Continuous k-Nearest Neighbour Strategies Using the mqrtree

Wendy Osborn(✉)

Department of Mathematics and Computer Science,
University of Lethbridge, Lethbridge, AB T1K 3M4, Canada
wendy.osborn@uleth.ca

Abstract. In this paper, two strategies for processing a continuous k-nearest neighbor query for location-based services are proposed. Both use a spatial access method, the mqrtree, for locating a safe region. The mqrtree supports searching within the structure, so searches from the root are not required - a property which is exploited in the strategies. However, the proposed strategies will work with most spatial access methods. The strategies are evaluated and compared against a repeated nearest neighbor search. It is shown that both approaches achieve significant performance gains in reducing the number of times a new safe region must be identified, in both random and exponentially distributed points sets.

1 Introduction

A location-based service provides information to a user of a mobile device (e.g. smartphone or tablet) based on their location, interests and the type of query issued by the user [22]. One example of such a query is to find the nearest k restaurants to the user. This query is referred to as a k-nearest neighbour (k-NN) query. Another important aspect to processing queries for location-based services is the aspect that the user is moving around as their query is being processed. As the user moves, this affects the result for their query - in the above case, the k nearest restaurants - and therefore this results needs to be continually updated. Therefore, an efficient continuous k-nearest neighbour query processing strategy is important, especially when the query is initiated from a mobile device [22].

Many strategies have been proposed that process nearest neighbour queries for location-based services. Several utilize spatial access methods, including [1,2,4,5,10,20,21,23]. In addition, several strategies have been proposed for processing continuous k-nearest neighbour queries in static point sets, including [3,7,8,11–17,19]. These strategies have several limitations, including repeated searching of a spatial access method, caching more data on the device than is desired, requiring that safe regions - regions where a query remains valid when it moves around - need to be constructed from scratch whenever a new one is needed, and knowing the query trajectory in advance.

The mqrtree [18] is a spatial access method with desirable properties, including two-dimensional nodes and the maintenance of spatial relationships between

L. Barolli et al. (Eds.): NBiS 2018, LNDECT 22, pp. 168–181, 2019.
https://doi.org/10.1007/978-3-319-98530-5_15

objects. This structure lends itself to continuous spatial query processing that requires no repeated searching from the root, no requirement to know the trajectory in advance, and no requirement to construct safe regions from scratch, since potential ones exist in the index structure. The mqrtree has been applied to process k-nearest neighbour queries [20], and preliminary results are promising. However, when used to process a continuous k-nearest neighbour queries, many adjacent queries produce the same results, which is undesirable and unnecessary.

There, this paper proposes two strategies for continuous k-nearest neighbour processing that utilize an approximation-based spatial access method (e.g. mqrtree). The strategies obtain safe regions from the spatial access method. In addition, the strategies determine where to begin the search *within the structure* for an updated safe region, when a new one is needed. An experimental evaluation and comparison shows that the strategies significantly reduce the number of new safe regions needed for processing a user trajectory of queries, and also reduces the computation required for the employed k-nearest-neighbour strategy.

2 Background

This section presents some background relevant to the work to be proposed. An overview of the mqrtree is provided here, along with a summary of its k-nearest neighbour search strategy. More details on the mqrtree (including validity, insertion, construction, and basic region searching algorithms) can be found in [18], while details on the k-nearest neighbour search can be found in [20].

The mqrtree [18] is a approximation-based spatial access method that uses two-dimensional nodes to organize objects in two-dimensional space. This allows the existing spatial relationships to be maintained between objects and the regions of space that contain them. After an object or point is inserted, a validity test is performed to ensure that all spatial relationships are maintained, and any objects or regions that violate the spatial relationship rules are relocated. In addition to traditional region searching and point searches, the features of the mqrtree also allow it to support k-NN searching. A very nice feature of the mqrtree that will lend itself nicely to k-NN searching is that zero overlap of regions (on the same level of the tree) occurs when the mqrtree is used to solely index point data [18].

