Dense Nearest Neighborhood Query

Hina Suzuki[1], Hanxiong Chen[2], Kazutaka Furuse[3], and Toshiyuki Amagasa[2]

[1] Degree Programs in Systems & Information Engineering, University of Tsukuba
suzuki.hina.ss@alumni.tsukuba.ac.jp
[2] Dept. Computer Science, University of Tsukuba
{chx,amagasa}@cs.tsukuba.ac.jp
[3] Faculty of Business Administration, Hakuoh University, Japan
furuse@fc.hakuoh.ac.jp

Abstract. A nearest neighbor (NN) query is a principal factor in applications that handle multidimensional vector data, such as location-based services, data mining, and pattern recognition. Meanwhile, a nearest neighborhood (NNH) query is a query to find dense neighborhoods. However, it cannot find desired groups owing to strong restrictions such as fixed group size in previous studies. Thus, in this paper, we propose a dense nearest neighborhood (DNNH) query, which is a query without strong constraints, and three efficient algorithms to solve the DNNH query. The proposed methods are divided into clustering-based and expanding-based methods. The expanding-based method can efficiently find a solution by reducing unnecessary processing using a filtering threshold and expansion breaking criterion. Experiments on various datasets confirm the effectiveness and efficiency of the proposed methods.

Keywords: Nearest Neighborhood query · Spatial database · Information retrieval · Grid index.

1 Introduction

In multi-dimensional vector data such as spatial databases, a nearest neighbor (NN) query is a fundamental and important query in many fields, such as data mining and information retrieval, and it is widely used in various applications, such as services using pattern recognition, facility information, and map and navigation services using location information. Fig.

Many neighbor searches have been developed from the NN search. Sometimes users want to search a "dense group" of neighboring points quickly. Examples are as follows:

- A tourist who wants to visit several stores without moving too much
- A user who wants to find a social networking community whose hobbies and interests match his own
- A user who wants to identify an accident-prone area near a school

Fig. 1 shows an example for some neighbor points from a given query q. In (b), four points nearest to the query are searched, which are obtained by repeating the

single neighbor point search four times. As indicated in this example, searched points may be scattered in the data space. In (c), a dense group including four points is searched. This paper addresses the latter type of searches.

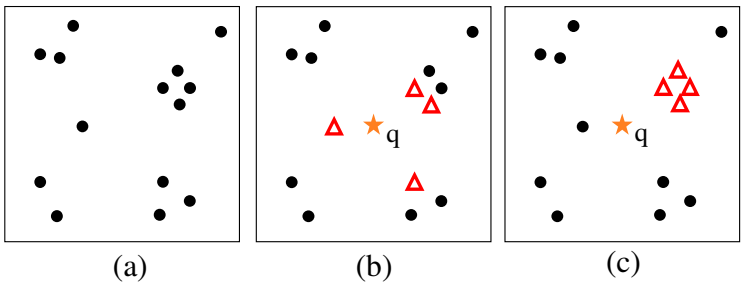

Fig. 1. (a) Points, (b) NN (k=4), (c) Dense neighborhood group

Among the many studies on the efficiency and extension of the NN query, the most relevant queries for this type are the nearest neighborhood (NNH) query [1] and balanced nearest neighborhood (BNNH) query [5], which is explained in detail in Section 2. The BNNH query is an extension of the NNH query that solves the query of the empty output owing to the strong constraints. However, the remaining constraints of the BNNH query interfere with users obtaining the desired output.

In this paper, we propose a novel flexible query, **dense nearest neighborhood (DNNH)**, which has no redundant constraints, and three algorithms for solving the query. One is an intuitive method that uses clustering techniques, and the other two algorithms provide faster search by exploiting filtering thresholds and expansion breaking criteria for group retrieval. To verify the usefulness and efficiency of the proposed methods, we conducted experiments on a large dataset and compared the performance of the proposed methods.

2 Related Work

The neighborhood search query starts with the most basic (k-)NN query, which searches for the (k) points closest to the query point, and has been studied in many ways to improve its efficiency and extension [2][4][6][7][9].

The queries that are most relevant to the search for dense neighborhood groups are the NNH query [1] and BNNH query [5]. When each of these queries is applied to the dataset in Fig. 1(a), the candidate groups are as shown in (a), (b) and (c) of Fig. 2.

