Time-dependent shortest path problem using MapReduce

František Kolovský[1], Jan Ježek[2]
[1,2]Department of Mathematics, University of West Bohemia in Pilsen, CZE

Abstract

Searching for the fastest path in a road network is influenced by many temporally changing circumstances such as a traffic density, weather conditions or road constructions. Even though many of these factors can be predicated in advance, most of the currently used solutions searching for a shortest path does not consider temporal traffic condition changes.

In this paper we focus on problem of searching a shortest path in a large graph where the cost of the edge is time dependent. In particular we focus on the selection of the best time of departure and the shortest path calculation to achieve the fastest travelling time from one node to all other nodes of the graph. To be able to find such information in a large graph exceeding a commodity hardware parameters, we will focus on distributed computing environment based on MapReduce model.

We test and evaluate a MapReduced algorithm derived from Label Correcting algorithm with respect to data size, number of cluster nodes, algorithm complexity as well as the result correctness. Moreover we applied the method on searching a time dependent path in a real word road network by using OpenStreetMap data and randomly generated time dependent edge cost functions. Finally a description of an implementation based on Apache Spark and GraphX framework is given and the benchmarking results measured on a production cluster are mentioned.

1 Introduction

Time-dependent shortest path search in a road network is a complex task often important not just for the car drivers, but also for traffic engineers. Road network analysis focusing on searching the time-dependant path from one place to all the other places for every departure time (one-to-all) might be a crucial analysis of a road network, that serves an important information for other analysis (e.g. calculation of traffic volume). Due to the complexity and potential size of the graph it might not be sufficient to use the basic approaches that are based on loading the graph to the operational memory of commodity hardware and sequential algorithm (e.g. Dijkstra).

In this work we describe stat-of-the art method (Label Correcting algorithm) and introduce a customization of it that suits well for MapReduce programming model. Finally we implement and test our approach by using in-memory MapReduce framework Apache Spark. The complexity of such a solution is described and the performance tests are shown. Due to the concept of MapReduce model, our approach enables to calculate a one-to-all time-dependent shortest path for all possible times of departure in a graph representing a road network of a size limited just by the sum of all the memory of the cluster nodes.

2 Related work

Searching for shortest path in a static road network is a deeply analysed problem (see e.g. [11, 13]). The available solutions addressing the time dependent cost of the edge can be divided into two main groups. The

first group considers the time function as a discrete function and second group as a continuous function. The algorithms based on discrete function are described by [3] and [2]. The approach based on continuous function is deeply described by B. Dean [8] and [4], where the Label Correcting Algorithm (LCA) is mentioned.

The state-of-the-art solutions can be also divided by a type of a static algorithm that the solution is based on. The solution based on Dijkstra (originally by [7]) is described in [8], other approach derived from A* is given by [14] and [12]. Other work has been focus on Bi-directional, Bellman-Ford [8] or ALT [14] algorithms.

Concerning the approaches how the perform algorithms in parallel, there are several studies available. The paralellization of discrete approaches is discussed in [9] and [3]. Parallel algorithm based on the space decomposition in described in [1]. The parallel approach for LCA is described by [10] including the C/C++, Matlab and MPI (Message Passing Interface) implementation. Even though mentioned studies focus on parallel processing, they are not focused on MapReduce programming model. A MapReduce (see [6]) is well known concept allowing to use not just a parallel processing but also a distributed and scalable data storage, allowing to store and process extremely large datasets.

Finally, as discussed by [5], the shortest paths algorithms focused on time-dependent graph can be categorised as follows:

- Searching for the shortest path for particular departure time:
  (a) Algorithm based on slight modification of static algorithms
- Searching for all departure times:
  (a) solutions in continuous time
  (b) solutions in discreet time

  (a) 'one-to-all' (from one node to all nodes)
  (b) 'all-to-one' (from one all nodes to one)
  (c) 'all-to-all' (from all nodes to all nodes)

  (a) parallel
  (b) serial.

In this paper we will focus on searching the shortest path from one-to-all for all departure times suitable for MapReduce model (results for all departure times enable a selection of a best departure time).

3 MapReduce algorithm based on Label Correcting (LCA)

The well known solution for searching the shortest path in a graph is Dijkstra algorithm, however its usage for time dependent graph is not easily achievable due to its utility of the priority queue. For such a reason we choose the Label Correcting algorithm, even thought its complexity is much higher comparing to the Dijktra's approach. The high level of complexity is reduced by customizing the algorithm so it can be executed in parallel. In upcoming sections the algorithm details are given and a customization to the MapReduce model is described.

