JOURNAL OF APPLIED
COMPUTER SCIENCE
Vol. 21 No. 1 (2013), pp. 25-38

Parallel Hierarchies for Solving Single Source Shortest Path Problem

Łukasz Chomątek[1]

[1]*Lodz University of Technology*
Institute of Information Technology
Wólczanska 215,90-924 Lodz, Poland
lukasz.chomatek@p.lodz.pl

Abstract. *The problem of route optimization is significant due to the fact of the high availablity of private transport as well as the need of efficient utilization of the corporate fleet. Optimal route can have different meaning for different users. In the existing algorithms designed for finding optimal route between two points on the map, hardly any preferences are reflected. In this article we present efficient algorithm for finding optimal route between points on the map which is designed to conforms Users' needs.*
Keywords: *Single Source Shortest Path, Parallel Hierarchies, Drivers' Preferences.*

1. Introduction

Problem of finding of shortest path (Single Source Shortest Path, SSSP) between two points in the graph is present for above 50 years in computer science. Firt algorithms was based on the priority queue which even nowadays remains the main idea. In section 2 main research tracks on SSSP problem are briefly described. Main contribution of this work is shown in section 3 which contains description of all phases of the Parallel Hierarchies algorithm. Obtained results are gathered in section 4. The last section contains conclusions and points the possibilities of further research.

2. Finding the shortest path

Problem of finding the optimal, in other words: shortest in the sense of some cost function is often solved by use of the priority queue [1]. Original version of the algorithm was designed to construct the minimum spanning tree for the graph. Resulting tree consisted of the shortest paths between selected start node and any other node in the analysed graph. Algorithm can be utilized both for directed and undirected graphs where edges have only non negative weights. Actual algorithm is presented below:

Algorithm 1 Algorithm of finding the shortest path

Require: o ▷ o - Start node
  $L_o \Leftarrow 0$ ▷ Cost of reaching the start node is set to 0
  $L_j \Leftarrow \infty \; \forall j \neq o$ ▷ For all other nodes it is set to ∞
  $P_o \Leftarrow NULL$ ▷ Start node has no ancestor
  $Q \Leftarrow \{o\}$ ▷ Set the node o as required to reach
  while $Q \neq \emptyset$ **do** ▷ While there are other nodes to reach
    $i = poll(Q)$ ▷ We choose first available node
    for all $l \in OutgoingEdges(i)$ **do**
      $j = IncomingNode(l)$
      if $L_i + c_l < L_j$ **then** ▷ If the cost of reaching node j is lower, if the path passes the node i
        $L_j \Leftarrow L_i + c_l$ ▷ Set proper values of cost and the ancestor node
        $P_j \Leftarrow l$
        $put(Q, j)$
      end if
    end for
  end while

where:

- L_i Cost of reaching the node i,
- P_i Edge used for reaching the node i,
- Q Data structure which holds nodes remaining to reach,
- l Currently visited edge,
- c_l Cost of travelling the edge l.

For storing nodes remaining to reach one can use either priority queue or a heap. Conducted research revealed that use of heap causes faster performance of the algorithm, but almost all implementations utilizes the priority queue as it is easier to manage [2].

Choosing between heap and priority queue is not the only method of the optimization of the shortest path algorithm. In the problem of finding the shortest paths in the road network graph, not only information about neighbouring nodes can be used but also one can focus on the network topology. Methods of handling this case can be divided into the three groups [3]:

- pruning based,
- decomposing the problem,
- limiting the number of available edges.

The idea of pruning in case of graph search is based on finding the probability for each node in the search scope of being a part of the shortest path. Karimi [4] proposed limitation of the search space a rectangle containing both source and the destination nodes. In the other work, authors proposed computation of the lower and upper bounds for the path length. The cost of visiting new node cannot exacerbate the ratio of cost of visiting current node with respect to the lower bound for the source node.

Decomposition of the problem is based on two independent searches [5]. One of the searches was a classical search from source to the destination node. The second search was performed from the destination to source. Query is finished when at least one node is visited by both searches. One should note that the first node when the searches meet can be not the best solution.

