# k-NN Search in Time-Dependent Road Networks

Ugur Demiryurek and Cyrus Shahabi
Computer Science Department, University of Southern California, Los Angeles, CA, USA

## Synonyms

Location Based Services; Nearest Neighbor Search; Route Planning; Spatial Networks; Spatio-temporal Networks; Time-Dependent Road Networks

## Definition

The ever-growing popularity of online map services and their wide deployment in smartphones and car-navigation systems have led to extensive use of location-based services. One of the most popular classes of such services is k-nearest neighbor (kNN) queries where users search for geographical points of interests (e.g., restaurants) and the corresponding directions and travel times to these locations in road networks. The online nature of these services requires almost instant response time. Accordingly, many algorithms have been developed to speed up kNN search in road networks by using a variety of precomputation techniques. However, all the existing approaches and commercial services for $k$NN search in road networks make the simplifying assumption that the fastest path between any two nodes in the network is unique by assuming the weight of each edge in the road network is constant. This assumption makes precomputation techniques feasible in terms of both preprocessing time and storage complexity. However, in the real world, the actual travel time on network edges depends on the arrival time to that edge – i.e., travel time is *time dependent*, and hence the fastest path between any nodes is not unique. It is infeasible to extend existing precomputation techniques to time-dependent road networks as input size (i.e., super-polynomial number of fastest paths) would increase drastically by yielding exponential precomputation time and prohibitively large storage requirements. This entry introduces the problem of $k$NN search in time-dependent spatial networks where the weight of each edge is a function of arrival time and studies an algorithm based on two novel indexing schemes – Tight Network Index ($TNI$) and Loose Network Index ($LNI$) – that enables efficient $k$NN search. The main idea of the algorithm is to localize the search and reduce the problem to a point location problem by decoupling the process of computing k-nearest neighbors from the invocation of expensive distance computation in network space.

## Historical Background

On static road networks where edge costs are constant, $k$NN search problem has been exten-

© Springer International Publishing Switzerland 2015
S. Shekhar et al. (eds.), *Encyclopedia of GIS*,
DOI 10.1007/978-3-319-23519-6_1528-1

sively studied, and many efficient speed-up techniques have been developed. In Papadias et al. (2008), introduced Incremental Network Expansion (INE) and Incremental Euclidean Restriction (IER) methods to support $k$NN queries in spatial networks. With INE, starting from query location $q$, all network nodes reachable from $q$ in every direction are visited in order of their proximity to $q$ until all k-nearest data objects are located. On the other hand, $IER$ exploits the Euclidean restriction principle in which the results are first filtered in Euclidean space and then refined by using the network distance. In Kolahdouzan and Shahabi (2004), proposed first-degree *network Voronoi diagrams* to partition the spatial network to network Voronoi polygons ($NVP$), one for each data object. They used the $NVP$s to efficiently find k-nearest neighbors in static road networks. Cho and Chung (2005) presented a system UNICONS where the main idea is to integrate the precomputed $k$NNs into the Dijkstra algorithm. Hu et al. [4] proposed a distance signature approach that precomputes the network distance between each data object and network vertex. The distance signatures are used to find a set of candidate results, and Dijkstra is employed to compute their exact network distance. Huang et al. addressed the $k$NN problem using *Island* approach 2005 where each vertex is associated to all the data points that are in radius $r$ (the so-called islands) covering the vertex. With their approach, they utilized a restricted network expansion from the query point while using the precomputed islands. In Huang et al. (2007), introduced *S-GRID* where they partition the spatial network to disjoint subnetworks and precompute the shortest path for each pair of border points. Recently, Samet et al. (2008) proposed a method where they associate a label to each edge that represents all nodes to which a shortest path starts with this particular edge. The labels are used to traverse *shortest path quadtrees* that enables geometric pruning to find the network distance. With all these studies, the network edge weights are assumed to be static (i.e., travel-time functions of all edges are constants), and hence, the fastest path is calculated on static networks (Zhang 2008). Unfortunately, once time-dependent edge weights are considered, all the proposed $k$NN solutions assuming constant edge weights and/or relying on distance precomputation would fail as the shortest path computation is invalidated with time-varying edge weights.

