Improvement of Thorup Shortest Path Algorithm by Reducing the Depth of A Component Tree

Yusi Wei and Shojiro Tanaka

***Abstract*—Wei and Tanaka have proposed a variant of the Thorup algorithm which showed a better result than the original Thorup algorithm and the Fibonacci-based Dijkstra algorithm in practice. In this paper, we propose a faster algorithm based on their work. Our new algorithm has a faster speed when visiting vertices, it is achieved by decreasing the depth of a component tree. The experimental result indicates, comparing to array-based Dijkstra, Fibonacci-based Dijkstra and the original Thorup algorithm, reduction by 7.5%, 72.9% and 85.6% of the time cost, respectively.**

***Index Terms*—Dijkstra, single-source shortest path, thorup, undirected weights.**

I. Introduction

Wei and Tanaka [1] have proposed a variant of the Thorup algorithm which reduces the time cost of pre-indexing and calculating shortest paths in practice. It is proven that their algorithm showed a better result than the original Thorup algorithm and the Fibonacci-based Dijkstra algorithm in practice, but still slower than the array-based Dijkstra algorithm. In this paper, we will propose a faster algorithm based on their algorithm further. Our algorithm enhances the performance of visiting vertices by reducing the depth of the structure used. The experiment result indicates, our algorithm is faster than thearray-based Dijkstra, the Fibonacci-based Dijkstra and the original Thorup algorithm.

Definitions and background of this paper are described in the rest part of this Section. Related works are introduced in Section II, and then we will review Wei and Tanaka's algorithm (named The Improved Thorup Algorithm) in Section III. Our new algorithm is introduced in Section IV. Section V includes the introduction of the experiment and analysis.

The single source shortest path (SSSP) is the problem of finding the shortest path from a source vertex to every other vertex. It has been applied in many fields such as navigation [2], keyword searching [3], computer network [4], and it is widely used for finding the optimal route in a road network. SSSP problem is described as follows throughout this research. Given a graph $G = (V, E)$ and a source vertex $s \in V$, suppose s can reach each vertex of the graph, then find the shortest path from s to every vertex $v \in V$, in which V and E represent the vertices and edges of G [5]. The m and n mentioned in the rest of this paper represent $|E|$ and $|V|$, respectively. Use $D(v)$ to represent the tentative distance from the source vertex to v, and use $d(v)$ to represent the ensured shortest distance from the source vertex to v, and let $W(v, w)$ represent the positive integer weightsof edge(v, w). At the beginning, $D(v) = \infty$ for every vertex except the source vertex, $d(s) = 0$. The length of a shortest path should be the sum of the weights of each edge on the shortest path.

One of the most influential shortest path algorithms is Dijkstra [5]-[7] which is proposed in 1959. Yen's paper [8] is considered to be the first one which implements Dijkstra with an array. In detail, an array is used to record the tentative distance of each vertex which is adjacent to visited vertices. Every time when a vertex is visited, the distances of the vertices which are adjacent to the vertex will be recorded in an array, and then, the vertex which has the shortest distance will be taken to try to relax the vertices which adjacent to this vertex. The word relax is described as follows. A vertex, say *A*, can relax another vertex, say *B*, means the distance from the source vertex to B can be shortened through A. It takes $O(1)$ time to insert a relaxed vertex and $O(n)$ time to delete a vertex from an array. To relax all vertices, Dijkstra algorithm runs in $O(m)$ plus the time of maintaining the array, overall, it costs $O(m + n^2)$.

Thorup [9] is an algorithm which theoretically proved that solving the SSSP problem in linear time with pre-processed indices. The paper proposed a hierarchy- and buckets-based algorithm to pre-process indices for performing shortest path calculations in undirected graphs with non-negative weights. Theoretically, this algorithm constructs the minimum spanning tree in $O(m)$, constructs the component tree in $O(n)$, constructs an unvisited data structure in $O(n)$, and calculates distances of all vertices based on the constructed structures in $O(m + n)$. But in practice, Thorup algorithm occasionally does not perform as expected according to the experimental result provided by Asano and Imai [10], and Pruehs [11].

