s* to node *w* through *arc(u, w)* with this restriction that each intermediate arc in the path belongs to $\bar{O} \cup arc(v, u)$. □

2.4. Time and Space Complexity of the Algorithm

As it was explained in section i, there are two proposed algorithms in this field. The first one is based on probability functions and the second one is based on time windows. Since our algorithm is based on time windows too, we compare our algorithm with chen and yang algorithm [15].

In proposed algorithm, there are two key operations: In line 8, we need to find the arc with minimum path length and TENTATIVE label and, label it as a PERMANENT arc which will be called EXTRACT-MIN. In line 15, we need to update the value of arcs which smaller values have been found for them in the recent iteration which will be called DECREASE-KEY. If as Yang and Chen [15] Fibonacci heap [22] is used, the time complexity of each EXTRACT-MIN operation will be *O(log m)* and the time complexity of

each DECREASE-KEY operation will be $O(1)$.

Therefore, the time complexity of EXTRACT-MIN operation in line 8 is $O(m \log m)$ and the time complexity of DECREASE-KEY operation in line 13 is $O(mnr)$ where r denotes the maximum number of time windows in a *TLT*, m denotes the number of arcs and n denotes the number of vertex in the network. Consequently, the total time complexity of the algorithm is $O(m \log m + mnr) \approx O(mnr)$. To calculate the space complexity of the algorithm, consider that every node needs space of order $O(n^2)$ for storing routes, labels, arc lengths and path lengths, $O(n^3)$ for storing classes and $O(rc)$ for storing the timings of different classes (*TLT*). Notice that c denotes the maximum number of classes. Consequently, the final space complexity of the algorithm is $O(n^2 + n^3 + rc) \approx O(n^3)$. Table 1 compares the time and space complexities of proposed algorithm with Yang and Chen Algorithm. As seen, the proposed algorithm has less time and space complexity.

TABLE I.
COMPARISON OF TIME AND SPACE COMPLEXITY BETWEEN YANG AND CHEN ALGORITHM AND PROPOSED ALGORITHM

Algorithm	Time Complexity	Space Complexity
Yang and Chen [15]	$O(m \log m + mn \log r + rn^3)$	$O(n^2 + rn^3)$
Faez and Khanjary	$O(m \log m + mnr)$	$O(n^2 + n^3 + rc)$

3. CONCLUSION

In this paper, a label-setting shortest path algorithm was proposed to consider the waiting times for green light (nodes cost) as well as required time to travel the streets (arcs cost) in calculation of optimal routes in synchronized traffic-light networks. The proposed algorithm uses classified traffic light timings (*TLT*) and simulates different timings by using timers. The proposed algorithm has less time and space complexities as compared to similar algorithms. As further work, finding shortest paths at presence of different criteria for arc cost and node cost could be considered.

4. REFERENCES

[1] A. Sedeño-Noda, C. González-Martín, "On the K shortest path trees problem," *Elsevier European Journal of Operational Research*, vol. 202, no. 3, pp. 628-635, 2010.

[2] M.H. Farahi, M. Zamirian, A.R. Nazemi, "An applicable method for solving the shortest path problems," *Elsevier Applied Mathematics and Computation*, vol. 190, no. 2, pp. 1479-1486, 2007.

[3] J. B. Orlin, K. Madduri, K. Subramani, M. Williamson, "A faster algorithm for the single source shortest path problem with few distinct positive lengths," *Elsevier Journal of Discrete Algorithms*, vol. 8, no. 2, pp. 189-198, 2010.

[4] D. Villeneuve, G. Desaulniers, "The shortest path problem with forbidden paths," *Elsevier European Journal of Operational Research*, vol. 165, no. 1, pp. 97-107, 2005.

[5] N. Deo, C. Pang, "Shortest path algorithms: Taxonomy and annotation," *Wiley Networks*, vol. 14, no. 2, pp. 275−323, 1984.

[6] R.K. Ahuja, T.L. Magnanti, J.B. Orlin, "Network Flows: Theory, Algorithms, and Applications," *Prentice-Hall*, Englewood Cliffs, NJ, 1993.

[7] J. M. Hill, H. D. Sherali, "Reverse time-restricted shortest paths: Application to air traffic management," *Elsevier Transportation Research Part C*, vol. 17, no. 6, pp. 631−641, 2009.

[8] S. Urrutia, R. F. Da Silva, "A General VNS heuristic for the traveling salesman problem with time windows," *Elsevier Discrete Optimization*, 2010, In Press.

[9] D. Feillet, T. Garaix, D. Josselin, C. Artigues, "Vehicle routing problems with alternative paths: An application to on-demand transportation," *Elsevier European Journal of Operational Research*, vol. 204, no. 1, pp. 62−75, 2010.

[10] L.R. Rilett, L. Fu, "Expected shortest paths in dynamic and stochastic traffic networks," *Elsevier Transportation Research Part B*, vol. 32, no. 7, pp. 499−516, 1998.

[11] P. Recht, A. Fabri, "On dynamic pickup and delivery vehicle routing with several time windows and waiting times," *Elsevier Transportation Research Part B*, vol. 40, no. 4, pp. 335−350, 2006.

[12] J. Albiach, D. Soler, E. Martínez, "A way to optimally solve a time-dependent Vehicle Routing Problem with Time Windows," *Elsevier Operations Research Letters*, vol. 37, no. 1, pp. 37−42, 2009.

[13] W. T. Ooi, Y. W. Wan, T. S. Chang, "A stochastic dynamic traveling salesman problem with hard time windows," *Elsevier European Journal of Operational Research*, vol. 198, no. 3, pp. 748−759, 2009.

[14] T. Yamada, E. Taniguchi, A. G. Qureshi, "An exact solution approach for vehicle routing and scheduling problems with soft time windows," *Elsevier Transportation Research Part E*, vol. 45, no. 6, pp. 960−977, 2009.

[15] Y. Chen, H. Yang, "Shortest paths in traffic-light networks," *Elsevier Transportation Research Part B*, vol. 34, no. 4, pp. 241−253, 2000.

[16] H. Yang, Y. Chen, "Finding K shortest looping paths with waiting time in a time-window network," *Elsevier Applied Mathematical Modeling*, vol. 30, no. 5, pp. 458−465, 2006.

[17] H. Yang, Y. Chen, "Minimization of travel time and weighted number of stops in a traffic-light network," *Elsevier European Journal of Operational Research*, vol. 144, no. 3, pp. 565−580, 2003.

[18] H. Yang, Y. Chen, "Finding the first K shortest paths in a time-window network," *Elsevier Computers & Operations Research*, vol. 31, no. 4, pp. 499−513, 2004.

[19] H. Yang, Y. Chen, "The First K Shortest Unique-Arc Walks in a Traffic-Light Network," *Elsevier Mathematical and Computer Modelling*, vol. 40, no. 13, pp. 1453−1464, 2004.

[20] H. Yang, Y. Chen, "Finding K shortest looping paths in a traffic-light network," *Elsevier Computers & Operations Research*, vol. 32, no. 3, pp. 571−581, 2005.

[21] B. Yang, E. Miller-Hooks, "Adaptive routing considering delays due to signal operations," *Elsevier Transportation Research Part B*, vol. 38, no. 5, pp.385−413, 2004.

[22] M. L. Fredman, R. E. Tarjan, "Fibonacci heaps and their uses in improved network optimization algorithms," Journal of ACM, vol. 34, no. 3, pp. 596−615, 1987.