NNH query. The NNH query [1] outputs the smallest distance between the center of a circle and a query point among the circles that contain more than a specified number (k) of points within the circle of a specified radius (ρ).

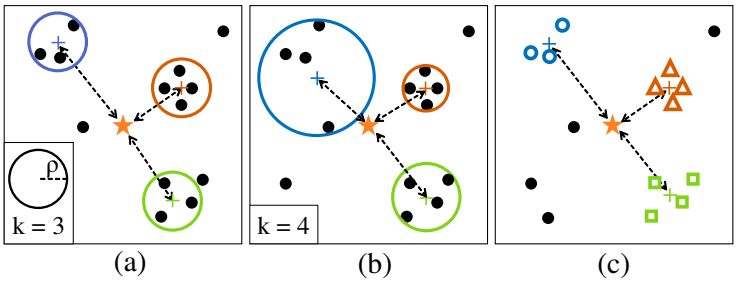

Fig. 2. The candidate grouFps. (a) NNH, with k specified to 3 and a fixed radius ρ (b) BNNH, where k is specified as 3 and ρ is changeable. (c) DNNH

As shown in Fig. 2(a), it is possible to obtain the desired group (triangle mark points in Fig. 1(c)) by specifying the appropriate parameters. However, in reality, it is not always the case that more than k points are found within the circle of fixed ρ, which implies that the query obtains an empty result if the appropriate parameters cannot be specified. Estimating the appropriate parameters in advance is particularly difficult for large datasets.

BNNH query. The BNNH query uses a circle of variable radius to guarantee that it contains the specified number (k) of points. This is implemented by finding the circles minimizing the evaluation $\Delta(C, q)$ expressed by the following equation:

$$\Delta(C, q) = \alpha \parallel q - c(C) \parallel + (1 - \alpha)\rho$$

Here, $c(C)$ is the center of C, and ρ is the radius of C. α is a value for $0 < \alpha < 1$, where the closer the value to 0, the greater the cohesion, and the closer it is to 1, the greater the distance to q. Unlike NNH queries, BNNH queries allow variable circle sizes instead of specifying the number k, and so it can always return a non-empty result.

However, there is still the problem that the group circle will not be dense unless an appropriate k is specified. For example, the candidate groups in the sample dataset (Fig. 2(b)) by BNNH with varying parameters are shown in Fig. 3. The best dense nearby group is group C with parameter $k = 4$, as shown in (a). However, in (b), for parameter $k = 5$, it returns group C', which is neither dense nor close to the query q.

Therefore, the BNNH query strongly depends on the parameter k, and there is a high possibility that the desired dense nearby group cannot be obtained if the appropriate value is not given. Nevertheless, it is difficult to estimate the value depending on the dataset and query location in advance. To address the problem that neither NNH nor BNNH guarantees that the answer groups found are dense, we propose DNNH, which finds a dense group without specifying the parameters k and ρ. DNNH queries are more flexible than BNNH queries because they compare dense groups that were retrieved regardless of the number of points in the group:

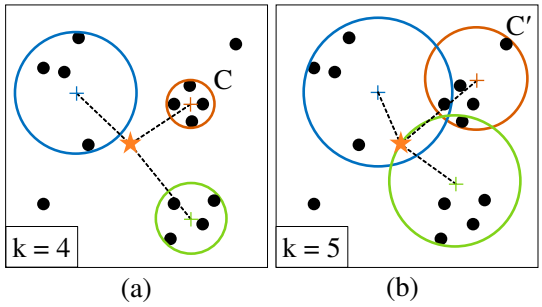

Fig. 3. The candidate groups by BNNH with varying parameters.
(a) $k = 4$, (b) $k = 5$

3 Dense Nearest Neighborhood Query

Definition 1. *(Dense nearest neighborhood query, DNNH). Given a set of points P and a query point q, the DNNH query returns a group C that minimizes the value of $\Delta(C, q)$.*

The total degree of approximation Δ (different from Δ in the BNNH query) is defined by the following equation:

$$\Delta(C, q) = \| q - c(C) \| + \mathrm{sd}(C) \tag{1}$$

Here, $c(C)$ is the center of gravity of C, and $\mathrm{sd}(C)$ is the standard deviation of C. Compared with previous studies that used the center and radius of a circle, this method can accurately represent the variation of points in a group. Moreover, DNNH overcomes the disadvantage of BNNH such that the searched groups are not necessarily dense, because it does not require the number of points in each group.