Further research shown that the most effective algorithms for finding the shortest path are those based on the limiting of the number of edges that can be visited. Such algorithms usually consist of two phases. In the first phase, some preprocessing on the graph is made to reduce number of edges or nodes to reduce the complexity of the queries.

First algorithm in this group is ALT (A-Star Landmarks Triangle inequality) introduced by Goldberg [6]. In the first phase small set of nodes in the graph is chosen. These nodes are called landmarks. After that, for each pair node-landmark its distance is calculated. Query processing is based on the fact that the path from the current node to the destination must not be longer that one containing a landmark.

Query complexity reduction can be obtained by temporal changes in the graph structure. Delling and Schultes [7] proposed in their work an algorithm which is based on the observations of real drivers' behaviour. When such a person need to travel far from the place he lives, he usually drives to the motorway. The he does not leave the motorway until he is close enough to the destination. After that he chooses some major roads to reach closer to the place.

3. Parallel Hierarchies algorithm

In this section we introduce Parallel Hierarchies algorithm which is designed for solving problem of finding optimal paths in the road network graph. This approach is based on the Highway Hierarchies mentioned in the previous section. Highway Hierarchies algorithm depends on the division of road network into certain number of hierarchy levels. Our solution is an improvement of this algorithm as it does not operate on the whole graph, but on its subsets named *sectors*.

3.1. Construction phase in Highway Hierarchies

Highway Hierarchies [8] algorithm requires two parameters: H that identifies the degree to which the requests for the shortest path are met without coming to a higher level in the hierarchy, and L, which represents the maximum permissible hierarchy level. The method iteratively generates a higher level with number $l + 1$ for a graph G_l in the way given below:

1. For each vertex $v \in V$, build the neighbourhood N_l^H for all vertices reached from v by using Dijkstra's algorithm in graph G_l, respecting the H constraint. Set the state of the vertex V to *active*.

2. For each vertex: Build the partial tree B(v) and assign to each vertex its state. The state of the vertex is inherited from the parent vertex every time when the vertex is reached or settled. Vertex becomes *passive* if on the shortest path $\langle v, u, \ldots, w \rangle$ where $v \neq u \neq w$: $\left|N_H^l(u) \wedge N_H^l(v)\right| \leq 1$ Partial tree is completed, when reached but unsettled vertices don't exist.

3. For each vertex t, which is a leaf node in the tree B(v) move each edge (u, w), where $u \in N_H^l(t); w \in N_H^l(v)$ to the higher hierarchy level.

During the first stage, a highway hierarchy is constructed, where each hierarchy level G_l, for $l < L$, is a modified subgraph of the previous level graph G_{l-1}. Therefore, no canonical shortest path in G_{l-1} lies entirely outside the current level for all sufficiently distant path endpoints. This ensures that all queries between far endpoints on level $l - 1$ are mostly carried out on level l, which is smaller, thus speeding up the search.

3.2. Graph partitioning in Parallel Hierarchies

The algorithm of graph partitioning is based on the Breadth First Search algorithm. [9] At first, K nodes from the road graph are randomly chosen and they're the points of start for the BFS-based algorithm. The algorithm assigns nodes and edges of the input graph to one or more sectors. When there are no unassigned nodes, the algorithm is completed.

1. For each node $k : k \in BFS_{start}$ create empty list E_k and N_k to store the information about edges and nodes that belong to the sectors.
2. For each node $k : k \in BFS_{start}$:
   (a) For the nodes from BFS queue, check if their children are assigned to any sector. If not, add them to BFS queue of the current node. Add adjacent nodes to N_k as well as corresponding edges to E_k
3. If there are unvisited nodes left, go to step 2.
4. Perform required post-processing for each set N_k

As one can see, some nodes from the input graph can be assigned to more than one sector. These are border nodes (denoted as B_k) which are needed for proper processing of the parallel queries. Each sector built from start node $k \in BFS_{start}$ has its own set of border nodes. Such an algorithm can be applied for connected graphs. If there is more than one connected component in the input graph, it should be performed for each connected component.

