An Efficient Path Computation Approach for Road Networks Based on Hierarchical Communities

Qing Song and Xiaofan Wang

Department of Automation, Shanghai Jiao Tong University, and Key Laboratory of System Control and Information Processing, Ministry of Education of China, Shanghai 200240 China (e-mail: qsong@sjtu.edu.cn, xfwang@sjtu.edu.cn).

Abstract: The problem of efficient path computation arises in several applications such as intelligent transportation and network routing. Although various algorithms exist for computing shortest paths, their heavy precomputation/storage costs and/or query costs hinder their application to real-time routing. By detecting hierarchical community structure in road networks, we develop a community-based hierarchical graph model that supports efficient path computation on large road networks. We then propose a new hierarchical routing algorithm that can significantly reduce the search space over the conventional algorithms with acceptable loss of accuracy. Finally, we evaluate our approach experimentally using a large, real-world road network to show the gain in performance.

1. INTRODUCTION

Computing the shortest path in real road networks is of great interest to us. In fact, we are dealing with such routing problems almost everyday. We want to get to a desired destination through the fastest way which may be the shortest travel time/distance, or we may balance among several influencing factors such as time, security, and toll charges. In network theory, this corresponds to the shortest path problem, and different influencing factors will only affect the form of arc weights based on user’s preference. The most classical shortest path algorithm is the Dijkstra algorithm (Dijkstra, 1959) with a complexity of \(O(|E|+|V|\log|V|)\), where \(|V|\) is the number of vertices and \(|E|\) is the number of arcs. Though Dijkstra algorithm computes the optimal solution in a theoretical sense, it is often far too slow for real-time route guidance applications.

Several speedup techniques such as bidirectional search (Pohl, 1971), goal-directed search (Hart, 1968), and hierarchical approach have been proposed, most of which perform some precomputation on the input data to reduce the search complexity. Naturally, combination of these techniques will yield in a faster response times, and recent researches have shown that it is most promising to combine hierarchical and goal-directed heuristics. Many powerful hierarchical approaches have been developed to strike a balance between preprocessing and query times. Liu (1997) and Jagadeesh (2002) cast the road network into hierarchies based on its inherent hierarchical topologies (e.g., road categories, road lengths, and speed limits) and then speed up the search by switching to a higher level of hierarchy. Rajagopalan (2008), Huang (1996), and Jung (2002) employ network partition techniques for structuring the network in a hierarchical fashion, and then constantly restrict the search to smaller and smaller networks of important edges.

In the past few years, complex networks have been studied across many fields of science (Kaza (2009); Li (2010)). A number of network features have been discovered (see Wang, 2003), among which the properties of hierarchical topology and community structure have attracted a great deal of interests recently (Newman (2004); Blondel (2008)). Communities are groups of vertices within which connections are dense but between which they are sparser. Networks often show a hierarchical structure of communities nested within each other. Identification of these communities can provide insight into better understanding and visualizing the structure of networks, and applications have ranged from technological networks to biological networks and social networks (Li (2010); Kaza (2009); Fortunato (2010)). In a road network where streets are mapped as edges and intersections as vertices, the intersections that locate close in a small region are more likely to form a community. The network is then decomposed, with adjacent subnetworks being loosely connected by the intergroup edges. All the shortest paths between different communities should go along one of these few edges, and this property can be used to accelerate the route computation process, which makes sense for the community detection of road networks.

A wide variety of methods have been developed for detecting communities in networks (see Fortunato (2010) for a recent review). Recently, fast algorithms for detecting hierarchical community structure in large networks have received increasing attention. A hierarchical community detection method proposed by Blondel et al. (2008), which is referred to as Louvain’s method, is adopted in this paper mainly due to its fast execution time and high quality of hierarchical communities detected.