Figure 1 presents an mqrtree for the given dataset. A node has the quadrants NW, NE, SW and SE. Each node has a corresponding nodeMBR, which encompasses all objects, points and regions in the subtrees accessible from the node. All objects and regions containing other objects, are placed in the appropriate quadrant based on their relationship to the centroid of the nodeMBR. For example, in the leaf node containing m1, m2 and p9, we observe that m1 is NW of the centroid for the nodeMBR that contains it Similarly, m2 is SW and p9 is NE of the centroid, respectively. Therefore, these objects are placed in the NW, SW, and NE quadrants of the node, respectively. All nodes, including the root node, are organized in this manner.

The mqrtree k-nearest neighbour search strategy [20] requires two steps: (1) location of a candidate nodeMBR that leads to at least k points, which requires

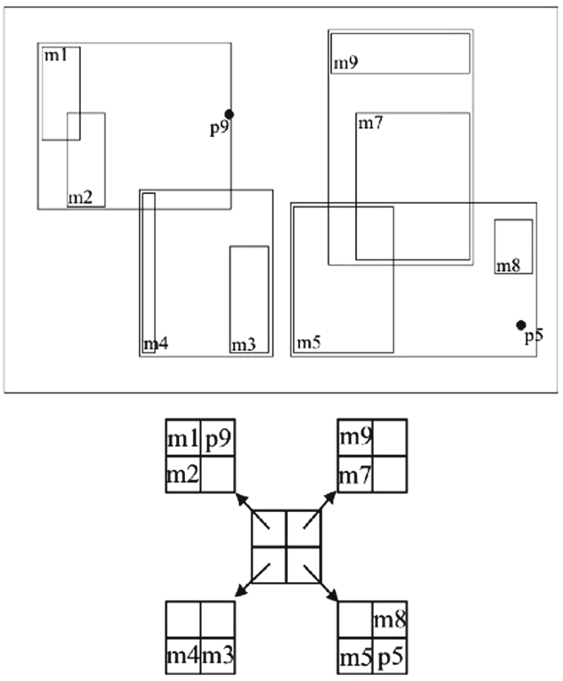


Fig. 1. mqrtree example [18]

a simple path search, and (2) a traversal of all nodes from that location in order to obtain the corresponding set of points that reside in the nodeMBR. If the obtained nodeMBR and point set contain a valid k-nearest neighbour result for the query, then the search is stopped. Otherwise, this process is repeated at the parent entry of the nodeMBR, until a suitable candidate is found. In the worst case, the nodeMBR at the Root node may be chosen. However, studies show that this case happens infrequently [20].

3 Continuous k-Nearest Neighbour Strategies

In this section, I propose two strategies for continuous k-nearest neighbour query processing. First, the concept of a safe region will be presented, along with how it can be obtained easily. Then, I propose two strategies - CKNN1 and CKNN2 - that deal with decisions on what to do when safe regions are no longer valid. I will first present the strategies respect to the mqrtree k-nearest neighbour strategy proposed in [20]. After, the application to any existing spatial access method will be discussed.

3.1 Safe Regions

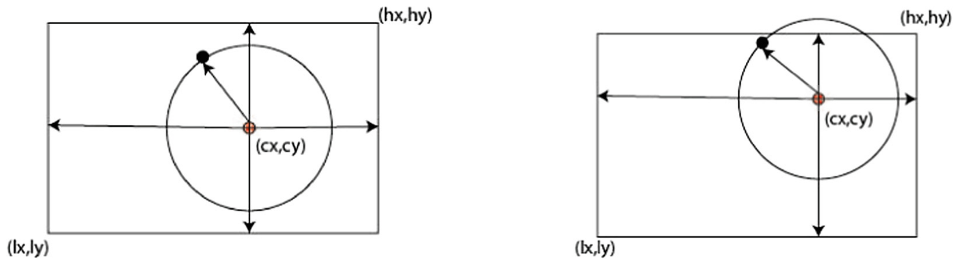
Intuitively, a region of space is defined as a *safe region* if the query remains in the region, as this guarantees that the query answer is valid. With respect to a k-nearest neighbour query, if a query remains in the safe region, then the current result (i.e. k nearest neighbours) remains valid. This work also utilizes the concept of a *safe region*. However, whether or not a region of space is considered safe will be determined differently than in other safe-region approaches. The trade off is that existing minimum bounding rectangles within a spatial access method can serve as safe regions.

