Geoinformatica (2019) 23: 105–161
https://doi.org/10.1007/s10707-018-00338-7

An overlapping Voronoi diagram-based system for multi-criteria optimal location queries

Ji Zhang[1] · Po-Wei Harn[2] · Wei-Shinn Ku[1] · Min-Te Sun[3] · Xiao Qin[1] · Hua Lu[4] · Xunfei Jiang[5]

Received: 26 October 2017 / Revised: 9 June 2018 / Accepted: 30 November 2018 /
Published online: 9 January 2019

Abstract
This paper presents a novel Multi-criteria Optimal Location Query (MOLQ), which can be applied to a wide range of applications. After providing a formal definition of the novel query type, we propose an Overlapping Voronoi Diagram (OVD) model that defines OVDs and Minimum OVDs (MOVDs), and an OVD overlap operation. Based on the OVD model, we design advanced approaches to answer the query in Euclidean space. Due to the high complexity of Voronoi diagram overlap computation, we improve the overlap operation by replacing the real boundaries of Voronoi diagrams with their Minimum Bounding Rectangles (MBR). Moreover, if there are changes to a limited number of objects, re-evaluating queries over updated object sets would be expensive. Thus, we also propose an MOVD updating model and an advanced algorithm to incrementally update MOVDs to avoid the high cost of query re-evaluation. Our experimental results show that the proposed algorithms can evaluate the novel query type effectively and efficiently.

Keywords Voronoi diagram · Optimal location query

1 Introduction

Numerous optimal location queries considering a wide range of criteria have been extensively studied. As an example of location decision making problems, Multi-criteria Optimal Location Query, or MOLQ in short, was proposed to find a location by taking multiple factors (*e.g.*, distance and reputation) into account [56]. Specifically, given a family of object sets in different types, the query returns an optimal location, which minimizes the total weighted distance from the location to one object in each type.

Making residential location decisions is a typical example of MOLQ that finds home locations with maximum residential satisfaction [36]. In order to attract more customers, an optimal location would be selected for minimizing the total distance from the location to a

✉ Xunfei Jiang
jiangxu@earlham.edu

Extended author information available on the last page of the article.

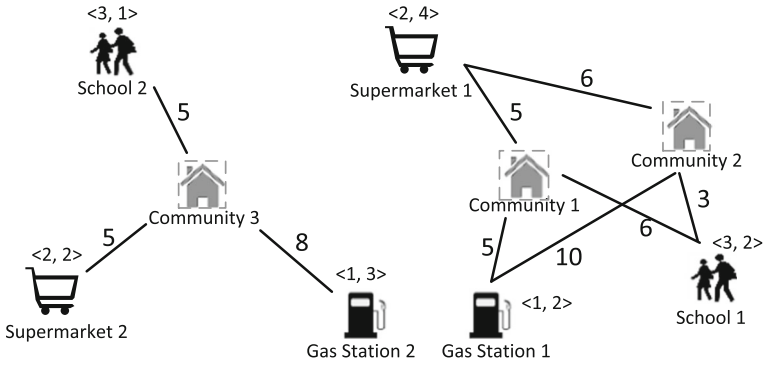


Fig. 1 An example of residential location selection. The object weights are indicated as $< w^t, w^o >$. Smaller weights indicate higher preference

supermarket, an elementary school, and a gas station. Figure 1 displays a simple example. There are two schools, two gas stations, and two supermarkets in the city. Their locations are indicated by symbols. The figure also shows three potential community locations. Lines connect communities to their closest gas station, school, and supermarket, respectively. The numbers on the lines indicate the distance between two locations in Euclidean space. If the optimal location for a new community is the place that minimizes the total distance to its closest school, gas station, and supermarket, the best place is $Community\ 1$, the total distance (16) of which is shorter than that of $Community\ 2$ (19) or $Community\ 3$ (18).

Tradeoffs of multiple factors are actually considered in real residential location selection [47]. The importance of schools, gas stations, and supermarkets varies greatly among people. For example, some people may prefer living near a school because it is convenient to drive their children to school. In addition, objects of a particular type are considered differently. When selecting a school, the ones that provide higher quality programs are more attractive than others. In order to take these differences into consideration, a *type weight* w^t and *object weight* w^o are associated with each object. Providing objects with weights in the location selection allows users to prioritize objects based on their preference. If the weights $< w^t, w^o >$ customized by users are as indicated in Fig. 1 (smaller weights indicate higher preference), the best choice is $Community\ 3$ (59), which has the smallest sum of weighted distance to the nearest school ($5 \times 3 \times 1 = 15$), gas station ($8 \times 1 \times 3 = 24$), and supermarket ($5 \times 2 \times 2 = 20$). We assume that the weighted distance of a community and an object is calculated as the product of the distance and the two weights. Instead of associating a single weight with an object, a *type weight* and *object weight* are set individually in the example because various weight functions are allowed to be applied to the *type weight* and *object weight* individually in the query. This will be described in Section 3. In the example, a multiplicatively-based weight function is applied to both *type weight* and *object weight*. Appropriately selecting the factors and their weight values is another interesting problem. More discussions can be found in [36, 47]. We focus mainly on the novel query type in this paper.

The proposed query is challenging due to the following reasons. First, the query searches an optimal location in the entire search space. There are no candidate locations available for the query. Second, the computational complexity of the query grows exponentially with larger input data sets. The cost of examining all object combinations would be considerably high. Third, various indexing methods have been proposed for the evaluation process of spatial queries. For example, Voronoi-Quad-tree (or VQ-tree in short) was developed to

improve the response time of k Nearest Neighbor (kNN) query, reverse kNN query, and closest pair query on road networks [10]. However, MOLQ allows users to specify the input object weights and type weights as preferences, which might be greatly varied in queries. Therefore, if Voronoi diagrams are used, they have to be dynamically generated according to object locations and such varying weights. Building any indices on the Voronoi diagrams at run-time would be expensive for query processing.

Therefore, motivated by properties of Voronoi diagrams, we propose an Overlapped Voronoi Diagram (OVD) model, which integrates location information and object weights w^o of spatial objects by overlapping the Voronoi diagrams generated from the objects. With the OVD model, the closest objects of different types to a particular location can be efficiently retrieved without checking all combinations of objects.

To efficiently answer the query in Euclidean space, we design two solutions based on the OVD model in two steps. First, the proposed solutions generate an OVD from input objects. Due to the difference in dominance regions of objects, a Real Region as Boundary (RRB) solution calculates the real overlapping region of two Overlapping Voronoi Regions (OVRs); while a Minimum Bounding Rectangle as Boundary (MBRB) solution approximates the overlapping region by using Minimum Bounding Rectangles (MBRs) for avoiding high cost of OVR overlapping computation. Then, by utilizing Fermat-Weber techniques, our solutions iterate all potential OVRs, and produce the global optimal location as the result of the query. Additionally, due to a surprisingly large number of OVRs output by RRB or MBRB methods, we propose a cost-bound iterative algorithm (Algorithm 5) that is able to significantly reduce the computational complexity of the original iterative method [48] (See Section 8.1.2).

Moreover, the object location or weights may vary over time in applications. The quality of programs in a school may be re-evaluated every year. More positive or negative reviews to a supermarket may be continuously posted. If there are changes in a limited number of objects, re-evaluating MOLQs over updated object sets by using either RRB or MBRB would be considerably expensive due to high cost of Voronoi diagram overlapping operations. Therefore, we propose a new problem, which focuses primarily on updating the result of MOLQ if locations, *object weights*, or *type weights* of objects are changed. After providing a formal definition of the MOLQ updating problem, we demonstrate a baseline approach, which incrementally updates Voronoi diagrams of input object sets that contain updated objects and generates a new MOVD by overlapping the Voronoi diagrams. To avoid re-computing MOVDs, we propose an MOVD updating model, in which the object insertion and deletion operations are defined to incrementally update MOVDs. Based on the updating model, we propose an advanced MOVD-based incremental updating approach, which only updates the Overlapping Voronoi Regions (OVRs) inside the dominance regions of updated objects and the neighbor OVRs of the objects. We further analyze the object updating algorithms over ordinary MOVDs (which are generated from ordinary Voronoi diagrams).