Different from the graph decomposition methods used in Rajagopalan (2008), Huang (1996), and Jung (2002), which simply focus on the minimization of boundary nodes or
balancing of subnetwork size, community-based network partitioning seems more reasonable since real communities are not of equal size, e.g., big cities generally have more streets than small cities. In addition, the community-based partition method may be extremely fast and can be applied to nonplanar graphs with underpasses or fly-overs. Based on the identified communities, a hierarchical graph model is developed for structuring the road networks, and a heuristic hierarchical routing algorithm is then proposed to reduce search and provide near-optimal solutions. The algorithm is tested on New York road network and is found to achieve good performance.

2. HIERARCHICAL GRAPH MODEL

A road network can be viewed as a graph \( G=(V, E, W) \), where \( V \) is the set of vertices representing road intersections, and \( E=\{<i, j>| (i, j) \in V \wedge (i \neq j)\} \) is the set of edges representing roads. \( W=\{w_{ij}| w_{ij}=f(i, j)\} \) is the set of edge weights, where \( w_{ij} \) denotes the cost (e.g., distance) of edge \(<i, j>\) with a cost function \( f \). Suppose that \( G(V, E, W) \) is partitioned into \( p \) communities at level \( l \), which is depicted by a partition \( P=\{G_1^l, G_2^l, \ldots, G_p^l\} \) of pairwise disjoint sets \( V_1^l \subseteq V \), \( E_i^l \subseteq E \) such that \( \cup_{i=1}^p V_i^l = V \) and \( \cup_{i=1}^p E_i^l = E \). For any node \( i \), let \( c_i \) denote the community to which \( i \) belongs in a level \( l \) partition. A node \( i \) in \( V_i^l \) is called a border node of \( G_i^l \) if there exists an edge \(<i, j>\in E \) with \( j \in V_j^l \) and \( V_i^l \neq V_j^l \), and subgraphs \( G_i^l \) and \( G_j^l \) are then said to be adjacent. The border node set for \( G_i^l \) is denoted by \( \text{BORDER}(G_i^l) \).

Definition 1: Given a partition \( P=\{G_1^l, G_2^l, \ldots, G_p^l\} \) of \( G \), edges that link adjacent subgraphs \( G_i^l \) and \( G_j^l \) are called the intercommunity edges, and the set is denoted by

\[
\text{INTERCOM}(G_i^l, G_j^l) = \{<i, j>| (i, j) \in E \wedge (i \notin \text{BORDER}(G_i^l)) \wedge (j \in \text{BORDER}(G_j^l)) \}.
\]

Obviously, \( \bigcup_{i=1}^p \text{INTERCOM}(G_i^l, G_j^l) = E - \bigcup_{i=1}^p E_i^l \) with \( u_1 \leq u_2 \leq \ldots \leq u_p \) and \( u \neq v \).

Definition 2: Given a partition \( P=\{G_1^l, G_2^l, \ldots, G_p^l\} \) of \( G \), the community edge set of subgraph \( G_u^l \) \( (1 \leq u \leq p) \) is defined by

\[
\text{COMU}(G_u^l) = \{<i, j>| (i, j) \in \text{BORDER}(G_u^l) \}
\]

\[
( i \rightarrow j \ w_{ij} \text{(i \rightarrow j in } G_i^l) \wedge (i \neq j))\).
\]

The cost function \( w_{ij}(i, j) \) gives the shortest path cost from node \( i \) to \( j \) computed only with respect to subgraph \( G_i^l \).

Definition 3: Given a partition \( P=\{G_1^l, G_2^l, \ldots, G_p^l\} \) of \( G \), the high-level community graph of \( G \) with respect to partition \( P \) is defined by \( G^p=(V^p, E^p, W^p) \).

1. \( V^p = \bigcup_{u=1}^p \text{BORDER}(G_u^l) \).
2. \( E^p = \bigcup_{u=1}^p \text{INTERCOM}(G_i^l, G_j^l)) \cup \bigcup_{u=1}^p \text{COMU}(G_u^l) \), with \( 1 \leq u, v \leq p \) and \( u \neq v \).

3) For any edge \(<i, j>\in E^p \), its edge weight

\[
w_{ij} = f_1(i, j) \cdot (1 - \delta(c_i, c_j)) + f_2(i, j) \cdot \delta(c_i, c_j),
\]

where \( \delta -\text{function } \delta(x, y) = 1 \) if \( x = y \) and 0 otherwise. Together, \((G, G^p)\) constitute a two-level graph hierarchy.

