Stanislav Paluch *

A MULTI LABEL ALGORITHM FOR *k* SHORTEST PATHS PROBLEM

The paper presents an algorithm for computing k shortest walks or k shortest paths in a directed graph G = (V, A). The proposed algorithm can be applied for solving the k shortest paths problem in an undirected graph G = (V, E), too, by transforming the graph G = (V,E) to the digraph G' = (V, A) where the arc set A contains a couple of arcs (u,v), (v,u) for every edge [u,v] E.

1. Introduction

Many problems related to transportation or communication lead to the problem to find in a network relatively short point to point path which has to fulfill certain special constraints that cannot be simply formulated by means of graph theory. Such problems can be solved by successive enumeration of k shortest paths and consequently by choosing that one from them which complies with given constraints.

This work was motivated by the problem of finding a bus and train connection using digitalized bus time table and train time table. The user requires not only one but several proposals of such connections and he chooses among them that one which fulfills his special personal requirements. The connection searching problem can be formulated as a shortest path problem in a huge acyclic digraph G having a vertex for every pair (*stop*, *minute*) where *stop* is a bus stop or railway station and *minute* $\in \{0, 1, ..., 1439\}$ is a minute of the day. The arc set of digraph G is the set of all ordered pairs

((*deperature_stop*, *deperature_time*), (*arrival_stop*, *arrival_time*)).

Such a digraph G for the Slovak Republic contains several millions vertices and enormous number of arcs.

The k shortest path problem is frequently studied in literature. Plesnik presents in [3] a procedure based on prohibiting edges of till now best paths, Yen gives in [4], [5] a deviation algorithm with running time $O(kn(m+n\log n))$ – this worst case assessment has been the best known for long time. Algorithm by Gotthilfa, and Lewenstein [1] (issued in March, 2009) runs in time $O(kn(m+ +n\log\log n))$.

Algorithms treated in literature have very good theoretical and practical complexity, but none of them seemed to us to be suitable for our problem – some of them seemed to be too tricky or seemed to be difficult for implementation namely because of time shortage. The requirements for our algorithm were

1. Theoretically clear and simple
2. Easy and fast implementable
3. Relatively fast even in huge graphs

The result is algorithm with complexity $O(Km(\log(Km)+n))$ for general directed graphs and $O(Km \cdot \log(Km))$ for directed acyclic graphs, where K is the number of shortest paths, n is the number of vertices and m is the number of arcs. The only nontrivial structure used in proposed algorithm is the Fibonacci heap with operations `insert` and `extract_min`,

2. Terminology

A **digraph** (a directed graph) is an ordered pair $G = (V, A)$, where V is a nonempty finite set and A is a set of ordered pairs of the type (u, v) such that $u \in V$, $v \in V$ and $u \neq v$.

The elements of V are called **vertices** and the elements of A are called **arcs** of the digraph G.

A **graph** (an undirected graph) is an ordered pair $G = (V, E)$, where V is a nonempty finite set and E is a set of unordered pairs of the type $\{u, v\}$ such that $u \in V$, $v \in V$ and $u \neq v$.

The elements of V are called **vertices** and the elements of E are called **edges** of the graph G.

A (v_1, v_2) – **walk** in digraph $G = (V, A)$ is an alternating sequence of vertices and arcs of the form

$$\mu(v_1, v_k) = (v_1, (v_1, v_2), v_2, (v_2, v_3)\, v_3, ..., v_{k-1}, (v_{k-1}, v_k), v_k)$$

A **trivial walk** is a walk $\mu(v, v) = v$ containing only one vertex.

* **Stanislav Paluch**
Faculty of Management Science and Informatics, University of Zilina, Slovakia, E-mail: Stanislav.Paluch@uniza.sk

A (v_1, v_k) - **trail** in G is a (v_1, v_k) - walk in G with no repeated arcs. A (v_1, v_k) - **path** in G is a (v_1, v_k) - walk in G with no repeated vertices.

A walk, trail and path in undirected graph can be defined in an analogical way.

An **arc weighted digraph** $G = (V, A, c)$ is an ordered triple where $G = (V, A)$ is a digraph and $c : A \to R$ is a real function defined on the arc set A, the value $c(a)$ for $a \in A$ is called the **weight** of the arc a (or sometimes the arc-weight, the length or the cost of the arc a). An **edge weighted graph** can be defined similarly. In this paper we will assume that $c(a) \geq 0$. This condition is fulfilled in many practical applications.

The **length of the walk** $\mu(u, v)$ in a digraph $G = (V, A, c)$ is the total sum of arc-weights of its arcs, whereas the arc weight is added to the total sum so many times how many times it appears in the walk $\mu(u, v)$. The length of the walk in undirected graph is defined similarly.

Suppose that $\mu(v_1, v_k) = (v_1, (v_1, v_2), ..., (v_{k-1}, v_k), v_k)$ and $\mu(u_1, u_l) = (u_1, (u_1, u_2), ..., (u_{l-1}, u_l), u_l)$ are two walks and let $v_k = u_1$.