The contributions of this study are summarized below:

1. We formulate a novel Multi-criteria Optimal Location Query (MOLQ) that is able to find optimal locations comprehensively by considering multiple criteria.
2. We build an OVD model, and analyze its properties and overlap operations systematically.
3. After introducing a Sequential Scan Combinations (SSC) solution as a baseline, we propose a Real Region as Boundary (RRB) solution and a Minimum Bounding Rectangle as Boundary (MBRB) solution based on the OVD model. RRB and MBRB can efficiently evaluate the novel query type in Euclidean space.

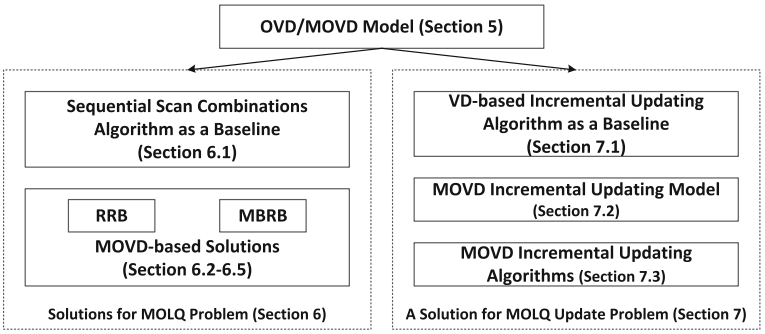


Fig. 2 The structure of this paper

4. We identify a novel Multi-criteria Optimal Location Query (MOLQ) updating problem.
5. We build an MOVD updating model, and propose an MOVD-based incremental updating approach to efficiently address the MOLQ updating problem. We analyze the computational complexity of our proposed object updating algorithms over ordinary MOVDs. The computational complexity of object insertion operation over an MOVD is bounded by $3 \times I$, where I denotes the average number of OVRs in the Voronoi dominance region of the new object. In the worst case, the computational cost becomes $O(n^2)$, where n indicates the number of objects in each object type.
6. We evaluate the performance of the proposed solutions through extensive experiments with real-world data sets.

The rest of this paper is organized as follows. Section 2 surveys related works. The proposed query and relevant mathematical tools utilized in our solutions are formally defined in Section 3. As Fig. 2 displays, the OVD/MOVD model is detailed in Section 5. After presenting a baseline solution for MOLQ query, we illustrate two MOVD-based solutions in Section 6. We create an MOVD updating model and propose a MOVD incremental updating solution for the MOLQ updating problem in Section 7. The experimental validation of our designs is presented in Section 8. We conclude the paper in Section 9.

2 Related work

In this section, we review works related to reverse nearest neighbor queries, optimal location queries, and incremental methods for Voronoi diagrams.

2.1 Reverse nearest neighbor query

Korn and Muthukrishnan [29] proposed the influence set notion based on reverse nearest neighbor (RNN) queries. They presented a precomputation-based approach for solving RNN queries and an R-tree based method (RNN-tree) for large data sets. In order to decrease index maintenance costs in [29], Yang and Lin [51] presented the Rdnn-tree which combines the R-tree with the RNN-tree and leads to significant savings in dynamically maintaining the index structure. The solutions in [29, 51] can be employed to evaluate both the monochromatic RNN query and the bichromatic RNN query; however, these precomputation-based techniques incur extra maintenance costs for data updates. Therefore, several solutions

without precomputation were proposed. For discovering influence sets in dynamic environments, Stanoi et al. [39] presented techniques to process bichromatic RNN queries without precomputation. The design is to dynamically construct the influence region of a given query point q where the influence region is defined as a polygon in space which encloses all RNNs of q. For the monochromatic RNN query, Tao et al. [41] developed algorithms for evaluating RkNN with arbitrary values of k on dynamic multidimensional data sets by utilizing a data-partitioning index. The algorithms were later extended to support continuous RkNN searches [42], which return the RkNN results for every point on a line segment.

There are some other works related to RNN query evaluation. Retrieving RNN aggregations (such as COUNT or MAX DISTANCE) over data streams was introduced in [30]. Yiu et al. [53] proposed pruning-based methods to find RNNs in large graphs. The algorithms for efficient RNN search in generic metric spaces were presented in [43]. The techniques require no detailed representations of objects and can be applied as long as the similarity between two objects can be computed and the similarity metric satisfies the triangle inequality. Cheema et al. [6] studied the problem of continuous monitoring of reverse k nearest neighbor queries in Euclidean space as well as in spatial networks. Parisa et al. [19] investigated a novel Continuous Maximal Reverse Nearest Neighbor (CMaxRNN) query on spatial networks. The query assumes that objects would frequently change their locations. Instead of calculating the optimal network location by time, their method incrementally updates the MaxRNN query results on spatial networks. Choudhury et al. [8] studied a bichromatic reverse k nearest neighbor queries on spatial-textual datasets. The query returns an optimal location and a set of keywords, which maximize the size of bichromatic reverse spatial texture k nearest neighbors (MaxBRSTkNN). While the aforementioned approaches work well for R(k)NN queries, they cannot be utilized to evaluate the unique query type studied in this paper for the following reasons. First, R(k)NN queries find objects from a given object set; while no optimal location candidates are given in MOLQ queries. Second, RNN queries only consider two types of objects; but MOLQ queries may take more than two types of objects into account. Third, the distance between objects in two types is used in object selection in R(k)NN queries; but MOLQ queries evaluate the total sum of distance between a location and many objects.

2.2 Optimal location query

One group of optimal location queries (OLQ) is defined with an optimization function which maximizes the influence of a facility. Given a set of sites, a set of weighted objects, and a spatial region Q, the optimal-location query defined in [14] returns a location in Q with a maximum influence based on the L_1 distance, where the influence of a location is the total weight of its RNNs. Xia et al. [49] proposed pruning techniques based on a metric named $minExistDNN$ to retrieve the top-t most influential sites according to the total weights of their RNNs inside a given spatial region Q. The Optimal Location Selection (OLS) search was introduced in [18], which retrieves target objects in a target object set D_B that are outside a spatial region R but have maximal optimality with a given data object set D_A and a critical distance d_c. Here, The optimality of a target object $b \in D_B$ located outside R is defined as the number of the data objects from D_A that are inside R and have distances to b not exceeding d_c.