The most difficult part of solving a DNNH query is to efficiently retrieve dense groups. For example, if we consider how to find a dense group to which a point p may belong, in the case of the BNNH query, because the number of data in the group k is specified, we can find it only by performing a number of NN searches based on k for p. Meanwhile, in a DNNH query, the number of data in a group is not specified; thus, we must find a dense group from a large number of combinations including p, and it is NP-hard to find the optimal solution. We propose two approximate solutions by heuristics – one by X-means clustering and the other by expanding – for an efficient DNNH search.

3.1 Clustering-Based Approach

As the simplest approach to solving DNNH queries, we propose calculating Δ after clustering all the points. It is important to note here that the DNNH query is parameter free; therefore, clustering should not take redundant parameters as well.

X-means [8] is an extension of k-means and is characterized by the fact that it can estimate the number of clusters and perform clustering simultaneously without the need to specify the number of clusters in advance, by using the recursive 2-means partitioning and the stopping criterion based on the information criterion BIC. BIC is calculated by the likelihood function and the size of a dataset, intuitively telling whether it is likely to split a set into two. In this study, we used an algorithm improved by Ishioka [3]. This algorithm differs from the original algorithm by Pellog and Moore [8] in that it considers the possibility that the variance differs among clusters, and it uses approximate computation for some calculations to enhance the efficiency.

Algorithm 1 X-means Clustering-based Algorithm

Input: P, q, k_0
Output: C

$C, \mathbb{C} \leftarrow \phi$
$C_1, C_2, ..., C_{k_0} \leftarrow k$-means++ (p, k_0) // partition P into k_0 clusters
for each $C_i \in \{C_1, C_2, ..., C_{k_0}\}$ **do**
    splitClusterRecursively(C_i)
for each $C_i \in \mathbb{C}$ **do**
    if $\Delta(C_i, q) < \Delta(C, q)$ **then**
        $C \leftarrow C_i$
return C

function splitClusterRecursively(C)
    $C_1, C_2 \leftarrow k$-means++$(C, 2)$
    if $BIC(C) > BIC'(C_1, C_2)$ **then**
        for each $C_i \in \{C_1, C_2\}$ **do**
            splitClusterRecursively(C_i)
    else
        Insert C into $\mathbb{C}$

3.2 Expanding-Based Approach

As clustering a large dataset requires a many computation times, our other approach attempts to retrieve groups from nearby the query q. Intuitively, points located near the query are more likely to form the result group. In this approach, we retrieve groups from the query's neighborhood and filter points that cannot be answered using the current degree Δ_e as a threshold. Using this approach, we briefly repeat the following: (1) Extract the query neighbor from the dataset, (2) retrieve the dense group to which the extracted point belongs (Section 3.2.1, 3.2.3), (3) update the threshold for filtering (Section 3.2.2), and (4) filter and remove the points that cannot be answered from the dataset (Section 3.2.2).

3.2.1 Evaluation Metric for Retrieving a Cluster

In this section, we explain how the retrieval of a dense group C_p to which a point p belongs is performed. In this study, this is achieved by selecting a dense preferred group from among the groups obtained by expansion using an NN search. The problem is to select a group from the enlarged ones based on the criteria, which we address by designing and using the enlargement index Δ_e.

The Δ_e is calculated by

$$\Delta_e(C, p_{next}) = \Delta(C, q) \cdot \pi_e(C, p_{next})$$

where $\pi_e(C, P)$ denotes the expandability of group C in dataset P and is defined by the following equation:

$$\pi_e(C, p_{next}) = \frac{pd_{mean}(C)}{pd_{mean}(C) + nd_{mean}(C, p_{next})}$$

$$pd_{mean}(C) = \frac{1}{\binom{|C|}{2}} \sum_{p_i, p_j \in C} \| p_i - p_j \|$$

$$nd_{mean}(C, p_{next}) = \frac{1}{|C|} \sum_{p \in C} \| p_{next} - p \|$$

$$p_{next} = arg \min_{p \in P - C} \| c(C) - p \|$$

pd_{mean} and nd_{mean} represent the average distance between the samples in the group and the average distance between the candidate points (p_{next}) and the samples in the group, respectively. The candidate point p_{next} is the point that has the smallest distance to the center $c(C)$ among the points not in C.