## Scientific Fundamentals

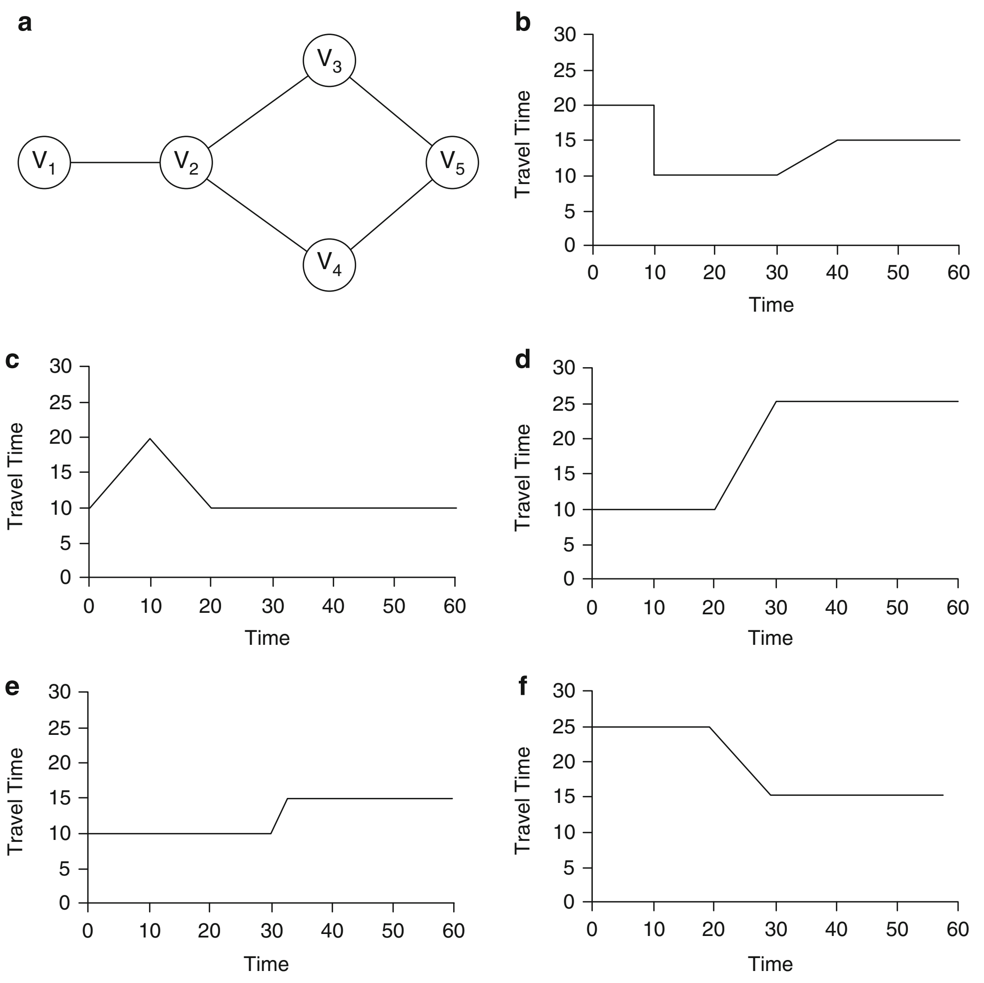
With $k$NN search in time-dependent road networks, the road network is modelled as a *time-dependent-weighted graph* where the nonnegative weights are time-dependent travel times (i.e., positive piecewise linear functions of time) between the nodes. The road network contains a set of data objects (i.e., points of interest such as restaurants) as well as query objects searching for their $k$NN. In this entry, the road network is assumed to satisfy the first-in, first-out (FIFO) property with which moving objects exit from an edge in the same order they entered the edge.

### Problem Definition

**Definition 1** A time-dependent graph ($G_T$) is defined as $G_T(V, E)$ where $V = \{v_i\}$ is a set of nodes and $E \subseteq V \times V$ is a set of edges representing the network segments each connecting two nodes. For every edge $e(v_i, v_j) \in E$, and $v_i \neq v_j$, there is a *cost function* $c_{(v_i,v_j)}(t)$, where $t$ is the time variable in time domain $T$. An edge cost function $c_{(v_i,v_j)}(t)$ specifies the travel time from $v_i$ to $v_j$ starting at time $t$.

Figure 1 illustrates a road network modeled as a time-dependent graph $G_T(V, E)$. While Fig. 1a shows the graph structure, Fig. 1b–f depict the time-dependent edge costs (i.e., travel time) as piecewise linear functions for the corresponding edges.

**Definition 2** Let $\{s = v_1, v_2, \ldots, v_k = d\}$ represent a path which contains a sequence of nodes where $e(v_i, v_{i+1}) \in E$ and $i = 1, \ldots, k-1$. Given a $G_T$, a path ($s \rightsquigarrow d$) from source s to destination d, and a departure-time at the source $t_s$, the time-dependent travel time $TT(s \rightsquigarrow d, t_s)$ is the time it takes to travel along the path. Since the travel time of an edge varies depending on the arrival time to that edge (i.e., arrival dependency), the travel time is computed as follows:



**k-NN Search in Time-Dependent Road Networks, Fig. 1** A time-dependent graph $G_T(V, E)$. (**a**) Graph $G_T$. (**b**) $c_{1,2}(t)$. (**c**) $c_{2,3}(t)$. (**d**) $c_{2,4}(t)$. (**e**) $c_{4,5}(t)$. (**f**) $c_{3,5}(t)$ change

$TT(s \rightsquigarrow d, t_s) = \sum_{i=1}^{k-1} c_{(v_i,v_{i+1})}(t_i)$ where $t_1 = t_s, t_{i+1} = t_i + c_{(v_i,v_{i+1})}(t_i), i = 1,..,k$.

