On Group Nearest Group Query Processing

Ke Deng, Shazia Sadiq, Xiaofang Zhou, *Senior Member*, *IEEE*,
Hu Xu, Gabriel Pui Cheong Fung, and Yansheng Lu

Abstract—Given a data point set D, a query point set Q, and an integer k, the Group Nearest Group (GNG) query finds a subset ω ($|\omega| \leq k$) of points from D such that the total distance from all points in Q to the nearest point in ω is not greater than any other subset ω' ($|\omega'| \leq k$) of points in D. GNG query is a partition-based clustering problem which can be found in many real applications and is NP-hard. In this paper, Exhaustive Hierarchical Combination (EHC) algorithm and Subset Hierarchial Refinement (SHR) algorithm are developed for GNG query processing. While EHC is capable to provide the optimal solution for $k = 2$, SHR is an efficient approximate approach that combines database techniques with local search heuristic. The processing focus of our approaches is on minimizing the access and evaluation of subsets of cardinality k in D since the number of such subsets is exponentially greater than $|D|$. To do that, the hierarchical blocks of data points at high level are used to find an intermediate solution and then refined by following the guided search direction at low level so as to prune irrelevant subsets. The comprehensive experiments on both real and synthetic data sets demonstrate the superiority of SHR in terms of efficiency and quality.

Index Terms—K-median clustering, group nearest group query, group nearest neighbor query.

1 Introduction

Many decision making applications need to consider multiple factors simultaneously. The spatial queries that consider multiple query points have received much attention recently [1], [2], [3], [4]. One such query is the group nearest neighbor (GNN) [4], [5] query which finds one data point p from a data point set D such that the total distance from p to the points in a query point set Q is minimal, that is, $\sum_{q\in Q} d(q,p) \leq \sum_{q\in Q} d(q,p'),\ p' \in D - p$. The applications of GNN query come from location-based services, e.g., finding a meeting venue for a group of people such that the traveling distance is minimized. In this paper, we study Group Nearest Group (GNG) query which can be regarded as a generic version of the GNN query. While a GNN query retrieves a single point in D, a GNG query aims to find a subset ω from D such that $|\omega| \leq k$ and $\sum_{q\in Q} d(q,\omega)$ is minimum where $d(q,\omega)$ is the shortest distance from q to ω (i.e., the euclidean distance from q to the nearest point in ω). GNG query has many applications in business analytic and decision support systems.

Example 1. A security service provider plans to set up several new branches to serve several business districts which can be represented by a set of landmarks, such as well-known buildings. The maximum number of branches is usually constrained by the factors such as business cost. Under this constraint, the provider wishes to select branch locations from many choices such that the response time to security alarms can be minimized, that is, the average distance from these landmarks to the nearest branch is minimal, mathematically, same as the minimum total distance. This is a typical GNG query.

Example 2. When planning a meeting for a group of people, a venue can be found from available locations by issuing a GNN query. With the maturity of teleconference techniques, people can aggregate to several convenient locations in order to further reduce the traveling cost. Due to considerations such as budget or administration costs, the maximum number of venues k is usually limited. Under this constraint, a GNG query can be used to find a set of venues such that the total traveling distance for people to the nearest venue is minimal. When $k = 1$, a GNG query is reduced to a GNN query.

The GNG query can be defined in a formal way. Given a set of data points D, a query point set Q, and an integer value k ($1 \leq k \leq min\{|D|, |Q|\}$), a GNG query finds a subset of points ω from D ($|\omega| \leq k$), such that $\sum_{q\in Q} d(q,\omega) \leq \sum_{q\in Q} d(q,\omega')$ for every $\omega' \in \mathcal{P}(D)$ ($|\omega'| \leq k$), where $\mathcal{P}(D)$ is the power set of D. A GNG query may return k points as shown in Fig. 1a. In some cases, a GNG query may return a solution with less than k points. For example in Fig. 1b, p_1 is the nearest location for all persons. No matter what the setting of k is, the minimal total traveling distance is the distance from all persons to p_1. If we are aware of the cardinality of the solution, say k', processing GNG query is to find the subset of k' points that produces the minimum total traveling distance. Unfortunately, k' is not known in advance when processing GNG query.

To avoid evaluating all subsets of cardinality from 1 to k, we observe that the subset of k points producing the

- *K. Deng, S. Sadiq, and X. Zhou are with the School of Information Technology and Electrical Engineering, The University of Queensland, Room 639, Building 78, St Lucia Campus, Brisbane, QLD 4072, Australia. E-mail: {dengke, shazia, zxf}@itee.uq.edu.au.*
- *H. Xu and Y. Lu are with the School of Computer Science and Technology, Huazhong University of Science and Technology, Wuhan, Hubei 430074, China. E-mail: hxu@smail.hust.edu.cn, lys@mail.hust.edu.cn.*
- *G.P.C. Fung is with the Arizona State University, Brickyard Suite 568, 699 S. Mill Ave., Tempe, AZ 85287. E-mail: gabriel.fung@asu.edu.*

Manuscript received 5 Nov. 2009; revised 19 Jan. 2010; accepted 14 July 2010; published online 16 Nov. 2010.
Recommended for acceptance by T. Sellis.
For information on obtaining reprints of this article, please send e-mail to: tkde@computer.org, and reference IEEECS Log Number TKDE-2009-11-0765.
Digital Object Identifier no. 10.1109/TKDE.2010.230.

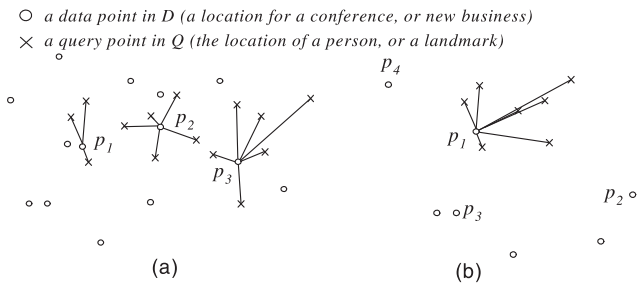

Fig. 1. Two examples of GNG query when $k = 3$.

minimum total traveling distance must contain the solution. For example, in Fig. 1b, $\{p_1, p_2, p_3\}$ produces the minimum total traveling distance but $\{p_4, p_2, p_3\}$ does not. The solution may be contained in many subsets of k points. Processing GNG query just needs to find one of them. In the obtained subset, the final solution of GNG query can be easily found by dropping those points to which no query point is grouped. For example, p_1 is the final solution by dropping p_2, p_3 from the subset $\{p_1, p_2, p_3\}$ in Fig. 1b. Thus, processing GNG query is to find one subset of k points in D such that the total distance is minimized.

When $k = 1$, a GNG query reduces to a GNN query which returns an individual point in D. The focus of GNN query processing is on minimizing access and evaluation of data points in D. Different from the GNN query, GNG query processing is to find a subset of $k(> 1)$ points in D, and the focus is on minimizing the access and evaluation of potential subsets.[1] The total number of such potential subsets is $\frac{|D|!}{(|D|-k)!k!}$ which is exponentially greater than $|D|$ when k increases from 2 (or decreases from $|D| - 2$). This paper studies the case when k increases from 2 since it is same when k decreases from $|D| - 2$. In this work, we suppose $k \leq |D|$.[2]

GNG query, also known as k-median clustering in operations research [6], is a partition-based clustering problem which groups data points into k (a given number) clusters based on an optimization objective function. The partition-based clustering problem is NP-hard [7]. The k-median clustering has been investigated using local search heuristics which are popular for hard combinatorial optimization problem [8]. The typical process of the local search heuristics finds an arbitrary solution and iteratively optimizes it with the operation of *swap* (i.e., replacing a point in the solution with another point in the search space) till improvement can be obtained. In practice, using local search heuristics for GNG query leads to a gap of a few percentage points between the obtained solution and the global optimum [9], [10]. In the worst case, the local search heuristics have been proved to achieve at most five times of the global optimum [11]. To achieve better performance, a widely used method is to represent the data set by a set of samples and then apply the local search heuristics [12], [13]. However, with this efficient method, the clustering quality is sacrificed and is unbound in the worst case.

In this work, database techniques are explored to boost the GNG query processing of local search heuristics without any loss on clustering quality. Two algorithms are proposed and they are Exhaustive Hierarchical Combination (EHC) algorithm and Subset Hierarchial Refinement (SHR) algorithm. In order to minimize the access and evaluation of potential subsets, the data points in EHC and SHR are hierarchically represented by data blocks, e.g., using R-tree. We process GNG query by treating the blocks as points to find an intermediate solution in higher hierarchical level first. To refine the solution, the search space in lower hierarchical level is minimized by following the guided search direction. EHC aims to find the optimal solution without evaluating all subsets of k points. In EHC, every set of k blocks is evaluated in high hierarchical level and the set with the current best value (i.e., the minimum total distance) are refined by visiting their children in next level. EHC is capable to provide the optimal solution. However, the performance of EHC drops down dramatically when k increases from 2.

SHR is a local search heuristic with support of the database techniques. In higher hierarchical level, each block is treated as a point by SHR to replace every element in the subset, and the resultant subset with the current best value is refined by visiting the children of the block. Being a local search heuristic, the solution of SHR is usually close to the global optimum and guaranteed to be within a factor of at most 5 from the global optimum in the worst case as discussed above. Compared to Partitioning Around Medoids (PAM) [12], a well-known local search heuristic for partition-based clustering, SHR provides the solution of same quality while the performance is better by 1-3 orders of magnitudes. In this work, we also compare SHR with Clustering Large Applications based upon RANdomized Search (CLARANS), a variant of PAM, which sacrifices quality in order to achieve a better processing efficiency by sampling. Hence, the solution quality of SHR is always better than or equal to that of CLARANS. The experiments show that SHR outperforms CLARANS by several times in most cases. For the simplicity of description, the GNG query discussed in this paper is in two-dimensional space with the support of R-tree. However, the proposed algorithms can be applied in multidimensional space with support of any hierarchical data structure as well.

Based on database techniques, a recent work [14] proposes a method called tree-based PArtition Querying (TPAQ) which can provide a quick answer for k-median clustering problem. However, the solution of TPAQ is unbounded in terms of clustering quality. Thus, TPAQ can be used to generate the initial solution of SHR and then SHR refines the initial solution to the final solution which is at most five times of the global optimum.