In Fig. 1 an example of road map divided into four sectors is given. Each colour represent a single sector. To create a division we use BFS algorithm which assigns graph nodes to sectors. However presented method of creating sectors is quite efficient, as it has low computational complexity, in future work we would like to examine other methods of the partitioning to get a division which better fits the

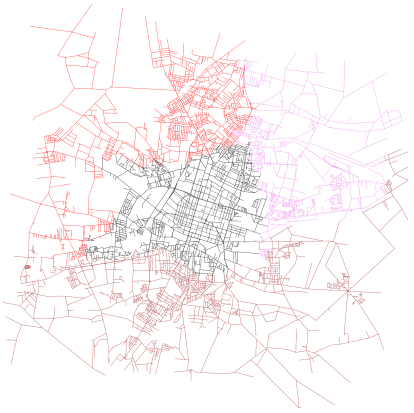

Figure 1. Road map divided into sectors

Figure 2. Hierarchical division of the road map

hierarchical algorithm. One of the problems is that number included road segments can differ quite much within the sectors, so that when the construction phase is performed in parallel way, some threads will have to wait until other threads will finish their work.

After graph partitioning phase is completed, for each sector a hierarchical division is performed by simultaneous instances of Highway Hierarchies algorithm (Fig. 2).

3.3. Querying

Highway Hierarchies algorithm is designed for solving SSSP problem in the graph which represents the road network, but both phases of the algorithm, the whole road graph must be taken into account. Parallel Hierarchies assumes that hierarchical division for all of the sectors obtained in the partitioning process is calculated independently. What is more, when hierarchical division is performed on the sectors, instead of the whole graph, hierarchy levels for edges in the sector, can be different to levels of corresponding edges in the input graph. Due to the fact, that in Highway Hierarchies the query algorithm depends on the proper hierarchical division of the graph edges, we had to adapt the algorithm to our method.

In HH algorithm, edges and nodes that are on the highest hierarchy level belongs to the one connected component. In case of parallel division, edges belong-

ing to the highest hierarchy level, make a connected component in each sector. When both source and target node are placed in the same sector, both search scopes can meet without any modification of the querying algorithm known from HH, as there is only one connected component in the highest hierarchy level for each sector.

Situation is more complicated, when the path between the source and target node crosses the borders of sectors. The idea of the new algorithm is that actual query is split into smaller subqueries, each of which is performed inside one sector. In this case one have to identify border nodes, which are source and a target node for a subquery. In the final step, results of all of the subqueries are merged to find a proper itinerary.

Algorithm 2 presents the method of finding the path between nodes s and t. List I contains source, target, and border nodes that are going to be traversed in the final result. Function *findClosest* is responsible for finding the border node which is closest to the line connecting nodes given as second and third parameters.

Algorithm 2 Algorithm for parallel querying

Require: s ▷ s - Start node
  $I \Leftarrow \emptyset$ ▷ I is $\emptyset$
  $I.add(s)$
  $b \Leftarrow findClosest(B_k, s, t)$
  $I.add(b)$
  while $sector(b) \neq sector(t)$ **do** ▷ b and t are not in one sector
    $b \Leftarrow findClosest(B_k, b, t)$
    $I.add(b)$
  end while
  $I.add(t)$
  Perform subquery for each neighbouring pair of nodes in I
  Merge results of all subqueries

4. Obtained results

Our algorithm was implemented in Java programming language. Authors utilized MATsim [1] project, where Parallel Hierarchies algorithm was added as a path

[1] www.matsim.org

calculator. For the research purposes, authors also implemented Highway Hierarchies and extended existing Dijkstra's algorithm to present number of visited nodes.

Input road network is a map of Lodz,Poland taken from OpenStreetMap[2] database. To have it read by the MATsim environment, we had to use the Osmosis tool, which is the part of MATsim project. Input network has over 7000 nodes and over 17000 links. Each link is treated as a one way road, so that if one can drive in two directions, there are to links for such a segment.

Modelled simulation of road traffic contained agents that need to drive from their houses placed in the suburbs to workplaces in the city centre. For each agent, optimal routes from and to home had to be calculated, as many streets in the city centre are one-way and this causes agents to have different routes in both directions.