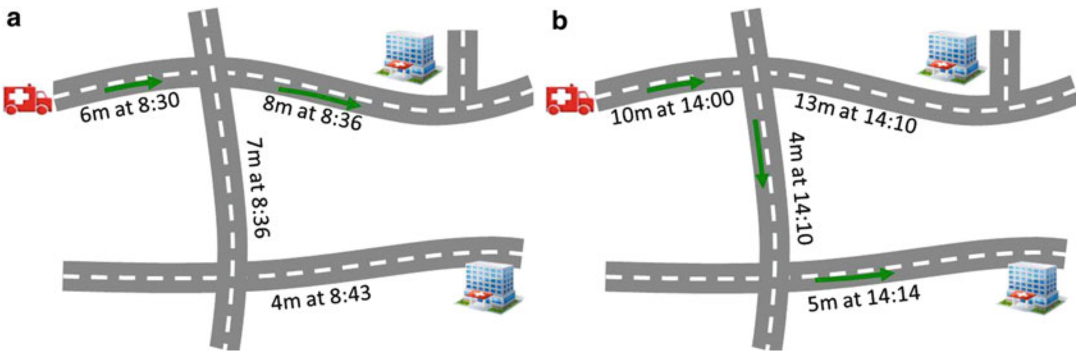
**Definition 3** Lower-bound graph $\underline{G}(V, E)$ is a graph with the same topology (i.e., nodes and edges) as graph $G$, where the weight of each edge $c_{(v_i,v_j)}$ is fixed (not time dependent) and is equal to the minimum possible weight $c^{\min}_{(v_i,v_j)}$ where $\forall$ $e(v_i, v_j) \in E, t \in T$ $c^{\min}_{(v_i,v_j)} \le c_{(v_i,v_j)}(t)$.

**Definition 4** Lower-bound travel time $LTT(s \rightsquigarrow d)$ of a path is less than the actual travel time along that path and computed w.r.t $\underline{G}(V, E)$ as $LTT(s \rightsquigarrow d) = \sum_{i=1}^{k-1} c^{\min}_{(v_i,v_{i+1})}, i = 1, \ldots, k$.

For example, in Fig. 1b, $c^{\min}_{(v_1,v_2)}$ = 10 units.

**Definition 5** Upper-bound graph $\bar{G}(V, E)$ is a graph with the same topology (i.e., nodes and edges) as graph $G$, where the weight of each edge $c_{(v_i,v_j)}$ is fixed (not time dependent) and is equal to the maximum possible weight $c^{\max}_{(v_i,v_j)}$ where $\forall$ $e(v_i, v_j) \in E, t \in T$ $c^{\max}_{(v_i,v_j)} \ge c_{(v_i,v_j)}(t)$.

**Definition 6** Upper-bound travel time $UTT(s \rightsquigarrow d)$ of a path is greater than the actual travel time along that path and computed w.r.t $\bar{G}(V, E)$ as



**k-NN Search in Time-Dependent Road Networks, Fig. 2** Time-dependent NN search. (**a**) 1-NN query at 8:30 AM. (**b**) 1-NN query at 2:00 PM

$UTT(s \rightsquigarrow d) = \sum_{i=1}^{k-1} c^{\max}_{(v_i,v_{i+1})}, i = 1,..,k$. For example, in Fig. 1, $c^{\max}_{(v_1,v_2)} = 20$ units.

As discussed, $UTT(s \rightsquigarrow d)$ and $LTT(s \rightsquigarrow d)$ are the maximum and minimum possible times to travel along a path, respectively. To illustrate, consider $t_s = 5$ and path $(v_1, v_2, v_3, v_5)$ in Fig. 1 where $TT(v_1 \rightsquigarrow v_5, 5) = 45$, $UTT(v_1 \rightsquigarrow v_5) = 65$, and $LTT(v_1 \rightsquigarrow v_5) = 35$. Note that time dependency is not considered when computing $UTT$ and $LTT$; hence, $t$ is not included in their definitions. Given the definitions of $TT$, $UTT$, and $LTT$, the following property holds for any path in $G_T$: $LTT(s \rightsquigarrow d) \leq TT(s \rightsquigarrow d, t_s) \leq UTT(s \rightsquigarrow d)$. This property will be used in subsequent sections to establish some properties of our algorithm.

**Definition 7** Given a $G_T(V, E)$, $s$, $d$, and $t_s$, the time-dependent fastest path $TDFP(s, d, t_s)$ is a path with the *minimum travel time* among all paths from $s$ to $d$ for starting time $t_s$.