This paper is the full version of a published poster [15]. There, we concisely introduce the idea of SHR and EHC. Here in this full version, new material and content are supplemented mainly on three aspects. First, the technical details of SHR and EHC are discussed. That includes the optimization techniques, the disk-based algorithms, and

1. GNN may also return k individual points which are known as k group nearest neighbors [4], [5]. These points are independent from each other. For example, the first group nearest neighbor is always the best choice without other $k - 1$ points. In contrast, the objective of GNG is to identify k points which together form the solution.

2. In the case $k = |D|$, the solution of a GNG query is D; in the case $k = |Q| \leq |D|$, the solution simply consists of the nearest points of Q in D. These two trivial cases are omitted.

comprehensive analysis. Second, the implementation difficulties and solutions are investigated with a much more detailed report on experiments. Third, the justification of this work is better supported through the review of related work.

The remainder of this paper is organized as follows: the related work is followed the introduction of EHC (Section 3) and SHR (Section 4). Sections 5 and 6 discuss the situations where query points are disk based and weighted separately. The paper is concluded after the report of experimental results in Section 7.

2 Related Work

2.1 R-Tree and Nearest Neighbor Query

R-tree is a spatial data access method which partitions data points in space. A leaf node of R-tree contains a data entry referring to a data point. A nonleaf node contains a minimum bounding rectangle (MBR) which includes all MBRs associated with the child nodes. To build an R-tree, the data points close to each other in space are recursively approximated by MBRs until a single MBR is reached.

To process a spatial query such as a nearest neighbor query, R-tree can be traversed in the best first or depth-first fashion. The best first nearest neighbor search [16] maintains a priority queue for entries being visited. The queue elements are sorted in the increasing order of their minimum distance from the query point. The entry with the smallest minimum distance is visited first. The best first nearest neighbor search is proved to be optimal in terms of the number of node accesses [17]. In addition, it has the advantage that we can obtain successive nearest neighbors incrementally without recomputing the query from the scratch. In this paper, the best first traverse fashion is used in the proposed algorithms. The depth-first algorithm [18] starts from the root of the tree and visits the entry whose minimum distance from the query point is the smallest. When it reaches a leaf node, it finds the candidate nearest neighbor and traces back. In the procedure of back tracing, the algorithm only visits entries whose minimum distance is smaller than the distance of the candidate nearest neighbor.

The group nearest neighbor query was proposed by Papadias et al. [4], [5]. The input of GNN consists of a set of points D in multidimensional space and a set of query points Q. The kth GNN is the point $p \in D$ with the kth minimum $\sum_{q \in Q} d(q, p)$. Three algorithms have been proposed [4], [5]. The multiple query method (MQM) performs incremental NN query for every query point and the summation of distances from query points to their current farthest NNs is computed as a threshold. For each retrieved point from D, its distances to all query points are computed and summed. The first point with distance sum less than the current threshold is the first GNN. The single point method (SPM) finds the centroid of Q which is a point in space with a small value of distance sum to all query points. Then, the centroid is used to prune the search space by exploring the triangular inequality. The minimum bounding method (MBM) performs in a similar way as SPM, but uses the minimum bounding rectangle of Q to further prune the search space. The optimization of GNN query is to minimize the access of D since the solution is a set of individual points.

2.2 Partition-Based Clustering

Given an integer k and a data point set D, database clustering is the process of grouping data points together based on a user defined similarity function. Data points are similar to each other when they are in the same cluster and dissimilar when they are in different clusters. Two well-known partition-based algorithms are the k-means and k-medoids. The k-means algorithm uses the centroid (a virtual point rather than a data point) of the objects in each cluster as the cluster center while k-medoids algorithm uses the most centrally located point as the center. The overall distance for data points to the associated cluster centers is minimum. The k-median clustering is the bichromatic version of k-medoids clustering where there are two data sets, and one data set is grouped to k cluster centers from the other data set. In general, the task of finding a global optimal k partition belongs to the class of NP-hard problems [7]. For this reason, heuristics are used to achieve a balance between the efficiency and effectiveness close to the global optimum as much as possible.

The algorithm for k-means clustering [19], [20] is relatively scalable and efficient in processing large data sets because the computational complexity of the algorithm is $O(nkt)$, where n is the total number of objects, k is the number of clusters, and t is the number of iterations. Normally, $k \ll n$ and $t \ll n$. The method often terminates at a local optimum. The k-means is very sensitive to noise and outlier data points since a small number of such data can substantially influence the mean value. K-medoids/median clustering is less sensitive to noise and outliers. However, this results in a higher running time. Next, we review the local search heuristics and database techniques for processing k-medoid/median clustering.

Local search heuristics. The local search heuristic is popular in clusterings of k partitions and this process includes two steps [21]. The first step arbitrarily selects k points as the initial cluster centers. The data points are grouped to the nearest center and cluster quality is computed. In the second step, the iterative relocation technique is adopted. In each iteration, some/all cluster centers are updated and the cluster quality is recomputed. The second step terminates when the cluster quality cannot be further improved.

A well-known local search heuristic for k-mediod/median clustering is PAM [12]. Using PAM, k centers are randomly selected in the first step to form the initial set of medoids/median, say ω. In each iteration of the second step, PAM performs following operations: tentatively replacing a current medoid/median $p \in \omega$ by a point $p' = D - \omega$, the cluster quality improvement is computed, e.g., the reduction of the distance sum; among all such tentative replacements, the one with the maximum improvement is carried out. PAM repeats this operation until the cluster quality cannot be improved any further. According to the analysis in [11], the solution of k-medoids/k-median clustering can be solved with local search heuristics within a factor of at most 5 from the global optimum. In practice, however, the gap is usually much smaller, just a few

percentage points [9], [10]. The complexity of PAM in one iteration is $O(k(|D|-k)^2)$. For a large D, such computation becomes costly.

To deal with this situation, a sampling-based method, called CLARA [12] was developed. Instead of taking the whole set of data into consideration, a small portion of the actual data is chosen as representative. The cluster centers are chosen from this sample using PAM. But a good clustering based on samples will not necessarily represent a good clustering of the whole data set if the sample is biased. To improve the quality and scalability of CLARA, another clustering algorithm called CLARANS was proposed in [13]. The first step of CLARANS, same as PAM, randomly selects a set of cluster centers ω as the initial solution. Their difference focuses on iterative relocation process when searching for a better cluster center. CLARANS randomly selects a point $p \in \omega$ and tentatively replaces it with a randomly selected point $p' \in D - \omega$. If the cluster quality can be improved, ω is updated by replacing p by p' immediately. For the updated ω, if no better solution is found after a certain number of attempts, the local optimal solution is assumed to be reached. CLARANS has parameters $maxneighbor$ and $numlocal$ which are specified by users. $maxneighbor$ is the number of attempts. The higher the value of $maxneighbor$, the closer CLARANS is to PAM, and the longer is each search of a local minimum. $numlocal$ is the number of local optimal solutions from which the best is selected as the final solution. The cluster quality of CLARANS depends on the sampling method used.

Database techniques. Two approaches have been proposed to exploit the locality property of R-tree to improve efficiency of k-medoids/median clustering. One is *focusing on representative* (FOR) developed by Ester et al. [22] and the other is called TPAQ [14]. Given the R-tree of a data point set, FOR takes the most centrally located point of each leaf node and forms a sample set, which is considered as representative of the entire set. Then, it applies CLARANS on this sample to find the k cluster centers. FOR is essentially a preprocessing step since the final solution is eventually calculated by applying CLARANS.

For k-medoids clustering problem, TPAQ visits the R-tree of the data point set top down and stops at the topmost level where the total number k' of entries has $k' \geq k$. These k' entries are then grouped into k clusters based on their locality. To do that, TPAQ utilizes *space-filling curves* which map a multidimensional space into a linear order, typically used to preserve the locality of points. The space-filling curves are divided into k partitions and the points in the same partition are in the same cluster. Then, the weighted center (a virtual point rather than a data point) of each cluster is calculated. In the data point set, the nearest neighbor of this weighted center is used as the cluster center. TPAQ is efficient since only necessary levels of R-tree are visited. For k-median clustering problem, TPAQ visits the R-tree of one data point set (i.e., the query point set Q in GNG query) and finds the cluster centers from the other data point set (i.e., the data point set D in GNG query).

The solution of TPAQ is unbounded in terms of clustering quality. In the first step, the cluster members are decided first based on the assumption that the distances between data points can be well indexed using space-filling curves, that is, the data points closer in space-filling curves should be closer in space. It is well known that this assumption can be very weak in many cases. In the second step, once the cluster members are decided, TPAQ finds the nearest neighbor of the weighted center from the data set representing the service locations as the cluster center. Since the solution of TPAQ is unbound, it can be used in SHR as the initial solution and then refined to be within five times of the optimum. In this work, the initial solution of SHR is based on k-means algorithm (introduced in detail in Section 4.2) since k-means algorithm is well studied and rich solutions are available. In addition, k-means algorithm does not require R-tree of Q while TPAQ does. When Q is given online by user of GNG query, the online construction of R-tree can be avoided using k-means algorithm.

Summary. Two types of exiting methods which are capable to process GNG query are local search heuristics and database techniques. The local search heuristics provide theoretical quality bound to solution but they are computationally expensive. The database techniques focus on the efficiency but the quality of solution is unbounded. The aim of SHR in this paper is to return a solution to GNG query where the quality is bounded as local search heuristics and the efficiency is supported by the proposed database techniques. In the special case of a very small k (i.e., $k = 2$), we also provide an exact algorithm EHC.

2.3 Others

Zhang et al. [23] studied a problem called Min-Dist optimal-location query where a set S of sites (e.g., McDonald's stores) and a set O of objects are given. When a new site l is added into S (a new McDonald's store), i.e., $S = S \cup l$, the Min-Dist optimal-location query aims to find the location for l such that the total distance from objects in O to the nearest sites in S is minimized. While Min-Dist optimal-location query is to find a single optimal location (other locations are fixed), GNG is to find a set of optimal locations. Thus, Min-Dist optimal-location query and GNG are different problems even though they are equivalent when $k = 1$. Furthermore, we cannot solve GNG by incrementally running Min-Dist optimal-location query. Suppose one site l is found by Min-Dist optimal-location query when $k = 1$. When $k = 2$, l' is found. So, $S = \{l', l\}$. In GNG when $k = 1$, the same l is returned. But when $k = 2$, the location of l may change. Thus, the solution of GNG is different from that using Min-Dist optimal-location query when $k > 1$.