The **concatenation** $\mu(v_1, v_k) \oplus \mu(u_1, u_l)$ of walks $\mu(v_1, v_k)$, $\mu(u_1, u_l)$ is the walk

$$\mu(v_1, v_k) \oplus \mu(u_1, u_l) = (v_1, (v_1, v_2), v_2, ..., (v_{k-1}, v_k), v_k) \equiv$$
$$\equiv (u_1, (u_1, u_2), ..., v_{l-1}, (v_{l-1}, v_l), v_l) .$$

Denote $V^+(v) = \{w | (v, w) \in A\}$, $A^+(v) = \{(v, w) | (v, w) \in A\}$. The **forward star** FSTAR(v) of the vertex $v \in V$ is the subgraph of $G = (V, A)$ with vertex set $V^+(v) \cup \{v\}$ with arc set $A^+(v)$, i. e. FSTAR(v) = $(V^+(v) \cup \{v\}, A^+(v))$.

3. Algorithm

Assume that $V = \{1, 2, \ldots, n\}$.

Denote

`n`	- the number of nodes of the digraph G = (V, A, c),	(1)
`m`	- the number of arcs of the digraph G = (V, A, c),	(2)
`s, f`	- starting and finishing vertex	(3)
`K`	- the number of searched shortest paths,	(4)
`c(u,v)`	- the length of the arc $(u, v) \in A$	(5)

The `k`-th shortest `(s,w)`-path will be denoted by μ_k `(s,w)` for arbitrary `w` $\in V$.

For every vertex `w` $\in V$ and for `k` = 1, 2, ..., `K` we will define step by step at most `K` definitive three-dimensional labels

```
deflab[w][1][1], deflab[w][1][2], deflab[w][1][3]
deflab[w][1][2], deflab[w][2][2], deflab[w][2][3]
. . . . . .
deflab[w][k][1], deflab[w][k][2], deflab[w][k][3]
. . . . . .
deflab[w][K][1], deflab[w][K][2], deflab[w][K][3].
```

Let `deflab[w][k][1-3]` be the `k`-th three-dimensional label assigned to a vertex `w`. The number `k` is called the rank of the label `deflab[w][k][1-3]`.

The above mentioned labels have the following meaning:

`deflab[w][k][1]` - the length of the `k`-th shortest `(s,w)`- path μ_k`(s,w)`	(6)
`deflab[w][k][2]` - the last but one vertex of the `k`-th shortest `(s,w)`- path μ_k`(s,w)`	(7)
`deflab[w][k][3]` - the rank of the label of the last but one vertex of the `k`-th shortest `(s,w)`- path μ_k`(s,w)`, i. e. `(s,w)` is a concatenation of the l-th shortest `(s,v)`- path μ_l`(s,v)` and the path `(v,(v,w),w)`, where l = `deflab[w][k][3]` and `v` = `deflab[w][k][2]`, i.e. μ_k`(s,w)` = μ_l`(s,v)` $\oplus$ `(v,(v,w),w)`.	(8)
`n_deflab[w]` - the number of determined definitive labels of the vertex `w` $\in V$.	(9)

When algorithm starts, we set initially `n_deflab[s]=1`, `deflab[s][1][1]=0`, `deflab[s][1][2]=0` and `deflab[s][1][3]=0` for the starting vertex `s`. Further we set initially for all `w` $\in V$ such that `w`$\neq$`s` `n_deflab[w]=0` (no definitive label is assigned to `w`) and we consider all labels `deflab[w][j][1]`, `deflab[w][j][2]` and `deflab[w][j][3]` to be undefined for all vertices `w` $\in V$ such that `w`$\neq$`s` and `j`$\neq$`1`.

Labels `deflab[w][k][1]`, `deflab[w][k][2]` and `deflab[w][k][3]` will be calculated step by step in course of algorithm work until `deflab[f][K][1]`, `deflab[f][K][2]` and `deflab[f][K][3]` are determined for the final vertex `f`.

The proposed algorithm will make use of an auxiliary set E of ordered quadruples of the form `(wtemp,ttemp,xtemp,ktemp)` where `wtemp` is a vertex `wtemp` $\in V$, `ttemp` is the length of the `(s,wtemp)`- path which was obtained by concatenation of the `ktemp`-th shortest `(s,xtemp)`- path $\mu_{ktemp}(s,xtemp)$ and the path `xtemp,(xtemp,wtemp),wtemp`, i.e.

$$\mu_{ktemp}(s,xtemp) \oplus \texttt{(xtemp,(xtemp,wtemp),wtemp)}.$$

The set E will contain initially exactly all the quadruples of the form `(w,c(s,w),s,1)` where `(s,w)` `FSTAR(s)`, i.e.

$$E = \{(w, c(s, w), s, 1 \mid (s, w) \in A\}$$

Now E contains all the quadruples `(w,t,s,k)` where `w` is a final vertex of a `(s,w)`- path -namely the path `s,(s,w),w`; where `t` is the length of this path, `s` is the last but one vertex of this path, `k=1` and this path is concatenation of k-th shortest `(s,s)`-path (i.e. shortest `(s,s)`-path) $\mu_k(s,s) = \mu_1(s,s)$ and `s,(s,w),w` = $\mu_k(s,s) \oplus$ `(s,(s,w),w)`.

