# V*-$k$NN: an Efficient Algorithm for Moving $k$ Nearest Neighbor Queries

Sarana Nutanong[†‡], Rui Zhang[†], Egemen Tanin[†‡], Lars Kulik[†‡]
[†]Department of Computer Science and Software Engineering
University of Melbourne, Victoria, Australia
{sarana,rui,egemen,lars}@csse.unimelb.edu.au
[‡]NICTA Victoria Laboratory, Australia

***Abstract*— This demonstration program presents the V*-$k$NN algorithm, an efficient algorithm to process moving $k$ nearest neighbor queries (M$k$NN). The V*-$k$NN algorithm is based on a safe-region concept called the V*-Diagram. By incrementally maintaining the V*-Diagram, V*-$k$NN continuously provides accurate M$k$NN query results and supports dynamically changing values of $k$. Our approach exploits information regarding the current location of the query point and the search space in addition to the data objects. As a result, the V*-$k$NN has much smaller IO and computation costs than existing methods.**

## I. Introduction

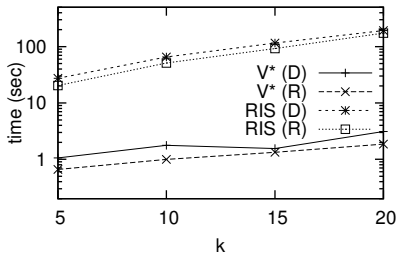
Location-based services provide position information with a high degree of temporal precision to provide services to the users. The two following scenarios are examples of location-based queries where the answer may change according to the location of the query issuer. An ambulance driver is keeping track of the nearest available emergency wards at all times. A delivery driver maintains a list of several nearest gas stations while driving around the city. The queries are sent to a server that processes the queries and returns the answers. The server also has to continuously maintain the answer set which may change depending on the location of the query point. These queries are *location-based continuous spatial queries* and the scenarios above are typical examples of *moving $k$ nearest neighbor queries* (M$k$NN).

To avoid unnecessary data access, one can define a region in which the query point can move without changing the result. Such region is known as a *safe region*. Using a safe-region-based method, an M$k$NN query can be processed by: (i) finding the current $k$ NNs; (ii) calculating a region that the current $k$ NNs are valid, i.e., a safe-region of the $k$NN; (iii) repeating the first two steps when the query point moves out of the safe region. Therefore, a safe-region-based method continuously provides accurate answers without the need for sampling.

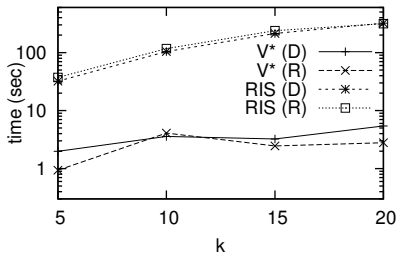
This demonstration program shows the internal mechanism of a safe-region-based technique called the V*-Diagram [1] and the associated algorithm, called V*-$k$NN. The V*-Diagram has the following key advantages:

(i) It requires no precomputation.
(ii) It incrementally computes answers and therefore efficiently adapts to changes – such as insertions and deletions of objects, as well as, dynamically changing values of $k$.

Figure 1 shows experimental results on the performance comparison between our V*-$k$NN algorithm and the *retrieve-influence-set $k$NN* (RIS-$k$NN) algorithm [2], which is the best existing method, with respect to $k$. The experiments were conducted on: (i) two types of trajectories, directional (D) and random (R); (ii) two real datasets of $65,743$ and $119,897$ postal addresses from California and North-Eastern USA, respectively. V*-$k$NN outperformed RIS-$k$NN by one order of magnitude.




(a) California




(b) North-Eastern USA

Fig. 1. Effect of $k$

The V*-Diagram is based on two types of safe regions, the *fixed-rank region* (FRR) and the *safe region with regard to a data point*, which are described in Sections II and III, respectively. A $k$NN safe region called the *integrated safe region* (ISR), which is formulated by calculating the intersection of the FRR and the safe region with regard to the current $k^{th}$ NN, is presented in Section IV. The V*-$k$NN algorithm is explained in Section V. Our demonstration program, which shows how the V*-$k$NN algorithm operates, is described in Section VI.