3 Exhaustive Hierarchical Combination Algorithm

The notations used in this paper are summarized in Table 1. In EHC, a GNG query is processed by traversing the R-tree of D. The pseudocode of EHC is shown in Fig. 3. At the root, every k entry forms a subset denoted as ω. For a query point $q \in Q$ and a subset ω, the minimum distance from q to an entry E in ω (E is the minimum bounding rectangle of data points under E in R-tree as shown in Fig. 2) is denoted as function $mindist(q, E)$ and the maximum distance is denoted as function $maxdist(q, E)$. Given a set Q of query

TABLE 1
Summary of Notations

Notation	Explanation
D	a data point set (e.g., conference venues)
Q	a query point set (e.g., people)
τ	a threshold
ω	a subset of k points in D
$d(q,\omega)$	the minimum Euclidean distance from a point q to ω
$mindist(q,E)$	the minimum Euclidean distance from a point q to an entry E in a node of R-tree
$maxdist(q,E)$	the maximum Euclidean distance from a point q to an entry E in a node of R-tree

points and a subset ω, the query points can be grouped to the closest entries in ω according to these two distance functions. For distance function $mindist(q,E)$, the distance sum from query points to the closest entries in ω is denoted as $\sum_{q\in Q} mindist(q,\omega)$, and $\sum_{q\in Q} maxdist(q,\omega)$ for distance function $maxdist(q,E)$; that is, ω has two distance sums. Let a leaf node be the child leaf node of ω if it is under one entry in ω. It is clear that any subset formed by k child leaf nodes of ω must have a distance sum in between $\sum_{q\in Q} mindist(q,\omega)$ and $\sum_{q\in Q} maxdist(q,\omega)$. Thus, we denote $\sum_{q\in Q} mindist(q,\omega)$ as $\sum_{lb}$ and $\sum_{q\in Q} maxdist(q,\omega)$ as $\sum_{ub}$.

At the root, let $\{\omega_1,..,\omega_n\}$ be all possible subsets of k entries. When each of these subsets computes $\sum_{lb}$ and $\sum_{ub}$, they are inserted into a heap H. The minimum $\sum_{ub}$ in H is assigned to a threshold τ. If a subset ω in H has lower bound distance $\sum_{lb}^{\omega} \geq \tau$, it can be safely pruned from H. That is, the child leaf nodes of ω cannot form a subset of size k with a distance sum less than τ. For the remaining subsets in H, following the best first traverse fashion, the ω with the minimum $\sum_{lb}$ is selected; the direct child nodes of ω (i.e., the nodes directly referred by entries in ω) are visited and every k entries of these child nodes form new subsets. The $\sum_{lb}$ and $\sum_{ub}$ are computed for each of these new subsets. After inserting them into H, the threshold τ is set as the current minimum $\sum_{ub}$ in H. Similarly, any subset in H with $\sum_{lb} \geq \tau$ can be safely pruned. These operations are repeated until the minimum $\sum_{lb}$ in H is associated with a subset of

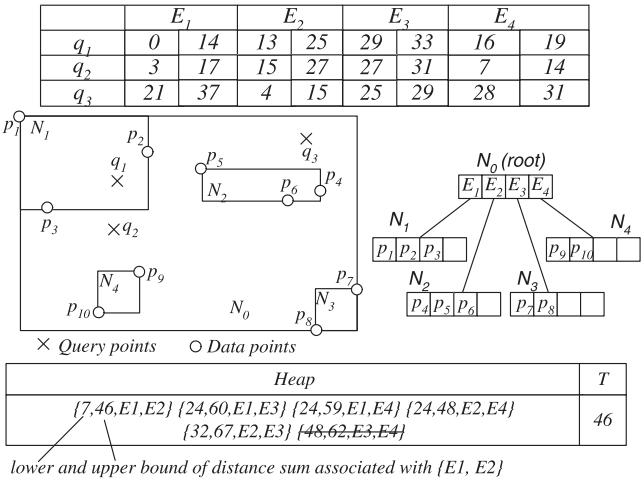

Fig. 2. An example of GNG query processing using EHC ($k = 2$).

Algorithm EHC(D,Q,k)
Input: a data set D, a query set Q, an integer k
Output: $\omega \in \Omega$ with the minimum total distance
1. ξ = entries in root of R-tree on D;
2. $H = \emptyset$; $\tau = +\infty$;
3. do
4. for each subset ω of k entries in ξ
5. compute $\sum_{lb}^{\omega}$ and $\sum_{ub}^{\omega}$;
6. $H \leftarrow \{\sum_{lb}^{\omega}, \sum_{ub}^{\omega}, \omega\}$;
7. τ = the minimum $\sum_{ub}$ of subsets in H;
8. for each subset $h \in H$
9. if $h.\sum_{lb}^{\omega} \geq \tau$, prune h from H;
10. remove $h \in H$ where $h.\sum_{lb}^{\omega}$ is minimum;
11. if $h.\omega$ consists of entries referring to leaf nodes, break;
12. $\xi = \emptyset$;
13. for each entry $E \in h.\omega$
14. ξ = ξ + entries in node referred by E;
15. while true
16. return $h.\omega$;
END EHC

Fig. 3. Pseudocode of EHC.

leaf nodes (in this case $\sum_{lb} = \sum_{ub}$). This subset is returned by EHC. For description clarity, the subsets are inserted into the heap and then pruned. However, the pruning can also take place before inserting subsets into H.

Fig. 2 shows an example of EHC. The entries in the root are combined into subsets of size $k = 2$ and these subsets are inserted into a heap. The minimum $\sum_{ub}$ in H is 46. Using $\tau = 46$ as the threshold, the subset $\{E_3, E_4\}$ can be pruned from the heap. Then, $\{E_1, E_2\}$ are replaced by all possible subsets of size 2 composed by direct child nodes of E_1, E_2.

4 Subset Hierarchial Refinement Algorithm

EHC is workable only for small k even though optimization is applied. In this section, we describe the Subset Hierarchial Refinement algorithm. It can provide a high quality solution while the performance is linear with k, D, and Q. The time complexity is $O(k(|D|-k)|Q|)$. SHR is inspired by PAM. SHR works basically as follows: first, an initial subset ω is identified; then, SHR tries to refine ω by tentatively replacing elements in ω by points in $D-\omega$ until the total distance cannot be reduced further.

4.1 Pruning Rules

Before going to the details of SHR, we first propose two pruning rules which explore the proximity of the query points in order to prune the irrelevant subsets.

Initial candidate pruning (ICP). Given D, Q, and a subset $\omega \subset D$, we tentatively use points in $D-\omega$ to replace the points in ω in order to obtain an updated ω which produces a smaller total distance. During this process, a point $p' \in D-\omega$ can be pruned if $d(q,p') \geq d(q,\omega)$ for $\forall q \in Q$.

ICP aims to reduce the search space when optimizing a given ω by replacing one point $p \in \omega$ with a data point $p' \in D-\omega$. The replacement should reduce the total distance from query points in Q to the closest points in ω. If a point $p' \in D-\omega$ cannot reduce the total distance by replacing it with any point in ω, p' is not relevant to the optimization of ω. The purpose of ICP is to identify such p' and prune it.

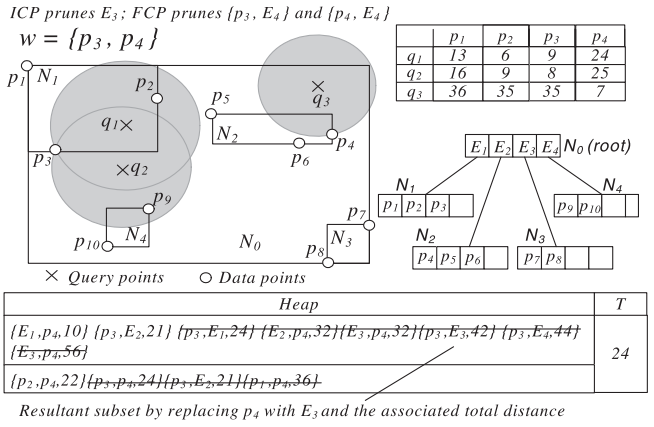

Fig. 4. An example of the first iteration of SHR.

ICP is valid when p' is an entry of a node in R-tree. When a node N is visited during the tree browsing, the entries in N are processed. To each entry, say E, the distance $mindist(q, E)$ is computed from all $q \in Q$. E can be pruned if $mindist(q, E) \geq d(q, \omega)$ for $\forall q \in Q$. The reason is that replacing any $p \in \omega$ by any data point under E in R-tree cannot reduce the total distance. We use the example shown in Fig. 4 to describe the process. ω is $\{p_3, p_4\}$ to which the points in Q are grouped: $\{q_1, q_2\}$ to p_3 and $\{q_3\}$ to p_4. When the node N_0 is visited, the entries $\{E_1, E_2, E_3, E_4\}$ are examined. E_3 has a longer distance to q_1, q_2 than p_3 and has a longer distance to q_3 than p_4. Replacing p_3 or p_4 by E_3 in ω cannot reduce the total distance. Thus, E_3 can be safely pruned.

Further candidate pruning (FCP). Given D, Q, and a subset $\omega \subset D$, we tentatively use points in $D - \omega$ to replace a point $p \in \omega$ in order to obtain an updated ω which produces a smaller total distance. During this process, $p' \in D - \omega$ can be pruned if the resultant total distance cannot be reduced.

If a point $p' \in D - \omega$ is not pruned by ICP, FCP is applied to check whether replacing p' with some points in ω can be avoided. The idea is to compute the total distance when replacing p' with a point in ω. Clearly, if p' is a data point, every possible replacement is processed and thus no computation is avoided. The pruning power of FCP takes effect when p' is an entry of an R-tree node. As shown in Fig. 4, when replacing p_3 by the entry E_4, q_1, q_2 will change to E_4 and q_3 is still to p_4. Since the resultant total distance cannot be reduced, any data point under E_4 cannot diminish the total distance if it replaces p_3 in ω. Thus, the replacement between E_4 and p_3 can be pruned safely and so does E_4 and p_4.