The essential idea of the proposed algorithm is as follows:

While n_deflab[wmin]<K and E ≠ ∅ repeat the following procedure:

Start procedure.

Extract from E a quadruple (wmin,tmin,xmin,kmin) having the minimum value of the second item tmin in E. This quadruple determines a (s,wmin)- path $\mu_{\text{kmin}}(s,x \text{ min}) \oplus$ (xmin,(xmin, wmin),wmin) with the length tmin and last but one vertex xmin.

If n_deflab[wmin] < K, we have just found the subsequent shortest (s,wmin)- path with the length tmin.

In this case:

1. Set: n_deflab[wmin]= n_deflab[wmin]+1;
   k=deflab[wmin];
   deflab[wmin][k][1]= tmin;
   deflab[wmin][k][2]= xmin;
   deflab[wmin][k][3]= kmin;

2. For all vertices w ∈ V^+(wmin) such that n_deflab[w]< K create the quadruple (w, tmin+c(wmin,w), wmin, k)

3. a) Examine whether the path μ_k(s,wmin) contains the vertex w. If yes, the walk

   μ_k(s,wmin) $\oplus$ (wmin,(wmin,w),w)

   contains a cycle and therefore it is not a path. In this case do nothing.

   b) If the path μ_k(s,wmin) does not contain the vertex w, the walk

   μ_k(s,wmin) $\oplus$ (wmin,(wmin,w),w)

   is a (s,w)-path with the length tmin+c(wmin,w), with the last but one vertex wmin and with the rank of the label of the last but one vertex equal to k.
   In this case insert the quadruple (w, tmin+c(wmin,w), wmin, k) into the set E.

End procedure

The k-th shortest (s,f) path $\mu_k(s,f) = (s \equiv v_1, (v_1, v_2), v_2, ..., v_{l-1}, (v_{l-1}, v_l), v_l \equiv f)$ can be calculated by making use of labels deflab[][][] as follows:

v_l = f;
k_l = k;

v_{l-1} = deflab[v_l][k_l][2];
k_{l-1} = deflab[v_l][k_l][3];

v_{l-2} = deflab[v_{l-1}][k_{l-1}][2];
k_{l-2} = deflab[v_{l-1}][k_{l-1}][3];
. . . . . .
v_2 = deflab[v_3][k_3][2];
k_2 = deflab[v_3][k_3][3];

v_1 = deflab[v_2][k_2][2];

The same procedure can be used in step 3. a) for checking whether the path μ_k(s,wmin) contains the vertex w.

4. The complexity of proposed algorithm

Recall that n = |V|, m = |A|, suppose m>n. A quadruple (w,t,x,k) can be inserted into the set E at most K.m times. In the case that E is organized as a Fibonacci heap all insertions require O(K.m) steps. Before every insertion a cycle occurrence check is necessary. A single cycle occurrence check requires O(n) steps; all cycle occurrence checks will require at most O(K.m.n) steps. Every quadruple can be extracted from E at most once, a single extraction requires (in the case of Fibonacci heap) O(log(K.m)) steps, all extractions will need at most O(Km log(Km)) steps. So the complexity of proposed algorithm is O(Km(log(Km)+n)). Let us remark that we get the same complexity if E is organized as a binary heap.

5. Modification of algorithm for directed acyclic graphs – DAGs

There are many applications where the studied digraph $G = (V, A, c)$ is a directed acyclic graph – DAG. For example – the underlying digraph for CPM and PERT methods is DAG, the digraph used for optimum bus and/or train connections search is also DAG. Since there are no cycles in DAGs step 3. a) is not necessary – it can be skipped – and therefore the complexity of the proposed algorithm is O(Km log(Km)).

Let's remark that omitting the step 3. a) in the proposed algorithm leads in a digraph which is not acyclic to an algorithm which computes K shortest walks.

6. Conclusion

The proposed algorithm was used for bus and/or train connection search problem with great success. Our further plans are to examine the efficiency and performance of this algorithm in huge general graphs and digraphs, where step 3.a) may cause considerable slow down.

Acknowledgement

The author is pleased to acknowledge the financial support of the Scientific Grant Agency of the Slovak Republic VEGA under the grant No. 1/0135/08.

References

[1] GOTTHILFA, Z., LEWENSTEIN, M.: *Improved algorithms for the k simple shortest paths and the replacement paths problems,* Information Processing Letters, Vol. 109, 7/2009, Pages 352–355
[2] MARTINS, E., Q., W., PASCOAL, M., M., B., SANTOS, J., L., E.: *A New Implementation of Yen's Ranking Looples Path Algorithm*, Investigacao Operacional, 2000
[3] PLESNIK, J.: *Graph Algorithms* (in Slovak), VEDA Bratislava, 1983
[4] YEN, J., Y.: *Finding k Shortest Loopless Paths in a Network,* Managements Science, 17:712 760, 1971
[5] YEN, J., Y.: *Shortest Paths Network Problems,* Mathematical Systems in Economics, Heft 18, Meisennheim am Glan, 1975.