Given a region (lx, ly, hx, hy), a query point q and the kth nearest neighbour kq, a region of space is identified as a safe region if it meets the following criteria:

1. it contains at least k points, and
2. the distance between q and kq is less than or equal to the distance between q and each side of (lx, ly, hx, hy):

$$dist(q, kq) < min(dist(q, lx), dist(q, hx), dist(q, ly), dist(q, hy))$$

Figure 2 depicts a valid and invalid safe region respectively for a 1-nearest neighbour query. In Fig. 2a, the distance from the query point to the k point (where $k = 1$) is less than all distances to the four sides of the region. Therefore, this is a valid safe region for this query point. In Fig. 2b, the distance between the query point and the kth point is greater than the distance to the north side of the region. This means that there may be a closer nearest neighbour that resides outside of the north side. Therefore, this region is not a safe region since the query result is not guaranteed.



(a) Valid (b) Invalid

Fig. 2. Valid and invalid cases

To obtain a safe region, one approach is to construct a new one whenever that is required. This can become costly if being created from scratch. Another approach - and the one adopted for this work - is to utilize a data structure that already contains pre-defined regions, in order to locate a suitable safe region. A spatial access methods (i.e. spatial index) [6] is a hierarchical structure that indexes regions of objects, and regions of regions, using minimum bounding rectangles. Therefore, a spatial access method is very suitable for obtaining candidate safe regions that will support multiple queries along a user trajectory without having to be updated.

3.2 CKNN1 and CKNN2

I now present the two approaches for continuous k-nearest neighbour query processing that utilize minimum bound rectangles (MBRs) from a hierarchical approximation spatial access method. For presentation purposes here, I assume the use of the mqrtree [18] and its corresponding k-nearest neighbour strategy [20].

The first strategy, called CKNN1, works as follows. First, for the first query point p, an initial safe region and corresponding set of m points (where $m >= k$) that reside within the safe region are identified using the k-nearest neighbour strategy. For each subsequent query point along the user trajectory, it is handled based on one of the following situations:

1. The safe region is still valid and is not the minimum bounding rectangle from the root node of the mqrtree. There is nothing further that needs to be done here until the next query arrives.
2. The safe region is still valid, but was derived from the minimum bounding rectangle that corresponds to the root node. The search starts over again from the root node for a new safe region and corresponding point set that contains the k nearest neighbours. The reasons for handling this case in this manner are the following. First, a safe region that corresponds to the root node will have a corresponding set of $m >= k$ points that includes all points from within the mqrtree. This can be up to thousands or millions of points, which most mobile devices will not (likely) have the storage capacity to handle. Second, and related, a safe region corresponding to the root node will always remain valid and never result in a new safe region being sought, unless the situation is forced. This will result in some additional processing cost, but the tradeoff is that a more management safe region and set of points can be send to the client.
3. The safe region is no longer valid. The search begins for a new safe region (and corresponding point set) from the parent of the node from which the safe region was derived. This approach is proposed in order to attempt to optimize the k-nearest neighbour search by not having it begin at the root of the mqrtree.

This strategy continues while query points are sent from the client and remain within the general region of space covered by the mqrtree. Figure 3 depicts the pseudocode for the CKNN1 strategy. This pseudocode utilizes the following functions: (1) knn_query(starting node, query point, safe region, result set), to find and return a new safe region and points (i.e. result set) that reside in it, and (2) node(region) to return the node that represents the region passed in.