Both FCP and ICP are used in SHR when browsing R-tree. When a node is visited, the entries in this node are pruned using ICP first, and then the remaining entries are pruned using FCP. The entries pruned by ICP can also be pruned by FCP. That is, FCP does decide the number of entries accessed and processed. The purpose of ICP is that it is cheaper to prune some entries and so FCP is avoided to do that same job. The effect of ICP to performance is stronger when query points are distributed in a small region (discussed in Section 4.3.2) and disk based (discussed in Section 5).

Time consumed to find k-mean centers (s)

k	10	20	30	40	50
|Q| = 64	0.5345	1.0078	1.4969	1.9781	2.4687

|Q|	4	16	64	256	1024
k = 6	0.3173	0.3281	0.3453	0.3358	0.3486

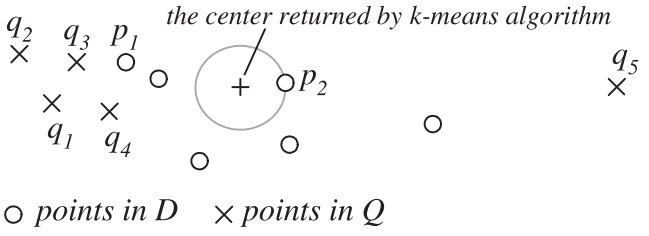

Fig. 5. An example of ω_{ini}.

4.2 GNG Query Processing

The query processing starts by selecting an initial subset ω_{ini}. We can arbitrarily select k points from D as ω_{ini}. However, if ω_{ini} produces a smaller total distance, fewer replacements are required to reach the final solution and thus a better overall performance. In this work, we employ an approach based on k-means algorithm which considers the distribution of both Q and D. First, Q is clustered using k-means algorithm. Then, for each obtained center (i.e., mean), the nearest neighbor from D is retrieved as a member of ω_{ini} until there are k distinct points in ω_{ini}. A noteworthy point is that ω_{ini} is not reliable to be an approximate solution of GNG. For example, in Fig. 5, p_2 is the nearest neighbor of the center of query points $\{q_1, .., q_5\}$. Even though p_1 can result in a smaller total distance, p_2 not p_1 is included in ω_{ini}. In the second step, the obtained ω_{ini} is refined. The pseudocode of SHR is illustrated in Fig. 6.

Let the current ω (ω_{cur}) be ω_{ini}. For the initial ω_{cur}, SHR computes a threshold $\tau = \sum_{q \in Q} d(q, \omega_{cur})$. Then, we try to find a better solution by replacing points in ω_{cur} by points in $D - \omega_{cur}$ with support of R-tree. This is an iterative process. At the root N for the first iteration, we tentatively replace each $p \in \omega_{cur}$ by every entry $E \in N$ and compute the resultant total distance sum. For an entry E and a point $p \in \omega_{cur}$, only if the resultant $sum < \tau$, we record $\{sum, p, E\}$ in a heap H (empty initially); otherwise, this replacement can be safely pruned according to the pruning rules. After processing tentative replacements between all entries in N and all points in ω_{cur}, if H is empty, SHR terminates by returning ω_{cur} since no replacement can further reduce the total distance; else if H is not empty, the element $h \in H$ with the minimum sum is removed from H for further processing. For h with $h.E$ referring to a nonleaf node, the entries in this node are processed in the similar way as we process the root. The only difference is that these entries only tentatively replace with $h.p$ rather than all $p \in \omega_{cur}$. For h with $h.E$ referring to a leaf node, say p', the first iteration is done by replacing $h.p$ with p' in ω_{cur} and resetting $\tau = h.sum$. In the next iteration, the R-tree is browsed again with the updated ω_{cur} and τ in the same way. SHR terminates until no further reduction of the total distance is possible as discussed above.

In Fig. 4, the same example of Fig. 2 is reused to illustrate the processing of the first iteration. The initial (current)

Algorithm SHR(D,Q,k)
Input: a data set D, a query set Q, an integer k
Output: $\omega \in \Omega$ with the best cluster quality
1. ω_{cur} = find ω_{ini};
2. τ = compute sum based on ω_{ini};
3. N = root of R-tree on D;
4. $H = \emptyset$;
5. for each entry E in N
6. for each $p \in \omega_{cur}$
7. compute sum when E replaces $p \in \omega_{ini}$;
8. if $sum < \tau$, $H \leftarrow \{sum, p, E\}$;
9. if $H = \emptyset$, SHR terminates by returning ω_{cur};
10. remove $h \in H$ where h has the minimum sum;
11. $p = h.p$; N = node referred by $h.E$;
12. while N is a non-leaf node
13. for each entry E in N
14. compute sum when E replaces $p \in \omega_{cur}$;
15. if $sum < \tau$, $H \leftarrow \{sum, p, E\}$;
16. if $H = \emptyset$, return ω_{cur};
17. remove $h \in H$ where h has the minimum sum;
18. $p = h.p$; N = node referred by $h.E$;
19. ω_{cur} = replace $p \in \omega_{cur}$ by N;
20. $\tau = h.sum$, go to line 3;
END SHR

Fig. 6. Pseudocode of SHR.

subset is $\{p_3, p_4\}$ and $\tau = 24$. The R-tree is browsed by visiting the root first. Replacing each point in the initial subset with every entry in the root, the resultant subset and the associated total distance are recorded in a heap. If the total distance is greater than τ, the resultant subset can be safely pruned.[3] Two subsets $\{E_1, p_4, 10\}\{p_3, E_2, 21\}$ remain in the heap. Since $\{E_1, p_4, 10\}$ has the minimum total distance, E_1 is replaced by every entry in node N_1 in this subset. Subsequently, the subset $\{p_2, p_4, 22\}$ has the minimum total distance. Since p_2, p_4 are leaf nodes, the first iteration stops; the current subset and τ are updated to $\{p_2, p_4, 22\}$ and 22.

As a further step to cut off the tentative replacements, τ can be updated during the R-tree traversed within one iteration. It is similar to the method used in EHC. When tentatively replacing a point $p \in \omega_{cur}$ with an entry E, we can also group query points based on their maximum distances to E. The consequential total distance is the upper bound of the total distance of the subset which is obtained by replacing p with any data point under E in R-tree. It is denoted as $\sum_{ub}$. If $\sum_{ub} < \tau$, τ is updated to $\sum_{ub}$. For an element $h \in H$, it can be pruned safely pruned if $h.sum \geq \tau$.

4.3 Analysis

4.3.1 SHR versus PAM and CLARANS

k-medoids algorithms PAM and CLARANS can be used to process GNG queries. Using PAM, k centers are randomly selected in the first step to form the initial set of medoids, say ω. In each iteration of the second step, PAM performs following operations. Tentatively replacing each current medoid $p \in \omega$ by every point $p' = D - \omega$, the cluster quality improvement is computed, e.g., the reduction of the distance sum. Among all such tentative replacements, the one with the maximum improvement is carried out. In next iteration, PAM repeats this operation until the cluster quality cannot be improved any further. The complexity of PAM in one iteration is $O(k(|D| - k)^2)$. For a large D, such computation becomes costly. If the initial subset is the same, SHR and PAM return the identical solution since SHR is equivalent to PAM without the proposed database techniques to improve the processing efficiency. Compared to PAM, SHR is more efficient as it avoids unnecessary tentative replacements with support of R-tree and pruning rules. As discussed in Section 2, PAM can always find solutions of GNG query that are within a factor of at most 5 from the optimum and for practical, nonpathological instances, the gap is usually much smaller, just a few percentage points.

In CLARANS, one randomly selected point in ω_{cur} is tentatively replaced by one randomly selected point in $D - \omega_{cur}$. If the total distance can be reduced, ω_{cur} is updated by carrying out this replacement immediately. For the updated ω_{cur}, if no better solution is found after $maxneighbor$ attempts, a local optimal solution is assumed to be reached. From $numlocal$ local optimal solutions, the global solution is returned. Therefore, the solution of CLARANS is unbounded in terms of quality. By setting a higher value of $maxneighbor$, the quality of solution can be improved to approach the quality level of PAM and SHR.

In order to compare the efficiency of SHR with CLARANS, suppose SHR and CLARANS have a common initial subset ω. In the best case of CLARANS, ω is the final solution such that no better point in $D - \omega$ can be found by conducting $maxneighbor$ tentative replacements. Then, CLARANS returns ω. As suggested in [13], $maxneighbor$ is 1.25 percent of the number of the possible replacements between all points in D and all points in ω, and thus the number of subsets evaluated by CLARANS is $1.25\% * k(|D| - k)$ in the best case. In SHR, each node at high level of R-tree tentatively replaces all $p \in \omega$ and, following the best first traverse fashion, we visit the children of the node by replacing which the total distance can be reduced the most. The total number of subsets evaluated by SHR is $m * n$ where m is the number of nodes visited and n is the fan-out of node. n is a constant in a given R-tree and the value is 50 in our experiments. In the best case of SHR, ω is the final solution such that m is $\log_n |D|$. For a medium size D ($|D| = 20k$ in the PP data set used in our experiments), $m * n$ in SHR is smaller than $1.25\% * k(|D| - k)$ in CLARANS by several times in the best case. The larger the D, the better the relative performance of SHR, compared to CLARANS. In general cases, SHR is deliberately designed to always refine the currently most promising subset when traversing the index structure while CLARANS randomly selects a subset to process. Thus, we expect the performance of SHR to be generally better than CLARANS.

4.3.2 Q Region and Pruning Rules

The distribution region of Q impacts on the pruning rules. In general, if Q is distributed in a relatively small region compared to the distribution region of D, the pruning rule ICP works very well. As shown in the example in Fig. 7a, the MBR of Q is around 10 percent of MBR of D. The data

3. Only FCP pruning rule is applied in the example. If ICP is used, $\{E3, p4\}\{p3, E3\}$ can be avoided.

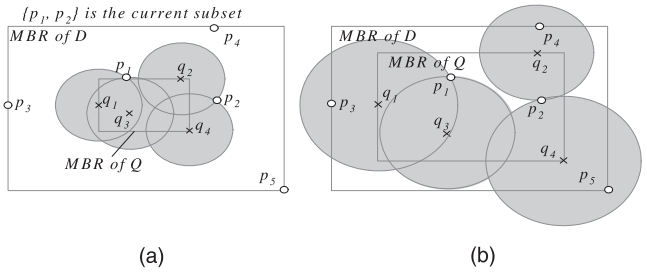

Fig. 7. Q region and pruning rules ($k = 2$).

points outside the gray area are pruned by ICP, such as p_3, p_4, p_5. In Fig. 7b, the MBR of Q is around 45 percent of MBR of D. The gray area increases significantly and there are no points, that can be pruned by ICP in this example.

