

Research paper analysis on

Continuous nearest neighbor monitoring in road networks

....

Seminar: Algorithms for database systems

Sergio Marsella

sergio.marsella@access.uzh.ch

Contents

1.	Introduction.....	3
2.	Problem statement.....	4
2.1.	kNN query.....	4
2.2.	CkNN query.....	4
3.	Assumptions and data structures.....	5
4.	IMA Algorithm	6
4.1.	Initial result computation	6
4.2.	Processing of updates.....	7
4.3.	The complete IMA algorithm.....	9
5.	GMA Algorithm.....	10
6.	Evaluation	11
7.	Conclusion	12
8.	References	13

1. *Introduction*

Nowadays, wireless communication devices including the wide area of its functionalities have become increasingly important. In the last decade, not only a huge progress in terms of performance and functionality has been achieved, but also have devices become smaller and smaller. Geographical positioning technologies like GPS, which are already counted among the common functionalities of a mobile device, can be used for several purposes. Here, we set the focus on the mentioned technology as a dynamic part of a road networks.

Starting with two simple examples of daily routines the main issue of this report is demonstrated, which includes the challenges of the research area.

In the first example, a certain person is located at a certain street of a town and needs to get to a pharmacy as soon as possible. Because the person is moving by foot, there exists no external influence like traffic jam that could decrease the speed of walking. So, the path distances to the closest pharmacies are enough to meet the requirements.

The second example is much more complex. During the rush hours, a woman is sitting in a bus and is searching for a taxi driver. For that purpose, there is a display in the bus that shows the currently nearest taxis with their phone numbers. The taxis and the bus are moving with varying speed depending on the type of road and can be influenced by traffic jams, road works and other obstacles in the road network. So, the current path distances from the different taxis to the bus are not sufficient. Therefore, for every section of the paths, aspects like road type or traffic jam have to be taken into account in addition to the pure distance. Some of those are more or less static, but some are highly dynamic. At last, the woman is not interested in the path distance of the taxis. Her final wish is to get the soonest opportunity of moving off from the point where the bus is currently located.

Both the examples show common situations that could not be solved by devices a couple of decades ago. But with the inventions of positioning technologies, wireless communication and the internet, new opportunities have been opened.

In the example with the pharmacies, most of the elements are static or slowly changing. The only dynamic element is the person that requests the closest pharmacies. This could also be done by looking it up in a conventional map. Nevertheless, computing those paths to the closest destinations

with an algorithm like Dijkstra's shortest path algorithm would make it much faster and more comfortable.

The second example has much more of those dynamic elements. The taxis and the bus are moving and there are lots of taxis. Additionally, all the mentioned factors that determine the speed of a certain section of the roads have to be updated regularly.

To enable such a computation of a continuous nearest neighbor search, there has to be introduced a database, which stores all the mentioned elements of the road network as well as the moving objects, which would be the taxis in the example. It also needs a central server and algorithms that efficiently monitor the queries, which are requested from the application in the bus. So, the bus always has the current results. Thus, the second example is where the paper headed to. The detailed description of two approaches called 'Incremental Monitoring Algorithm' and 'Group Monitoring Algorithm' will follow in the next sections.

Before moving on to the problem statement, it has to be emphasized that road networks of a larger town imply huge number of roads, moving objects and not to mention the high dynamisms of the road conditions, which makes it very complex for the algorithms. Therefore, such an algorithm has to be highly scalable.

2. Problem statement

There are two basic definitions in the context of this work. The kNN query represents the fundamental idea and the CkNN query is the extended version of it. The new approaches presented in the paper face the challenges of the extended version.

2.1. kNN query

A kNN query is defined as nearest neighbor search of the k nearest data objects to the requesting object. This type of query is restricted to snapshot queries over static data.

2.2. CkNN query

CkNN stands for continuous kNN and deals with the dynamism of mobile data. This means, that the queries run for a long time and demand a constant update of the result. In the example, this would be the application on a bus, that always gets updated with the currently nearest taxis.