3.1 Problem formulation

In upcoming section following expressions will be used:

$G(V, E)$	graph,
$w_{ij}(t)$	travel time function from node i to j,
$a_{ij}(t)$	arrival time function from node i to j,
EA_{si}	earliest arrival time function from source node,
T	starting time interval,
n	number of edges.

We find optimal departure time and optimal path from source node (s) to all other nodes in graph. Graph ($G = (V, E)$) is time-dependent and FIFO with non-negative edge. Time-dependent means: $\forall(i, j) \in E : \exists w_{ij}(t)$ where w_{ij} is travel time function. The function return travel time dependence of departure time from source node. Algorithm uses arrival function (a_{ij}), where return arrival time depends of departure time. The function can be expressed as:

$$a_{ij}(t) = w_{ij}(t) + t \tag{1}$$

The FIFO property of a graph can be expressed as:

$$\forall(i, j) \in E, t_1 < t_2 : a_{ij}(t_1) < a_{ij}(t_2) \tag{2}$$

Road network have FIFO property. In FIFO network there are no waiting nodes [8]. The solution in non-FIFO case is not polynomial [5].

3.2 Static LCA (Label correcting algorithm)

Our MapReduce algorithm is based on LCA (Algorithm 1), where. w_{ij} stands for a weight of the edge and $dist(i)$ express a length from the source node. Initialization of the graph is the same as in Dijksta's algorithm. LCA takes an arbitrary edge and tests condition $dist(i) + w_{ij} < dist(j)$, if this is true then $dist(j) = dist(i) + w_{ij}$. While it is true $\forall(i, j) \in E : dist(j) \leq dist(i) + w_{ij}$ loop ends.

Algorithm 1 : Static LCA

1: For all $i \in V : dist(i) = \infty$
2: $dist(s) = 0$
3: **while** $\exists(i, j) \in E : dist(i) + w_{ij} < dist(j)$ **do**
4: Select some $(i, j) \in E$
5: **if** $dist(i) + w_{ij} < dist(j)$ **then**
6: $dist(j) = dist(i) + w_{ij}$
7: **end if**
8: **end while**

3.3 MapReduce algorithm

In this work we present an algorithm focused on MapReduce model. The algorithm consists from 4 stages. The first stage is initialization of the graph. Initialization is done through one map function on data-set that contains all the nodes. The map function is depicted in Algorithm 3. Initialization is derived from the standard LCA.

The second stage is a map function performed on all the edges (see Algorithm 4). Such a function is the most time-consuming step in whole process. First step of this function performs operation $f(t) = a_{ij}(EA_{si})$ after the condition $piece\ of\ f(t) < EA_{sj}$, if such a condition is true the $f(t)$ is returned, else the null is returned.

Algorithm 2 : MapReduce algorithm

map nodes (initial function)
map edges (map edge function)
reduce updates (reduce node function)
updates count
while number of updates > 0 **do**
    join updates with nodes
    map nodes
    map edges
    reduce updates
    message updates
end while

The third stage is a reduce operation. The reduce function aggregates new arrival functions that is used to update a node. This function is $min(f_a(t), f_b(t))$, where f_a and f_b are functions from previous operation (map). Number of functions for aggregation can be in the interval from 0 to number of node's neighbours. Now we have one or zero arrival function in each node. The process terminates in the case that no arrival functions needs to be updated (see 3)).

$$\forall (i,j) \in E : EA_{sj} \leq a_{ij}(EA_{si}) \quad \Leftrightarrow \quad number\ of\ updates\ is\ 0 \tag{3}$$

The fourth stage joins arrival functions with original function for each node. Input for such an operation is the old arrival function store in the node (old $EA_si(t)$) and arrival function calculated in the third stage. Output is minimum from these function ($min(EA_{si}^{old}, EA_{si}^{update})$). Stage 2 - 4 are in a 'while loop'.

Algorithm 3 : initial function

Input: $i \in V$
Output: arrival function in node i

if $i = sourcenode$ **then**
    return $EA_{ss}(t) = t$
else
    return $EA_{si}(t) = \infty$
end if

Algorithm 4 : map edge function

Input: $(i,j) \in E$
Output: arrival function in node j for update

if piece of $a_{ij}(EA_{si}) < EA_{sj}$ **then**
    return $a_{ij}(EA_{si})$
else
    return nothing
end if

In the algorithm there are three operations that process the arrival function (these functions are minimum, comparison and $f(g(x))$). This operations are desirable to be implemented with a complexity that doesn't depend on the size of the graph. These functions are designed as piece-wise linear function.

By using MapReduce model all 4 stages (initialization, map edges, reduce updates, join functions) can be applied in parallel.