3.2.2 Bounding the Expanding Group

In this section, we explain the conditions under which a point p is removed using Δ of an already retrieved group C. Let C_p denote the group to which p belongs. If $\min \Delta(C_p, q) > \Delta(\exists C, q)$ holds, the group to which p belongs will not be preferred to the existing groups, and no further processing of p is necessary. Therefore, if we can derive $\min \Delta(C_p, q)$ from the information of p, we can determine whether the group is removed by filtering using $\Delta(C, q)$. However, in a DNNH query where the number of data in a group is not specified, Δ may be as small as possible depending on the distribution of the data, and it is difficult to determine the exact filtering threshold. Evidently, it makes no sense to find dense groups in a uniform distribution; therefore, we assume that the data of C_p follow a normal distribution.

Based on the assumption that the data of C_p follow a normal distribution, $arg \min \Delta(C_p, q)$ as C_p^{min}, we can approximate its center of gravity and standard deviation. The α indicates the sigma rule coefficient for the 68-95-99.7 rule of the normal distribution; for example, when $\alpha = 2$, it indicates that approximately 95% of the points of C_p exist within the radius $2sd(C)$ from the center of gravity.

In this case, min is the center of gravity. In this case, $\min \Delta(C_p, q)$ can be calculated as follows:

$$\min \Delta(C_p, q) = \| q - c_p^{min} \| + sd(C_p^{min}) = \frac{\| q - p \|}{2} + \frac{\| q - p \|}{2\alpha} = \frac{\alpha + 1}{2\alpha} \| q - p \|$$

The α indicates the sigma rule coefficient for the 68-95-99.7 rule of the normal distribution; for example, when $\alpha = 2$, it indicates that approximately 95% of the points of C_p exist within the radius $2sd(C)$ from the center of gravity. Substituting this into $\min \Delta(C_p, q) \leq \Delta(C, q)$, we obtain $\| q - p \| > \frac{2\alpha}{\alpha+1} \Delta(C, q)$. This leads to the following conclusion.

Theorem 1. *A point p locating further than a bound, that is,*

$$\| q - p \| > \frac{2\alpha}{\alpha + 1} \min_C \Delta(C, q) \tag{2}$$

can be removed by the filtering process.

3.2.3 Expansion Breaking Criteria

The bound given above is inefficient because it works only in the second and subsequent retrieval of clusters. For the computation of pd_{mean}, nd_{mean}, the first group retrieval always continues to expand until the entire dataset is included, and $\mathcal{O}(|P|^2)$. This is a problem because the DNNH query does not specify the group size, which affects the efficiency.

By the definition of Δ_e, we know that p of small Δ_e suggests that the corresponding group is desired so that we can stop the enlargement process, thereby reducing the computational cost. Then, we determine that Δ_e is small enough. Under the assumption that C to which p belongs follows a normal distribution, and they exist within the radius $\alpha \cdot sd(C)$ from the center of gravity c, when Eq. 3 holds for the expansion point p_{next}, we can conclude that further expansion is meaningless.

$$\| p_{next} - q \| \geq \alpha \cdot sd(C) \tag{3}$$

In addition, in the latter half of the cluster retrieval, where the solution is less likely to be obtained, we aim to further speed up the process by terminating the expansion when the cluster is found to be less preferable than the current most preferable cluster C_{best} among the retrieved clusters. However, as mentioned in section 3.2.2, because the DNNH query does not specify the group size, Δ may be as small as possible depending on the distribution of the data. For C of a certain size, it is reasonable to assume that $sd(C)$ monotonically increases with each expansion. Let C' be the cluster of C expanded an arbitrary number of times; then, $sd(C') > sd(C)$ holds by assumption. Here, if

$$sd(C) > \Delta(C_{best}) \tag{4}$$

then by definition of Δ, $\Delta(C') > sd(C') > sd(C') > sd(C) > \Delta(C_{best})$
This means that Eq. 4 can be used as expansion breaking criteria to stop expanding C.

3.2.4 Basic Expanding Algorithm

The first method is shown in Algorithm 2. In this method, the points of dataset P are first sorted in order of their distance from the query point q (second line). The points are extracted from the sorted dataset in order from the top, and groups are retrieved as described in Section 3.2 (lines 4–10). After the retrieval is finished, the points in the group are removed from P_{sort} as processed (line 11), the threshold *bound* is updated and filtered (lines 12–14), and if there are still unprocessed points, the group is retrieved again (line 3). The process is terminated when there are no more unprocessed points.