3. Assumptions and data structures

First of all, there is a network which represents a road network of a certain town. It consists of edges and nodes. Both are static elements, but the edges have an additional weight that can fluctuate depending on traffic conditions. Moreover, there are data objects and CkNN queries, both part of the system, too. The data objects are the entities of interest and the queries correspond to users or applications that request continuous monitoring of their k closest data objects. Both the coordinates of the data objects or queries and the weights of the edges change frequently, so those changes have to be updated to the central monitoring server. Figure 1 shows an example of a small road network that will be used for further explanations of the algorithm. Every static node, object and query has a coordinate and the edges have an current weight of 1, which is indicated in brackets (i.e. E1(1)).

The final goal is to continuously compute the network distances between a query and a data object, which is defined as the total cost along the shortest path connecting them. Consider that the shortest path is not the shortest path in meters, but that one combined with the weights on that path.

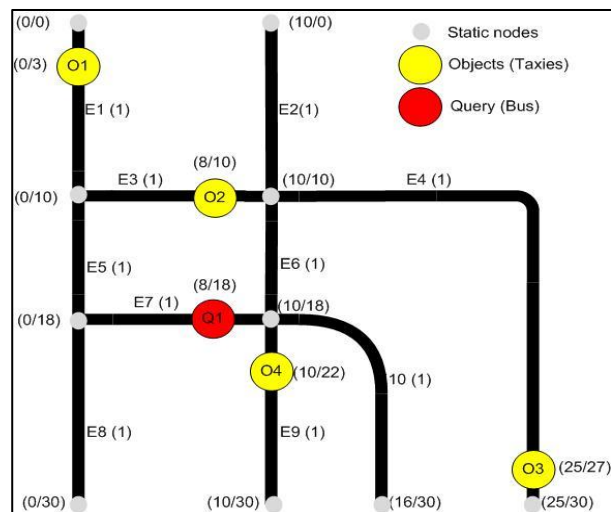


Figure 1: Road network example with a query and 4 objects

The actual information about the network, the data objects and the queries is stored in three memory-resident data structures. The first one is used for the existing network edges and is called the *spatial index SI*. The edges are stored in a PMR quadtree which allows to insert edges iteratively and to efficiently retrieve an edge in a top-down way. The second structure is the *edge table ET* which maintains network and data object information. Every edge has two endpoints, a set of adjacent edges to each of its endpoints, its weight, a list of objects currently on e , and the influence list containing queries affected by e . A query is affected by an edge if the edge contains an interval where the network distance is less than the query's $q.kNN_dist$, which is defined by the furthest nearest neighbour of the kNN set. The third structure is the *query table QT*. It maintains the information about the existing queries. Every query consists of its coordinates, the number $q.k$ of the required NNs, its current k -NN set $q.result$ and its expansion tree $q.tree$. The expansion tree holds all the shortest paths to the current k NN and stores it in a tree structure with the root q .

4. IMA Algorithm

After a short overview of the data structure, this section will show how that is actually used. The explanation of the Incremental Monitoring Algorithm consists of three main parts. The first one deals with the initial result computation for a new query. After that, the processing of the different updates is focused. As the third and last part, all the single parts of the algorithm are combined to a complete algorithm.

4.1. Initial result computation

When a query q arrives the first time to the system, the initial result has to be computed. This step is also called the computation from scratch. The initial k -NN set is processed by a technique based on the Dijkstra algorithm, which is applied on the current snapshot of the database. Expanding the network around the query q until k data objects are found, it builds up the expansion tree of q and updates all the influence lists of the affected edges. These are the two goals so far: the continuity of the algorithm will be seen in the following sections. Figure 2 shows the computed expansion tree after the initial result computation.