1084-4627/09 $25.00 © 2009 IEEE
DOI 10.1109/ICDE.2009.63

## II. Fixed-rank Region

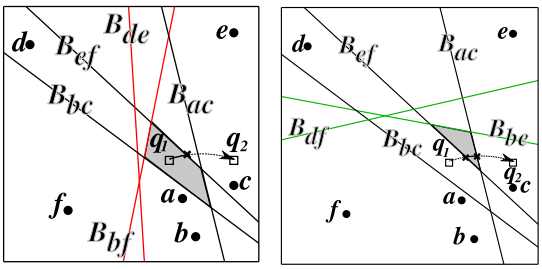
Kulik and Tanin [3] introduced an algorithm called *incremental rank updates* (IRU) to compute regions where the ranking of all the objects (based on their distances) is the same, i.e., *fixed-rank regions*.

Based on the observation that only rank-adjacent objects can swap their ranks, defining the FRR of $n$ objects requires at most $(n-1)$ bisectors[1] of the $(n-1)$ pairs of rank-adjacent objects. Continuously monitored objects are ranked according to their distances to a moving query point. Updates are performed by maintaining a rank-sorted list of objects and its corresponding list of bisectors of pairs of rank-adjacent objects (*rank-adjacent bisectors*). Each time the query point crosses a bisector, the ranks of the two corresponding objects are swapped and the list of rank-adjacent bisectors are updated.

Given a list $L$ of objects, $\langle \boldsymbol{p_1}, \boldsymbol{p_2}, ..., \boldsymbol{p_n} \rangle$, the FRR of $L$ is the set of all points such that for any point $\boldsymbol{v}$ in the set, $\boldsymbol{p_1}, \boldsymbol{p_2}, ..., \boldsymbol{p_n}$ are already sorted in ascending order by their distances to $\boldsymbol{v}$. Let $H_{\boldsymbol{p_i p_j}}$ be defined as $\{\boldsymbol{v} \in DS : dist(\boldsymbol{v}, \boldsymbol{p_i}) \leq dist(\boldsymbol{v}, \boldsymbol{p_j})\}$, where $DS$ is the data space. An FRR is a function of a list and is defined as follows.

*Definition 1 (Fixed-rank region):*

$$F\langle \boldsymbol{p_1}, \boldsymbol{p_2}, ..., \boldsymbol{p_n} \rangle = \bigcap_{i=1}^{n-1} H_{\boldsymbol{p_i p_{i+1}}}$$




(a) Initial step (b) First update

Fig. 2. Incremental rank update

Figure 2 shows how the FRR is maintained by keeping track of the $(n-1)$ rank-adjacent bisectors. The current FRR is shown as a grey region. Let us assume that $\boldsymbol{q}$ is the location of a moving query point which starts at $\boldsymbol{q_1}$. In Figure 2(a), $\boldsymbol{q}$ is at $\boldsymbol{q_1}$ and the ranking is initially $\langle \boldsymbol{a}, \boldsymbol{c}, \boldsymbol{b}, \boldsymbol{f}, \boldsymbol{e}, \boldsymbol{d} \rangle$. The corresponding list of bisectors is $\langle\ B_{\boldsymbol{ac}},\ B_{\boldsymbol{bc}},\ B_{\boldsymbol{bf}},\ B_{\boldsymbol{ef}},\ B_{\boldsymbol{de}} \rangle$. Then $\boldsymbol{q}$ crosses $B_{\boldsymbol{ef}}$ in Figure 2(b). This causes $\boldsymbol{e}$ and $\boldsymbol{f}$ to swap their ranks. Therefore $B_{\boldsymbol{bf}}$ and $B_{\boldsymbol{de}}$ are replaced by $B_{\boldsymbol{be}}$ and $B_{\boldsymbol{df}}$, respectively.

## III. Safe region with regard to a data point

In the V*-Diagram, the incremental NN query is called repeatedly to help maintain $(k+x)$ NNs, where $x$ is the number of auxiliary objects. Let us assume that $\boldsymbol{z}$ is the farthest known object to $\boldsymbol{q_b}$, i.e., the $(k+x)^{th}$ NN of $\boldsymbol{q_b}$. Any objects that has the distance to $\boldsymbol{q_b}$ smaller than that of $\boldsymbol{z}$ is guaranteed to be discovered via the query. Hence, the area enclosed by the circle centered at $\boldsymbol{q_b}$ with the radius of $dist(\boldsymbol{q_b}, \boldsymbol{z})$ is called the *known region*, $W(\boldsymbol{q_b}, \boldsymbol{z})$ (shown as the area enclosed by the dotted circle in Figure 3).