In this section we gathered results concerning the performance of the Parallel Hierarchies algorithm. At first we examined we present how do the architecture of prepared hierarchical division changes. After that duration of the construction phase (section 4.2) depending on number of sectors. In section 4.3, we discuss the efficiency of adapted querying algorithm.

For visualization of obtained data we used StreetVis tool, which was implemented in C# 4.0 language. We decided to use the C# language as the application can be easily scaled and converted to the web application. On Fig. 3 classes that represent graphs in StreetVis are shown. `Graph` consists of two lists. One is a list of nodes, and the other is a list of edges. `GraphEdge` represent directed edge in the graph. When one need an undirected edge, two directed edges must be added instead. `GraphNode` keeps information about its coordinates as well as incoming and outgoing edges. Both `GraphNode` and `GraphEdge` classes are base classes for task-dependent representation for both structures.

In the Fig. 3 one can see `HierarchizedGraphEdge` which specifies edge with additional property named `HierarchyLevel`. This is needed for proper handling of Parallel Hierarchies algorithm.

4.1. Different hierarchy levels depending on the size of the sector

Number of nodes in each hierarchy level can differ depending on sectors count and the parameters used for construction of hierarchical division for each sector. The research was conducted to acknowledge if number of sector influences the

[2] www.openstreetmap.org

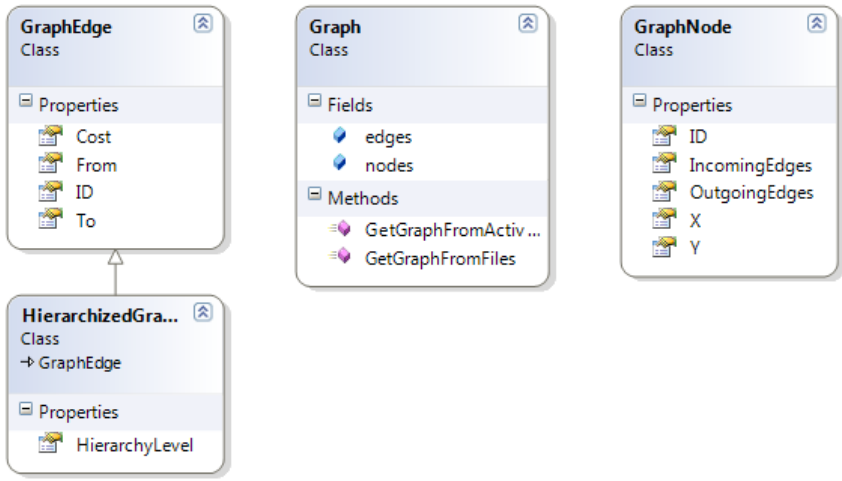

Figure 3. Classes for graph representation

obtained division. We decided to temporarily modify the Highway Hierarchies algorithm in two aspects:

- contraction was disabled after promoting edges to the new level,
- first phase algorithm was forced to stop after processing certain number of levels.

In Table 1 results for construction phase for different algorithm parameters are shown. As one can see, for constant neighbourhood size (denoted as N), number of edges that belong to the certain hierarchy level can differ significantly. For $N = 10$, we observe the major number of road segments is placed on the highest level. It is not a normal situation in HH algorithm, as its task is to reduce number of edges in the top levels. This happened because of small neighbourhood size - if HH construction for the sector is not stopped, the maximum hierarchy level would be greater for the lower neighbourhood size than for larger one. For $N = 20$, we obtained almost correct division (no further hierarchy levels were needed in a construction phase in some cases). Number of edges on different levels gets lower from level to level, so that search will be performed faster as we cut many of the road segments when traversing to the higher levels.

Table 1. Number of nodes on each hierarchy level and processing time gain, depending on sectors count.