II. Related Works

Asano and Imai [10], and Pruehs [11] implemented Thorup algorithm with their modifications. Asano and Imai [10] uses an array to realize the function of atomic heaps, which is used in the Thorup algorithm; uses the union with size and find with path compression algorithm instead of the union and find algorithm [12] to construct the component tree, and designs a three-levels tree as an unvisited data structure to maintain the tentative distance of each vertex.

Pruehs [11] uses Kruskal's algorithm [13] to generate a

Mancscript received Novemeber 20, 2013; revised January 6, 2014.

Yusi Wei is with the Information Systems Course, Interdisciplinary Graduate School of Science and Engineering, Shimane University, Matsue, Japan (e-mail: wayis@ live.com).

Shojiro Tanaka is with the Information Systems Division, Interdisciplinary Graduate School of Science and Engineering, Shimane University, Matsue, Japan(e-mail: tanaka@cis.shimane-u.ac.jp).

DOI: 10.7763/JACN.2014.V2.99

minimum spanning tree, and uses Tarjan's union-find algorithm [14] to construct the component tree. Then it uses Gabow's Split-findmin data structure [12] instead of atomic heaps to maintain the tentative distance of each vertex.

Hagerup [15] proposed an improved hierarchy-based algorithm which theoretically solves SSSP problem in $O\ (m \log\log C + n \log\log n)$, and also solves all-path shortest path (APSP) problem in $O(mn + n^2 \log\log n)$, where *C* represents the maximum edge weights. Pettie [16] proposed a hierarchy-based algorithm which theoretically solves APSP problems in $O\ (mn + n^2 \log\log n)$, and proved that no SSSP algorithm can solve the problem in $\Omega(m + n \log n)$.

Wei and Tanaka [1] proposed a variant of the Thorup algorithm which enhances the performance of the original algorithm by using component tree instead of unvisited structure to maintain the tentative distances, so that to accelerate the algorithm by reducing the number of the structures used. The detailed mechanism will be introduced in Section III, which is related to this paper.

III. The Improved Thorup Algorithm

Since the method of using both of the component tree and the unvisited structure will reduce the efficiency of the algorithm in practice, the improved Thorup algorithm maintains the tentative distances with a modified component tree, for avoid using the unvisited structure. This method reduces the time cost of pre-index and query shortest path.

Same as the original algorithm, the improved Thorup algorithm is a hierarchy- and buckets-based algorithm which preprocesses indices for accelerating the performance of queries in undirected graphs with non-negative weights. It consists of two phases: construction of index and calculating shortest path by visiting vertices. In phase one, it firstly constructs a minimum spanning tree [14], and then constructs a component tree in linear time based on the constructed minimum spanning tree. In a component tree, leaf-component always contains only one vertex. Non-leaf components are created by a number of connected vertices, in which their weights are smaller than 2^i, where i represents the level of the component tree that increases from 0 to $n, 2^n >$ the largest weights of *G*. Leaf-components are on level 0. Level 1 includes the components created by the vertices which the weights between them are smaller than 2^1, Level 2 includes the components created by the vertices which the weights between them are smaller than 2^2, and so forth. In the second phase, the tentative distance of each reached component will be mapped to its parent's buckets, for deciding the order to visit the vertices in different components. This hierarchy- and buckets-based method overcomes the sorting bottleneck of priority queue-based algorithms. Since when trying to get the element which has the smallest value, currently, there is not any method can sort the values in linear time. This problem reduces the performance of the Dijkstra algorithm every time when extracting the vertex with the minimum tentative distance from a priority queue.