As the query point $\boldsymbol{q}$ moves away from $\boldsymbol{q_b}$, some objects are no longer *reliable*. Specifically, an object is reliable when we know that there are no objects outside the known region nearer to $\boldsymbol{q}$ than the object. The region that contains all reliable objects is termed as the reliable region. Assume that the query point is at $\boldsymbol{q'}$ in Figure 3. Mathematically, $\boldsymbol{p}$ being in the reliable region with regard to $\boldsymbol{q'}$ is expressed as:

$$dist(\boldsymbol{q'}, \boldsymbol{p}) \leq dist(\boldsymbol{q_b}, \boldsymbol{z}) - dist(\boldsymbol{q_b}, \boldsymbol{q'}). \quad (1)$$

The reliable region with regard to $\boldsymbol{q'}$ is the area enclosed by the dashed circle in Figure 3.

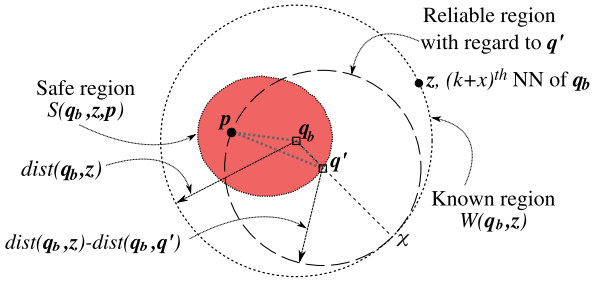



Fig. 3. The known, reliable, and safe regions

The *safe region with regard to the data point* $\boldsymbol{p}$ is the region in which the query point can move while $\boldsymbol{p}$ remains reliable. Figure 3 shows the safe region with regard to $\boldsymbol{p}$, $S(\boldsymbol{q_b}, \boldsymbol{z}, \boldsymbol{p})$. Based on Inequality (1), we formally define the safe region with regard to $\boldsymbol{p}$ and a given know region $W(\boldsymbol{q_b}, \boldsymbol{z})$ as:

*Definition 2 (Safe region with regard to a point):*

$$S(\boldsymbol{q_b}, \boldsymbol{z}, \boldsymbol{p}) = \{\boldsymbol{q'} : dist(\boldsymbol{q'}, \boldsymbol{p}) + dist(\boldsymbol{q_b}, \boldsymbol{q'}) \leq dist(\boldsymbol{q_b}, \boldsymbol{z})\}$$

## IV. Integrated Safe Region

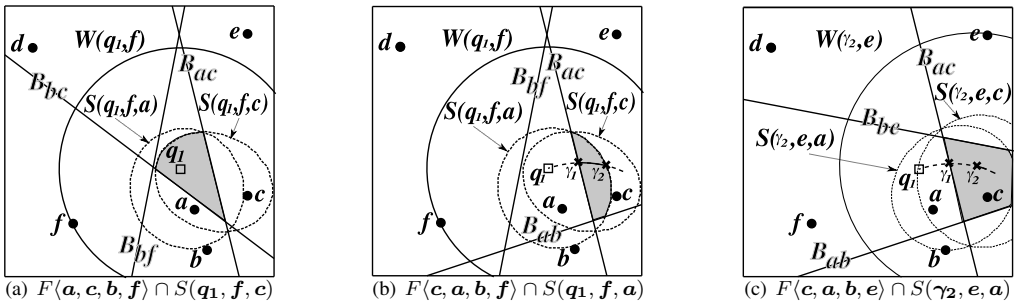
An integrated safe region (ISR) is a region that the order-sensitive list of $k$ NNs is unchanged as long as the query point stays inside. The definition is given as follows.