The second proposed strategy, called CKNN2, is a simplified version of CKNN1. It works as follows. An initial search for a safe region and corresponding set of points that reside in the safe region is still carried out. Then, for each subsequent query point along the user trajectory, it is handled in one of two ways:

```
knn = k (from client)
search_point = NULL (from client)
result_set = NULL (send to client when updated)
safe_region = NULL (sent to client when updated)

/*obtain first point on user trajectory*/
search_point = obtain_next_point();

While exists(search_point)
   /*is k-nn result still valid in the current node region?*/
   If exists(safe_region) and and node(safe_region) != Root
        and safe_region_valid(safe_region,search_point,knn)
        /*nothing to do - fetch the next point

   Else if not_exists(safe_region) or node(safe_region) = Root
        /*perform k-nn query from root node*/
        knn_search(Root, search_point, knn, safe_region,
                    result_set)

    Else     /*safe region not valid*/
        /*perform k-nn query from parent of node corresponding
           to safe_region*/
        knn_search(node(safe_region)->parent, search_point,
                    knn, safe_region, result_set)

    End If

    /*obtain next point on user trajectory*/
    search_point = obtain_next_point()
End While
```

Fig. 3. First continuous k-NN search strategy - CKNN1

- The safe region is still valid and is not the minimum bounding rectangle from the root node of the mqrtree. There is nothing further that needs to be done here until the next query arrives.
- For all other cases, the search for a new safe region and corresponding point set begin at the Root node.

The reason for proposing a simplified version of CKNN1 is to determine if starting new searches from within the mqrtree structure, as opposed to returning to the root, results in significant savings in computation.

3.3 Application to Other Spatial Access Methods

Although I assumed above that the mqrtree *k*-nearest neighbour strategy [20] is the ad-hoc strategy of choice, both strategies will also work with other spatial access methods that are hierarchical-based and utilize minimum bounding rectangles for approximations of regions of points and regions that contain other regions. The only requirement are: (1) a *k*-nearest neighbour strategy exist for a chosen spatial access method that can identify and return both a safe region and a corresponding superset of points that reside in the safe region, and (2) that pointers from all nodes (except the root) exist that point back to the parent nodes. If these requirement are met, then the above strategies CKNN1 and CKNN2 can be applied.

4 Evaluation

This section presents the empirical evaluation of the mqrtree-based k-NN strategy, including a comparison against repeated k-nearest neighbour searching. I first present the framework and evaluation methodology. Then, I will present the outcome of the evaluation and the resulting discussion of the outcome.

4.1 Methodology

The mqrtree-based k-NN strategy is implemented in C on a PC running the Centos 7 version of Linux. It was evaluated using several synthetic point sets for both the data and the queries. These were chosen so that certain characteristics, such as the number of points and distribution, could be controlled. Altogether, 18 points sets were used for the experiments. The first are 6 point sets of uniform distribution, each of which 500, 1000, 5000, 10000, 50000, and 100000 points respectively. Each set of n points is drawn from a two-dimensional region of space of dimensions $(\sqrt(n) * 10)x(\sqrt(n) * 10)$. For example, the points in the 1000-point set are drawn from a 310×310 region.

Next are 6 point sets of exponential (i.e., skewed) distribution. Each also contains the same number of points as the uniform datasets, and are drawn from the same sized regions of space as the uniformly distributed sets. Finally, there are 6 sets of query points. Each contain the square root of the number of points of the data set they are applied to. For example, the query point set corresponding to the 1000-point data sets contains 31 query points, while the one corresponding to the 100000-point data set contains 316 query points. The reason for this is because the query point set in each file represents a trajectory that proceeds diagonally through the region of space that contains the points.

The two continuous strategies proposed above are evaluated by using the mqr-based k-nearest neighbour strategy proposed in [20]. This strategy is also employed for comparison against the two strategies by performing repeated searching. This strategy was chosen for comparison due to promising performance improvements shown in its preliminary evaluation. The expectation is that using the continuous strategies with it will result in further performance improvements.

The following tests are performed by applying the point query sets to their respective point data sets, mentioned above:

- 1-NN, first continuous strategy. 12 tests, with 6 using the uniform point sets and 6 using the exponential point sets.

- 1-NN, second continuous strategy. 12 tests, with 6 using the uniform point sets and 6 using the exponential point sets.
- 1-NN, repeated individual search. 12 tests, with 6 using the uniform point sets and 6 using the exponential point sets.
- k-NN ($k = 1$ to 10), first continuous strategy. 20 tests, with 10 using the uniform 10000-point set and 10 using the exponential 10000 point set.
- k-NN ($k = 1$ to 10), second continuous. 20 tests, with 10 using the uniform 10000-point set and 10 using the exponential 10000 point set.
- k-NN ($k = 1$ to 10), repeated individual search. 20 tests, with 10 using the uniform 10000-point set and 10 using the exponential 10000 point set.