**Definition 8** A time-dependent $k$-nearest neighbor query (TD-kNN) is defined as a query that finds the $k$-nearest neighbors of a query object which is moving on a time-dependent network $G_T$. Considering a set of $n$ data objects $P = \{p_1, p_2, \ldots, p_n\}$, the TD-$k$NN query with respect to a *query point* $q$ finds a subset $P' \subseteq P$ of $k$ objects with minimum time-dependent travel time to $q$, i.e., for any object $p' \in P'$ and $p \in P - P'$, $TDFP(q, p', t) \leq TDFP(q, p, t)$.

Figure 2 shows an example of time-dependent $k$NN search where an emergency vehicle is looking for the nearest hospital (with least travel time) at 8:30 AM and 2:00 PM on a particular road network. The time-dependent travel time (in minutes) and the arrival time for each edge are shown on the edges. Note that the travel times on an edge change depending on the arrival time to the edge in Fig. 2a, b. Therefore, the query issued at 8:30 AM and 2:00 PM would return different results.

## Time-Dependent kNN Search

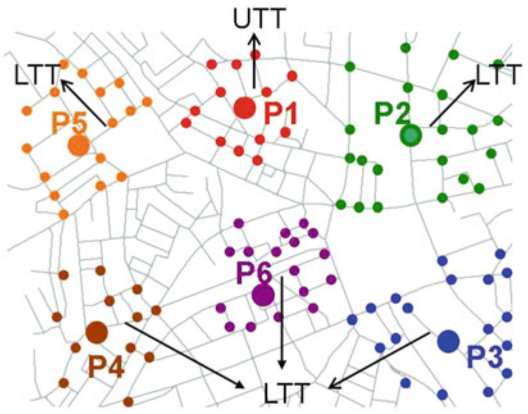
Several naive approaches can be used to evaluate $k$NN queries in time-dependent road networks. Firstly, Dreyfus (1969) has studied the relevant problem of time-dependent shortest path planning and showed that this problem can be solved by a trivially modified variant of Dijkstra algorithm. Consequently, a primitive solution for the time-dependent $k$NN problem can be developed based on the incremental network expansion (INE (Papadias et al. 2008)) approach where Dreyfus's modified Dijkstra algorithm is used for time-dependent distance calculation. However, considering the prohibitively high overhead of executing blind network expansion particularly in large networks with a sparse (but perhaps large) set of data objects, this approach is too slow to scale for real-time $k$NN query processing. Secondly, one can use time-expanded graphs (George et al. 2007) to model the time-dependent networks. The time-expanded model discretizes the time domain $\mathrm{T} = [t_0,\ t_n]$ into $n$ points of time and constructs a static graph by making n copies of each node and each edge, respectively. Specifically, time-expanded network replicates the original network for each discrete time unit $t = 0, 1, \ldots, t_n$, where $t_n$ is determined by the total duration of the time interval under consideration. This model connects a node and its copy at the next instant in addition to the edges in the original network, replicated for every time instant. The weight of an edge in time-expanded network is the time difference between the time events associated with its endpoints. With this model, the time-dependent $k$NN problem is reduced to the problem of computing the minimum-weight paths through a series of static networks. Although this approach allows for exploiting the existing algorithms for $k$NN computation on static networks, it often fails to provide the correct results (may return false nearest neighbors) because the model misses the state of the network between any two discrete time instants. Moreover, since the original network is replicated across time instants, the size of the network increases, hence, resulting in a very high storage overhead and slower response time (see Demiryurek et al. 2010). Finally, with a third baseline approach, one can consider to precompute time-dependent shortest paths between all possible sources and destinations in the network. However, shortest path precomputation on time-dependent road networks is challenging, because with time-dependent road networks, the shortest path depends on the departure time from the source, and therefore, all possible shortest paths between all possible source and destination nodes for *all possible departure-times* should be precomputed and stored. Obviously, this is not a viable solution because of unpractical precomputation time, and the storage requirements for the precomputed paths would quickly exceed reasonable space limitations.

Demiryurek et al. developed a comprehensive TD-$k$NN algorithm (2010) that (a) efficiently answers the time-dependent kNN queries in near real time, (b) is independent of density and distribution of the data objects, and (c) effectively handles the database updates where nodes, links, and data objects are added or removed. Their approach involves two phases: an off-line spatial network indexing phase and an online query processing phase. During the off-line phase, the spatial network is partitioned into *tight cells (TC)* and *loose cells (LC)* for each data object $p$, and two complementary indexing schemes *Tight Network Index (TNI)* and *Loose Network Index (LNI)* are constructed. During the online phase, TD-$k$NN finds the first nearest neighbor of $q$ by utilizing the $TNI$ and $LNI$ constructed in the off-line phase. Once the first nearest neighbor is found, TD-$k$NN expands the search area by including the neighbors of the nearest neighbor to find the remaining *k-1* data objects.