*Definition 3 (Integrated safe region (ISR)):* Let $O$ be the $(k+x)$NN set of $\boldsymbol{q_b}$, $L$ be the list of these $(k+x)$ objects sorted by their distances to the query point, $\boldsymbol{z}$ be the farthest retrieved object to $\boldsymbol{q_b}$, and $\boldsymbol{p_k}$ be the $k^{th}$ object in $L$. The integrated safe region with respect to $\boldsymbol{q_b}$, $\boldsymbol{z}$, $\boldsymbol{p_k}$ and $L$ is defined as:

$$I(\boldsymbol{q_b}, \boldsymbol{z}, \boldsymbol{p_k}, L) = F(L) \cap S(\boldsymbol{q_b}, \boldsymbol{z}, \boldsymbol{p_k})$$

As exemplified in Figure 4(a), four objects retrieved by a 4NN query ($k=2$ and $x=2$) at $\boldsymbol{q_1}$ are $\langle \boldsymbol{a}, \boldsymbol{c}, \boldsymbol{b}, \boldsymbol{f} \rangle$. Point $\boldsymbol{q_1}$ is the most recent point of retrieving $(k+x)$ NNs. As long as the query point $\boldsymbol{q}$ remains in $F\langle \boldsymbol{a}, \boldsymbol{c}, \boldsymbol{b}, \boldsymbol{f} \rangle \cap S(\boldsymbol{q_1}, \boldsymbol{f}, \boldsymbol{c})$ (the grey region), we are ensured that (i) no object outside $W(\boldsymbol{q_1}, \boldsymbol{f})$ is nearer to the two NNs, $\boldsymbol{a}$ and $\boldsymbol{c}$; (ii) the ranking of $\langle \boldsymbol{a}, \boldsymbol{c}, \boldsymbol{b}, \boldsymbol{f} \rangle$ is unchanged.

[1]The *bisector* of two objects $\boldsymbol{a}$ and $\boldsymbol{b}$ is the set of points where each point is equidistant to $\boldsymbol{a}$ and $\boldsymbol{b}$.




(a) $F\langle \boldsymbol{a}, \boldsymbol{c}, \boldsymbol{b}, \boldsymbol{f}\rangle \cap S(\boldsymbol{q_1}, \boldsymbol{f}, \boldsymbol{c})$ (b) $F\langle \boldsymbol{c}, \boldsymbol{a}, \boldsymbol{b}, \boldsymbol{f}\rangle \cap S(\boldsymbol{q_1}, \boldsymbol{f}, \boldsymbol{a})$ (c) $F\langle \boldsymbol{c}, \boldsymbol{a}, \boldsymbol{b}, \boldsymbol{e}\rangle \cap S(\boldsymbol{\gamma_2}, \boldsymbol{e}, \boldsymbol{a})$

Fig. 4. Example for Algorithm 1, ($k = 2$, $x = 2$)

## V. Algorithm

The ISR is maintained through the V*-$k$NN algorithm (Algorithm 1). The initialization steps (Lines 1-3) involve retrieving the $(k + x)$ NNs at the initial location of the query point and creating the FRR and the safe region ($S_k$) of the current the $k^{th}$ NN. In the event loop (Lines 4-13), we check if the query point exits the FRR. If yes, updates are performed accordingly. We also check if the query point exits $S_k$. If yes, we retrieve the new $(k+x)$ NNs at the current location of the query point, then the FRR and $S_k$ are accordingly created.

**Algorithm 1** V*-$k$NN

```
1: Retrieve (k + x) NNs
2: Create the FRR of the (k + x) NNs
3: Create the safe region (S_k) of the k^th NN
4: while more events do
5:    if the query point exits the FRR then
6:       Update the ranking and bisectors
7:       Update S_k (if the k^th NN changed)
8:    else if the query point exits S_k then
9:       Retrieve new (k + x) NNs
10:      Create the FRR of the (k + x) NNs
11:      Create S_k
12:   end if
13: end while
```