For the points that cannot be pruned by ICP, the pruning rule FCP is applied. Given the current subset ω, when tentatively replacing a point $p \in \omega$ by $p' \in D - \omega$, p' is pruned according to FCP if the resultant total distance cannot be reduced. In the case of large MBR of Q, the point $p' \in D - \omega$ not pruned by ICP may be pruned by FCP when tentatively replacing a point $p \in \omega$. For example, the point p_5 pruned by ICP in Fig. 7a can be pruned by FCP in Fig. 7b when tentatively replacing p_1 by p_5. That is, even though ICP loses pruning power in case of large MBR of Q, FCP pruning rule still works effectively.

4.3.3 Processing GNN Query

When setting $k = 1$, a GNG query is reduced to a GNN query. Thus, the algorithm EHC and SHR can be directly used to process GNN queries. It is clear that the GNN solution reported by either EHR or SHR is optimal. It is possible to compare SHR and EHC against the existing algorithms [4] in terms of query processing efficiency. Since SHR and EHC aim to process GNG query, the generic form of GNN, we do not expect that EHC and SHR perform better than those algorithms specifically designed for GNN.

4.4 Further Optimization of SHR

In SHR algorithm shown in Fig. 6, R-tree needs to be traversed from scratch in each iteration in order to further refine ω_{cur}. This is similar to PAM and CLARANS. In each iteration, PAM always accesses all points in $D - \omega_{cur}$ to tentatively replace every point in ω_{cur}. In CLARANS, a point is randomly selected in $D - \omega_{cur}$ to replace a randomly selected point in ω. For each iteration, the same procedure is repeated. The results from previous iterations are not used in the following iteration to reduce the processing cost. To further optimize the efficiency of SHR, this section provides an approach to make use of the results of previous iteration such that the same GNG solution can be achieved by traversing R-tree only once.

Recall in the first iteration, a heap H is built up. H records the tentative replacements in the form $\{sum, p, E\}$ where p is a point in ω_{cur} and E is an entry in a node of R-tree. When the first iteration is done, ω_{cur} is updated to ω'_{cur} by replacing one point $p_{old} \in \omega_{cur}$ with a point $p_{new} \in D - \omega_{cur}$. To avoid traversing the R-tree from scratch in the second iteration, the current H, the legacy of the first iteration, can be used. H is transformed to H' which is the heap by traversing R-tree from scratch based on ω'_{cur}. To correctly make the transformation, two aspects need to be considered.

The first aspect is to transform the elements currently in H. There are two situations. 1) The element $h\{sum, p, E\} \in H$ where $h.p \neq p_{old}$, the point in ω_{cur} but not in ω'_{cur}. To transform such element, $h.sum$ is recomputed by tentatively replacing $h.p \in \omega'_{cur}$ with $h.E$. 2) The element $h\{sum, p, E\} \in H$ where $h.p = p_{old}$. This element is transformed by setting $h.p = p_{new}$. The $h.sum$ does not change since p_{new} has replaced p_{old} in ω'_{cur}.

The second aspect is the pruning rules. For the entries pruned by ICP, replacing any $p \in \omega_{cur}$ by any such entry cannot produce a smaller total distance. But this may not be true when ω_{cur} is updated to ω'_{cur}. We must generate $\{sum, p, E\}$ for every $p \in \omega_{cur}$ and every entry E pruned by ICP, and then transform it in the same way as the elements currently in H. In order to do that, all entries like E are kept rather than pruned. The extra storage required is relatively small since these pruned entries by ICP are in a level as high as possible in R-tree. For the tentative replacement $\{sum, p, E\}$ pruned by FCP, $sum \geq \tau$ is held based on ω_{cur}. When ω_{cur} is updated to ω'_{cur}, it is not necessarily true. $\{sum, p, E\}$ needs to be transformed in the same way as we transform the elements currently in H. Therefore, $\{p, E\}$ needs to be kept rather then pruned. This introduces a relatively small storage cost since E is an entry in the highest possible level of R-tree. We use a heap I to store the entries pruned by ICP and another heap F to store $\{p, E\}$ pruned by FCP. When the size of the heap cannot fit in main memory, they can be stored on disk.

Now, we introduce how to apply this optimization in SHR. When one iteration is done and ω_{cur} is updated to ω'_{cur}, if next iteration is necessary, SHR transforms current H to H' and examines the heap I and F in the way described above. If a part of I and F is stored on disk, it is read in main memory by pages. For an element $\{p, E\} \in F$, it is moved from F to H' if it produces a $sum < \tau$ based on ω'_{cur}; for an element $\{sum, p, E\} \in H'$, it is moved to F if $sum \geq \tau$. For an entry $E \in I$, if it cannot be pruned by ICP based on ω'_{cur}, an element $\{p, E\}$ is created for every $p \in \omega'_{cur}$ and is inserted to F and H' according to the sum produced. After the transformation is done, SHR continues the second iteration by selecting the element $h \in H'$ with the minimum sum followed by a visit to the children referred by $h.E$ as usual.

Using this optimization, the operations performed in order to transform H to H' and process I and F are usually necessary when an iteration is conducted by traversing R-tree from scratch. The performance improvement of this optimization comes from not redoing the access and evaluation of R-tree nodes that have been accessed and evaluated in previous iteration(s).

5 Disk-Based Q

This section discusses the GNG query processing techniques when Q is too large to fit in main memory. In this situation, Q is stored on disk and each time a fraction of Q is read into main memory. The space of Q is divided into rectangular regions at a desired level of granularity. The criteria are that the data points in the densest partition can reside on one disk page. All partitions are organized in a

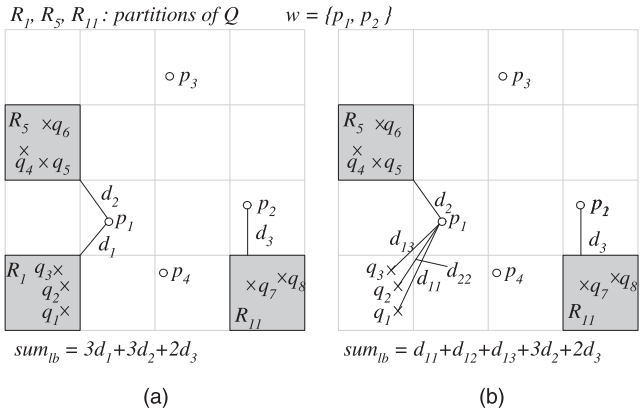

Fig. 8. An example of disk-based Q processing in EHC ($k = 2$).

single sequence by using the space-filling curve in Morton order (Z-order). Thus, Q is a set of partitions $\Phi = \{R_1, .., R_z\}$ where i is the Z-value of partition R_i. Using Z-value, the rectangle of each partition can be easily identified in space. B^+-tree is used to index these partitions on Z-value.

In EHC, when a set of new subsets are generated during R-tree traverse, each of them needs to be evaluated. To reduce the I/O of Q, these subsets are evaluated together by accessing Q once. First, the boundary information (i.e., rectangle) of all partitions is read into main memory. For each subset ω, the partitions are grouped to the nearest point $p \in \omega$ according to the minimum distance from p to the boundary of each partition, and $sum_{lb} = \sum_{R \in \Phi} |R| * d(R, \omega)$ is computed where $|R|$ is the number of query points in R (note if $p \in \omega$ is an entry of an R-tree node, the distance between p and the boundary of a partition R is the minimum distance between them). Clearly, sum_{lb} is a lower bound of the total distance from points in Q to the nearest point in ω. If a subset has the total distance lower bound $sum_{lb} \geq \tau$, this subset can be safely excluded. If ω is not pruned, the first sequence of partitions $\Phi_1 = \{R_1, .., R_n\}$ which is on the same disk page is read in main memory. For a subset ω, we group every query point $q \in R_i, .., R_n$ to the nearest point in ω and $sum_1 = \sum_{q \in R_1, .., R_n} d(q, \omega)$ is computed. The sum_{lb} of ω is updated to $sum_{lb} + sum_1 - \sum_{R \in \Phi_1} |R| * d(R, \omega)$, that is, the distances from $q \in R_1, .., R_n$ to the nearest point in ω substitute the distances from the boundaries of $R_1, .., R_n$ to the nearest point in ω. If ω has $sum_{lb} \geq \tau$, ω is pruned. If ω is not pruned, similar operations are performed by reading next sequence of partitions in main memory. When entire Q has been processed, the total distance from Q to ω is computed ($= sum_{lb}$). sum_{ub} can be computed for each ω in a similar way. Whenever there is a $\sum_{ub} < \tau$, $\tau = \sum_{ub}$.

An example is shown in Fig. 8. $\omega = \{p_1, p_2\}$ is evaluated when Q is accessed by partitions. The total distance lower bound sum_{lb} is computed by boundary of partitions as shown in Fig. 8a. If $sum_{lb} < \tau$, the points in R_1 are read in main memory and sum_{lb} is updated as shown in Fig. 8b.

In SHR, an initial subset ω_{ini} needs to be found first. To do that, the query points inside a partition are represented by the center of the partition, and the number of the query points is attached with this center. Then, ω_{ini} can be found using k-means approach based on such centers of all partitions. Following the same way in EHC, the total distance from disk-based Q to ω_{ini} can be computed. Next, we try to improve the initial solution by replacing each point in ω_{ini} by each point in $D - \omega_{ini}$. During R-tree traverse, when a node is visited, several new subsets need to be evaluated. The method is the same as the one in EHC.

6 Weighted Query Points

In the GNG query discussed so far, all query points in Q have unit weight (i.e., equal to 1). However, the query points with different weights are useful in many situations. In the conference venue example, a query point q may indicate a suburb where 60 people live. Assigning each query point a proportional weight to represent the number of people, the GNG query can be used to find the venues that minimize the distance sum by all persons (as opposed to locations).