For example, Fig.1a shows a graph \( G \) and its four subgraphs with border node sets \{\(b_1, b_2\), \{\(b_3, b_4\), \{\(b_5, b_6, b_7, b_8\), and \{\(b_9, b_{10}\)\}, respectively. Fig.1b shows the high-level graph constructed from \(G_1^l\) to \(G_5^l\), where each subgraph is represented as a complete graph composed of its border node set and the community edge set. The intercommunity edge can be thought of as forming bottlenecks between subgraphs. This can be extended to a \((k+1)\)-level graph hierarchy by introducing \(k\) levels of partitions \(P^1, P^2, \ldots, P^k\). Generally, with the increase in hierarchy level, the corresponding graph contains far fewer nodes and edges. Thus, by switching to a higher level graph hierarchy, the search space can be greatly reduced, and the time efficiency will be improved.

3. HIERARCHICAL ROUTING ALGORITHM

In this section, we introduce an efficient routing algorithm involving two possible scenarios, i.e., within-community routing (WICR) and between-community routing (BCR), depending on whether the source and destination nodes belong to the same community or not.

We first introduce some basic notations to be used later:

\[
w_c(i, j) \quad \text{— weight of edge } <i, j> \text{ in graph } G; \]

\[
SP_t(i, j) \quad \text{— shortest path from } i \text{ to } j \text{ within graph } G;\]
Before formally presenting the hierarchical routing algorithm, we introduce the precomputation task involving three steps:

Step 1) Community finding. Louvain’s method is adopted in this paper for decomposing the road network (see Blondel (2008) for a detailed description).

Step 2) Graph hierarchy construction. After the network is decomposed, we extract the border nodes and construct the so-called community edges between every pair of border nodes of each subgraph, which are equivalent to the shortest paths computed within that subgraph, and finally we link up adjacent subgraphs through intercommunity edges. These intercommunity edges, together with border nodes and the preprocessing-generated community edges, construct the high-level graph, thereby forming a two-level hierarchy.

Step 3) Community modification. Note that the weights of those community edges are just minimal within the context of one subgraph; therefore, we need some local modifications to get the globally optimal value. This involves the construction of a local shortest path tree from each node in the high-level community graph. Starting with the border node as the root, a Dijkstra search is stopped as soon as the distance of the currently visited node (from the root) is greater than the maximal weight of the community edge incident from the root. During this process, all border nodes in the source subgraph (the subgraph where the root located) have been visited. If the distance of a border node (from the root) is less than the weight of its corresponding community edge, then we store the edge in the modified community edge set with a modified weight.

3.2 WICR Algorithm

If the source and destination nodes are located within the same community, we use WICR algorithm to retrieve the optimal shortest path. There are two scenarios listed here.

Case 1) Both the source and destination nodes are border nodes. For this case, the shortest path cost is equivalent to the weight of the corresponding community edge based on Definition 2, and the path can be retrieved directly since it has been stored in memory.

Case 2) At least one of the source or destination node is not a border node. This can be resolved by applying the Dijkstra algorithm (or SPFPA (Duan, 1994)) for all pairs shortest path problem) on a rebuilding search area formed by the nodes and edges of that subgraph, together with the modified community edges within that subgraph. By restricting the search area, the average computational time is reduced compared with an exhaustive search over the original network, whereas the path retrieved is the same.

The optimality of the WICR algorithm is guaranteed by the following theorem that can be easily proven.

Theorem 1: Let $P = \{G^1, G^2, \ldots, G^p\}$ be a partition of graph $G$ and $MCOMU(G^i)$ be the modified community edge set of subgraph $G^i$ where $1 \leq u \leq p$. For any node pair $s, t \in V^i$, we have $SPC_G(s, t) = SPC_{G[U \cup MCOMU(G^i)]}(s, t)$.