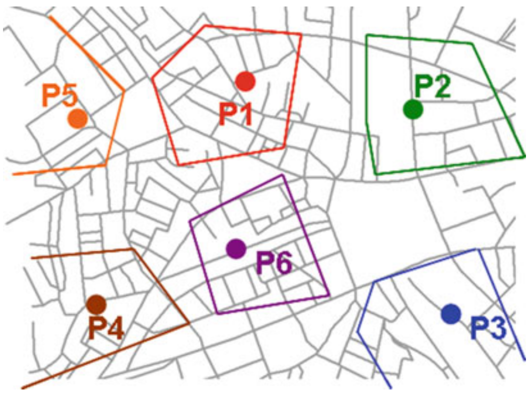
## Indexing Time-Dependent Network

This section explains the construction of tight and loose network cells (subgraphs) and corresponding index structures used to evaluate $k$NN queries in time-dependent road networks. The main idea behind partitioning the network to tight and loose network cells is to localize the $k$NN search and minimize the costly time-dependent shortest path computation. These index structures efficiently find the data object(s) (i.e., generator of a tight or loose cell) that is in the shortest time-dependent distance to the query object $q$.

**Tight Network Index (TNI):** The tight cell $TC(p_i)$ is a subnetwork around $p_i$ in which any query object is guaranteed to have $p_i$ as its nearest neighbor in a time-dependent network. The tight cell of a data object is computed by using simultaneous Dijkstra algorithm that grows shortest path trees from each data object. Specifically, the algorithm expands from $p_i$ (i.e., the generator of the tight cell) assuming maximum (upper-bound) travel time (corresponding to minimum speed on road segments) between the nodes of the network (i.e., UTT), while in parallel, it expands from each and every other data object assuming minimum travel time (corresponding to maximum speed on road segments) between the nodes (i.e., LTT). The expansions stop when the shortest path trees meet. The main rationale is that if the upper-bound travel time between a query object $q$ and a particular data object $p_i$ is less than the lower-



**k-NN Search in Time-Dependent Road Networks, Fig. 3** Tight cell construction for $P_1$



**k-NN Search in Time-Dependent Road Networks, Fig. 4** Tight cells

bound travel times from $q$ to any other data object, then obviously $p_i$ is the nearest neighbor of $q$ in the time-dependent network. We repeat the same process for each data object to compute its tight cell. Figure 3 demonstrates the network expansion from the data objects during the tight cell construction for $p_1$. The expansion (i.e., visited nodes) for each data object is illustrated with different colors, where UTT is used for expanding from $p_1$ and LTT is used other data objects. The same process is repeated for each data object to find the corresponding tight cells. Figure 4 shows the end result with which the tight cell of each data object is represented as a polygon. The edges of the polygons are generated by connecting the adjacent border nodes (i.e., nodes where the shortest path trees meet) of a generator to each other.

**Lemma 1** *Let P be a set of data objects* $P = \{p_1, p_2, \ldots, p_n\}$ *in* $G_T$ *and* $TC(p_i)$ *be the tight cell of a data object* $p_i$*. For any query point* $q \in TC(p_i)$*, the nearest neighbor of* $q$ *is* $p_i$*, i.e.,* $\{\forall q \in TC(p_i), \forall p_j \in P, p_j \neq p_i, TDFP(q, p_i, t) < TDFP(q, p_j, t)\}$.

Lemma 1 states that if a query point $q$ is inside a specific tight cell, one can immediately identify the generator of that tight cell as the nearest neighbor for $q$ (see Demiryurek et al. (2010) for proof). Although the tight cells are constructed based on the network distance metric, each tight cell is actually a polygon in Euclidean space as illustrated in Fig. 4. Therefore, the problem is reduced to a point location problem in Euclidean space. One can implement polygon inclusion algorithm to find the tight cell that contain $q$. However, polygon inclusion may be slow with a large number of data objects, and hence, this stage can be expedited by using a spatial index structure generated on the tight cells. In particular, the tight cells can be indexed using spatial index structures (e.g., R-tree, quadtree). This way, a function (i.e., $contain(q)$) invoked on a spatial index structure on the tight cells would efficiently return the tight cell whose generator has the minimum time-dependent network distance to $q$. Tight Network Index is defined as follows.

**Definition 9** Let P be the set of data objects $P = \{p_1, p_2, \ldots, p_n\}$; the Tight Network Index is a spatial index structure generated on $\{TC(p_1), TC(p_2), \ldots, TC(p_n)\}$.