For all tests, the following performance factors are recorded:

- The CPU time is recorded for each query. Two average times are calculated and recorded: the average time over all queries, and the average time for the queries that have less than $2\,\mu$s CPU time. The reason for this will be explained later.
- The number of queries that have less than $2\,\mu$s CPU time.
- The number of pages hits. The number of page hits (i.e., nodes that are checked) for each query is recorded.
- The number of queries with invalid tests over the trajectory. Tt is possible that the initial set of candidate points that is fetched may not contain a valid set of k nearest neighbours, due to other members of the true result being outside of the corresponding superMBR. Therefore, the number of queries that produce at least one invalid result during the search is recorded.
- The number of queries that traverse from the root. In [20] it is possible that the chosen safe region corresponds to the root node, which means that the entire spatial access method must be traversed.

In addition, for the tests that utilize the continuous query processing approaches, the number of times a safe region remained valid when the next query point was evaluated is also recorded. This is not specifically recorded for repeated k-nearest neighbour searching, since all queries are executed from the beginning (i.e. Root).

4.2 Results

I first present the comparisons of most results across all of the performance factors above, followed by a comparison of how the retention of safe regions over many query points affect the number updates to the safe region that must be performed. In all charts presented here: strat is the strategy applied (cknn1, cknn2, knn is for repeated knn), k is the number of nearest neighbours #points is the number of points in the data set, #queries is the number of queries, max#phits is the number of page hits (i.e. node accesses) required by the worst performing query point, avg#phits is the average number of page hits over all queries, #valid is the number of times a safe region remained valid when one or more subsequent query points arrived, #invalid is the number of queries that had last least one invalid safe region generated during the search for a valid safe region (see [20] for more info), #root is the number of queries that ultimately chose a safe region that corresponded to the root of the mqrtree, avgtime is the average CPU time over all queries, #u2ms is the number of queries that required less than 2 μ of CPU time, and tu2ms is the average running time for the queries that ran in less than 2 μ.

Figures 1 and 2 present the results of the 1-nearest neighbour tests across all uniform and exponential point sets, respectively. Due to space limitations, we only present the results for 5000 to 100000 points. For the Uniform data sets, we observe a significantly high percentage of times that safe regions remained valid when subsequent queries were processed - from approximately 68% for queries in the 500 point set, up to approximately 95% for queries in the 100000 point set. In the Exponential data sets, we see similar trends, with the only exception being for the exponential 500-point set. This number also contributes to improvements in the number of queries that identified at least one invalid safe region during the search for one is significantly lower in both CKNN1 and CKNN2, over repeated knn searching.

We also see that for the Uniform data set, there exists no noticeable difference between the average number of page accesses between CKNN1 and CKNN2, although both have modest improvements over repeated knn searching. However, a surprise is with the difference in the average number of pages accesses for the Exponential data sets. The average for CKNN2 is actually lower than that for CKNN1.

Finally, we see the high maximum number of page accesses for both the Uniform and Exponential data sets. Although we observe a high number of times that a safe region remains valid, unfortunately in some cases performing a k-nearest neighbour query still incurs a high number of page hits. This does have more to do with the k-nearest neighbour strategy chosen [20] than the proposed approaches here.

Next, Figs. 3 and 4 present the results of the mqrtree-based k-NN tests on both the 10000-point Uniform and Exponential point sets, respectively. Due to space limitations, only the even nearest neighbour results (i.e. 2,4,6,8 and 10) are shown. However, this subset still represents the outcome across all 10 cases. Here, we can observe another surprising finding - that the number of nearest neighbours does not affect the number of times a safe region remains valid when subsequent queries are processed! In addition, although the number of queries that find at least one invalid safe region does increase with k, the increase is modest when compared to repeated k-nearest neighbour searching. Finally, the trends with the average number of page hits found in the 1-nearest neighbour tests (Tables 1 and 2) also exist here.