The initial result computation is an incremental process. At the beginning, an empty heap is initialized, which is only necessary for the current computation process. The algorithm then first en-heaps the edge, where the query lies, to the heap. As an expansion tree has to be built up, it sets the query as the root and the two endpoints of that edge are the first two children. The data objects (i.e. the taxis) are inserted into $q.result$. Then, if not already k objects are reached in $q.result$, the iterative expanding process starts. In each step, the next nearest static node is considered, which gets de-heaped and verified if it has a weighted distance to q is not further then the currently furthest nearest neighbor. Otherwise it gets unverified and remains in the heap. If the node is verified, the expansion tree gets updated. Next, the influence list of the edge from the current considered node to its predecessor in the expansion tree is updated with the query q . The data objects on that edge are inserted into $q.result$ step by step. The $q.kNN_dist$ gets also updated after inserting an object to $q.result$. Lastly, it en-heaps the adjacent edges of the currently considered static node if it has been verified. This iterative process continues until k data objects have been inserted into the

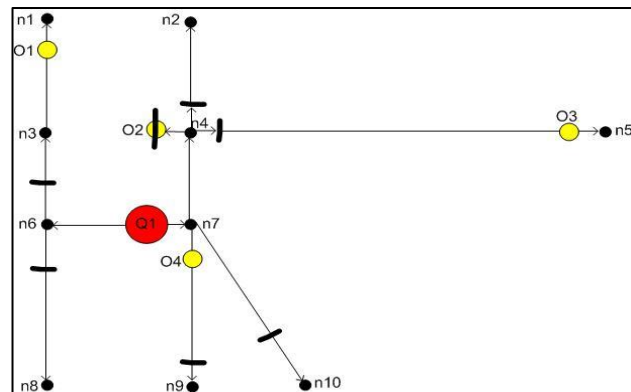


Figure 2: Expansion tree example after initial result computation

current result of the query and all the remaining static nodes in the heap are unverified. So these unverified nodes get deleted from the heap and the marks that show the weighted distance $q.kNN_dist$ are computed.

In the end, the expansion tree and the result are stored in the query table. This first initial computation that is called “*from scratch*” is a very costly operation, so it has to be avoided if possible. Therefore, the algorithm needs to be extended in a way, which will be explained in the following sections.

4.2. Processing of updates

There are three types of updates that have to be respected in the system. Objects and queries are those elements that are moving in the network and the weight of edges can vary. It is important to emphasize in advance that one of the advantages of this approach is that not every update influences the current result of a query. So, if a taxi far away from the bus moves, the result of the query used to display the nearest taxi is not influenced and remains the same. Or it could also be that only a part of the expansion tree is influenced, so not the whole expansion tree has to be re-computed.

Invalidation

Before explaining the different updates, it is important to know the concept of invalidation. The algorithm can invalidate parts of the current expansion tree. Valid parts are guaranteed to be in the new tree again, as the updates did not influence those parts. But the invalidated parts

have to be re-computed in the same manner as the initial result computation, so it expands towards that invalidated part. As the tree does often not get invalidated entirely or even not at all, this can save a lot of computing time.

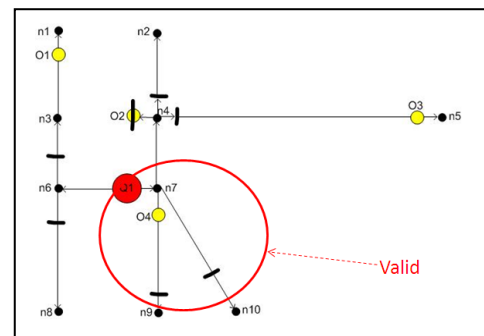


Figure 3: Invalidation of the expansion tree

Object Updates

Objects in the system are dynamic elements that can move, appear or disappear in the network. Basically, every object update contains the id of the object, its new and the old coordinates. To update the database there have to be considered two points:

1. An object has to be transferred from the object list of the old edge to the new one with the new coordinates.

2. An object movement could also influence queries, as it could leave or join a kNN-set of a certain query.

The second point is much more complex than the first one, because the influenced queries have to be updated in certain cases. There are three types of object updates influencing a certain query. The incoming objects lied out of the distance $q.kNN_dist$ of a query before the update and got closer to q than that distance. The outgoing objects are exactly the contrary of those. The moving objects of the third type are the current NNs that move within that distance and do not have any influence on the $q.result$.