Let the weight associated with a query point $q \in Q$ be $q.wei$, a nonnegative real number. By treating each weighted query point q as $q.wei$ query points of unit weight, the GNG query with weighted query points can be directly processed by EHC and SHR. Even though the number of query points is increased in this case, the processing efficiency of EHC and SHR is not affected. The extra computation cost for processing $q.wei$ query points is incurred only when computing their distances to cluster centers. This extra cost is trivial. The reason is that the $q.wei$ query points of unit weight are at the same location. They must be in the same cluster and their distances to the cluster center must be same. In specific, the total distance from these $q.wei$ query points to the cluster center p is $d(q, p) * q.wei$; if E is an entry when browsing R-tree, the summation of the $mindist(q, E)$s (or $maxdist(q, E)$s) from $q.wei$ query points to E is $mindist(q, E) * q.wei$ (or $maxdist(q, E) * q.wei$).

7 Experiments

7.1 Experiment Setup

The experiments have been conducted using a PC (Intel Core2 6600 CPU 2.4 GHz, 2 GB memory). Four data sets are tested:

1. A real data set PP with 24,493 points is obtained from www.rtreepotral.org which consists of the populated places of the North America.
2. A real data set NE (www.rtreepotral.org), containing 123,593 postal addresses (points), which represents three metropolitan areas (New York, Philadelphia, and Boston).
3. A synthetic data set $UN1$ with 6×10^4 uniformly distributed points.
4. A synthetic data set $UN2$ with 500 uniformly distributed points.

These data sets are unified into a unit region. Q is distributed in an area whose MBR is a percentage of the whole data space, denoted as M. The disk page is set to 1,024 Bytes and a 1M Bytes buffer is used. All the data sets are indexed by R-trees for EHC and SHR. For PAM and CLARANS, these data sets are stored on a number of disk pages. We implemented the PAM and CLARANS algorithms and the k-means algorithm [19], [20]. To make a fair

comparison, a common initial subset is computed for SHR, PAM, and CLARANS using the k-means approach. For CLARANS, the ratio of $maxneighbor$ is set to 1.25 percent as suggested in [13]. To minimize I/O, CLARANS randomly selects a data page and then randomly selects 1.25 percent points from this page until $maxneighbor$ is reached. $numlocal$ is set as 1 due to common initial subset ($numlocal = 2$ in [13] where the initial subset is arbitrarily selected). The experimental results reported are the average of 100 queries. The default settings of experiments are $Q = 64$, $k = 6$, $M = 10\%$, and $D = PP$. In all experiments, the "average page accesses" refer to the total I/O including the disk flushing for heaps and the R-tree accesses.

7.2 Priority Queue Management

Both EHC and SHR use the priority queue in GNG query processing in order to apply best first search strategy of the index. The optimized SHR requires two more heaps to keep the combinations pruned by FCP and the nodes pruned by ICP. The complexity of EHC in memory requirement is $O(\frac{|D|!}{(|D|-k)!k!})$ and it is $O(|D||Q|)$ for SHR (including the optimized SHR). It is not always feasible to store queues in memory and thus the queue management is an important implementation issue. A simple memory-based implementation might slow down query processing due to excessive disk flushing. We implement the hybrid memory/disk scheme proposed by Hjaltason and Samet [16]. This scheme is briefly introduced here to make this paper self-contained. The idea is to partition the queue elements based on sum. The entire range of sum is $[0, \tau)$ on which we define a set of intervals $[D_0(=0), D_1)$, $[D_1, D_2)$, .., $[D_{p-1}, D_p)$, $[D_{p+1}, D_m(=\tau))$. Initially, tier 1 is associated with the interval $[0, D_1)$, tier 2 with the intervals $[D_1, D_2)$, .., $[D_{p-1}, D_p)$, and tier 3 with $[D_{p+1}, \tau)$. We only keep tier 1 in main memory. The elements in the same interval of tier 2 are grouped in the same disk pages. When a new element with $sum = r$ is inserted into the priority queue, we add this element to the tier whose associated interval matches r. When tier 1 is exhausted, we move the elements in interval $[D_1, D_2)$ from tier 2 to tier 1 and associate tier 1 with this interval. The next time when tier 1 is exhausted, we move elements in interval $[D_2, D_3)$ into tier 1, and so on. If this happens often enough, eventually we will exhaust tier 2. When this happens, we scan the entire contents of tier 3 and rebuild tiers 1 and 2 with new ranges. If the number of subsets to be evaluated is small, the elements in tier 2 may never need to be placed in main memory.

In our implementation, we use the following way to estimate the boundaries between tiers. Suppose n is the number of elements that can be stored in main memory and m is the total number of subsets that produce the total distance less than the initial threshold τ. The boundary between tiers 1 and 2 is $D_0 + \frac{n}{m}\tau$ and the boundary between tiers 2 and 3 is $D_0 + \frac{1}{2}\tau$. The ith interval in tier 2 is $[D + i\frac{n}{m}\tau,\ D + (i+1)\frac{n}{m}\tau)$. We estimate m as follows: when the initial τ is computed, we draw a circle around every query point q with radius $\frac{1}{k}\tau$. If a point in D is enclosed by any circle, it is counted once and let the total number of such points be η. We estimate $m = \frac{\eta!}{(\eta-k)!k!}$ in EHC and $m = \eta(\eta - k)$ in SHR. Since m is a rough estimation, the number of elements in tier 1 may be over

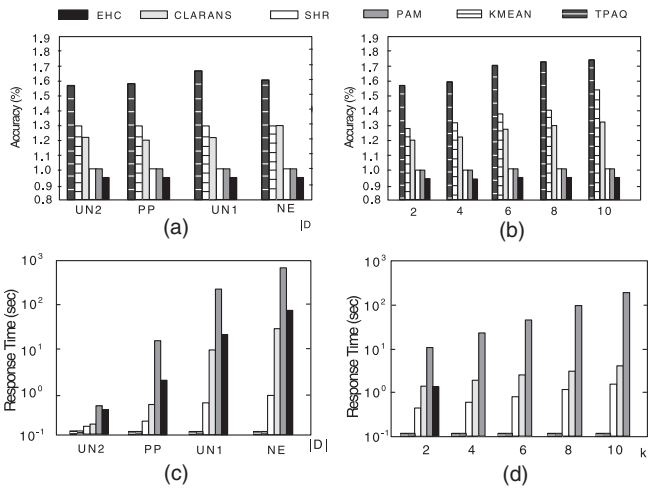

Fig. 9. Effect of $|D|$ and k ($|Q| = 64$, $M = 10\%$). (a) Accuracy versus $|D|$. (b) Accuracy versus k (PP). (c) Response time versus $|D|$. (d) Response time versus k (PP).

the memory constraint n. We allow the boundary between tiers 1 and 2 to be changeable. In this case, some elements in tier 1 are moved to tier 2 and accordingly the boundary between tiers 1 and 2 is adjusted.

In the experiment charts shown in the rest of this section, the page accesses are for R-tree accesses. But if the given buffer is not enough and thus disk flushing occurs in heap management, the page accesses are broken into I/O for R-tree accesses and I/O for heap management. For the default buffer (1M Bytes) in our experiments, only the experiments related to EHC (Section 7.5) require disk flushing.

7.3 Solution Quality

In this experiment, the quality of solution returned by EHC, SHR, PAM, CLARANS, k-means, and TPAQ is examined by comparing the average total distances. The initial solutions of SHR, PAM, and CLARANS are identified by k-means approach as described in Section 4.2. GNG query aims to find a subset of points in D which produces the minimum total distance. Therefore, we say that the solution producing a smaller average total distance has a higher quality. Since EHC is only workable for small k, we use PAM not EHC as the baseline to normalize the quality, i.e., the total distance returned by any algorithm is divided by the total distance returned by PAM.

For TPAQ in this test, since Q is small, it is not worth indexing by R-tree. We treat each query point in Q as an entry selected from R-tree, i.e., $k' = |Q|$, and these k' entries are used to find k centers to form the final solution of TPAQ (see details in Section 2). Since the query points are used directly instead of their approximations, the quality of the solution returned should be same or better. Therefore, the comparison in solution quality shown in Figs. 9a and 9b is fair to TPAQ.

Fig. 9a demonstrates the experimental results by setting $k = 2$ on four data sets, and Fig. 9b demonstrates the experimental results by setting k from 2 to 10 on the data set PP. In Fig. 9a, PAM and SHR demonstrate the same quality which is very close to that of optimal solution using EHC. While PAM is 1, the optimal quality is within 0.95 in average. The k-means approach returns solutions which

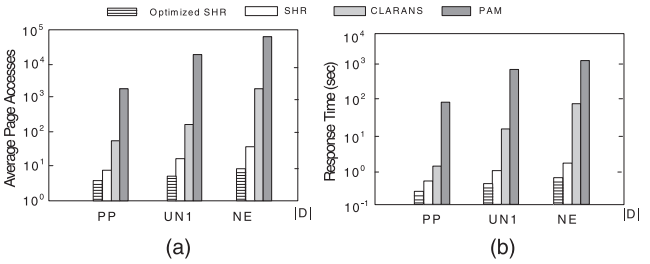

Fig. 10. Effect of $|D|$ ($|Q| = 64,\ k = 6,\ M = 10\%$). (a) PA versus $|D|$. (b) Response time versus $|D|$.

produce a varying quality around 1.3-1.35 in average when $|D|$ increases from $UN2$ to NE. When k increases from 2 to 10, the quality of the TPAQ gets worse to 1.5-1.6. The quality of TPAQ is obviously worse than that of k-means in all settings.

As shown in Figs. 9c and 9d, the costs of both TPAQ and k-means approach are trivial. Compared to TPAQ, k-means can generate a better solution and thus it is preferable for the initial solution of GNG. The reason is that the initial solution of higher quality tends to decrease the cost of subsequent operations based on local search heuristics which overwhelm GNG processing, see Figs. 9c and 9d.

CLARANS always returns solutions with quality in between those returned by k-means and SHR. As aforementioned, CLARANS can achieve a better quality when $maxneighbor$ is large enough at the expense of performance. When k increases, the quality of CLARANS tends to decrease.