Algorithm 2 Basic Expanding Algorithm

Input: P, q, α, k_{min}, k_{max}
Output: C_{best}
1: $bound \leftarrow \infty$
2: **while** P is not empty **do**
3: $p \leftarrow$ nearest point $\in P$ from q that is nearer than the bound in Eq. (2)
4: $C \leftarrow$ RetrieveCluster(p, k_{min}, k_{max})
5: $P \leftarrow P - C$
6: **if** $\Delta(C) < \Delta(C_{best})$ **then**
7: update the bound of Eq. (2)
8: $C_{best} \leftarrow C$
return C_{best}

9: **function** RetrieveCluster(p, k_{min}, k_{max})
10: $C \leftarrow \{p\}$, $C_{best} \leftarrow C$
11: **while** P is not empty $\wedge$ Eq. (4) is not satisfied **do**
12: $p_{next} \leftarrow$ nearest point $\in P$ from c
13: **if** Eq. (3) is not satisfied **then break**
14: **if** $|C_{best}| = 1 \vee \Delta_e(C, p_{next}) < \Delta_e(C_{best}, {p_{next}}^{best})$ **then**
15: $C_{best} \leftarrow C$
16: $C \leftarrow C \cup p_{next}$
return C_{best}

3.2.5 Grid Expanding Algorithm

The problem with the basic method is that the entire process, from indexing to filtering of the dataset, is point-based, which is inefficient. Therefore, we propose a grid-based method for preferential search from the neighboring points of query points and further reduction of the search space in the NN search. The images are presented in Fig. 4. The figure shows an example of a grid that divides

the space into 4×4 cells. The grid structure allows us to directly refer to the points in the cells, thereby enabling us to achieve a more efficient refinement of the search space in the NN search and coherent filtering process for each cell. The pseudocode is shown in Algorithm 3.

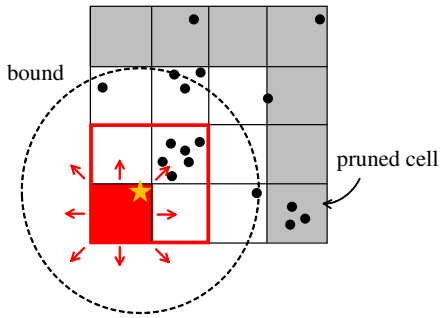

Fig. 4. Grid-based algorithm

4 Experiments

We conducted experiments to verify the efficiency of the proposed method. All algorithms for the solutions presented were implemented in C++. The experiments were conducted on a Windows operating system with the following specifications: Windows 10 Home, with a 2.9 GHz 8-Core Intel Core i7 processor and memory of 64 GB 1466 MHz DDR4. The real data NE (123,593 data), RR (257,942 data), and CAS (196,902 data) were provided by the U.S. Census Bureau's TIGER project*. In addition, to measure the correspondence to the datasets with various distributions, we prepared uniform random data UN (10000-− 200000 pts) and a cluster dataset RN. RN is a composite dataset of random numbers that follow the standard normal distribution and are scaled and arranged in space as clusters, and the effect of changes in cluster size was also measured. All these datasets were two-dimensional and normalized to $[0, 1]$. Experiments compare the performance of the three proposed methods (x-means clustering-based, basic expanding, and grid expanding) on real data, scalability, changes in cluster size, α, time, and distance of clusters.. In the grid expanding algorithm, we vary the grid size n and investigate the appropriate value of n. q was selected randomly from the dataset. Unless otherwise stated, $\alpha = 2$, $n = 100$, the distance between clusters $= 1.0$, and $k_{min} = 10$. All results are reported as the average processing time for conducting DNNH queries 10 times.