3.4 Complexity

Our algorithm is based on LCA. LCA have worse-case complexity $O(nm)$, where m is number of nodes. How many iteration MapReduce algorithm need? One iteration find all of the shortest paths having the distance equal to one edge. But as the number of edges of optimal path to the target node is not know in advance, the worst-case scenario needs n iterations (worst case graph is depicted in (a) of Fig.1). In the best-case just one iteration is needs (part (b) of Fig. 1). In every iteration there are n operations performed. Due to the topology of the of a real road network the algorithm is supposed to need less iterations then n (based on our experiments road network with 1600 edges (cities of Europe) usually needs roughly 98 iteration).

Algorithm performance also depends on the complexity of the three operations (minimum, comparison and $f(g(x))$). First two operations have complexity $O(p+q)$, where p and q is a number of pieces of the piece-wise linear arrival function. Last operation ($f(g(x))$) has the same complexity as the previous operation, but the number of pieces increases and thus depends on the size of the graph. The output arrival function has $p+q$ pieces. In large graph the size of the arrival function can reach intolerable limits. There are two ways of implementation: accurate and approximate. Approximate version return same number of linear parts like input function, but is not 100% correct. In our approach we implemented both ways. The tests has been performed with approximate version.

The final worst-case complexity is $O(n^2)$ (approximate version), where n is number of edges.

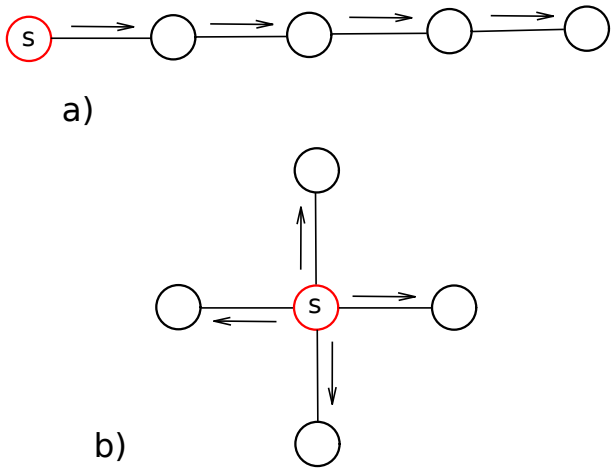

Figure 1: worst-case (a) and best-case (b)

4 Tests

The algorithm was implemented in distributed computation system Apache Spark (version 1.2.0) using GraphX. GraphX is framework for distributed computation for graph.

We performed 2 tests dependence of size of graph (Figure 2) and speed-up test (Figure 3). We made 7 data-set with another size. The biggest data-set has 16k edges. The smallest data-set have 400 edges. We use road network from Open Street Map (Pilsen and its parts). Arrival functions ware randomly generated with regard to real traffic.

Tests ware performed on computer with 2 x Intel(R) Xeon(R) CPU E5-2630 v2 @ 2.60GHz.

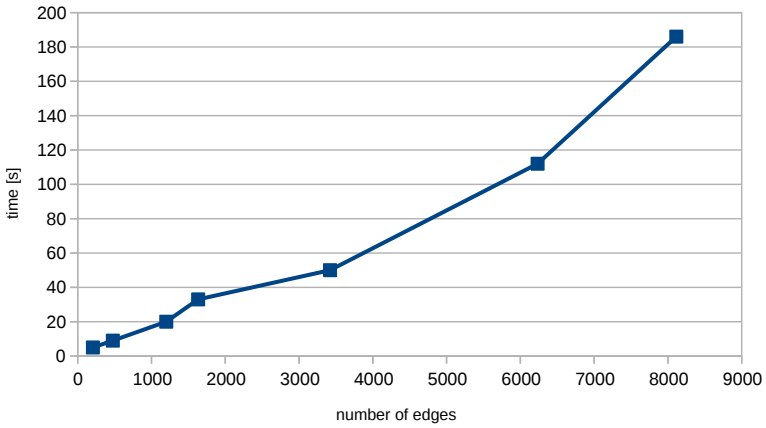

Figure 2: dependence of the size of graph (1 thread)