N	Sectors Count	Hierarchy level				Time Reduction (sum based)	Parallel TR
		0	1	2	3		
10	1	4702	2169	1766	8502	1	1
	2	5440	2532	1940	7423	1.55	2.82
	3	5405	2562	2060	7277	1.65	4
	4	5798	2819	2085	6677	2.01	5.33
	5	5773	2877	2220	6404	2.18	6.15
20	1	7952	4457	3129	1601	1	1
	2	8899	4612	2773	1051	1.51	2.61
	3	8890	4793	2497	1124	1.74	4.27
	4	9409	4844	2349	777	1.88	5.52
	5	9547	4888	2283	556	2.23	6.71

What is more number of road segments on certain levels depends on the number of sectors. As one can see, the more sectors are present, the smaller number of road segments remains in the lowest hierarchy levels. This situation is caused by the fact, that in smaller sectors, building 'isolated' neighbourhoods is easier (neighbourhoods overlaps each other more often for far nodes), so that road segment promotion is also easier.

The third important thing is that larger neighbourhood size obstructs the possibility of promotion of the road segments to higher level. If there is large number of road segments in the lowest levels, the search process can get slower. It is important to adapt the neighbour size to the size of the sector.

4.2. Construction phase duration

We expected that parallel computation of the hierarchy levels for road segments in the road graph will be faster for graph divided into sectors. Corresponding results are gathered in Table 1. Value of time reduction for one sector was set to 1 as then the algorithm works like plain Highway Hierarchies. For larger number of sectors, value in the table shows the time gain from performing a division. However in Parallel Hierarchies, the construction phase for each sector is done independently, in the table we show gain obtained from summing up times needed to process each sector (T_i). What is more, time needed to divide road network to

sectors ($T_{division}$), as well as identification of border nodes ($T_{borders}$) is added too (Eq. 1).

$$TR_{sectorsCount} = \frac{T_{division} + \sum_{i=1}^{sectorsCount} T_i + T_{borders}}{T_{wholeGraph}} \quad (1)$$

In general, when algorithm is performed in parallel, processing time gain can be calculated as follows:

$$PT_{sectorsCount} = \frac{max(T_{division}(i)) + max(T_{procesing}(i)) + max(T_{borders}(i))}{T_{wholeGraph}} \quad (2)$$

Equation 2 is valid only if the processing of all sectors is performed in parallel and does not take into account time needed for the synchronisation of the processing threads.

As one can see, Highway Hierarchies algorithm parameter (neighbourhood size) does not affect significantly the time gain obtained by dividing the road network graph into sectors. The number of sectors affects the total preprocessing time - we obtain higher gain when we have larger number of sectors. Please note, that the total processing time for all sectors cannot be easily associated with the parallel processing time, as sectors can have different shapes, number of nodes and road segments, and, of course different weights of the graph edges. All of this parameters affect the speed of hierarchical division for the sector.

4.3. Number of visited nodes

Number of nodes visited during the search is an important criterion in solving SSSP problem, either for systems, where number of queries is very high or when system resources are very low. Our research revealed, that in Parallel Hierarchies, search algorithm needs to visit lower number of nodes than other examined algorithms.

For our research we counted number of visited nodes for about 40 agents. Obtained results are gathered on Fig. 4. As one can see, the highest number of nodes needed to be visited before a solution is found, is for Dijkstra's algorithm. Number of nodes visited for Highway Hierarchies is more than five times lower. For Parallel Hierarchies, we obtained even better results. Division of the road graph into more than two sectors can result in their further improvement.