3.3 BCR Algorithm

For the source and destination nodes in two distinct communities, we are inclined to a heuristic approach to significantly reduce the search space. The BCR algorithm is based on the assumption that coordinates of all nodes on a road map are available. It employs bidirectional search by using the knowledge on the location of currently visited subgraphs such that only relatively close subgraphs (among all adjacent forward and backward subgraph pairs) are selected for the next visit and thereby achieves acceleration. The algorithm is described as follows:

Step 1) Evaluation of subgraph distance. We define subgraph coordinates by averaging the $x$- and $y$-coordinates of all border nodes within that subgraph, which to some extent depict the shape of subgraphs. Let $(G^u^i.x, G^u^i.y)$ be the coordinates of subgraph $G^i_u$; then, the subgraph distance between $G^u^i$ and $G^v^i$ is given by

$$\text{DIST}(G^u^i, G^v^i) = \sqrt{(G^u^i.x - G^v^i.x)^2 + (G^u^i.y - G^v^i.y)^2}.$$  

For convenience, we precompute the preceding Euclidean distance for every pair of subgraphs and store them in memory.

Step 2) Selection of subgraph pair. The algorithm first identifies the source and destination subgraphs, which contain the source node $s$ and the destination node $t$, respectively. Then, starting with the source-destination subgraph pair, we consider all possible adjacent subgraph pairs between the source and the destination, and instead of selecting all, we select only $\alpha$ pairs of subgraphs with minimum subgraph distance. The value of $\alpha$ should not exceed the number of available subgraph pairs in order to reduce the search space. Then, for each selected subgraph pair, an upper bound is computed by enlarging the distance of its previously visited subgraph pair $\beta$ times. Only the subgraph pair whose distance is smaller than its upper bound...
would be added to the next-visit subgraph pair set. An appropriate value of $\beta$ can add a sense of direction to the search process, thus avoiding the sharp distance increase caused by network structure. In addition, for each pair of subgraphs in the next-visit subgraph pair set, we then repeat the preceding procedure to select the subgraphs for visit.

As shown in Fig.2, subgraph pairs ($G_1'$, $G_2'$) and ($G_3'$, $G_4'$) are selected for visit because their subgraph distances are minimal among all adjacent subgraph pairs with $DIST(G_1', G_2') < DIST(G_3', G_4')$ and $DIST(G_3', G_4') < DIST(G_2', G_1')$, where $\alpha = 2$ and $\beta = 1$. Apparently, the forward and backward search meet at subgraphs $G_2'$ and $G_3'$.

Next, we describe how to retrieve paths within each subgraph and how to connect these paths to form the final route. For convenience, we call the previously visited subgraph the parent subgraph and its next visit subgraph the children subgraph.

Step 3) Path construction. To compute a path from the parent subgraph $G_1'$ to a children subgraph $G_2'$, we proceed as follows.

1) Find the intercommunity edges from $G_1'$ to $G_2'$, and mark the endpoints $p_{11}$ and $p_{12}$ in the parent subgraph.

2) Find the optimal shortest paths from the source node $s$ to the marked endpoints $p_{11}$ and $p_{12}$ using the WICR algorithm.

3) Scan the intercommunity edges $<p_{11}, c_{11}>$ and $<p_{12}, c_{12}>$. If the distance to $c_{11}$ ($c_{12}$) through $p_{11}$ ($p_{12}$) is less than the previously recorded distance (infinity in the beginning), overwrite the distance, and concatenate the edge $<p_{11}, c_{11}>$ ($<p_{12}, c_{12}>$) with the path previously obtained.

Once the path to subgraph $G_2'$ is determined, its next-visit subgraphs $G_4'$ and $G_5'$ are released, and $G_2'$ becomes a tentative parent subgraph, etc. For each parent subgraph (other than the source or destination subgraph), we repeat the preceding procedure to compute the path, except that the second task is modified as follows.

2') Scan the border nodes in the parent subgraph. For each marked endpoint, check if the distance to it through any border node is less than the previously recorded distance (infinity in the beginning). If so, overwrite the distance and concatenate the corresponding community edge with the path previously obtained.