Finally, Figs. 4 and 5 depict the percentage of times that a new safe region is needed due to it being no longer valid. As the number of points increases, this percentage decreases - and is especially noticeable when compared to the 100% that is required for repeated 1-nearest neighbour searching. Also, the number of nearest neighbours does not affect the percentage of updates required to the

Table 1. 1-nn results - uniform distribution

#points	#queries	strat	max#phits	avg#phits	#valid	#invalid	#root	avgtime	#u2ms	tu2ms
5000	70	cknn1	3854	91.10	58	6	1	1.26	67	0.51
		cknn2	3854	91.41	58	6	1	1.25	67	0.51
		knn	3858	193.59	0	34	2	1.94	65	0.52
10000	100	cknn1	5759	157.23	87	4	2	3.26	96	0.52
		cknn2	5759	157.56	87	4	2	3.28	96	0.52
		knn	5759	199.63	0	38	2	3.30	96	0.56
50000	223	cknn1	28815	215.52	211	2	1	21.77	218	0.51
		cknn2	28815	215.64	211	2	1	21.71	218	0.51
		knn	28815	277.70	0	81	1	21.88	216	0.57
100000	316	cknn1	76980	403.92	301	9	1	63.10	309	0.50
		cknn2	76980	404.05	301	9	1	62.60	309	0.50
		knn	76980	531.31	0	136	1	63.58	299	0.53

Table 2. 1-nn results - exponential distribution

#points	#queries	strat	max#phits	avg#phits	#valid	#invalid	#root	avgtime	#u2ms	tu2ms
5000	70	cknn1	6626	630.50	51	14	8	5.08	59	0.50
		cknn2	6627	553.14	48	15	8	4.63	60	0.50
		knn	6627	993.74	0	39	13	7.79	53	0.51
10000	100	cknn1	13420	945.49	78	14	7	13.29	89	0.50
		cknn2	13420	788.22	76	13	7	11.70	90	0.50
		knn	13420	1061.04	0	50	8	14.90	87	0.51
50000	223	cknn1	69652	2597.97	198	17	6	165.65	212	0.50
		cknn2	69652	2006.54	194	15	6	140.46	214	0.50
		knn	69652	3114.56	0	115	7	209.86	210	0.54
100000	316	cknn1	137292	3173.13	296	14	5	410.00	305	0.50
		cknn2	137292	2526.88	292	15	5	349.52	306	0.50
		knn	137292	5761.48	0	171	7	788.15	286	0.56

safe region, and overall is significantly lower than required for repeated searching (Tables 3 and 4).

4.3 Discussion

Overall, we observe some significant improvements in performance when a continuous k-nearest neighbour query processing strategy is used in conjunction with a hierarchical approximation-based spatial access methods, and related k-nearest neighbour algorithm. Notably, that identifying appropriate safe regions from a spatial access method leads to significant savings in the number of times a new safe region needs to be found, as well as significant savings in the number of invalid attempts when trying to locate a safe region.

However, some surprising findings include a lower number of page hits in Exponential data when using what would appear to be a less efficient algorithm. CKNN2 restarts all searches for a new safe region from the root of the spatial

Table 3. knn results - uniform distribution

k	strat	avg#phits	max#phits	#valid	#invalid	#root	avgtime	#u2ms	tu2ms
2	cknn1	157.28	5759	87	9	2	3.27	96	0.52
	cknn2	157.61	5759	87	10	2	3.26	96	0.52
	knn	240.24	5759	0	88	2	3.45	95	0.59
4	cknn1	388.69	7713	84	26	5	8.56	93	0.52
	cknn2	388.92	7713	84	26	5	8.51	93	0.52
	knn	501.43	7713	0	120	5	8.80	91	0.60
6	cknn1	415.80	7713	88	36	5	8.66	93	0.52
	cknn2	396.86	7713	88	32	5	8.56	93	0.52
	knn	564.56	7713	0	144	5	9.12	89	0.65
8	cknn1	473.26	7713	88	34	6	10.32	92	0.52
	cknn2	473.33	7713	88	34	6	10.25	92	0.52
	knn	564.56	7713	0	144	5	9.12	89	0.65
10	cknn1	552.07	7713	84	40	7	12.87	91	0.52
	cknn2	552.22	7713	84	40	7	12.88	91	0.52
	knn	756.05	7713	0	142	7	13.47	86	0.67