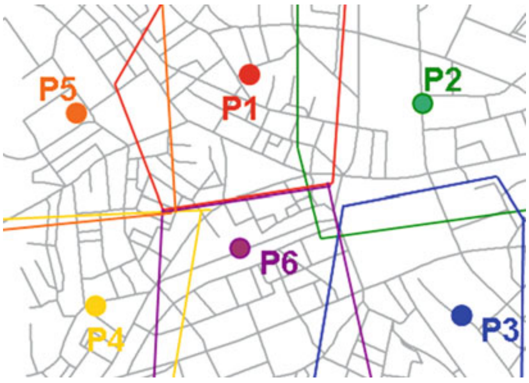
As illustrated in Fig. 4, the set of tight cells often does not cover the entire network. For the cases where $q$ is located in an area which is not covered by any tight cell, Loose Network Index ($LNI$) is used to identify the candidate nearest data objects.

**Loose Network Index (LNI):** The loose cell $LC(p_i)$ is a subnetwork around $p_i$ outside which any point is guaranteed *not* to have $p_i$ as its nearest neighbor. Similar to the construction process for $TC(p_i)$, parallel shortest path tree expansion is used to construct $LC(p_i)$. However, this time, minimum travel time between the nodes of the network (i.e., $LTT$) is used to expand from $p_i$ (i.e., the generator of the loose cell) and maximum travel time (i.e., $UTT$) to expand from every other data object.

**Lemma 2** *Let $P$ be a set of data objects $P = \{p_1, p_2, \ldots, p_n\}$ in $G_T$ and $LC(p_i)$ be the loose cell of a data object $p_i$. Data object $p_i$ is guaranteed not to be the nearest neighbor of $q$ if $q$ is outside of $LC(p_i)$, i.e., $\{\forall q \notin LC(p_i), \exists p_j \in P, p_j \neq p_i, TDFP(q, p_i, t) > TDFP(q, p_j, t)\}$.*

Lemma 2 states that data object $p_i$ is guaranteed *not* to be the nearest neighbor of $q$ if $q$ is outside of the loose cell of $p_i$ (see Demiryurek et al. (2010) for the proof). In other words, if $q$ moves with the minimum travel time on a path (corresponding to the fastest speed on each edge of the path) in the loose subnetwork toward $p$ and minimum travel time toward some other site $p'$, it will arrive at $p'$ before $p$. As illustrated in Fig. 5, loose cells, unlike $TC$s, collectively cover the entire network and have some overlapping regions among each other.

Based on the properties of tight and loose cells, loose cells and tight cells have common edges. The data objects that share common edges are referred to as *direct neighbors*, and loose cells of the direct neighbors always overlap. For example, consider Fig. 5 where the direct neighbors of $p_2$ are $p_1$, $p_3$, and $p_6$. This property is especially useful for processing *k-1* neighbors (see section "*k*NN Query:") after finding the first nearest neighbor. The direct neighbors are determined during the generation of the loose cells and stored in a data component. Therefore, finding the neighboring cells does not require any complex operation.



**k-NN Search in Time-Dependent Road Networks, Fig. 5** Loose cells

Similar to $TNI$, one can use spatial index structures to access loose cells efficiently. Loose Network Index is defined as follows.

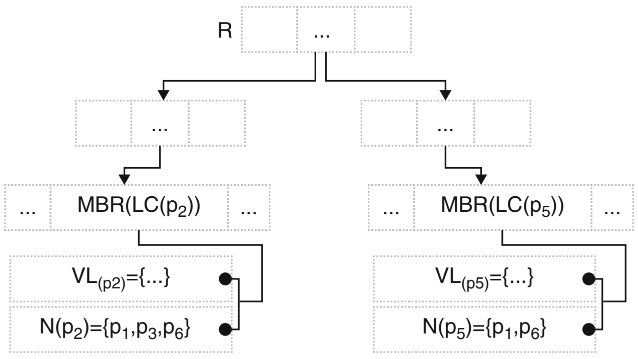
**Definition 10** Let P be the set of data objects $P = \{p_1, p_2, \ldots, p_n\}$; the Loose Network Index is a spatial index structure generated on $\{LC(p_1), LC(p_2), \ldots, LC(p_n)\}$.

$LNI$ and $TNI$ are complementary index structures. This means that if a $q$ cannot be located with $TNI$ (i.e., $q$ falls outside of any $TC$), then $LNI$ is used to identify the $LC$s that contain $q$. Based on Lemma 2, the generators of such $LC$s are the only NN candidates for $q$.

**Index Structures and Network Updates:** As we discussed, different spatial index structures can be used to index tight and loose cells. In this entry, we show how loose cells can be indexed using R-tree (Guttman 1984). The index construction for tight cells is very similar. Figure 6 depicts Loose Network Index structure based on R-tree (termed LN R-tree) to index loose cells. As shown, LN R-tree has the basic structure of an R-tree generated on minimum bounding rectangles of loose cells. The difference is that we modify R-tree by linking its leaf nodes to the the pointers of additional components that facilitate TD-kNN query processing. These components are the direct neighbors ($N(p_i)$) of $p_i$ and the list of nodes ($VL_{p_i}$) that are inside $LC(p_i)$. While $N(p_i)$ is used to filter the set of candidate nearest neighbors where $k > 1$, $VL_{p_i}$ is used to prune the search space. Tight Network Index is a similar data structure without extra pointers at the leaf nodes.