Next, we give a running example of the algorithm. Recall the example in Figure 4(a). At the starting point $\boldsymbol{q_1}$, 4 NNs are retrieved in the order of $\langle \boldsymbol{a}, \boldsymbol{c}, \boldsymbol{b}, \boldsymbol{f}\rangle$. The ISR is $F\langle \boldsymbol{a}, \boldsymbol{c}, \boldsymbol{b}, \boldsymbol{f}\rangle \cap S(\boldsymbol{q_1}, \boldsymbol{f}, \boldsymbol{c})$. Figure 4(b) shows how the ISR changes after $\boldsymbol{q}$ crosses $B_{\boldsymbol{ac}}$. At the instant that $\boldsymbol{q}$ is crossing $B_{\boldsymbol{ac}}$ at $\boldsymbol{\gamma_1}$, $\boldsymbol{a}$ and $\boldsymbol{c}$ swap their ranks. The object list $L$ becomes $\langle \boldsymbol{c}, \boldsymbol{a}, \boldsymbol{b}, \boldsymbol{f}\rangle$, and this causes both $F(L)$ and $S_k$ to change. Now $\boldsymbol{a}$ becomes $\boldsymbol{p_k}$ ($2^{nd}$ NN), and hence the ISR becomes $F\langle \boldsymbol{c}, \boldsymbol{a}, \boldsymbol{b}, \boldsymbol{f}\rangle \cap S(\boldsymbol{q_1}, \boldsymbol{f}, \boldsymbol{a})$ (the grey region). The current 2 NNs, $\boldsymbol{c}$ and $\boldsymbol{a}$, are reported to the user in order.

Figure 4(c) shows how the ISR changes after $\boldsymbol{q}$ exits $S(\boldsymbol{q_1}, \boldsymbol{f}, \boldsymbol{a})$. At the instant that $\boldsymbol{q}$ is exiting $S(\boldsymbol{q_1}, \boldsymbol{f}, \boldsymbol{a})$ at $S(\boldsymbol{q_1}, \boldsymbol{f}, \boldsymbol{a})$, the new $(k + x)$ NNs, $\langle \boldsymbol{c}, \boldsymbol{a}, \boldsymbol{b}, \boldsymbol{e}\rangle$, are retrieved. The ISR becomes $F\langle \boldsymbol{c}, \boldsymbol{a}, \boldsymbol{b}, \boldsymbol{e}\rangle \cap S(\boldsymbol{\gamma_2}, \boldsymbol{e}, \boldsymbol{a})$ (the grey region).

## VI. V*-$k$NN Demonstration Program

The demonstration program is a Java applet consisting of three panels:

(i) *Main Panel.* This is located in the middle part of the applet. This panel displays the locations of the data objects and the query points. The user can also interact with this panel by dragging the query point $\boldsymbol{q}$ around the screen to observe how the V*-$k$NN algorithm operates by updating the ISR.

(ii) *Control Panel.* Using this panel, one can change the values of $k$ and $x$, as well as, the appearance of the main panel. This panel is located at the top of the applet.

(iii) *Text Panel.* This panel displays descriptions of changes happened in the main and control panels.

Figure 5 displays four screen shots of the demonstration program, where each shows: (i) known objects as labelled dots; (ii) unknown objects as unlabelled dots; (iii) the known region after a query was executed at $\boldsymbol{q_b}$; (iv) the reliable region after $\boldsymbol{q}$ moved away from $\boldsymbol{q_b}$. Depending on the combination of display settings, the program may display the fixed-rank region and/or the safe region with regard to the $k^{th}$ NN. The default display mode is exemplified in Figure 5(a) where the fixed-rank and safe region are not displayed.

As can be seen in Figure 5(b), by checking the `Safe Region` box, the safe region with regard to the current $k^{th}$ NN is shown. Similarly, to display only the fixed-rank region, the user can check only the `Fixed-rank Region` box as can be seen in Figure 5(c). The integrated safe region which is the intersection of the two regions can be shown by checking both boxes as shown in Figure 5(d).

## VII. Acknowledgments

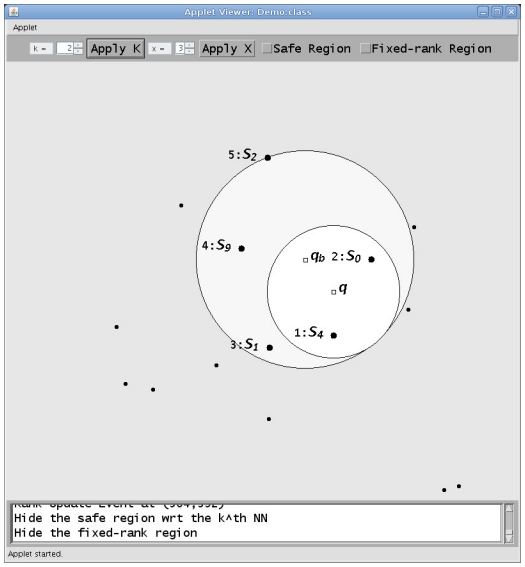
We would like to thank the anonymous reviewers for their comments that improved our paper. This work is supported under the Australian Research Council's Discovery funding scheme (project number DP0880215).