The whole process alternates between Steps 2 and 3 until forward and backward searches meet, or we may define a benchmark $\gamma$ such that the search will terminate if at least $\gamma$ pairs of subgraphs have met. The distance judgement in Step2 guarantees that the search will not continue from any meet subgraph pair with an appropriate value of $\beta$. The final path is the cost minimal one among these meet routes.

This approach is also applicable to a directed network, with the only difference being that the backward search from the destination node $t$ is executed on the reverse graph (Essam, 1970) of $G$ and $G'$. The performance of this heuristic algorithm is controlled by the parameters $\alpha$, $\beta$, and $\gamma$. The larger the values, the more likely the optimal path will be found and the longer the computation time will be. Generally, we choose $\alpha = 2$–6, $\beta = 1.0$–1.5, and $\gamma = (1$–3)$\alpha$, which yield a satisfactory balance between efficiency and accuracy.

One problem pertaining to the BCR algorithm is that the algorithm may stop without providing any solution if the value of $\alpha$ or $\beta$ is too small so that no pair of subgraphs is available for visit in Step2 before the forward and backward searches meet. This results from the differences in network structure and can be resolved by adding a self-adjustment loop such that the algorithm will automatically increase the value and restart the search if the next-visit subgraph pair set becomes null. The performance of the algorithm is discussed in the next section.

4. EXPERIMENTAL EVALUATION

To verify the validity of our hierarchical routing algorithm, we consider the New York road network with up to 264346 vertices and 733846 arcs (http://www.dis.uniroma1.it/~challenge9/download.shtml). We evaluate the efficiency of preprocessing and the performances of WICR and BCR in terms of computation time and accuracy, compared with two well-known approaches, i.e., the SPAH algorithm (Jung, 2002), which utilizes our hierarchical graph model and the A* algorithm with Euclidean distance as the cost function. All the algorithms were developed in Matlab 7.8.0 (R2009a) and conducted on an Intel Xeon X5482 Dual Core processor with 32GB of RAM. The system ran Microsoft Windows Vista.

4.1 Preprocessing

Louvain’s method (Blondel, 2008) is used in this paper with a variation of edge weight in the form of 100000 divided by the cost of the road such that close intersections are more likely to belong to the same community. The algorithm identifies a hierarchy of seven levels, with the modularity (Newman, 2004) being increased from 0.74 to 0.99. Here, the
top level of hierarchy is extracted to construct the high-level community graph with 460 communities and 8686 border nodes.

Table 1 shows the time required for each task. It is noted that the community modification task consumes more time than community finding and graph hierarchy construction, and it is needed only when we expect accurate results for within-community finding and graph hierarchy construction, and it can also be omitted in preprocessing and moved directly to the route computation part. Then, the time consumed in preprocessing would be 113.90s, and additional computation time would be added, i.e., probably 0.28s, to WICR.

4.2 WICR

To evaluate the average efficiency of the WICR algorithm, we randomly select five subgraphs and compute all pairs of shortest paths of each subgraph. The subgraph size involved in Tests 1-5 is 131, 270, 524, 826, and 1295, respectively. From Table 2, we can see that WICR requires an average computation time of 5.27s, whereas A* and SPAH require 20.95s and 31.56s, respectively. The superiority becomes more clear with the increase in subgraph size. It is also noted that A* outperforms SPAH for small subgraphs in Tests 1-4, and its performance deteriorates as the subgraph size increases in Test 5. The reason is that the search space of SPAH and A* varies little for small subgraphs, whereas SPAH searches on a more dense graph. However, for large subgraphs, the efficiency of SPAH is gradually enhanced since it searches less space. The experimental results also verify the accuracy of the WICR algorithm thereby are consistent with theoretical predictions.

4.3 BCR

A total of five tests are conducted, and each test is made to solve a set of 300 problems using the randomly generated source and destination nodes of different communities. The initialization parameters of the BCR algorithm are selected as follows: $\alpha = 4$ for the source and destination subgraphs and $\alpha = 2$ for others, $\beta = 1.1$, and $\gamma = 8$.