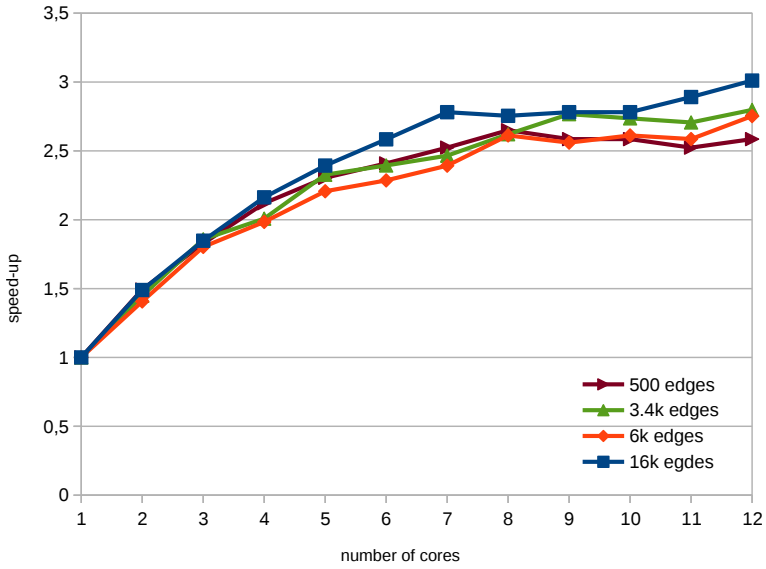

Figure 3: dependence of number of cores (threads)

The test proofs expected performance relevant to MapReduce model. The performance speed-up is the most significant up to cca 7 cores (for the used size of the graph). Adding more cores doesn't bring too much of speed-up due to the performance needed for the cluster nodes communication.

5 Conclusion

The contributed customization of LCA algorithm provide an algorithm dedicated to one-to-all, time-dependant shortest path search. Designed algorithm provide a quadratic complexity (with respect to number of graph edges) with possibility to be executed in MapReduce parallel environment.

The tests has been performed on single machine having 12 cores. In the future work we focus on testing out approach on production MapReduce cluster, where we focused on processing of a very larger graph.

Acknowledgment

This article was supported by the European Regional Development Fund (ERDF), project "NTIS – New Technologies for the Information Society", European Centre of Excellence, CZ.1.05/1.1.00/02.0090. Access to computing and storage facilities owned by parties and projects contributing to the National Grid Infrastructure MetaCentrum, provided under the programme "Projects of Large Infrastructure for Research, Development, and Innovations" (LM2010005), is greatly appreciated. We also thank you for providing computing power company virtufy.cz.

References

[1] H. AYED, Z. HABBAS, and D. KHADRAOUI. A parallel time-dependent multimodal shortest path algorithm based on geographical partitioning. In *Nature and Biologically Inspired Computing (NaBIC), 2011 Third World Congress on*, pages 213–218, Oct 2011.

[2] I. CHABINI. Discrete dynamic shortest path problems in transportation applications: Complexity and algorithms with optimal run time. *Transportation Research Records*, 1645:170–175, 1998.

[3] I. CHABINI and S. GANUGAPATI. Parallel algorithms for dynamic shortest path problems. *International Transactions in Operational Research*, pages 279–302, 2002.

[4] B. C. Dean. Shortest paths in fifo time-dependent networks: Theory and algorithms. *Rapport technique, Massachusetts Institute of Technology*, 2004.

[5] C. DEAN, Brian. Continuous-time dynamic shortest path algorithms, 1999.

[6] J. DEAN and S. GHEMAWAT. Mapreduce: simplified data processing on large clusters. *Communications of the ACM*, 51(1):107–113, 2004.

[7] E. W. Dijkstra. A note on two problems in connexion with graphs. *Numerische mathematik*, 1(1):269–271, 1959.

[8] B. DING, J. X. YU, and L. QIN. Finding time-dependent shortest paths over large graphs. In *Proceedings of the 11th International Conference on Extending Database Technology: Advances in Database Technology*, EDBT '08, pages 205–216, New York, NY, USA, 2008. ACM.

[9] S. V. GANUGPATI. Dynamic shortest paths algorithms : parallel implementations and application to the solution of dynamic traffic assignment models. *Massachusetts Institute of Technology*, 1998. Thesis (M.S.)–Massachusetts Institute of Technology, Dept. of Civil and Environmental Engineering, 1998.Includes bibliographical references (leaves 183-186).

[10] G. LAWSON, S. ALLEN, G. ROSE, D. NGUYEN, and M. NG. Parallel label-correcting algorithms for large-scale static and dynamic transportation networks on laptop personal computers. *Transportation Research Board 92nd Annual Meeting*, 2013.

[11] D. Luxen and C. Vetter. Real-time routing with openstreetmap data. In *Proceedings of the 19th ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, GIS '11, pages 513–516, New York, NY, USA, 2011. ACM.

[12] T. OHSHIMA. A landmark algorithm for the time-dependent shortest path problem. 2006.

[13] D. Wagner and T. Willhalm. Speed-up techniques for shortest-path computations. In *STACS 2007*, pages 23–36. Springer, 2007.

[14] L. ZHAO, T. OHSHIMA, and H. NAGAMOCHI. A* algorithm for the time-dependent shortest path problem. 2008.