7.4 Efficiency of SHR

This section compares SHR, optimized SHR, PAM, and CLARANS in terms of disk page accesses and the response time. The effects of $|D|$, $|Q|$, k, and M are investigated. Since all settings of k are six or more, EHC is extremely expensive and thus excluded in this section. (The performance of EHC is examined separately in Section 7.5 by setting $k = 2$.)

Fig. 10 demonstrates the performance of these algorithms on different data sets while $k = 6$, $|Q| = 64$, and $M = 10\%$. Along with the increase of $|D|$, the performance of all algorithms shows an obvious trend to increase linearly but at different rates. The increase of PAM is the fastest since PAM tentatively replaces every point in the initial subset ω with all other points in $D - \omega$. Thus, the size of data set D impacts on the performance in both I/O and CPU of PAM directly. CLARANS shows a much better performance and a relative slower increase rate. This is because CLARANS evaluates at most 1.25 percent of the subsets examined by PAM in each iteration. SHR and optimized SHR show the superior performance and the best scalability in all settings. The main reason is that the search of solution in SHR is always guided to the most promising subsets; thus, the search space is effectively pruned. Compared to SHR, the optimized SHR shows an improved performance since it browses the R-tree once instead of multiple times as in SHR.

In Fig. 11, the performance with different settings of k from 10 to 50 is illustrated. We omit PAM in Figs. 11c and 11d because of its extremely high cost on large data set NE. Given $|Q| = 64$ and $M = 10\%$, the experimental results demonstrate that the average performances of SHR,

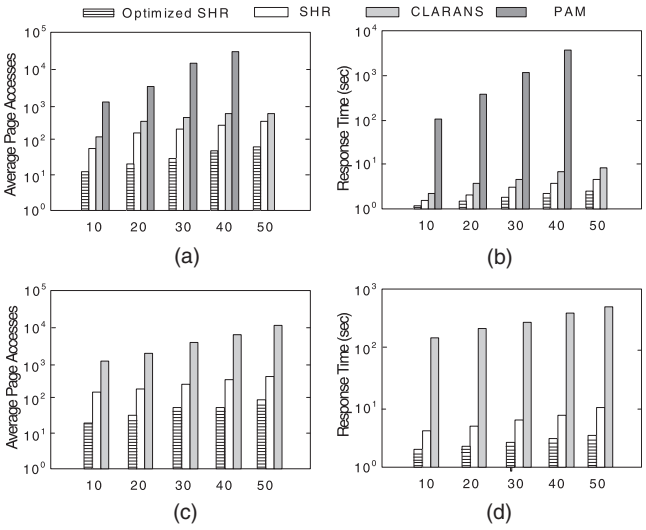

Fig. 11. Effect of k ($|Q| = 64,\ M = 10\%$). (a) PA versus k (PP). (b) Response time versus k (PP). (c) PA versus k (NE). (d) Response time versus k (NE).

optimized SHR, CLARANS, and PAM increase in a virtually linear way with the increase of k. SHR and optimized SHR demonstrate the best performances in all settings. The increase rate of PAM is the highest when k increases. This is because refining an initial subset with greater k generally requires not only more subsets in each iteration to be examined but also more iterations. This can also explain why CLARANS shows a worse performance compared to SHR and optimized SHR. Compared to PAM, CLARANS is obviously better since CLARANS carries out the replacement whenever the total distance can be reduced and thus the cost for each replacement in CLARANS is much lower than that in PAM.

The effect of $|Q|$ on the performance is shown in Fig. 12. In this experiment, the points in Q are all stored in main memory and $|Q|$ varies from 16 to 1,024. Figs. 12a and 12c show that the number of subsets that need to be assessed is almost a constant for given D, M, and k. When evaluating

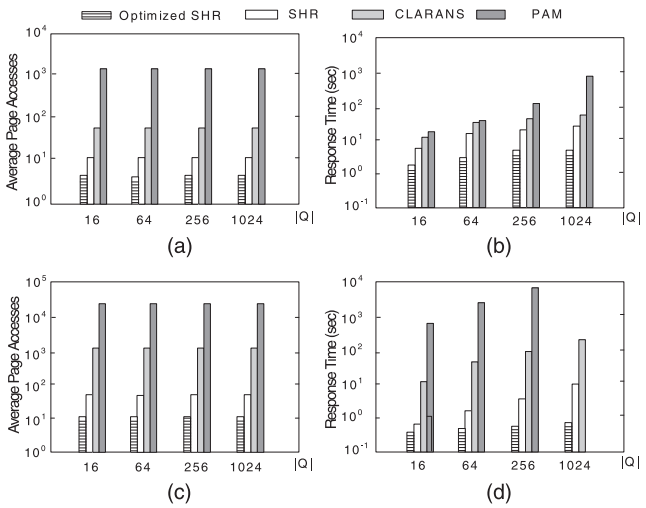

Fig. 12. Effect of Q ($k = 6,\ M = 10\%$). (a) PA versus $|Q|$ (PP). (b) Response time versus $|Q|$ (PP). (c) PA versus $|Q|$ (NE). (d) Response time versus $|Q|$ (NE).

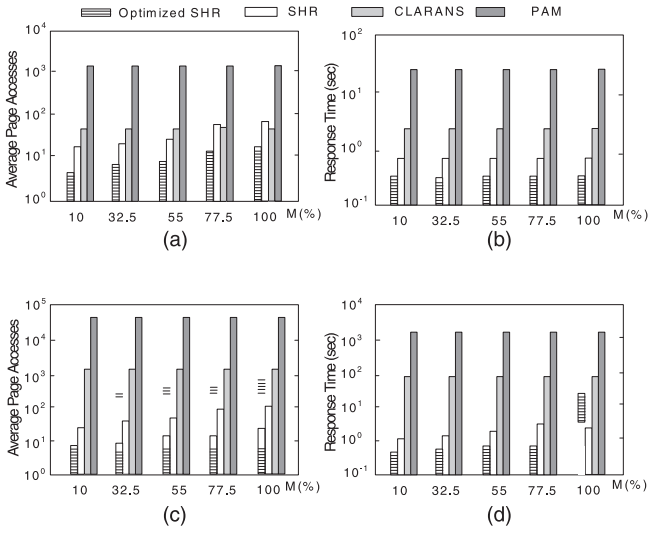

Fig. 13. Effect of M ($k = 6$, $|Q| = 64$). (a) PA versus M (PP). (b) Response time versus M (PP). (c) PA versus M (NE). (d) Response time versus M (NE).

each subset, the number of query points, $|Q|$, influences the response time. The reason is that, for PAM and CLARANS, the number of subsets to be processed is only determined by $k(|D| - k)$. In SHR and optimized SHR, the number of subsets to be processed is influenced mainly by D, M, and k. Even though the distributions of Q may have a slight impact, the average of 100 query results makes the difference unobservable.

The distribution of Q affects the performance remarkably. As aforementioned, this factor is represented by M. $M = 10\%$ means Q is distributed within an MBR covering 10 percent of the region where D is distributed. We examine the performance of SHR, optimized SHR, PAM, and CLARANS when M varies from 10 to 100 percent. The experimental results are shown in Fig. 13. PAM and CLARANS are independent of the change of M since the number of subsets processed is only determined by $k(|D| - k)$. In contrast, the page access of SHR and optimized SHR increases when M increases. Two reasons contribute to this situation. First, more nodes need to be tested. Second, the pruning power of ICP decreases and so more entries are tested using expensive FCP. We can see that the response time of SHR and optimized SHR increase obviously slower as compared to the increase rate of page accesses. This verifies that the pruning ability of FCP is well preserved for greater M.

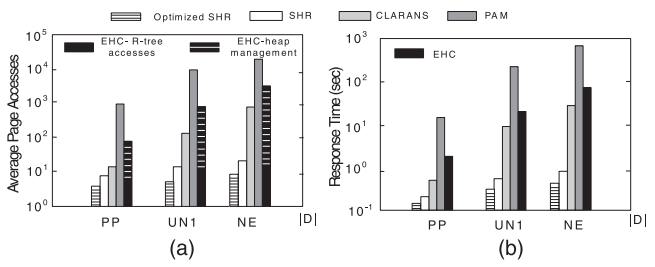

Fig. 14. Effect of $|D|$ ($|Q| = 64$, $k = 2$, $M = 10\%$). (a) PA versus $|D|$. (b) Response time versus $|D|$.

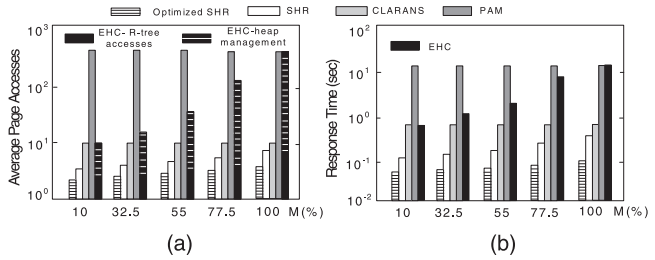

Fig. 15. Effect of M ($|Q| = 64$, $k = 2$). (a) PA versus M (PP). (b) Response time versus M (PP).

7.5 Efficiency of EHC

EHC is the only algorithm which can provide an exact solution. When k is greater than 2, EHC is extremely costly. Thus, the focus of this experiment is to examine the efficiency of EHC when $k = 2$. The experimental results on varying D and M are presented. Since the buffer is not enough to hold the heap of the priority queue and disk thrash occurs, the page accesses shown in the charts consist of I/O for R-tree accesses and I/O for heap management.

Since $|Q|$ is a weaker factor in EHC same as in SHR, the experimental result is omitted in this section. In Fig. 14, EHC, PAM, CLARANS, and SHR are compared on different D. While SHR has an increasingly better relative performance as $|D|$ increases, EHC does not. The reason is that EHC tries to examine all subsets. The number of all subsets in case $k = 2$ is $0.5 * |D|(|D| - 2)$. Even though optimization techniques are used, the number of subsets that need to be examined increases faster than SHR when $|D|$ increases.

Fig. 15 compares the performances on different settings of M from 10 to 100 percent. Similar to Fig. 12, PAM and CLARANS are generally independent from the change of M. In the case of large M, the performance of EHC degrades since the pruning capacity of the spatial index becomes weaker. The difference between SHR and EHC is that I/O and CPU cost of EHC gets worse much quicker than that of SHR. This is caused by the situation that all subsets of entries in nodes not pruned will be examined by EHC.