Table 3 reflects the efficiency and computational accuracy of the three algorithms. Both A* and SPAH produce optimal solutions while using Euclidean distance as the cost function. We can see that BCR achieves an average speedup factor of 7.2 over the A* algorithm and 2.9 over SPAH while providing routes that are longer than the optimal routes by 3.65% on average. It is also noted that the execution time of BCR increases more slowly than SPAH as the distance increases, which makes BCR more suitable for long-distance trip computations on large networks.

Fig. 3 shows the distribution of relative errors for the five test sets. It is shown that more than 90% of the node pairs are within the 10% error in all five tests, and more than 99% are within the 20% error. However, it cannot be ignored that the relative errors between few node pairs are still quite large, which is mainly due to the fact that the initialization parameters we set are relatively small, or the node pairs are close to each other, i.e., they reside in adjacent communities, and thus the shortest path may pass through another subgraph and bypass the intercommunity edges between the source and destination subgraphs.

4.4 Discussion

Our algorithm can be applied to dynamic scenarios due to its light preprocessing. This occurs in most practical situations when a new event (e.g., traffic congestion) happens that may affect the edge weights of several streets. Since communities are relatively stable structure, changes in a certain number of unimportant edges may not significantly affect the community structure of the network. Thus, there is no need to construct the communities every time in dynamic scenarios. Modularity can be used as a measure to assess the quality of the communities. The gain in modularity can be easily computed (see Blondel, 2008), and a community reconstruction process is performed only when the quality
degradation of modularity goes beyond a given value. The update cost is reduced by re-computing the community edges within affected subgraphs (containing weight-changing edges) and modifying the communities that involve accurate within-community computations. The update cost is quite low compared with initial precomputation, yielding fast queries for real time routing.

We compare the worst case runtime complexity of our heuristic hierarchical routing algorithm with Dijkstra and SPAH to show the gain in performance. In a graph with $|V|$ nodes and $p$ subgraphs, the average number of nodes in each subgraph is $\bar{z}=|V|/p$, and the computational complexity of the Dijkstra algorithm is $O(|V|^2)$. Suppose that the high-level community graph contains $|P^h|$ nodes and $|E^h|$ edges. Then, the path computation cost in SPAH is of complexity $O(\bar{z}^2)+O(|E^h|)$. In our hierarchical routing algorithm, the route computation process is broken into two parts, i.e., route computations within the source and destination subgraphs, which have a run time of $O(\bar{z}^2)$, and route computations over the high-level community graph. Considering the worst case, all the adjacent subgraph pairs will be selected for visit when $\alpha$ is large, then route computation in the high-level community graph takes $O(|E^h|)$ time. Hence, the worst-case computational complexity of our algorithm is $O(\bar{z}^2)+O(|E^h|)$ (or $O(\bar{z}^2)$ for WICR scenarios). Since the high-level community graph contains far fewer edges than that of the original network and for sparse networks $|E^h|<<|V|^2$ and $|E^f|<<|V|^2$; thus, we have $|E^h|<<|P^h|^2<<|V|^2$, where $|E|$ is the number of edges in the original network. Therefore, the worst case runtime complexity of our heuristic hierarchical routing algorithm is much lower than SPAH and the Dijkstra algorithm, and time efficiency is greatly improved.

5. CONCLUSION

In this paper, we employ community detection algorithm for retrieving the hierarchical structure of road networks. We develop a hierarchical graph model based on the decomposed network, and propose a new hierarchical routing algorithm, which could compute optimal routes for within-community node pairs and near-optimal routes for between-community node pairs on large road networks. The proposed algorithm is also applicable to networks with dynamic link weights, which makes it promising for real-time routing. The community detection method used for decomposing the network is not unique; thus, it is possible to try other new techniques and use different community partitions (with different modularity values) in future and choose the one with best performance.

ACKNOWLEDGMENT

The authors are grateful for the financial support for this work provided by the Major State Basic Research Development Program of China (973 Program) under Grant 2010CB731400 and the NSF of China under Grant 61074125.

REFERENCES