## References

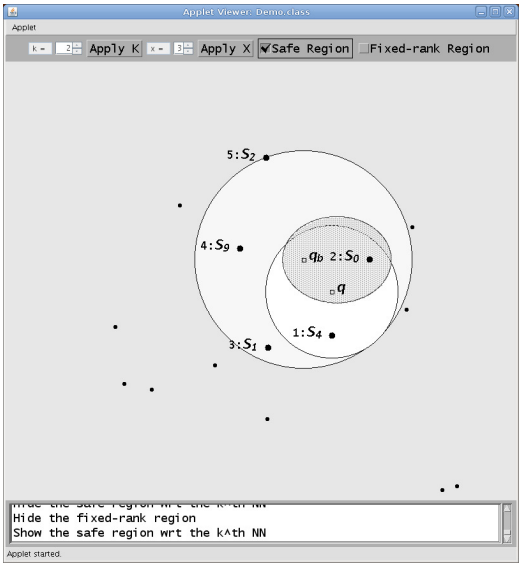
[1] S. Nutanong, R. Zhang, E. Tanin, and L. Kulik, "The V*-Diagram: A query dependent approach to moving kNN queries," in *VLDB*, 2008, pp. 1095–1106.

[2] J. Zhang, M. Zhu, D. Papadias, Y. Tao, and D. L. Lee, "Location-based spatial queries," in *SIGMOD*, 2003, pp. 443–454.

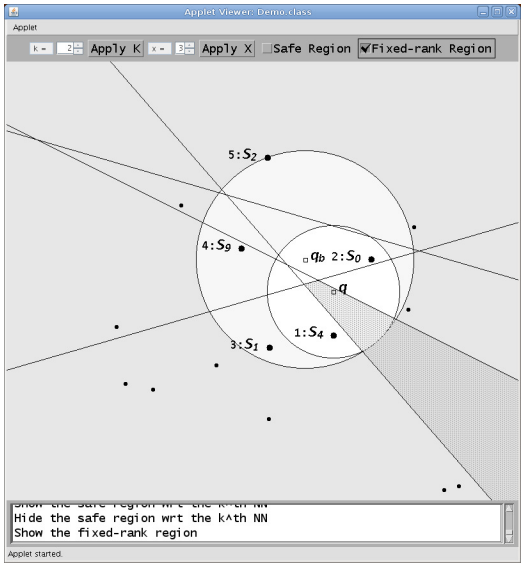
[3] L. Kulik and E. Tanin, "Incremental rank updates for moving query points," in *GIScience*, 2006, pp. 251–268.



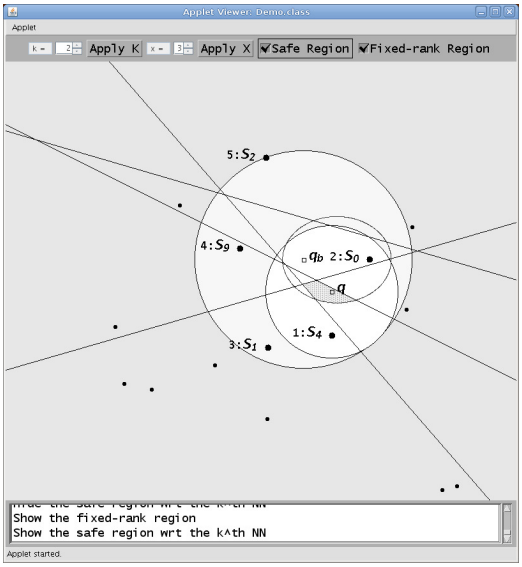

(a) No FRR or safe region displayed




(b) Safe region of the $k^{th}$ NN displayed




(c) FRR displayed




(d) Both FRR and the safe region of the $k^{th}$ NN displayed

Fig. 5. V*-$k$NN with $k = 2$ and $x = 3$