7.6 Disk-Based Q

When Q is stored on disk, the processing technique proposed in Section 5 is used. Fig. 16 demonstrates the effect of different sizes of Q to the performance of SHR. MSHR denotes SHR with Q in main memory and DSHR denotes SHR with Q on disk. We evaluate the entries in one node of R-tree in one batch, that is, Q is read in main memory once. If ICP pruning rule is not applied, the

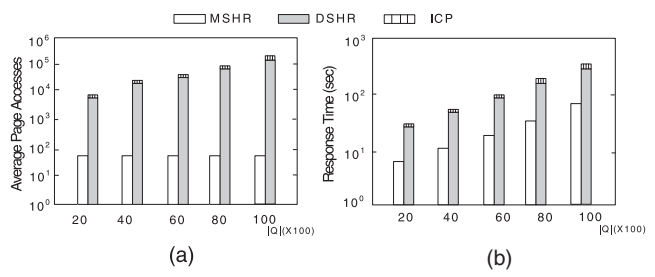

Fig. 16. Effect of disk-based Q ($K = 6$, $M = 10\%$). (a) PA versus $|Q|$. (b) Response time versus $|Q|$.

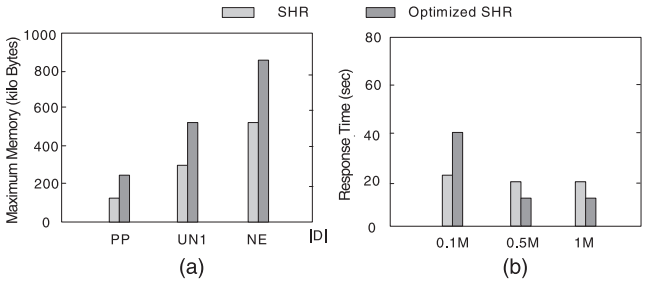

Fig. 17. Effect of buffer size ($K = 6$, $M = 10\%$). (a) Max Memory versus $|D|$. (b) Response time versus Buffer size (PP).

extra cost is also shown and is denoted as ICP. The purpose of ICP is that it is cheaper to prune some entries and so FCP is avoided to do that same job. The effect of ICP to performance is stronger when query points are distributed in a small region and disk based. It shows around 10 percent of the overall performance if ICP is not applied in the given settings.

7.7 Effect of Buffer Size

The priority queues are maintained by EHC and SHR due to the application of the best first search strategy of R-tree. The optimized SHR requires two more heaps to keep the subsets pruned by FCP and the entries pruned by ICP. In all above experiments, the default buffer (1M Bytes) is large enough to meet the memory requirement of SHR and optimized SHR, but EHC needs disk flushing as shown in Section 7.5. In this section, we demonstrate the maximum memory requirement of SHR and optimized SHR; and how the performance is influenced by disk flushing in the case of small buffer. As shown in Fig. 17a, the optimized SHR almost needs twice memory of SHR. The maximum memory increases with large data set with same settings on M, k. Fig. 17b illustrates that SHR shows better performance by comparing to optimized SHR in the case of small buffer.

8 Conclusion

The studied GNG query can be found in various business analytic and decision support systems. Since GNG query is a problem computationally intractable, both efficiency and effectiveness are critical to the practical value of GNG query. We use hierarchical blocks instead of data points to optimize the number of subsets evaluated. This work proposes two GNG query processing algorithms, Exhaustive Hierarchical Combination algorithm and Subset Hierarchial Refinement. EHC provides the optimal solution when $k = 2$ and SHR provides fast approximate solution for any setting of k. Compared to PAM [12], SHR has the same quality while the performance is better by 1-3 orders of magnitude. Compared to CLARANS [13], SHR returns quality bounded solution and SHR outperforms CLARANS by several times in most cases.

Acknowledgments

This work is supported by grants DP1096826, DP0773122 from the Australian Research Council.

References

[1] M. Yiu, N. Manoulis, and D. Papadias, "Aggregate Nearest Neighbor Queries in Road Networks," *IEEE Trans. Knowledge and Data Eng.*, vol. 17, no. 6, pp. 820-833, June 2005.
[2] K. Deng, X. Zhou, and H. Shen, "Multi-Source Skyline Query Processing in Road Networks," *Proc. 23th IEEE Int'l Conf. Data Eng.*, 2007.
[3] M. Sharifzadeh and C. Shahabi, "The Spatial Skyline Queries," *Proc. 32nd Very Large Data Bases Conf.*, 2006.
[4] D. Papadias, Q. Shen, Y. Tao, and K. Mouratids, "Group Nearest Neighbor Queries," *Proc. 20th IEEE Int'l Conf. Data Eng.*, 2004.
[5] D. Papadias, Y. Tao, K. Mourstidis, and C.K. Hui, "Aggregate Nearest Neighbor Queries in Spatial Databases," *ACM Trans. Database Systems*, vol. 30, no. 2, pp. 529-576, 2005.
[6] P. Hansen, *Systems of Cities and Facility Location.* Harwood Academic Publishers GmbH, 1987.
[7] M. Garey and D. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness.* Freeman and Company, 1979.
[8] *Local Search in Combinatorial Optimization*, E. Aarts and J.K. Lenstra, eds. Princeton Univ. Press, 2003.
[9] K.E. Rosing, "An Empirical Investigation of the Effectiveness of a Vertex Substitution Heuristic," *Environment and Planning B*, vol. 24, pp. 59-67, 1997.
[10] R. Whitaker, "A Fast Algorithm for the Greedy Interchange of Large-Scale Clustering and Median Location Problems," *INFOR*, vol. 21, pp. 95-108, 1983.
[11] V. Arya, N. Gary, R. Khandekar, A. Mayerson, K. Munagala, and V. Pandit, "Local Search Heuristics for k-Median and Facility Location Problems," *Proc. 33rd ACM Symp. Theory of Computing*, 2001.
[12] L. Kaufman and P. Rousseeuw, *Finding Groups in Data: An Introduction to Cluster Analysis.* John Wiley & Sons, 1990.
[13] R. Ng and J. Han, "Efficient and Effective Clustering Method for Spatial Data Mining," *Proc. 20th Very Large Data Bases Conf.*, 1994.
[14] K. Mouratidis, D. Papadias, and S. Papadimitriou, "Tree-Based Partition Querying: A Methodology for Computing Medoids in Large Spatial Datasets," *The VLDB J.*, vol. 17, no. 4, pp. 923-945, 2008.
[15] K. Deng, H. Xu, S. Sadiq, Y. Lu, G. Fung, and H. Shen, "Processing Group Nearest Group Query," *Proc. 25th IEEE Int'l Conf. Data Eng.*, 2009.
[16] G. Hjaltason and H. Samet, "Distance Browsing in Spatial Databases," *ACM Trans. Database Systems*, vol. 24, no. 2, pp. 265-318, 1999.
[17] C. Bohm, S. Berchtold, and D. Keim, "Searching in High Dimensional Spaces: Index Structures for Improving the Performance of Multimedia Databases," *ACM Computing Surveys*, vol. 33, no. 3, pp. 322-373, 2001.
[18] K. Cheung and A.W.C. Fu, "Enhanced Nearest Neighbor Search on the R-Tree," *ACM SIGMOD Record*, vol. 27, no. 3, pp. 16-21, 1998.
[19] I. Dhillon and D. Modha, "A Data-Clustering Algorithm on Distributed Memory Multiprocessors," *Proc. Large-Scale Parallel Data Mining*, pp. 245-260, 1999.
[20] J. Macqueen, "Some Methods for Classification and Analysis of Multivariate Observations," *Proc. Fifth Berkeley Symp. Math. Statistics and Probability*, pp. 281-297, 1967.
[21] J. Han, M. Kamber, and A. Tung, "Spatial Clustering Methods in Data Mining: A Survey," *Geographic Data Mining and Knowledge Discovery*, pp. 1-29, 2001.
[22] M. Ester, H.P. Kriegel, and X. Xu, "A Database Interface for Clustering in Large Spatial Databases," *Proc. First ACM SIGKDD Conf. Knowledge Discovery and Data Mining*, 1995.
[23] D. Zhang, Y. Du, T. Xia, and Y. Tao, "Progressive Communication of the Min-Dist Optimal-Location Query," *Proc. 32nd Very Large Data Bases Conf.*, 2006.

Ke Deng received the PhD degree in computer science from the University of Queensland (UQ), Australia, in 2007. After attaining the PhD degree, he joined CSIRO ICT Centre. Currently, he is an Australian postdoctoral fellow (2010-2012) funded by Australian Research Council in the ITEE School at UQ. His research focus is on spatial data management, data quality, and high performance query processing.

Shazia Sadiq received the master's degree in computer science from the Asian Institute of Technology, Thailand, and the PhD degree in information systems from the University of Queensland (UQ), Australia. She is an associate professor in the ITEE School at the University of Queensland. She is involved in teaching and research in databases and information systems. Her main research interests are innovative solutions for business information systems including business process management, governance, risk and compliance, and data quality management.

Xiaofang Zhou is a professor of computer science at the University of Queensland, Australia. His research focus is to find effective and efficient solutions for managing, integrating, and analyzing very large amount of complex data for business, scientific, and personal applications. He has been working in the areas of spatial and multimedia databases, data quality, high performance query processing, web information systems, and bioinformatics. He is a senior member of the IEEE.

Hu Xu received the BS degree in computer science from Wuhan University in 2002 and the MS degree in computer science from Huazhong University of Science and Technology in 2006. He has been working toward the PhD degree at the School of Computer Science and Technology, Huazhong University of Science and Technology, since 2006. His research interests include spatiotemporal databases and information retrieval.

Gabriel Pui Cheong Fung is working as a research fellow in Arizona State University. He was a lecturer in The Chinese University of Hong Kong (Aug. '05-Aug. '07) and was a research scientist in The University of Queensland (Jan. '08-Dec. '09). His research interest is data and text mining, especially multisource data mining, bioinformatics, and social media and network analysis.

Yansheng Lu is a professor of computer science and technology at Huazhong University of Science and Technology. His research interests include spatiotemporal databases, software testing, data mining, and data fusion.