4.1 Experimental Results

Performance of the real datasets. The results are presented in Fig. 5. The results for the x-means clustering-based algorithm are "xmeans"; basic expanding

Algorithm 3 Grid Expanding Algorithm

Input: P, q, k_{min}, k_{max}, α
Output: C_{best}

```
1: bound ← ∞, P_cur ← φ
2: cells ← get surround cells of q
3: while cells locate in bound do
4:     P_cur ← points in cells
5:     while P_cur is not empty do
6:         p ← nearest point ∈ P_cur from q that is nearer than the bound in Eq. (2)
7:         C ← RetrieveCluster(p, k_min, k_max)
8:         P ← P − C
9:         if Δ(C) < Δ(C_best) then
10:            bound ← update Eq. (2)
11:            C_best ← C
12:    cells ← the next round of cells
   return C_best

13: function RetrieveCluster(p, k_min, k_max)
14:    C ← {p}, C_best ← C
15:    cells ← get surround cells of p
16:    while cells locate in bound do
17:        P_cur ← points in cells
18:        while P_cur is not empty ∧ Eq. (4) is not satisfied do
19:            p_next ← nearest point ∈ P_cur from c
20:            if Eq. (3) is not satisfied then break
21:            if |C_best| = 1 ∨ Δ_e(C, p_next) < Δ_e(C_best, p_next^best) then
22:                C_best ← C
23:            C ← C ∪ p_next
24:        cells ← the next round of cells
   return C_best
```

and grid expanding algorighm are "basic" and "grid," respectively; and "U100" and "U500" indicate that the cluster size limit k_{max} is set to 100 and 500, respectively. First, it can be observed that the x-means clustering-based algorithm is the slowest, and the basic expanding and grid expanding algorithms are the fastest for all datasets. This is especially true for RR, where even the basic expanding algorithm ($k_{max} = 500$), which is the slowest among the basic expanding and grid expanding algorithms, is 10 times faster than the x-means clustering-based algorithm. The fastest algorithm was the grid expanding algorithm, which was up to three times faster than the basic expanding algorithm.

Effect of dataset size. For both the UN and RN datasets, experiments were conducted with data sizes varying between 10000 and 200000. For the RN dataset, the cluster size was fixed at 50. The results are shown in Figs. 6 and 7. First, the experimental results for the UN dataset (Fig. 6) show that the x-means clustering-based algorithm is the fastest when the data size is 10000. However, after 50000, the basic expanding and grid expanding algorithms are reversed

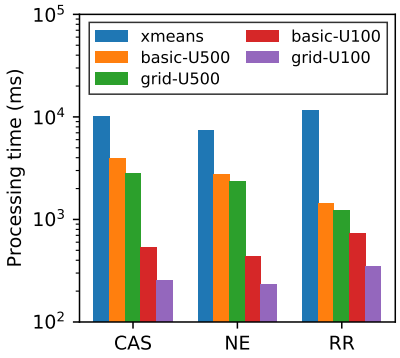

Fig. 5. Performance on real datasets

and become faster. All algorithms showed a linear or gradual increase in the execution time. The fastest algorithm is grid expanding, which shows a higher performance than basic expanding, as the data size increases. The experimental results for the RN dataset (Fig. 7) show that the basic expanding and grid expanding algorithms are about 10 times faster than x-means clustering-based when the cluster size = 50 and the increase in the execution time was also slow. Again, the fastest algorithm was the grid expanding algorithm, and the effect increased as the data size increased.

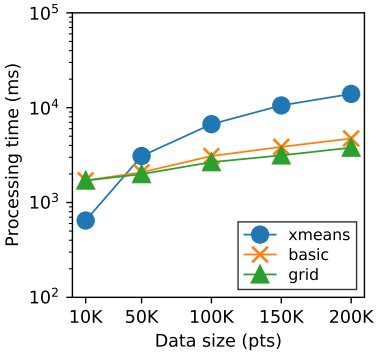

Fig. 6. Effect of data size (UN) ($k_{max} = 500$)

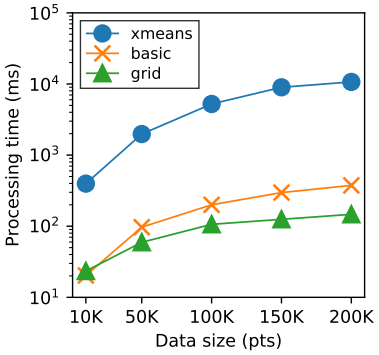

Fig. 7. Effect of data size (RN) ($k_{max} = 1000$; $|C| = 50$)

Effect of cluster size. For the RN dataset, we experimented by varying the cluster size between 10 and 1000. The data size was fixed at 100000. The results are shown in Fig. 8. From 10 to 500, the basic expanding and grid expanding algorithms were up to 100 times faster than the x-means clustering-based algo-

rithm. However, the execution time of the basic expanding and grid expanding algorithms increased significantly as the cluster size increased. However, the execution time of the basic expanding and grid expanding algorithms increased significantly with an increase in the cluster size and reversed when the cluster size was 1000. For the x-means clustering-based algorithm, the decrease in execution time as the cluster size increases can be attributed to the fact that the number of clusters in the entire dataset decreases owing to the fixed data size, which reduces the number of divisions by k-means and the amount of BIC computation.