Dinic [17] has demonstrated that, if Δ is the minimum edge weights, bucketing vertices tentative distances by $\lfloor D(v)/\Delta \rfloor$, then in Dijkstra's algorithm, we can visit vertices in the right order according to their positions in these buckets. Thorup's algorithm splits a graph into many small graphs, then using Dijkstra's algorithm and Dinic's method to finish whole problem. The size of buckets of each component *c* is calculated by dividing the interval $2^{c.\text{level}}$ into the total length of the edges in the same component. The formula is listed as follows, $W(edge)$ represents the weights of *edge*,

$$\left\lceil \sum_{edge \in c} W(edge)/2^{c.\text{level}\ -1} \right\rceil$$

For deciding the order of visiting vertices, except the root, each component should be mapped to its father component's buckets depends on its minimum tentative distance. A minimum tentative distance of a component is the smallest tentative distance among all the vertices in the component. It is retrieved by using a pre-constructed heap, which is called unvisited structure in Thorup. The unvisited structure is implemented with the algorithm called split-findmin [12].To map a sub-component, say *c*, to its father, we need to get the index of the bucket which stores component *c* with *c*'s tentative distance ($D(c)$) by the formula as follows,

$$D(c)\ >>\ (c.\text{level}\ -1)$$

Since unvisited structure used in the original Thorup algorithm reduces the efficiency in practice, the improved Thorup algorithm modified the component tree to make it able to maintain tentative distances, so that avoid of using the unvisited structure. This change has two following benefits,

1) Save the time cost of constructing the unvisited structure, which is in the first phase of the Thorup algorithm.
2) Accelerate the process of calculating shortest paths, which is in the second phase of the Thorup algorithm.

Two variables should be added to each component of a component tree,

1) *distance*, which is used to record the tentative distance of each component. The distance of a father component is equal to the minimum tentative distance of all of its children.
2) *expanded*, which is used to mark that whether the tentative distance of a component is not needed to be updated. It happens when we start to bucket the component's children.

All information of tentative distances is stored in the component tree. Since the vertices are indexed by their IDs. When updating the tentative distance of a vertex, it will be found in $O(1)$. The updated value should be reported to all of the vertex's unexpanded father components. If the value is smaller than the current father component's value, update the value of it too.

IV. The New Improvement

Our new algorithm is based on the variant of the Thorup algorithm introduced in Section III. The improvement intends to reduce the depth of a component tree. In Section III, we have studied that the components are created at different levels depends on the weights of the edges. The edges will be

included in the same component if their weights are greater than 2^i and smaller than 2^{i+1}, ($0 \leq$ i). In the real world, the information of roads could be enormous and various. This method might create a component tree with high depth. This will reduce the advantage provided by using buckets. To visit components frequently through such a structure is inefficient.

Accordingly, we try to reduce the depth of the component tree by extending the limitation for edges' weights of components in different levels. That is, the edges which their weights are greater than $base^i$ and smaller than $base^{i+1}$ will be accumulated in the same component, $base$ should be a number which is power of 2, such as 4, 8, 16 and so forth.

```
1.  Visit(v)
2.  Set j= the word length if v is the root of the tree.
    Otherwise let j equal to v's parent's level.
3.  if v is a leaf-component of the component tree then
4.      VisitLeaf(v)
5.      Remove v from the bucket of v's parent
6.      return
7.  end if
8.  if v has not been visited previously then
9.      Expand(v)
10.     v.ix = v.ix0
11. end if
12. repeat until v has no child or v.ix >> ((j −
    v.level) * log2 base) is increased
13.     while the bucket B[v.ix] is not empty
14.         let wh equal to the component in bucket B
            [v.ix]
15.         Visit(wh)
16.     end while
17.     v.ix = v.ix + 1
18. end repeat
19. if v has any child then
20.     move v to bucket B[v.ix >> ((j − v.level) *
        log2 base)] of v's parent
21. end if
22. if v do not has child and v is not the root of the
    component tree then
23.     remove v from the bucket of v's parent
24. end if
```

Fig. 1. Algorithm of visit.

```
1. Expand (v)
2. v.ix0 = v.distance ≫ ((v.level − 1) * log2 base)
3. v.expanded= TRUE
4. for each child wh of v
5.     store wh in bucket B[wh.distance >> ((v.level −
       1) * log2 base)]
6. end for
```

Fig. 2. Algorithm of expand.