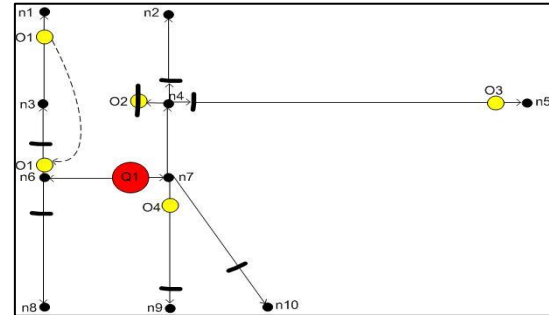


Figure 4: Incoming Object Update example

As there are normally a lot of updates of object movement for each timestamp, the result maintenance of the queries depends on the relative number of incoming and outgoing objects. Both cases may change the value of $q.kNN_dist$. If the outgoing objects are not more than the incoming objects, at least k objects lie within the current distance $q.kNN_dist$. So, the outgoing objects are removed from $q.result$ first and then the incoming ones are combined with the remaining objects of the $q.result$. Now, the expansion tree gets shrinked to get a kNN set again and the influence lists of the affecting edges get updated. The case of more outgoing objects than incoming objects is slightly different. As there are fewer objects left than k , the expansion tree has to be expanded, which is done by utilizing the knowledge of the remained objects to avoid a re-computation from scratch.

Query Updates

The movement of a certain query in the network means that the expansion tree is no more up-to-date, so it has to be updated in an efficient way avoiding a re-computation from scratch. This is only possible if the new position of the query is still in the area of the marks in the expansion tree and lies on an edge of it. If so, q has still a sub-tree of the old one that remains valid (assuming no object and edge updates). But the nodes, which are located in the invalid tree part, do not necessarily have the same shortest path as before, so this part is discarded and q has to be deleted

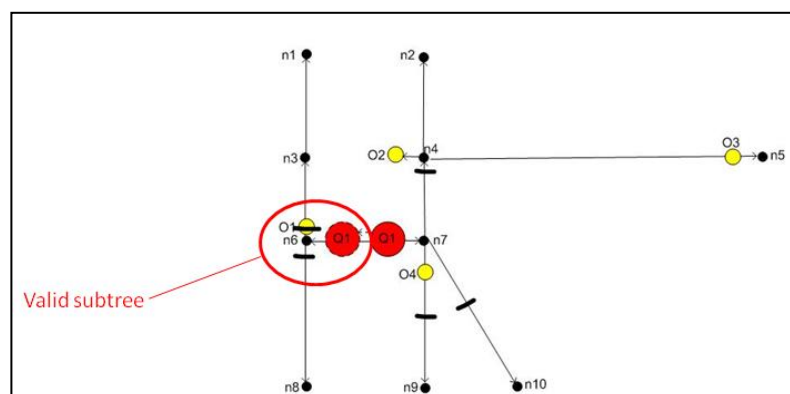


Figure 5: Query movement example

from the influence lists of the edges it spans. After that, the remaining NNs can be computed in the way it is done at the initial result computation.

Edge Updates

The last type of updates is done on the edge weights of a road segment. In case of an update, the edge weight increase implies that the sub-tree below that edge is in terms of time consumption further away. Therefore, those nodes or objects in that sub-tree could be reached faster through an alternative path. So, this part of the tree is not any longer valid. On the other hand, if an edge weight decreases, that affected sub-tree is closer in terms of time consumption. Outside that sub-tree, this update could imply that a new path to an object through the updated edge would be quicker than before, which is only possible for those objects/nodes that lie further then the shortest path from q to the endpoint of the updated edge. So, in the case of a decreasing edge weight, the affected sub-tree and the parts outside the sub-tree that are closer than the endpoint of the updated edge remain valid. Finally, the invalid part of the current $q.result$ is discarded and the query is removed

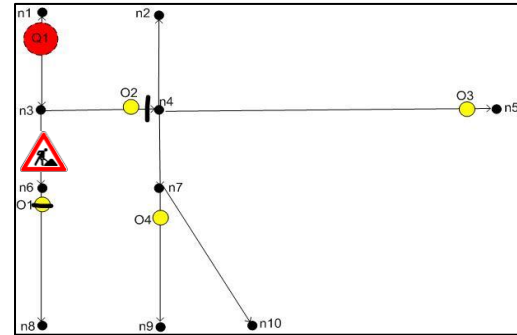


Figure 6: Before update of edge weight (n3-n6)