Table 4. knn results - exponential distribution

k	strat	avg#phits	max#phits	#valid	#invalid	#root	avgtime	#u2ms	tu2ms
2	cknn1	1014.65	13420	78	15	8	14.35	88	0.50
	cknn2	857.36	13420	76	14	8	12.75	89	0.50
	knn	1061.35	13420	0	53	8	14.82	87	0.51
4	cknn1	1273.40	13420	81	16	10	17.72	85	0.50
	cknn2	1173.58	13420	81	15	10	16.77	86	0.50
	knn	1598.87	13420	0	73	11	21.88	82	0.52
6	cknn1	1339.14	13420	80	17	11	18.77	84	0.50
	cknn2	1239.32	13420	80	16	11	17.90	85	0.50
	knn	2341.27	13420	0	73	19	32.89	73	0.52
8	cknn1	1738.94	13416	77	21	15	29.41	80	0.50
	cknn2	1639.10	13416	77	20	15	23.45	81	0.50
	knn	2679.39	13416	0	74	21	37.69	70	0.52
10	cknn1	1946.78	13416	74	24	18	27.55	77	0.50
	cknn2	1846.94	13416	74	23	18	26.77	78	0.50
	knn	3076.22	13416	0	76	24	44.15	64	0.52

access method, where CKNN1 starts this same search from the parent of the node where the safe region came from. The problem lies in the fact that in the Exponential data set, points are clustered around the positive x- and y-axes, with very few elsewhere. When a search starts from the parent, a traversal to