Effect of the cluster distance. For the RN dataset, we conducted experiments by varying the distance of clusters between 0.0 3.0. The data size was fixed at 100000, and the cluster size was set to 50. In the RN dataset, the clusters are equally spaced. However, when the distance between clusters is x, there is an interval of x clusters between the clusters. Consequently, while the basic expanding and grid expanding algorithms are faster than the x-means clustering-based algorithm, the performance of basic expanding and grid expanding deteriorated rapidly when the distance of the clusters was 0.0. This is because expansion breaking criteria can no longer function due to the loss of distance between clusters, but it does function after 0.5, indicating that the proposed expansion breaking criteria is effective even for small intervals.

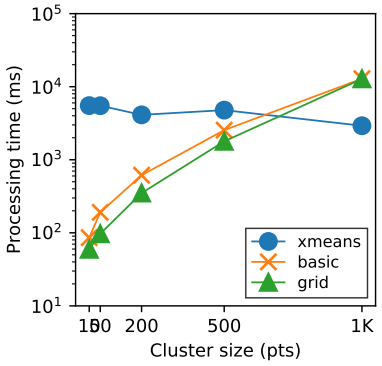

Fig. 8. Effect of cluster size ($|P| = 10000$; $k_{max} = 1000$)

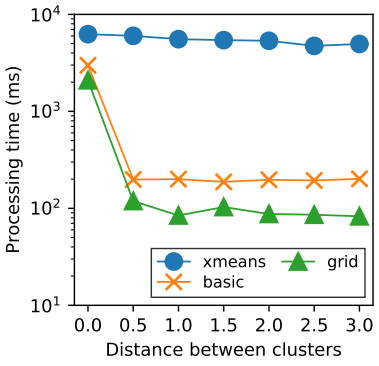

Fig. 9. Effect of distance of clusters ($|P| = 100000$; $|C| = 50$; $k_{max} = 500$)

Effect of sigma rule coefficient α. For both the UN and RN datasets, we experimented by varying the sigma rule coefficient α used in the basic expanding and grid expanding algorithms between 1 and 5. The data size was 100000, and the cluster size of the RN dataset was fixed at 50. Consequently, the slowest speed is obtained when $\alpha = 1$. For the RN dataset, it was the fastest when $\alpha = 2, 3$. However, when $\alpha = 5$, it was as bad as the result for the UN dataset when $\alpha \geq 2$. This is because α is quite large and expansion breaking criteria no longer works.

Effect of grid size n. For the NE, UN, and RN datasets, we experimented by varying the grid size n used in the grid expanding algorithm between 10 and 500. For example, because the datasets are two-dimensional, when $n = 100$, the maximum number of cells is n^2. The data size of UN and RN is 100000, and the cluster size of RN is fixed at 50 (RN-50P) and 500 (RN-500P). Hence, the fastest execution time was obtained when $n = 10, 50$, and the execution time increased slowly. This is because, when the grid size becomes quite large, the cells become smaller than necessary, and the amount of search and expansion processing of each cell increases.

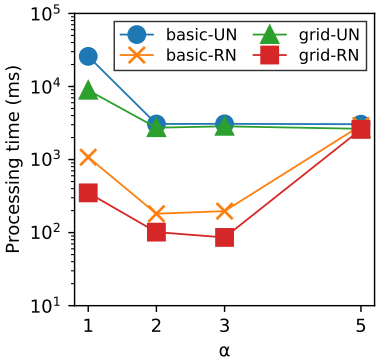

Fig. 10. Effect of α ($|P| = 100000$; $|C| = 50$ (RN))

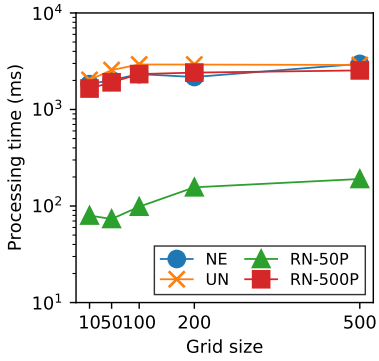

Fig. 11. Effect of grid size n ($|P| = 100000$ (UN, RN))