When the set of data objects and/or the travel-time profiles change, TNI and LNI need to be updated. Fortunately, due to local precomputation

K



**k-NN Search in Time-Dependent Road Networks, Fig. 6** LN R-tree

nature of TD-kNN, the affect of the updates with both cases is *local*, hence requiring minimal change in tight and loose cell index structures. Each update operation and corresponding modifications on the index structures are discussed below.

*Data Object Updates*: Two types of data object update are possible, insertion and deletion (object relocation is performed by a deletion following by insertion at the new location). With a location update of a data object $p_i$, only the tight and loose cells of $p_i$'s neighbors are updated (hence, local update). In particular, when a new data object $p_i$ is inserted (deletion is similar and hence not discussed), first the loose cell(s) $LC(p_j)$ containing $p_i$ is found and $LC(p_j)$ is shrunk. In addition, since the loose cells and tight cells share common edges, the region that contains $LC(p_j)$ and $LC(p_j)$'s direct neighbors needs to be adjusted. Toward that end, all the neighbors of $LC(p_j)$ are found and recompute; the tight and loose cells of these direct neighbors are the only ones affected by the insertion. In sum, the new tight and loose cells for $p_i$, $p_j$, and $p_j$'s direct neighbors are updated.

*Edge Travel-Time Updates*: The tight and loose cells are generated based on the minimum (LTT) and maximum (UTT) travel times of the edges in the network that are time independent. The only case that these index structures need an update is when minimum and/or maximum travel time of an edge changes, which is not that frequent. Moreover, similar to the data object updates, the affect of the travel-time profile update is local. When the maximum and/or minimum travel time of an edge $e_i$ changes in the network, we first find the loose cell(s) $LC(p_j)$ that overlaps with $e_i$ and thereafter recompute the tight and loose cells of $LC(p_j)$ and its direct neighbors.

## TD-*k*NN Query Processing (Online)

This section explains how TNI and LNI are used to process *k*NN queries in time-dependent road networks. The very first step in *k*NN query is to find the nearest neighbor (i.e., $k$=1) and then search for remaining $k-1$ data objects.

**Nearest Neighbor Query:** Given the location of $q$, first, a depth-first search from the $TNI$ root to the node that contains $q$ is performed. If a tight cell that contains $q$ is located, the generator of that tight cell is returned as the first NN. If $q$ cannot be located in $TNI$ (i.e., when $q$ falls outside all tight cells), the algorithm proceeds to search $LNI$. At this step, one or more loose cells that contain $q$ may be found. Based on Lemma 2, the generators of these loose cells are the only possible candidates to be the NN for $q$. Therefore, TDFP is computed to find the distance (time-dependent travel time) between $q$ and each candidate in order to determine the first NN. It is important to note that in this step either no (when $q$ is found in TC) or minimum number (when $q$ is located in overlapping LCs) of distance computation is performed, which in turn the performance is improved significantly.

***k*NN Query:** The algorithm for finding the remaining *k-1* NNs is based on the direct neighbor property discussed above. It has been shown with Lemma 3 that the second NN is among the direct neighbors of the first NN (see Demiryurek et al. (2010) for proof). Once the second NN is identified, the search continues by including the neighbors of the second NN to find the third NN and so on.

**Lemma 3** *The i-th nearest neighbor of $q$ is always among the direct neighbors of the i-1 nearest neighbors of $q$.*

To exemplify, consider Fig. 5 where $p_2$ is the first NN of $q$. The second NN will be among the direct neighbors (i.e., $\{p_1, p_3, p_6\}$) of $p_2$. The time-dependent fastest path from $q$ to $\{p_1, p_3, p_6\}$ must be computed in order to find the second NN. Let's assume that after time-dependent fastest path computation, $p_6$ is identified as the second NN. Then, the third NN will be among the direct neighbors of $p_2$ and $p_3$.

**Time-Dependent Fastest Path Computation:** As explained, once the nearest neighbor of $q$ is found and the candidate set is determined, the time-dependent fastest path from $q$ to all candidates must be computed in order to find the next NN. A time-dependent A* algorithm is used in Demiryurek et al. (2010) to find the fastest path between $q$ and candidate set. The time-dependent A* algorithm takes advantage of a very useful property of loose cells stated in Lemma 4.

**Lemma 4** *If $p_i$ is the nearest neighbor of $q$, then the time-dependent shortest path from $q$ to $p_i$ is guaranteed to be inside the loose cell of $p_i$.*

That is, Demiryurek, et al. prove that given $p_i$ is the nearest neighbor of $q$, the time-dependent shortest path from $q$ to $p_i$ is guaranteed to be in $LC(p_i)$. This property indicates that only the edges contained in the loose cell of $p_i$ is considered when computing TDFP from $q$ to $p_i$. This property allows to localize the time-dependent shortest path search by extensively pruning the search space. Since the localized area of a loose cell is substantially smaller as compared to the complete graph, the computation cost of TDFP is significantly reduced.