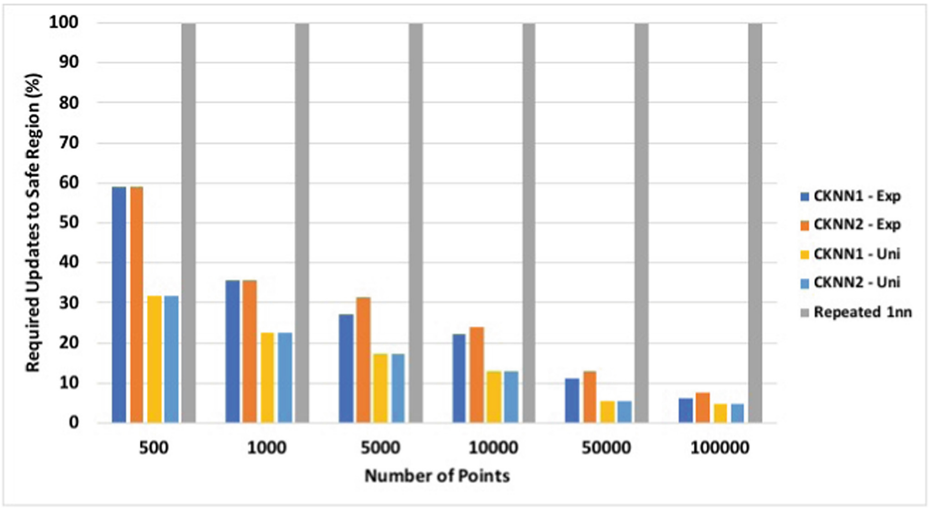


Fig. 4. Required update to safe region - 1NN

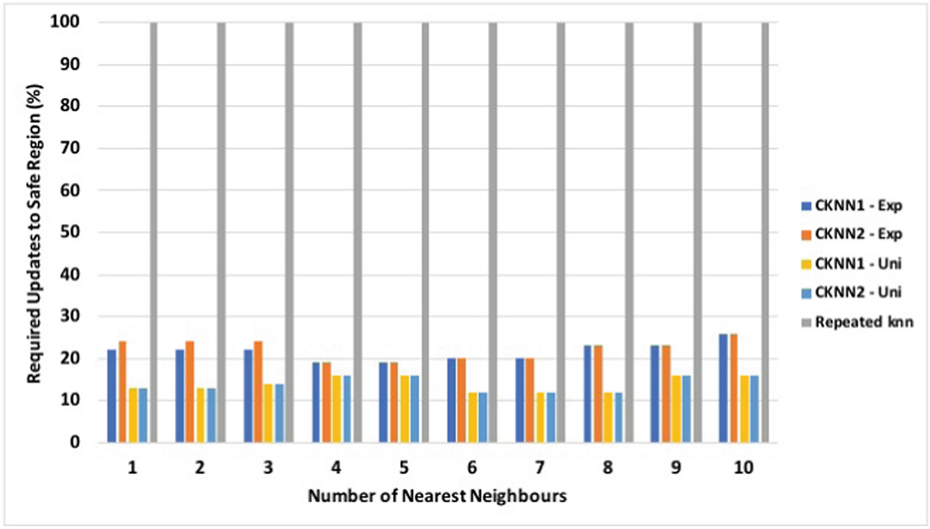


Fig. 5. Required update to safe region - kNN

obtain a point set may result in an invalid point set right from the start, and this search would proceed further up the tree in the same matter. Starting from the root will allow for drill-down to a lower node before the traversing needs to start. Therefore, this result is expected.

In addition, for the average CPU time it was noticeably lower for CKNN1 and CKNN2 than for repeated k-nearest neighbour searching. This is due to the elimination of multiple repetitions of really costly searches. If one search was costly but produced a safe region that was valid for several queries that followed, than these extra costly searches were successfully eliminated by the proposed strategies.

5 Conclusion

This paper proposes two strategies for continuous k-nearest neighbour processing that utilize an approximation-based spatial access method. The strategies obtain safe regions - regions where a query remains valid when it moves around - from the spatial access method. In addition, the strategies determine where to begin the search *within the structure* for an updated safe region, when a new one is needed. An experimental evaluation and comparison shows some significant results. In particular, the number of new safe regions that are required is not more than 20%, which result in other significant reductions in costs. In addition, the number of nearest neighbours does not affect the performance of the strategies.

Some directions of future work include the following: (1) some optimizations to both CKNN strategies and to the k-NN strategies in [20] have been identified and can further improve performance, (2) using other spatial access methods, such as the R-tree [9], and determining their performance in the proposed frameworks, and (3) a comparison versus other strategies that must create safe regions from scratch every time one is required.

References

1. Arya, S., Mount, D., Netanyahu, N., Silverman, R., Wu, A.: An optimal algorithm for approximate nearest neighbor searching fixed dimensions. J. ACM **45**(6), 891–923 (1998)
2. Brinkhoff, T., Kriegel, H.P., Seeger, B.: Efficient processing of spatial joins using R-trees. In: Proceedings of the 1993 ACM SIGMOD International Conference on Management of Data, SIGMOD 1993, pp. 237–246. ACM, New York (1993)
3. Cheng, R., Lam, K.Y., Prabhakar, S., Liang, B.: An efficient location update mechanism for continuous queries over moving objects. Inf. Syst. **32**(4), 593–620 (2007)
4. Friedman, J.H., Baskett, F., Shustek, L.J.: An algorithm for finding nearest neighbors. IEEE Trans. Comput. **24**(10), 1000–1006 (1975)
5. Fukunage, K., Narendra, P.M.: A branch and bound algorithm for computing k-nearest neighbors. IEEE Trans. Comput. **24**(7), 750–753 (1975)
6. Gaede, V., Günther, O.: Multidimensional access methods. ACM Comput. Surv. **30**, 170–231 (1998)
7. Gao, Y., Zheng, B.: Continuous obstructed nearest neighbor queries in spatial databases. In: Proceedings of the 2009 ACM SIGMOD International Conference on Management of Data, pp. 577–590. ACM (2009)
8. Gupta, M., Tu, M., Khan, L., Bastani, F., Yen, I.L.: A study of the model and algorithms for handling location-dependent continuous queries. Know. Inf. Syst. **8**(4), 414–437 (2005)
9. Guttman, A.: R-trees: a dynamic index structure for spatial searching. In: Proceedings of ACM SIGMOD International Conference on Management of Data, pp. 47–57 (1984)