Another group of location optimization queries is defined with a different optimization function which minimizes the average distance between a client and the nearest facility. Zhang et al. [54] proposed the Min-Dist Optimal Location Query (MDOLQ). Given

a set S of sites, a set O of weighted objects, and a spatial region Q, MDOLQ returns a location for building a new site in Q, which minimizes the average distance from each object to its closest site according to the L_1 distance. They provide a progressive algorithm that quickly suggests a location, tell the maximum error the outcome may have, and continuously refine the result. When the algorithm finishes, the exact answer can be found. Because user movements are usually confined to underlying spatial networks in practice, Xiao et al. [50] extended OLQ to support queries on road networks. They designed a unified framework that addresses three variants of optimal location queries. By observing that users can only choose from some candidate locations to build a new facility in many real applications, Qi et al. [37] introduced the Min-dist Location Selection Query (MLSQ) based on the studies in [50, 54]. Given a set of clients and a set of existing facilities, MLSQ finds a location from a given set of potential locations for establishing a new facility where the average distance between a client and her nearest facility is minimized. MND, a method for efficiently solving MLSQ, employs a single value to describe a region that encloses the nearest existing facilities of a group of clients. MND can achieve close performance to the fastest common methods without extra indices. Chen et al. [7] re-visited the optimal location query problem based on road networks. They introduced a novel idea of nearest location component in their method, and applied it to three types of problems, namely the optimal multiple-location query problem, the optimal location query on 3D road networks, and the optimal location query problem with another objective. Yao et al. [52] proposed a unified framework to address three variants of optimal location queries. Moreover, the framework was extended to support the incremental monitoring of the query results when the locations of facilities and clients have been changed. Liu et al. [31] investigated the optimal location queries for finding more than one new server or facilities. And they also developed an approximation algorithm for the cases when a large number of new servers needed to set up. However, these studies differ from the proposed query type in definition and optimization functions. Consequently, we cannot use them for answering our novel query type.

2.3 Incremental methods for Voronoi diagram

A natural way to construct a Voronoi diagram is to incrementally insert objects to the Voronoi diagram [21]. The object insertion operation of Voronoi diagram is a process that finds a new Voronoi cell enclosing the new point. The process often consists of walking through the neighbors of the new point and splitting Voronoi cells of the neighbors. Ohya et al. studied the order of object insertions and developed an improved incremental construction method that finds an optimal insertion order in a pre-processing stage [34]. Sugihara and Iri focused on the topological structure of objects rather than their numerical values, and proposed a method for generating Voronoi diagrams from millions of objects [40]. Guibas and Stolfi proposed a Voronoi diagram construction method and an object insertion method by using the Voronoi dual and Delaunay triangulation. The computational complexity of the two methods are $O(n \log n)$ and $O(n)$ in average cases [22]. Guibas et al. developed a randomized incremental construction algorithm, which randomizes the insertion sequence of objects [23]. The computational complexity of their method is bounded by $O(n \log n)$ for any collection of objects regardless of their distribution. The incremental construction algorithms have been applied to many applications [1, 17, 20, 24]. Object deletion operation of Voronoi diagram is largely the reverse of the object insertion process with specific difficulties. Devillers presented that Heller's algorithm is false, and proposed an efficient algorithm with cost $O(n \log n)$ for vertex deletion operation in a planar Delaunay triangulation by

Table 1 Symbolic notations

Symbol	Meaning
P_i	An object set of a particular type
G	An object group
p_i^u	A spatial object in P_i
w^t, w^o	Type weight and object weight
ς^t, ς^o	A type weight function and an object weight function
$\|S\|$	The number of elements in the set S
ϵ	An error bound
η	A distance bound
γ	A stopping rule used in iterative approaches
$d^E(.,.)$	Euclidean distance between two objects
$d^N(.,.)$	The distance between two objects on the road
$\mathbb{E}$	A family of object sets or groups
$\mathbb{R}$	The search space
$VD(P_i)$	Voronoi diagram of P_i
$Dom(p_j)$	Dominance region of p_j in a Voronoi diagram
OVD	An overlapped Voronoi diagram
OVR	An overlapped Voronoi region
$MOVD$	A minimum overlapped Voronoi diagram
$\mathcal{S}$	A subdivision of a search space
$\mathcal{V}, \mathcal{E}, \mathcal{F}$	A Set of vertices, edges and faces in a subdivision
$\mathcal{W}$	A set of boundary points in a subdivision
$\boxplus, \boxminus$	Object insertion and deletion operators over a family of object sets
$\dotplus, \dot{-}$	Object insertion and deletion operators over MOVDs
$\oplus$	MOVD overlapping operator

utilizing ear elimination, where n indicates the degree of the deleted vertex [11]. Mostafavi et al. improved the method by considering the empty circumcircle property of the Delaunay triangulation [32]. They found that any triangle removed by the method must be empty of vertices except the deleted point. Dinis and Mamede utilized the sweep line technique in his Voronoi diagram updating algorithms, in which an object can be added or deleted in linear time [12].

3 Preliminaries

A spatial object is defined by the triple $< l, w^t, w^o >$, where l is its location in the search space, and w^t and w^o are the type weight and object weight associated with the object. Without loss of generality, w^t and w^o are positive numbers. Smaller values indicate higher preference. $\mathbb{E} = \{P_1, ..., P_n\}$ denotes a family of object sets, where $P_i = \{p_i^1, ..., p_i^m\}$ denotes a set of objects of a particular type. $G = \{p_1^u, ..., p_n^v\}$, where $p_1^u \in P_1, ..., p_n^v \in P_n$, denotes an object group, in which the objects are in different types. ς^t and ς^o are monotonic weight functions applied to *type weight* and *object weight*. Notations used in this paper are summarized in Table 1.

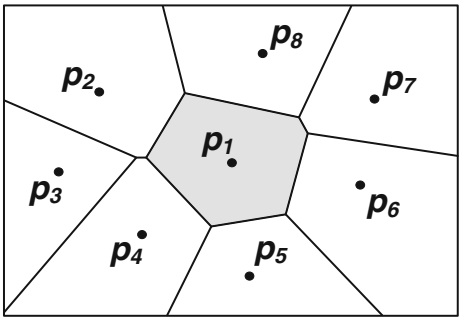


Fig. 3 An example of ordinary Voronoi Diagrams

3.1 Voronoi diagram

3.1.1 Ordinary Voronoi diagram

Given a set of objects $P_i = \{p_i^1, ..., p_i^m\}$, the ordinary Voronoi diagram $VD^O(P_i)$ is defined as a collection of dominance regions $\{Dom^O(p_i^u) \mid p_i^u \in P_i\}$, each of which is dominated by an object p_i^u. All locations in $Dom^O(p_i^u)$ are closer to p_i^u than other objects. $d^E(.,.)$ denotes the distance between two points in Euclidean space.

$$Dom^O(p_i^u) = \{\, l \mid d^E(l, p_i^u.l) \;\leq\; d^E(l, p_i^v.l), u \neq v, p_i^u, p_i^v \in P_i\} \tag{1}$$

Figure 3 shows an example of ordinary Voronoi diagrams, which is generated by eight objects (generators) in Euclidean space. The dominance region of p_1 is highlighted by the shaded polygon. By the properties of Voronoi diagrams, p_1 is closer to any object in the shaded polygon than other generators (e.g., p_2).

An incremental updating method for Voronoi diagrams was proposed to avoid high cost of Voronoi diagram re-generation if only a small number of objects are added to or removed from the initial object set [21]. Figure 4 shows an example of inserting a new generator into an ordinary Voronoi diagram. An object p' in the dominance region of p_1 ($Dom(p_1)$) is added to the object set P. We observe that the bisector line of p_1 and p' intersects with the boundary of $Dom(p_1)$ at two points b_1 and b_4. The bisector line decomposes $Dom(p_1)$ into two sub-regions, which will become $Dom(p_1)$ and a part of $Dom(p')$ in the new Voronoi diagram. In addition, if the bisector line is extended at one end, say b_1, the line will go into $Dom(p_6)$. Then, the bisector line of p' and p_6 can be created to specify the boundary of $Dom(p_6)$ and $Dom(p')$ in the new Voronoi diagram. The process continues until $Dom(p')$ is produced. Any boundary inside $Dom(p')$ is removed in the process. The details of the

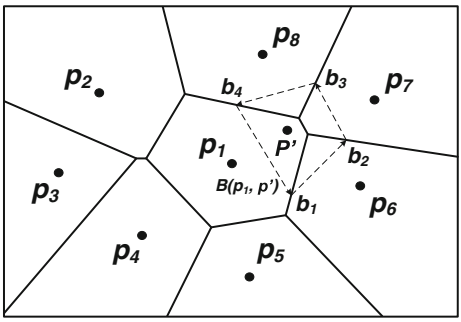


Fig. 4 An example of object insertion over a Voronoi diagram

deletion method can be found in [35]. The computational complexity of the insertion and deletion operations over Voronoi diagrams is linear to the number of neighbor Voronoi cells of inserted/deleted objects in average cases. The insertion and deletion methods can also be easily applied to weighted Voronoi diagrams.