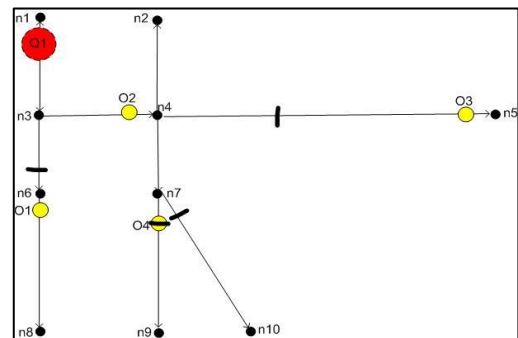


Figure 7: After update of edge weight (n3-n6)

from the corresponding influence lists. The valid part now serves to compute the new expansion tree in the manner of the initial result computation. The figures 6 and 7 show a case of an increasing edge weight due to road works.

4.3. The complete IMA algorithm

In the previous subsections, we only discussed separate update types. Now, all those updates have to be considered together for a single query. The main goal of the whole algorithm is to maximize the valid part of the expansion tree in order to reduce the cost of updating the tree.

The IMA initially checks whether the query itself has moved out of the area of the marks of $q.tree$. If this is the case, the whole expansion tree has to be re-computed from scratch, which is equal to an initial result computation. Remaining inside the area, the edge weight changes are considered before the query movement. It is important that the decreasing weights are considered before the increasing ones to ensure correctness. After that, the algorithm takes the query

movements into account as discussed in the subsection before. Then, the relevant object updates are taken into consideration. The outgoing ones are deleted from the $q.result$ and the incoming ones are inserted into it.

Finished with all updates, there is a remaining part of $q.tree$ that is still valid. This part is now the basis for the re-computation of the result of the query. The explained algorithm assumes that there are no more than one update per timestamp for each edge, object and query to guarantee the correct result. In an environment that does not support this timestamp issue, there has to be made a pre-computation step before every timestamp that summarizes the overall change of each element with respect to the timestamp before.

5. *GMA Algorithm*

The group monitoring algorithm extends the IMA by a new concept with shared executions of sequences. A sequence has been defined as follows: "A sequence is a path between two nodes n_i and n_j , such that (i) the degrees of n_i and n_j are not equal to 2 and (j) all intermediate nodes in the path have degree 2." Therefore, sequences lie between intersection nodes and/or terminal nodes, which are static nodes of the road network. The general approach of the GMA is to group together all the queries that lie in the same sequence and to monitor the static nodes in the network. This is done by storing all the sequences that end in a static node in its $n.S$ (sequences) and all the queries that lie in one of those sequences in the $n.Q$ (queries) for each static node. Hence, not the queries themselves are monitored anymore, but the static nodes are monitored and only the active ones. An active node both has at least one query in $n.Q$ and is not a terminal node. Each active node monitors the k nearest objects, which is the maximum k of the queries that are in $n.Q$.

The active nodes are monitored with the IMA except that the query movements are not treated. An *active node table* NT is invented to store the NNs and the expansion trees of the active nodes. It is similar to the *query table* QT of IMA. The influence list of an edge e now contains the ids of the active nodes affecting them and the queries in the sequence influenced by e . In addition, a *sequence table* ST stores all the information about the available sequences and the sets of edges.

The actual queries are maintained in a different way than IMA does. The initial NN search is done by expanding around the query. First, the objects in the current edge where the query lies are inserted to the $q.result$. Next, the NNs of the closer active node are inserted into $q.result$ if not already added. Note that the $q.kNN_dist$ is now used, but not mentioned in the paper, how it was set. Logically, it would be the distance to the furthest object in the current result. Now, the other active node of the sequence is taken into account and all the objects that lie closer than $q.kNN_dist$

and are not already in $q.result$ are inserted. The paper does not describe it further, but at this point, more than k objects could be in $q.result$ as every active node contains the k nearest objects, which is the maximum k of the queries that are in $n.Q$. Eventually, the $q.result$ should be shrunked at this point.