The *base* should not be restricted, but changes along the sizes of different graphs. The bucket size of each component c is then increased to,

$$\left\lceil \sum_{edge \in c} W(edge)/base^{\text{c.level}\ -1} \right\rceil$$

And we also need to right-shift $\log_2 base$ times to calculate the positions of components in their father components' buckets. The formula used to calculate the index of the bucket which stores component c with c's tentative distance ($\mathrm{D}(c)$) is accordingly changed as follows,

$$\mathrm{D}(c) \ >> \ (c.\text{level} - 1) * \log_2 base$$

Our algorithm is shown from Fig. 1 to Fig. 4, starting from the one named Visit (Fig. 1). Since the base is changed from 2 to a number which is power of 2, when manipulating buckets, the right-shift times is also increased in our algorithm.

```
1. VisitLeaf(v)
2. for each vertex w connected with v, if v.distance +
   W(v,w) < w.distance
3.     Let wh be the unvisited root of leaf w
4.     Let wi be the unvisited parent of wh
5.     Decrease(w, v.distance + W(v,w))
6.     if this decreases wh.distance >> ((v.level −
       1) * log2 base) then
7.         Move wh to bucket B[wh.distance >>
           ((v.level − 1) * log2 base)] of wi
8.     end if
9. end for
```

Fig. 3. Algorithm of visitleaf.

```
1. Decrease (v, newValue)
2. if v.distance > newValue and v.expanded != TRUE
   then
3.     v.distance=newValue
4.     let n to be the parent of v
5.     while n. expanded!= TRUE and n. distance >
       newValue
6.         n.distance=newValue
7.         let n to be the parent of n
8.     end while
9. end if
```

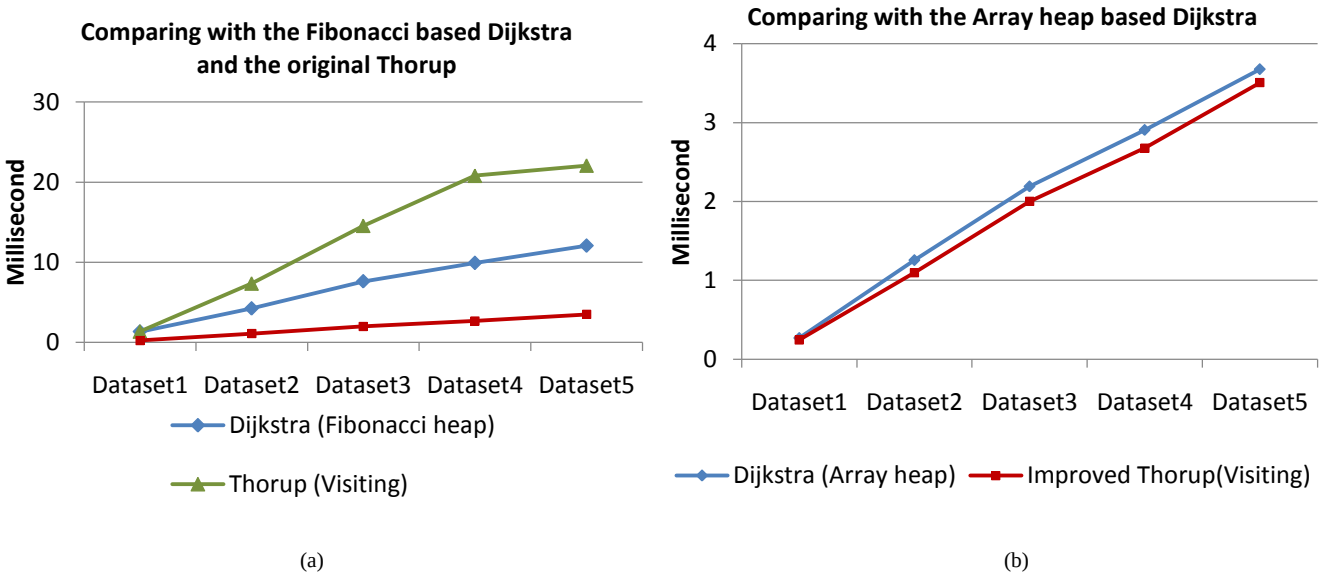
Fig. 4. Algorithm of decrease.