3.1.2 Weighted Voronoi diagram

In a weighted Voronoi diagram, generators have different weights reflecting their variable properties. Given a set of objects $P_i = \{p_i^1, ..., p_i^m\}$ and a weight function ς, the dominance regions are measured by weighted distance. The generation methods for ordinary or weighted Voronoi diagrams can be found in [2, 35].

$$\begin{aligned} VD^W(P_i) = &\quad \{Dom^W(p_i^u) \mid p_i^u \in P_i\} \quad \text{where} \\ Dom^W(p_i^u) = &\{ l \mid \varsigma(\, d(\, l,\ p_i^u.l),\ p_i^u.w^o\,) \leq \varsigma(\, d(\, l,\ p_i^v.l),\ p_i^v.w^o\,), u \neq v,\ p_i^u, p_i^v \in P_i\} \end{aligned} \tag{2}$$

3.2 Fermat-Weber point

Given a point group $G = \{p_1^u, ..., p_n^v\}$ in a d-dimensional space $\mathbb{R}^d$, the Fermat-Weber point is the point q which minimizes the following cost function [5]:

$$c(q, G) = \sum_{p_i^s \in G} p_i^s.w^t \times d(q, p_i^s.l) \tag{3}$$

The point exists for any point set and is unique except in the event that all the points lie on a single line [25]. In the non-collinear case, the cost function is strictly convex [45].

The solution to the three-point Fermat-Weber problem has been proposed in [27]. In the collinear case of any point set, an optimal point can be found in linear time [5]; however, to the best of our knowledge, if the number of points is greater than three, no exact solution has been reported for non-collinear cases. Instead, an iterative approach is used as an approximate solution proposed in [45, 48]. This approach converges monotonically to the unique optimal location during iterations.

The iterative approach starts with an arbitrary location q_0 ($q_0 \notin G$) in $\mathbb{R}^d$. In each iteration, a new location $q_i = f(q_{i-1}, G)$ is produced based on a location q_{i-1} found before the iteration. According to the monotonic convergence property, q_i is closer to the Fermat-Weber point than q_{i-1}; hence, theoretically, the Fermat-Weber point is located at $\lim_{n\to\infty} f^n(q_0, G)$, which indicates a location obtained after infinite iterations. The function f is described below.

$$f(q, G) = \begin{cases} \sum_{p_i^s \in G} \{g_i^s(q) \times p_i^s.l\} & \text{if } q \notin G \\ q & \text{Otherwise} \end{cases} \tag{4}$$

where

$$g_i^s(q) = \frac{p_i^s.w^t}{d(q, p_i^s.l)} \times \left\{ \sum_{p_{i'}^{s'} \in G} \frac{p_{i'}^{s'}.w^t}{d(q, p_{i'}^{s'}.l)} \right\}^{-1} \tag{5}$$

Three stopping rules for the iterative method are widely adopted. Üster and Love developed a generalized bounding method, by which the result is limited within a specified rectangular distance to the optimal location [44]. Verkhovsky and Polyakov adopted the

difference of the costs between two successive iterations as the stopping rule in their experiments [46]. Setting an acceptable deviation from the cost of the optimal location as the stopping rule is widely used in applications [38]. For example, given an error bound ϵ, the location after the n^{th} iteration l^n, and the optimal location l^∞, the iteration procedure will stop when

$$\frac{c(l^n,G)-c(l^\infty,G)}{c(l^\infty,G)} \leq \epsilon \tag{6}$$

is satisfied, where $c(l^\infty, G)$ is approximated by a lower bound of the cost at l^n:

$$lb(l^n) = \sum_{k=1}^{d}\left(\min_{x}\left(\sum_{p_i^s \in G} p_i^s.w^t \frac{|l^n.x_k - p_i^s.l.x_k||x-p_i^s.l.x_k|}{d(l^n,p_i^s.l)}\right)\right) \tag{7}$$

4 Problem definition

4.1 Definition of multi-criteria optimal location query (MOLQ)

4.1.1 Weighted distance of two points

Given a point q, a spatial object p, a type weight function ς^t, and an object weight function ς^o, weighted distance considers the distance between two points $d(.,.)$ and the weights of p. The formal definition is as follows:

$$WD(q,\ p,\ \varsigma^t,\ \varsigma^o) = \varsigma^t(\ \varsigma^o(\ d(q,\ p.l),\ p.w^o\ ),\ p.w^t) \tag{8}$$

Here, d(.) is the distance between two locations in Euclidean spaces.

4.1.2 Weighted distance from a query point to an object group

Given a point q, an object group $G = \{p_1^u, ..., p_n^v\}$, a type weight function ς^t, and object weight functions $\sigma = \{\varsigma_1^o, ..., \varsigma_n^o\}$, we define the weighted distance from q to G as the sum of WD(q, p_i^s, ς^t, ς_i^o), where $p_i^s \in G$, $\varsigma_i^o \in \sigma$. The formal definition is

$$WGD(q, G, \varsigma^t, \sigma) = \sum_{p_i^s \in G, \varsigma_i^o \in \sigma} WD(q, p_i^s, \varsigma^t, \varsigma_i^o) \tag{9}$$

4.1.3 Minimum weighted distance from a query point to object groups

Given a point q, a family of object sets $\mathbb{E} = \{P_1, ..., P_n\}$, a type weight function ς^t, and object weight functions $\sigma = \{\varsigma_1^o, ..., \varsigma_n^o\}$, we define the minimum weighted distance from q to object combinations of $\mathbb{E}$ as:

$$MWGD(q, \mathbb{E}, \varsigma^t, \sigma) = \min(\{WGD(q, G, \varsigma^t, \sigma) \mid G \in P_1 \times ... \times P_n\}) \tag{10}$$

4.1.4 Multi-criteria optimal location query (MOLQ)

Given a family of object sets $\mathbb{E} = \{P_1, ..., P_n\}$, a type weight function ς^t, and object weight functions $\sigma = \{\varsigma_1^o, ..., \varsigma_n^o\}$ where ς_i^o is applied to an object $p_i^u \in P_i$, the purpose of the query is to find an optimal location l in the search space $\mathbb{R}$ that minimizes $MWGD(l, \mathbb{E}, \varsigma^t, \sigma)$. There is no candidate location provided for the query.

$$\begin{aligned} &MOLQ(\mathbb{E}, \varsigma^t, \sigma) = l, \quad \text{where } l \text{ satisfies the condition} \\ &MWGD(l, \mathbb{E}, \varsigma^t, \sigma) = \min(\{MWGD(l', \mathbb{E}, \varsigma^t, \sigma) \mid l' \in \mathbb{R}\}) \end{aligned} \tag{11}$$

In the example of MOLQ shown in Fig. 1, the query receives three object sets $\mathbb{E} = \{P_{School}, P_{Supermarket}, P_{Gasstation}\}$. Multiplicatively-based functions are used as type and object weight functions. The weighted distance between a location q and an object p can be calculated as $WD(q, p, \varsigma^t, \varsigma^o) = d(q, p.l) \times p.w^o \times p.w^t$. An object group is an object set containing a school, a supermarket, and a gas station. Equation 9 represents the weighted distance from a location q to all objects in an object group. Equation 10 finds an optimal object group from $\{P_1 \times ... \times P_n\}$, which minimizes the weighted distance from q to an object group. MOLQ (11) aims to find an optimal location and an object group, which minimize the weighted distance from the location to all objects in the object group. In the example, Community 3 is the best location in the search space, because the distance from Community 3 to the object group $G = \{School\ 2, Supermarket\ 2, Gas\ Station\ 2\}$ is shorter than the distance from any other location to any object group.