4.2 Evaluation of the Proposed Methods

Overall, the grid expanding algorithm is the fastest. In particular, in comparison between the basic expanding and grid expanding algorithms, the grid expanding algorithm is faster in all results, unless the data size is small or inefficient parameter settings (such as $\alpha = 5$) are used . Therefore, the grid expanding algorithm should be chosen unless there is concern about memory usage or the small overhead of building the cellular data. Depending on the distribution of the data, a grid size of approximately 10–100 is considered the most suitable for achieving a good balance between memory usage and processing efficiency.

However, depending on the distribution of the dataset and the purpose of the search, the x-means clustering-based algorithm may be a better choice. For example, the dataset may be sparsely distributed and of small size, or the cluster size may be larger than 500, or the search may be for a large cluster with $k_{max} \geq 1000$. However, in this study, we consider finding a set of nearby facilities in a location-based service using a spatial database, or finding a set of objects with attributes similar to those of a particular user in a social network service (SNS).

In these situations, the basic expanding or grid expanding algorithm is preferable because it can rapidly detect small- to medium-sized neighborhood clusters.

5 Conclusion and Future Work

In this paper, we proposed a DNNH query, which finds dense groups without severe constraints, and efficient methods for solving the query: x-means clustering-based algorithm, basic expanding algorithm, and grid expanding algorithm. The DNNH query can flexibly find more desirable groups for users, which cannot be achieved by strongly constraining existing problems. Among the proposed methods, the grid expanding algorithm is the fastest, and it can contribute to many applications that deal with large datasets. For future work, we are going to investigate the effect of expansion breaking criteria in distributions with overlapping clusters. We will also extend the grid based method to high dimension data.

References

1. Choi, D., Chung, C.: Nearest neighborhood search in spatial databases. In: 2015 IEEE 31st International Conference on Data Engineering. pp. 699–710 (April 2015). https://doi.org/10.1109/ICDE.2015.7113326
2. Deng, K., Sadiq, S., Zhou, X., Xu, H., Fung, G.P.C., Lu, Y.: On group nearest group query processing. IEEE Transactions on Knowledge and Data Engineering **24**(2), 295–308 (2012). https://doi.org/10.1109/TKDE.2010.230
3. Ishioka, T.: Extended k-means with an efficient estimation of the number of clusters. Data Mining, Financial Engineering, and Intelligent Agents (2000). https://doi.org/10.1007/3-540-44491-2_3
4. Jang, HJ., H.K.C.J.e.a.: Nearest base-neighbor search on spatial datasets. Knowl Inf Syst **62**, 867–897 (2020). https://doi.org/10.1007/s10115-019-01360-3
5. Le, S., Dong, Y., Chen, H., Furuse, K.: Balanced nearest neighborhood query in spatial database. In: 2019 IEEE International Conference on Big Data and Smart Computing (BigComp). pp. 1–4 (Feb 2019). https://doi.org/10.1109/BIGCOMP.2019.8679425
6. Malkov, Y.A., Yashunin, D.A.: Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. IEEE Transactions on Pattern Analysis and Machine Intelligence **42**(4), 824–836 (2020). https://doi.org/10.1109/TPAMI.2018.2889473
7. Papadias, D., Tao, Y., Mouratidis, K., Hui, C.K.: Aggregate nearest neighbor queries in spatial databases **30**(2) (2005)
8. Pelleg, D., Moore, A.: X-means: Extending k-means with efficient estimation of the number of clusters. Machine Learning, p (2002), https://www.researchgate.net/publication/2532744_X-means_Extending_K-means_with_Efficient_Estimation_of_the_Number_of_Clusters
9. Stanoi, I., Agrawal, D., Abbadi, A.E.: Reverse nearest neighbor queries for dynamic databases. In: ACM SIGMOD Workshop on Research Issues in Data Mining and Knowledge Discovery. pp. 44–53 (2000), https://www.semanticscholar.org/paper/Reverse-Nearest-Neighbor-Queries-for-Dynamic-Stanoi-Agrawal/cb60aef9f2187d4052b36f99aba6e1b8eca9f4ca