V. Experiment

We use the same method and datasets with [1]and added our algorithm to the experiment to evaluate the performance. The value of *base* is set to 16.The experiment is originally proposed by Pruehs [11] and modified by Wei and Tanaka [1] to make it possible to run with real datasets. The experiment compared the performance of Dijkstra and Thorup algorithm with real datasets. The time cost of finding the distance of every vertex to a given source vertex is compared among the following three algorithms: Dijkstra with array-based primary queue, Dijkstra with Fibonacci-based primary queue, and Thorup.

TABLE I: RESULT OF EXPERIMENT (MILLISECOND)

Algorithms\Datasets	1	2	3	4	5
Array heap based Dijkstra	0.2711	1.2571	2.1913	2.9065	3.6762
Fibonacci heap based Dijkstra	1.3535	4.2393	7.6227	9.9158	12.0687
Thorup Construct Structures	0.9035	3.3064	5.8803	7.7166	9.3685
Thorup Visiting	1.3887	7.3706	14.5664	20.8024	22.0718
Improved Thorup Construct Structures	0.8546	3.2951	5.6091	7.1455	9.1381
Improved Thorup Visiting	0.2467	1.0990	2.0009	2.6768	3.5072



(a) (b)

Fig. 5. Chart of the results. (a) Comparing with the Fibonacci based Dijkstra and the original Thorup. (b) Comparing with the array heap based Dijkstra.

The datasets we used originally comes from the Geospatial Information Authority of Japan, which can be found from the link[1].

For making it suitable for the experiment, the dataset is cut into five parts. All the datasets are pruned to delete single lines which both sides of this kind of line do not connect to other lines. Otherwise the minimum spanning tree cannot be constructed. The source code of this project can be found from the link[2].

The result of comparison between Dijkstra, Thorup and our algorithm is given in Table I and Fig. 5. Because of the Thorup algorithm can respond in arbitrary times of shortest path query from any vertex by constructing an index only one time, here we focus on the comparison of visiting part. Comparing to the array-based Dijkstra, Fibonacci-based Dijkstra and the original Thorup algorithm, our algorithm reduced 7.5%, 72.9% and 85.6% of time cost, respectively. Table II shows the comparison of memory usage among four algorithms. The result is obtained by using JConsole. Comparing to the original Thorup algorithm, since our algorithm does not use unvisited structure, the memory usage is greatly saved. But still takes about 75% more memory than the array based Dijkstra algorithm. The experimental environment is listed in Table III. Detailed information of datasets is given by Table IV. The datasets can be found from the link[3].

TABLE II: COMPARISON OF MEMORY USAGE

Algorithms	Memory Usage (bytes)
Array heap based Dijkstra	6,429,896 (24.99%)
Fibonacci heap based Dijkstra	54,912,976 (213.45%)
Thorup	51,544,952 (200.35%)
Improved Thorup	25,726,824 (100%)

TABLE III: EXPERIMENTAL ENVIRONMENT

CPU	Intel(R) Core(TM) i7-3630QM CPU @ 2.40GHz, 2394 MHz
Memory	16GBytes, PC3-12800 (800 MHz)
OS	Microsoft Windows 7 64-bit Service Pack 1

TABLE IV: INFORMATION OF DATASETS

Info\Datasets	1	2	3	4	5
Vertices	1889	6913	11478	14295	16670
Edges	5920	21798	36262	44904	53318

VI. CONCLUSION

We have proposed a practically-improved Thorup-based algorithm. It enhances the performance of visiting vertices by

[1]http://www1.gsi.go.jp/geowww/globalmap-gsi/download/data/gm-japan/gm-jpn-trans_u_2.zip.

[2]http://weiyusi.com/resource/ShortestPaths.7z

[3]http://weiyusi.com/resource/gm-jpn-trans_u_2.7z

reducing the depth of a component tree. According to the experimental result, comparing to array-based Dijkstra, Fibonacci-based Dijkstra and the original Thorup algorithm, our algorithm reduced 7.5%, 72.9% and 85.6% of time cost for all the five datasets, respectively. About the memory usage, since we avoid using unvisited structure, our algorithm takes about only a half of the memory usage of the original Thorup algorithm. But still takes about 75% more memory than the array based Dijkstra algorithm. In the future work, we shall try to find structure that performs better than a component tree in maintaining the tentative distance of each vertex.