4.2 Definition of multi-criteria optimal location query updating problem

To define MOLQ updating problem, we first introduce object insertion and deletion operations over object sets as two types of object updating operations over a family of object sets. Then, given a family of object sets, $\mathbb{E} = \{P_1, ..., P_n\}$, the query result $MOLQ(\mathbb{E}, \varsigma^t, \sigma)$, and a set of changes on $\mathbb{E}$, the MOLQ updating problem can be defined as a process that finds the result of MOLQ after the changes have been applied to $\mathbb{E}$. The changes on $\mathbb{E}$ are abstracted by a set of object insertion or deletion operations on $\mathbb{E}$. Any updates on a particular object (e.g., the changes in its location, type weight, or object weight) are equivalent to deleting the object from $\mathbb{E}$ and then adding it back with new attributes.

4.2.1 Object updating to a family of object sets

Object insertion operation Given a family of object sets $\mathbb{E} = \{P_1, ..., P_n\}$ and an object q ($\nexists\ P_i \in \mathbb{E}, q \in P_i$), we assume there exists an object set $P_j \in \mathbb{E}$, in which the objects are in the same type with q. Then we define the process of inserting q to $\mathbb{E}$ as follows:

$$\mathbb{E}' = \mathbb{E} \boxplus \{q\} = \{P_1, ..., P_j \cup \{q\}, ..., P_n\} \tag{12}$$

If P_j does not exist, then q is the only object in its type. This is a case of MOLQ overlapping, in which the MOLQ of {q} is overlapped with the MOLQ of $\mathbb{E}$.

If an object set $Q = \{q_1, ..., q_k\}$ is given to insert into $\mathbb{E}$, the insertion process is equivalent to sequentially inserting each object in Q to $\mathbb{E}$. Note that there must exist one and only one object set in $\mathbb{E}$, which contains objects in the same type with the newly inserted object. Objects in Q can be in different types. We define the process that inserts an object set Q to $\mathbb{E}$ as follows ($\boxplus$ is left associative):

$$\mathbb{E}' = \mathbb{E} \boxplus Q = \mathbb{E} \boxplus \{q_1\} \boxplus ... \boxplus \{q_k\} \tag{13}$$

Object deletion operation The object deletion is an inverse operation of the object insertion $\boxplus$. Given a family of object sets $\mathbb{E} = \{P_1, ..., P_n\}$ and an object $p_i^k \in P_i$, the object deletion operation that removes p_i^k from $\mathbb{E}$ is defined as follows:

$$\mathbb{E}' = \mathbb{E} \boxminus \{p_i^k\} = \{P_1, ..., P_i \setminus \{p_i^k\}, ..., P_n\} \tag{14}$$

Deleting a set of objects $Q = \{q_1, ..., q_k\}$ from $\mathbb{E}$ can be completed by removing each object in Q from $\mathbb{E}$. Every objects in Q must be contained by an object set in $\mathbb{E}$. Objects in

Q can be in different types. The formal definition of deleting Q from $\mathbb{E}$ can be presented as ($\boxminus$ is left associative):

$$\mathbb{E}' = \mathbb{E} \boxminus Q = \mathbb{E} \boxminus \{q_1\} \boxminus ... \boxminus \{q_k\} \tag{15}$$

4.2.2 Multi-criteria optimal location query updating problem

We assume that the MOLQ has been addressed over a family of object sets. Each object is assigned with an object weight and a type weight. However, for any reasons, there are changes applied to a small number of objects, and the MOLQ is required to be re-evaluated over the updated object sets. Thus, the multi-criteria optimal location query updating problem can be defined as follows: given a set of object sets $\mathbb{E} = \{P_1, ..., P_n\}$, a type weight function ς^t, and object weight functions $\sigma = \{\varsigma_1^o, ..., \varsigma_n^o\}$, where ς_i^o is applied to an object $p_i^u \in P_i$, let $l = MOLQ(\mathbb{E}, \varsigma^t, \sigma)$ be the answer to the MOLQ query, Q be a set of objects updated to $\mathbb{E}$, then the multi-criteria optimal location query updating problem is to find an optimal location l', which minimizes the total weighted distance from l' to one object in each type after updating Q to $\mathbb{E}$.

If Q is inserted into $\mathbb{E}$, then

$$\begin{aligned} &MOLQ(\mathbb{E} \uplus Q,\ \varsigma^t,\ \sigma) = l', \quad \text{if } l' \text{ satisfies the condition} \\ &MinD_G^W(l',\ \mathbb{E} \uplus Q,\ \varsigma^t,\ \sigma) = min(\{MinD_G^W(l'',\ \mathbb{E} \uplus Q,\ \varsigma^t,\ \sigma) \mid l'' \in \mathbb{R}\}) \end{aligned} \tag{16}$$

If Q is deleted from $\mathbb{E}$, then

$$\begin{aligned} &MOLQ(\mathbb{E} \boxminus Q,\ \varsigma^t,\ \sigma) = l', \quad \text{if } l' \text{ satisfies the condition} \\ &MinD_G^W(l',\ \mathbb{E} \boxminus Q,\ \varsigma^t,\ \sigma) = min(\{MinD_G^W(l'',\ \mathbb{E} \boxminus Q,\ \varsigma^t,\ \sigma) \mid l'' \in \mathbb{R}\}) \end{aligned} \tag{17}$$

It is worth noting that any change on an object is equivalent to deleting the object from $\mathbb{E}$ and adding it back with new attributes. The object insertion operation only applies to Q if, $\forall q_i \in Q$, there must exist an object set $P_j \in \mathbb{E}$, which contains objects in the same type of q_i. The object deletion operation only applies to Q if every object in Q must be in an object set of $\mathbb{E}$. The query result over updated object sets could be l or a better location.

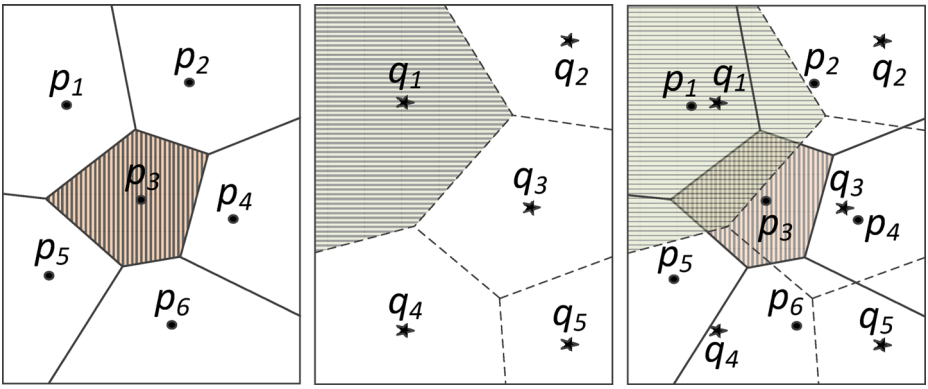
5 OVD and MOVD models

Before describing our MOVD-based solutions, we will first introduce the OVD and MOVD models. In this section, we start with a simple OVD example which provides a basic understanding of the model. Then, we formally define OVD and Minimum OVD (MOVD) and systematically analyze their properties, which not only highlight the difference from and relationship with Voronoi diagrams, but also provide correctness analyses of our MOVD-based solutions. More OVD/MOVD properties will be provided in Appendix A.1-A.3.