The result of a query can be changed due to four different events: (i) movement of the query, (ii) changes of the NNs of active nodes, (iii) object updates and (iv) edge updates. The movement of a query q is the only event that always affects q . In this case, a NN search from scratch is definitively done. The other three events do not necessarily affect the result of a query, only if the edge of the query and its influencing intervals around it are influenced by one of events (ii), (iii) or (iv). In the case of influencing updates, the q is not anymore valid and has to be recomputed from scratch similar to the query movement case.

6. Evaluation

For the experimental evaluation of the IMA and the GMA, sub-networks with 10K edges containing N objects and Q queries were used, where N and Q varied. With a clear definition of initial positions of objects and queries as well as the weights of the edge, the experiments started with some rules for the different update types. The queries then requested continuous monitoring of their k NNs for 100 timestamps. IMA and GMA were compared and additionally, there was a benchmark against them, which is an overhaul method that computes each query from scratch at every timestamp.

As expected, the GMA outperforms IMA which also outperforms the overhaul method. An important observation is that all algorithms scale well with the increasing number of objects. The biggest difference in terms of scalability was observed in the effect of an increasing number of queries on the CPU time and the memory requirements for increasing number of queries or increasing k . Other experiments with varying object agility and edge agility also show that GMA outperforms all the other algorithms. The only case, where IMA is the

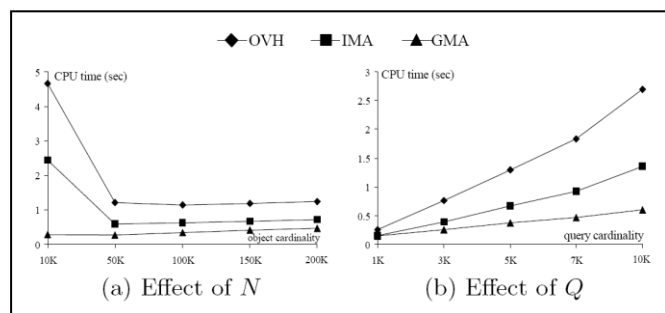


Figure 8: The effects of varying number of N/Q [1]

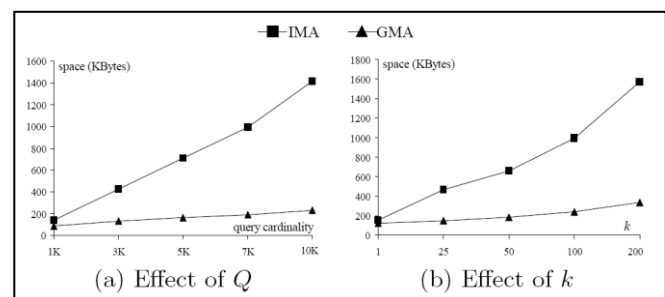


Figure 9: Effects on memory requirements [1]

winner of the competition, is in the experiment with an initial Gaussian distribution of the objects and an initial uniform distribution of the queries, which leads to a better CPU time for IMA.

The evaluation clearly shows that the GMA is much better than IMA and both outperform the overhaul method, which was clearly expected.

7. Conclusion

The paper elaborated two approaches of continuously monitoring the k nearest neighbors in a road network, which are the first attempts on that important problem. Both algorithms aim at reduction of the CPU cost of the central processing server. IMA monitors all the queries whereas GMA monitors intersections and uses some of the ideas of IMA. With the grouping of queries in sequences, the IMA could be optimized, which was well shown in realistic evaluations.

The overall impression of the paper is very good. First, the topic of the paper focuses on a problem that has never been researched on before. This is a kind of new step into a research field and gives the opportunity to other researchers to find other ways and better algorithms. Second, the observations of the evaluation showed good results and confirmed the expectations. Regarding the experiments, the question arises if the choice of the overhaul method as a benchmark is good enough. As there was no other work on this problem, it was probably not possible to take something else, but the two algorithms IMA and GMA can now be taken as benchmarks in future works.

8. *References*

- [1] K. Mouratidis, M. Liu, D. Papadias, and N. Mamoulis. Continuous Nearest Neighbor Monitoring in Road Networks. In VLDB, 2006.