REFERENCES

[1] Y. Wei and S. Tanaka, "An improved thorup shortest paths algorithm with a modified component tree," in *Proc. The ninth International Conference on Natural Computation*, 2013, pp. 1172-1177.

[2] D. En, H. Wei, J. Yang, N. Wei, X. Chen, and Y. Liu, "Analysis of the shortest path of GPS vehicle navigation system based on genetic algorithm," in *Proc. 2011 International Conference on Electrical, Information Engineering and Mechatronics*, 2012, pp. 413-418.

[3] G. Bhalotia, A. Hulgeri, C. Nakhe, S. Chakrabarti, and S. Sudarshan, "Keyword searching and browsing in databases using BANKS," in *Proc. 2002 ICDE Conf*, 2002, pp. 431-440.

[4] R. Sivakumar, P. Sinha, and V. Bharghavan, "CEDAR: a core-extraction distributed ad hoc routing algorithm," *IEEE Journal on Selected Areas in Communications*, vol. 17, no. 8, pp. 1454-1465, Aug. 1999.

[5] R. Sedgewick, *Algorithms in Java Part 5, Graph Algorithms*, 3rd ed., Addison-Wesley Professional, 2003, ch. 21.

[6] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, vol. 1, no. 1, pp. 269-271, Dec. 1959.

[7] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to Algorithms*, 3rd ed. MIT Press and McGraw-Hill, 2009, pp. 573-574.

[8] J. Y. Yen, "A shortest path algorithm," Ph.D. dissertation, University of California, Berkeley,California, United States, 1970.

[9] M. Thorup, "Undirected single source shortest paths in linear time," in *Proc. The 38th Symposium on Foundations of Computer Science*, 1997, pp. 12-21.

[10] Y. Asano, and H. Imai, "Practical efficiency of the linear time algorithm for the single source shortest path problem," *Journal of the Operations Research, Society of Japan*, vol. 43, no. 4, pp. 431-447, Dec. 2000.

[11] N. Pruehs, "Implementation of thorup's linear time algorithm for undirected single-source shortest paths with positive integer weights," B. S. thesis, University of Kiel, Kiel, Germany, 2009.

[12] H. N. Gabow, "A scaling algorithm for weighted matching on general graphs," in *Proc. the 26th Annual IEEE Symposium on Foundations of Computer Science*, 1985, pp. 90-100.

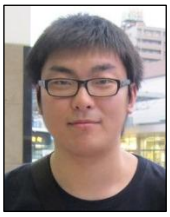
[13] J. B. Kruskal, "On the shortest spanning subtree of a graph and the traveling salesman problem," in *Proc. Am. Math. Soc*, Feb. 1956, vol. 7, pp. 48-50.

[14] R. E. Tarjan, "Efficiency of a good but not linear set union algorithm," *Journal of the ACM*, vol. 22, no. 2, pp. 215-225, Apr.1975.

[15] T. Hagerup, "Improved shortest paths on the word RAM," in *Proc. the 27th International Colloquium on Automata, Languages and Programming*, 2000, pp. 61-72.

[16] S. Pettie, "A new approach to all-pairs shortest paths on real-weighted graphs," *Theoretical Computer Science*, vol. 312, no. 1, pp. 47-74, Jan. 2004.

[17] E. A. Dinic, "Economical algorithms for finding shortest paths in a network," *Transportation Modeling Systems*, pp. 36–44, 1978.



Yusi Wei is currently a Ph.D student of computer science in Shimane University, Japan. He received his master's degree of computer science from the University in 2012. His current research interests include spatial databases and shortest paths.

Shojiro Tanaka received the MSc and PhD degrees from Hiroshima University, Japan, in 1986 and 1992, respectively. Presently, he is a professor at the Faculty of Science and Engineering, Shimane University, Japan.