We use the OVD model in Euclidean space as an example for better illustration in this paper; however, the model can be easily extended to road networks or other search spaces.

5.1 An OVD example

Figure 5a and b display two ordinary Voronoi diagrams generated by schools and supermarkets, respectively. The shaded areas in the figures are dominance regions of generators p_3 and q_1. Figure 5c shows an OVD that overlaps the two ordinary Voronoi diagrams. Apparently, the OVD is comprised of a number of overlapped regions, each of which is



(a) Schools (b) Supermarkets (c) An OVD

Fig. 5 Ordinary Voronoi diagrams and OVDs in Euclidean space

generated by overlapping two ordinary Voronoi polygons. For example, the doubly shaded area in Fig. 5c is the overlapped region in both shaded regions of two ordinary Voronoi diagrams. According to the properties of Voronoi diagrams, p_3 and q_1 are the closest school and supermarket to any locations in the doubly shaded region.

As an introductory example, all schools and supermarkets are assumed to be of equal weight in Fig. 5. An example of assigning different weights to objects in an OVD will be presented in Fig. 8 in Section 6.

5.2 Overlapped Voronoi diagram definition

5.2.1 Overlapped Voronoi diagram (OVD)

Given a family of object sets $\mathbb{E} = \{P_1, ..., P_n\}$ and a set of Voronoi diagrams $\mathbb{V} = \{VD(P_i) \mid P_i \in \mathbb{E}\}$, where $VD(P_i)$ can be either an ordinary or a weighted Voronoi diagram generated by P_i in the search space $\mathbb{R}$, Overlapped Voronoi Diagram (OVD) is a set of Overlapped Voronoi Regions (OVR),

$$OVD(\mathbb{E}) = \{OVR_j \mid 1 \leq j \leq t\} \tag{18}$$

where OVR_j is

$$OVR(p_1^u, ..., p_n^v) = \{l \mid l \in Dom(p_1^u), ..., l \in Dom(p_n^v), p_1^u \in P_1, ..., p_n^v \in P_n\} \tag{19}$$

Take Fig. 5c for example, $MOVD(\{P, Q\}) = \{OVR(p_1, q_1), ..., OVR(p_6, q_1), ..., OVR(p_1, q_5), ..., OVR(p_6, q_5)\}$.

In Eq. 18, $t = \prod_{P_i \in \mathbb{E}} |P_i|$, which denotes the number of OVRs in $OVD(\mathbb{E})$. This will be further discussed in Theorem 7.

Property 1 *An OVD may have one or more empty set OVRs (e.g., $OVR_j = \emptyset$).*

By definition, an OVR is the intersection of dominance regions from different Voronoi diagrams. These dominance regions may not overlap each other (see the dominance regions of p_1 in Fig. 5a and q_5 in Fig. 5b). If this is the case, no location falls into both dominance regions, thus their overlapping region is an empty set.

5.2.2 Minimum OVD (MOVD)

A Minimum Overlapped Voronoi Diagram (MOVD) is an OVD in which all empty OVRs have been removed. An OVD is an MOVD if it does not have any empty OVRs. The formal definition of MOVD is:

$$MOVD(\mathbb{E}) = OVD(\mathbb{E}) \setminus \{OVR_j \in OVD(\mathbb{E}) \mid OVR_j = \emptyset\} \tag{20}$$

In the extreme case that $\mathbb{E}$ is an empty set, no Voronoi diagrams overlap, and the search space is not decomposed into subregions. We define this case as:

$$MOVD(\emptyset) = OVD(\emptyset) = \{\mathbb{R}\} \tag{21}$$

6 MOVD-based algorithms in Euclidean space

After introducing the OVD model, we now propose our MOVD-based algorithms for the query in Euclidean space in this section. In particular, we first present a sequential scan combination algorithm as a baseline solution. Then, we illustrate the framework of our MOVD-based solutions in Section 6.2. Two algorithms for OVD overlapping operations, RRB and MBRB, are presented in Sections 6.3 and 6.4, respectively. Finally, we describe a cost-bound approach to optimize the cost of solving a large number of Fermat-Weber problems. In this research we mainly focus on applying the properties of OVD and MOVD models to answer the proposed novel query type. The proposed algorithms primarily rely on main memory for data storage.

6.1 Sequential scan combinations algorithm

One basic algorithm to solve MOLQ is to sequentially check optimal locations of all object combinations. Given $\mathbb{E} = \{P_1, ..., P_n\}$, the optimal locations l's of all combinations $\{p_1^u, ..., p_n^v\}$, where $p_1^u \in P_1, ..., p_n^v \in P_n$, in Euclidean space can be calculated by a Fermat-Weber method. The answer to the query is the best location among these l's. We call this algorithm the *Sequential Scan Combinations* (SSC) algorithm. The computational complexity of SSC is $O(\mu \times \prod_{P_i \in \mathbb{E}} |P_i|)$, where μ denotes the cost of finding the optimal location with a given object combination. The detailed steps of SSC are shown in Algorithm 1.

Algorithm 1 SSC($\mathbb{E}$, ς^t, σ)

```
1: Ubound = ∞
2: l = < 0, 0 >
3: for < p_1^u, p_2^s..., p_n^v > ∈ P_1 × ... × P_n do
4:     Calculate the optimal location l_1 of < p_1^u, ..., p_n^v >
5:     Cost = WGD(l_1, {p_1^u, ..., p_n^v}, ς^t, σ)
6:     if Cost < Ubound then
7:         Ubound = Cost
8:         l = l_1
9:     end if
10: end for
11: return l
```

In Euclidean space, since the computation of SSC is expensive, we can set an upper bound to reduce the complexity of the algorithm by filtering out a portion of combinations whose optimal locations cannot be the answer. For example, two combinations (object groups), G_1 and G_2, will be evaluated sequentially in SSC. We assume the optimal location of G_1 is at l_1. The weighted distance from l_1 to G_1 is denoted by d_1. Before processing $G_2 =< p_1^u, p_2^s..., p_n^v >$, we first set d_1 as an upper bound and calculate the optimal location l_2 of $< p_1^u, p_2^s >$, which costs much less than computing an optimal location of multiple points. If the weighted distance from l_2 to $< p_1^u, p_2^s >$ is greater than d_1, the weighted distance from any location to G_2 must be greater than d_1. Thus, calculating the optimal location of G_2 can be avoided. During SSC processing, the upper bound is initialized to infinity and will be reduced to the total weighted distance of the best solution found so far.

6.2 Framework of the MOVD-based solutions in Euclidean space

Figure 6 illustrates the framework of our solutions in Euclidean space. The inputs of the solution are Point of Interest (POI) data sets ($P_i \in \mathbb{E}$), object weight functions $\sigma = \{\varsigma_1^o, ..., \varsigma_n^o\}$, and a type weight function ς^t. The result is an optimal location of the query in the search space.

In the evaluation system, the query is sequentially processed by three modules. In particular, based on POIs of particular types and the object weight functions, *VD Generator* generates Voronoi diagrams that are the basic MOVDs used in the next step (see Theorem 10). Then, a new MOVD is produced by overlapping the basic MOVDs with *MOVD Overlapper* (see Eq. 44). A significant number of impossible object combinations are filtered out, which reduces the computation cost in the next step. Finally, *Optimizer* sequentially scans OVRs in the new MOVD, finding a locally optimal location in each OVR, and returns the best of these locations as the query result.