## Key Applications

### Online Maps and Car Navigation Systems

The applications of k-nearest neighbor search are of great interest to online maps such as Google Maps and Bing Maps as well as to car navigation systems. These systems provide k-nearest neighbor solutions based on the static information ignoring the time-dependency inherit in road networks, and hence, they fail to provide optimal results. Therefore, online map services and car navigation systems can adopt the ideas presented in this entry to provide more accurate results for their users.

### Emerging Mobile Applications

Smartphone-based resource sharing services – such as ride-sharing applications like Uber – are becoming more and more ubiquitous. These services make their planning and optimization based on nearest neighbor search and its variations. Time-dependent kNN search – with unique "time-ahead" view of traffic information on road networks – can certainly result in more optimal planning for such services.

### Other Network Applications

We are witnessing a data explosion era, in which huge data sets of billions or more are represented by (time-dependent) graphs such as internet routing, social networks, targeted marketing, surveillance sensor systems, and so on. On these large-scale data sets, nearest neighbor search is fundamental for lots of applications including proximity search, similarity search, clustering, as well as many other machine learning and data mining problems.

## Future Directions

This entry studied a generalized type of k-nearest neighbor query where the edge weights of the network are time varying rather than fixed. Given the importance of time dependency for accurate and realistic spatial query processing inroad networks as well as increasing the use of traffic sensors, we believe that there will be rapid growth of interest in developing various spatial query processing in time-dependent road networks. Therefore, one direction for future work is to investigate novel data structures and models for effective representation of time-dependent road networks. This is crucial in supporting the development of efficient and accurate time-dependent algorithms, while minimizing the precomputation and storage and cost of the algorithms. Second direction is to extend this work to a variety of other spatial queries – such as reverse nearest neighbor, range, and skyline queries – in time-dependent road networks.

## References

Cho H-J, Chung C-W (2005) An efficient and scalable approach to cnn queries in a road network. In: An efficient and scalable approach to CNN queries in a road network (VLDB'05), Trondheim, pp 865–876

Demiryurek U, Banaei-Kashani F, Shahabi C (2010) Efficient k-nearest neighbor search in time-dependent spatial networks. In: Bringas PG, Hameurlain A, Quirchmayr G (eds) Database and expert systems applications: proceedings of the 21st international conference, part I (DEXA'10), Bilbao, 30 Aug–3 Sept 2010 Springer, Berlin/Heidelberg, pp 432–449

Demiryurek U, Banaei-Kashani F, Shahabi C (2010) Towards k-nearest neighbor search in time-dependent spatial network databases. In: Kikuchi S, Sachdeva S, Bhalla S (eds) Databases in networked information systems: proceedings of the 6th international workshop (DNIS'10), Aizuwakamatsu, 29–31 Mar 2010, pp 296–310

Dreyfus P (1969) An appraisal of some shortest path algorithms. Oper Res 17(3): 395–412. New York

George B, Kim S, Shekhar S (2007) Spatio-temporal network databases and routing algorithms: a summary of results. In: Proceedings of the 10th international conference on advances in spatial and temporal databases (SSTD'07), Boston, pp 460–477

Guttman A (1984) R-trees: a dynamic index structure for spatial searching. In: Proceedings of the 1984 ACM SIGMOD international conference on management of data (SIGMOD'84), Boston, pp 47–57

Huang X, Jensen CS, Saltenis S (2005) The island approach to nearest neighbor querying in spatial networks. In: Proceedings of the 9th international conference on advances in spatial and temporal databases (SSTD'05), Angra dos Reis, pp 73–90

Huang X, Jensen CS, Lu H, Saltenis S (2007) S-grid: a versatile approach to efficient query processing in spatial networks. In: Proceedings of the 10th international conference on advances in spatial and temporal databases (SSTD'07), Boston, pp 93–111

Kolahdouzan M, Shahabi C (2004) Voronoi-based k nearest neighbor search in spatial networks. In: Proceedings of the thirtieth international conference on very large data bases (VLDB'04), Toronto, vol 30, pp 840–851

Papadias D, Yiu M, Mamoulis N, Tao Y (2008) Nearest neighbor queries in network databases. In: Encyclopedia of GIS. Springer, pp 772–776

Samet H, Sankaranarayanan J, Alborzi H (2008) Scalable network distance browsing in spatial databases. In: Proceedings of the 2008 ACM SIGMOD international conference on management of data (SIGMOD'08), Vancouver, pp 43–54

Zhang D (2008) Fastest-path computation. In: Encyclopedia of GIS. Springer, pp 309–313