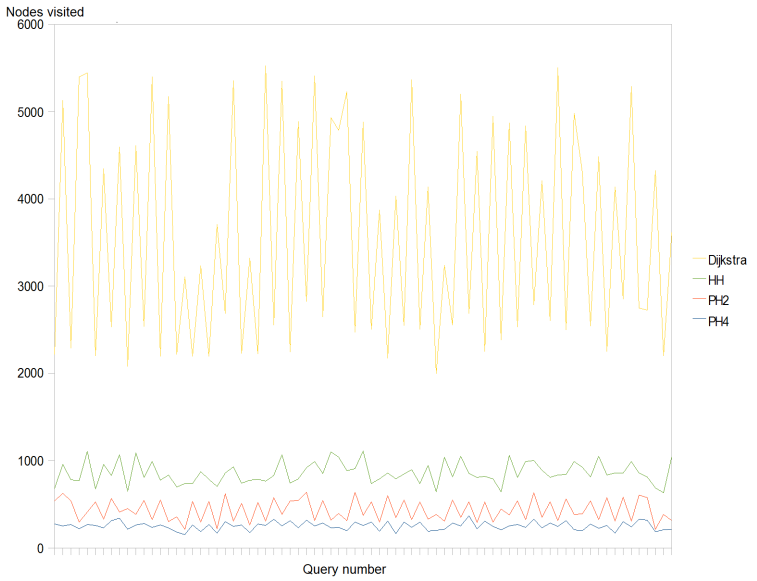

Figure 4. Number of visited nodes depending on algorithm

Moreover, for Parallel Hierarchies performed for four sectors, number of visited nodes does not differ a lot from query to query. On the other hand, for Dijsktra's algorithm dispersion of values is very high.

5. Conclusions

Presented Parallel Hierarchies algorithm is an efficient way of solving SSSP problem. Conducted research shown that both preprocessing and querying phase can be performed faster than in the initial version of Highway Hierarchies. As one can see, number of road segments in each sector depends on the sector size, which also affects the hierarchical division of the road segments. This causes the need to examine chosen properties of the algorithm for hierarchical division.

Our querying algorithm performs better in comparison to the known querying algorithms. Number of visited nodes is significantly lower, so that User gets faster responses for his queries. Efficiency of the querying algorithm depends also on the number of sectors. Number of sectors must be adjusted by the constructor of the division, but two main remarks must be taken into account: at first, greater number

of sector causes the faster switching the querying algorithm between sectors, so that a single sub-query is supposed to be performed faster. On the other hand, when sectors are smaller, if the chosen neighbourhood size is rather large, included Highway Hierarchies algorithm can build lower number of the hierarchy levels, so that most edges will fall to the lowest levels and the query will not be fastened.

Presented querying algorithm can be also utilized with other algorithms for preprocessing the sectors. In future, we would like to check the performance of other known routing algorithms like Contraction Hierarchies or make improvements known from ALT to the actual querying procedure. Extensibility of proposed querying algorithm applied to the divided road network graph is the main contribution in this paper, as independent research can be conducted on its optimization and in other fields.

References

[1] Dijkstra, E. W., *A note on two problems in connexion with graphs*, Numerische Mathematik, Vol. 1, 1959, pp. 269–271.

[2] Jacob, R., Marathe, M., and Nagel, K., *A computational study of routing algorithms for realistic transportation networks*, J. Exp. Algorithmics, Vol. 4, Dec. 1999.

[3] Fu, L., Sun, D., and Rilett, L., *Heuristic shortest path algorithms for transportation applications. State of the art*, Computers and Operations Research, Vol. 33, No. 11, 2006, pp. 3324–3343.

[4] Karimi, H., *Real-time optimal route computation: A heuristic approach*, ITS Journal, 1996.

[5] Dantzig, G., *On the shortest route through a network*, Management Science, Vol. 6, 1960, pp. 187–90.

[6] Goldberg, A. V. and Harrelson, C., *Computing the shortest path: A search meets graph theory*, In: Proceedings of the sixteenth annual ACM-SIAM symposium on Discrete algorithms, SODA '05, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2005, pp. 156–165.

[7] Sanders, P. and Schultes, D., *Engineering Highway Hierarchies.* In: ESA, edited by Y. Azar and T. Erlebach, Vol. 4168 of *Lecture Notes in Computer Science*, Springer, 2006, pp. 804–816.

[8] Delling, D., Sanders, P., Schultes, D., and Wagner, D., *Engineering Route Planning Algorithms*, In: ALGORITHMICS OF LARGE AND COMPLEX NETWORKS. LECTURE NOTES IN COMPUTER SCIENCE, Springer, 2009.

[9] Chomatek, L. and Poniszewska-Maranda, A., *Multi-agent System for Parallel Road Network Hierarchization*, In: ICAISC (2), 2012, pp. 424–432.