Essentially, two solutions are proposed in Fig. 6, illustrated by two paths from the *VD Generator* to the *Optimizer*. The solutions apply either Real Region as Boundary (RRB) or Minimum Bounding Rectangle as Boundary (MBRB) approaches in the *MOVD Overlapper*. The RRB approach provides real boundaries of OVRs in the new MOVD by calculating the overlapping regions, which is expensive if the regions are complex. The MBRB approach

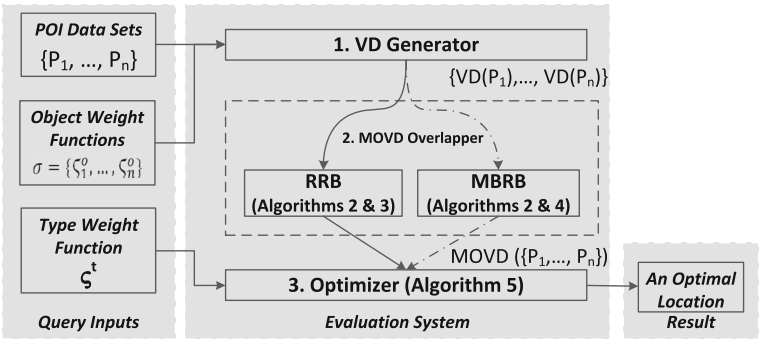


Fig. 6 Framework of the MOVD-based solution in Euclidean space. The paths of RRB and MBRB solutions are indicated by solid and dashed arrows, respectively

can avoid the real region calculation, but it produces false positives that would incur unnecessary calculation while overlapping the next MOVD. Which approach performs better depends on the number and the complexity of MOVDs generated by the *VD Generator*. The two MOVD overlapping approaches will be described in the following two subsections. A cost-bound approach that can reduce the complexity of finding locally optimal locations in *Optimizer* will be presented in Section 6.5. The Voronoi diagram generation approaches used in the *VD Generator* can be found in [2, 16].

6.3 RRB approach

In this subsection, we describe the RRB approach for MOVD overlapping operations. Since basic MOVDs are identical to Voronoi diagrams (see Theorem 10), the generation methods of which have been extensively studied, we will mainly focus on the process of creating an MOVD from two MOVDs. Moreover, MOVDs or Voronoi diagrams are special types of maps or subdivisions. A method that computes the overlay of two subdivisions is presented in [9, 15]. However, the method primarily focuses on the subdivisions that consist of line segments. Extending the method to overlap arbitrary subdivisions is non-trivial. Therefore, the RRB approach is proposed as a general design of MOVD overlapping operation. For a better explanation, the overlapping of two basic MOVDs is illustrated by the simple example in Fig. 7.

A plane-sweep-based algorithm is designed in the RRB approach. As the typical plane sweep approach [2, 16], the RRB approach maintains an event queue and two sweeping statuses. The event queue consists of a number of event points that are the maximum and minimum values of projections of OVRs on the y axis. These maximum and minimum points are called start and end points, which indicate that when the sweeping line arrives at these points, the corresponding OVR starts or ends its intersection with the sweeping line. The event points of both MOVDs are sorted by their y-coordinates in descending order. The sweeping line vertically scans the plane from top to bottom, so that the start point of an OVR will be reached before its end point. The status structures are set up to record OVRs that intersect with the sweeping line. Two status structures are maintained individually and respectively for MOVDs. To efficiently detect if OVRs in the two status structures are overlapped, we also calculate the range (minimum and maximum values) of projections of OVRs

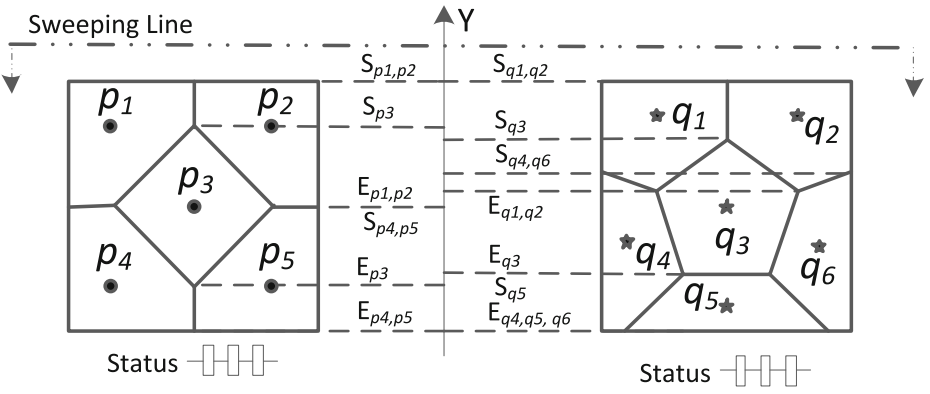


Fig. 7 Overlapping two MOVDs in Euclidean space

on the x axis. The event points and the projection on the x axis are pre-determined before the overlap calculation.

During the sweeping process, when an end point is arrived at, the corresponding OVR is removed from the status structure. When the sweeping line reaches a start point, the corresponding OVR is inserted into the status structure. Moreover, overlapping regions of the new OVR and OVRs in the other status structures are required to be detected. The detection process first identifies potential OVRs, the range of which overlaps with the new OVR on the x axis. Then, the overlapped region of the two OVRs is calculated. The details are described in Algorithms 2 and 3.

The essential idea of the algorithms is that the minimum and maximum values on the x and y axes are an outer boundary of OVR. Two OVRs cannot overlap each other if the area inside their outer boundaries does not overlap. Overlapped outer boundary detection significantly reduces overlapping region calculation by avoiding the overlapping of two OVRs (e.g., regions of p_1 and q_5 in Fig. 7), which are actually far away from each other.

As shown in Algorithm 2, the overlap operation receives two MOVDs as input parameters and produces a new MOVD. From lines 1-2, $Result$, $EventQueue$, $Status$, and $Status'$ are initialized to be empty sets. $Status$ keeps the status for $MOVD(E)$, and $Status'$ for $MOVD(E')$. Then, in lines 3-4, events are inserted into $EventQueue$ and sorted. Finally, from lines 5-12, all events are iteratively handled by Algorithm 3.

Algorithm 2 Overlap($MOVD(E)$, $MOVD(E')$)

```
1:  Result = ∅, EventQueue = ∅
2:  Status = ∅, Status' = ∅
3:  Push events of MOVD(E) and MOVD(E') into EventQueue
4:  Sort(EventQueue)
5:  while ( EventQueue ≠ ∅ ) do
6:      e = EventQueue.pop()
7:      if ( e is from MOVD(E) ) then
8:          EventHandler(e, Status, Status', Result)
9:      else
10:         EventHandler(e, Status', Status, Result)
11:     end if
12: end while
13: return Result
```

Algorithm 3 describes the event handler that receives the following four parameters. e is an event object. $Current$ is the status structure of MOVD from which the event occurs. $Other$ refers to the other status structure. $Result$ is the MOVD produced by the overlap operation. As shown in Fig. 9, an MOVD manages a list of OVRs, each of which is represented as $< region, pois >$, where $region$ maintains the shape of the OVR and $pois$ is a list of objects associated with the OVR. If a start event occurs, the corresponding OVR is first inserted into the $Current$ status. Then, potentially overlapped OVRs in $Other$ are detected by comparing their $Range_X$ with the current OVR. $Range_X$ denotes the range of possible x-coordinates of OVRs. If their $Range_X$ overlap, the overlapped region is calculated in line 5. If the newly generated overlapped region is not empty, a pair of the region and its associated $pois$ will be appended to $Result$. In the second branch, an end event takes place (in line 13) and the corresponding OVR is removed from $Current$.

Algorithm 3 EventHandler(e, $Current$, $Other$, $Result$)

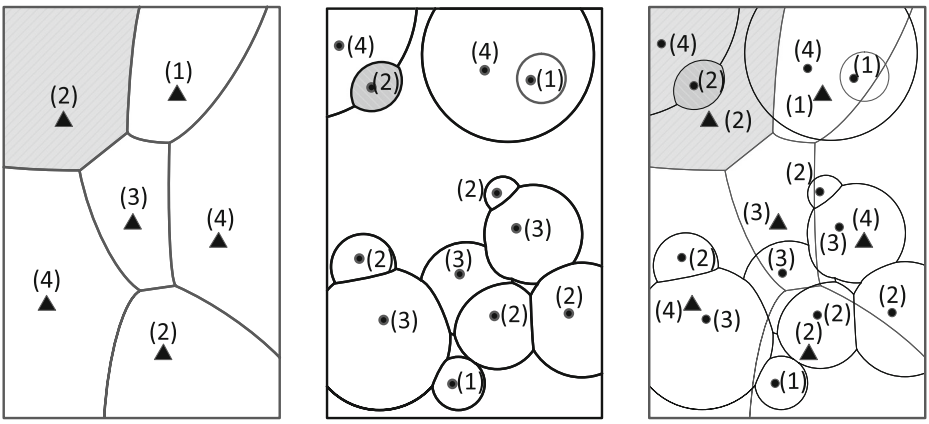
```
1:  if e is a start event then
2:      Insert e.ovr into Current
3:      for ovr ∈ Other do
4:          if Range_X(e.ovr) ∩ Range_X(ovr) ≠ ∅ then
5:              region = e.ovr.region ∩ ovr.region
6:              if region ≠ ∅ then
7:                  pois = e.ovr.pois ∪ ovr.pois
8:                  Result.append(< region, pois >)
9:              end if
10:         end if
11:     end for
12: else
13:     Remove e.ovr from Current /* e is an end event */
14: end if
15: return
```

It is worth noting that a general overlapping approach is not presented; however, the RRB approach can be modified to be a general approach used for the OVD model if line 7 is removed and only $region$ is appended to $Result$ in line 8. $pois$ contains the additional information for our specific query type. Algorithm 3 does not specify any methods for overlapping region calculation in line 5. The reason is that the shape of OVRs in a general model is difficult to predict. The case is worse after overlapping because the OVRs become more complex. Furthermore, overlap methods for regions vary greatly as well. The overlap methods for polygons are different from the ones for circles (Voronoi cells could be circles in multiplicatively Voronoi diagrams, see Fig. 8b). The overlap methods applied in the model cannot be determined until the shapes of regions have been decided. We will discuss this issue in Section 6.4.

The RRB approach is an output-sensitive algorithm, the complexity of which depends on the size of the results, or more exactly the number of OVRs existing in the new MOVD. We use the average size of MOVDs in the analysis instead of the number of objects, the number of object types, or the complexity of Voronoi diagrams, because (1) the inputs of the RRB approach are two MOVDs; (2) the complexity of MOVDs depends on both the number of Voronoi diagrams overlapped and the complexity of the Voronoi diagrams; (3) Voronoi diagrams may vary greatly due to the variety of sizes of input data sets and the ways of measuring distance. Thus, the computational complexity of operation $\oplus$ in the worst case is

$$4n \times \lg(4n) + 2 \times 2n \times \lg(2n) + \theta \times n^2 = O(n^2) \tag{22}$$

where θ denotes the cost of region overlapping computation and n indicates the number of OVRs in the input MOVDs. Specifically, the first part at the left side of Eq. 22 calculates the cost of sorting all events in order at line 4 of Algorithm 2. There are $4n$ events in total, and sorting them in order takes $4n \times \lg(4n)$ time. Then, the second part at the left side of Eq. 22 indicates the cost of maintaining status structures by inserting/deleting OVRs. There are $2n$ start and end events handled by Algorithm 3. If status structures are organized as a balanced search tree that sorts OVRs in order by their start x-coordinates, inserting or deleting an OVR from the status can be completed in $O(\lg n)$ time. Thus, the total cost of maintaining the status is $2 \times 2n \times \lg(2n)$ as well. If status structures record the start and end x-coordinates of OVRs, a range specified by the points that are either immediately



(a) Additively (b) Multiplicatively (c) An OVD of two weighted Voronoi diagrams.

Fig. 8 Weighted Voronoi diagrams in Euclidean space (the numbers indicate weights)

smaller than the minimum or greater than the maximum x-coordinate of the current OVR can be figured out in $O(\lg n)$ time. The OVRs, whose event points are located at the range, are potentially required to overlap the current OVR. Moreover, let θ be the cost of OVR overlapping computation, the cost of calculating the overlapped regions is $\theta \times n^2$, because there could be n^2 OVRs in the output MOVD.

6.4 MBRB approach

According to the variety of weight functions specified in the query inputs, various Voronoi diagrams are generated by the *VD Generator*. In addition to the ordinary Voronoi diagrams, two typical weighted Voronoi diagrams are displayed in Fig. 8. The generation methods of additively and multiplicatively Voronoi diagrams have been presented in [4, 13, 28, 33].

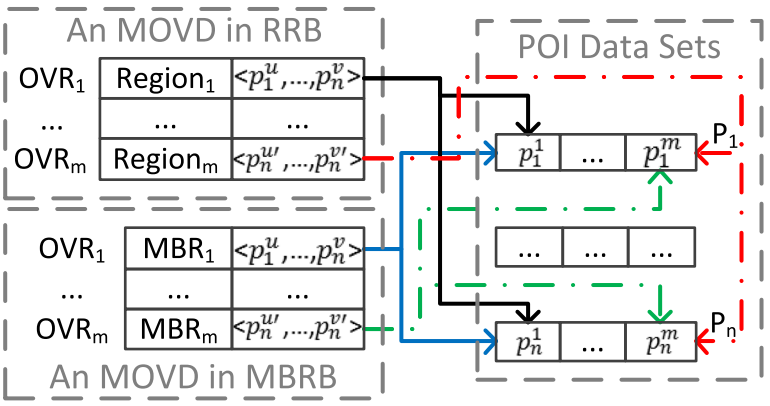


Fig. 9 Data structure

More practical Voronoi diagrams, such as network Voronoi diagrams, can be found in [2, 35].

Although the generation methods of weighted Voronoi diagrams have been extensively studied, efficiently maintaining the shape of OVRs is difficult since they are not in regular shapes. In general, their boundaries have to be modelled by a number of curves. More importantly, overheads of overlapping region calculation would be highly expensive due to the complexity of boundary representation.

Algorithm 4 MBRBHandler(e, $Current$, $Other$, $Result$)

```
1:  if e is a start event then
2:      Insert e.ovr into Current
3:      for ovr ∈ Other do
4:          if Range_X(e.ovr) ⋂ Range_X(ovr) ≠ ∅ then
5:              mbr = e.ovr.MBR ⋂ ovr.MBR
6:              pois = e.ovr.pois ⋃ ovr.pois
7:              Result.append(< mbr, pois >)
8:          end if
9:      end for
10: else
11:     Remove e.ovr from Current /* e is an end event */
12: end if
13: return
```

To overcome this difficulty, we propose the MBRB approach that combines Algorithm 2 with an alternative event handler, MBRBHandler, for the overlap operation. The MBRB approach is motivated by an observation that the shapes of OVRs are not used in *Optimizer*. Instead, the POI locations and their weights are the criteria for optimal location selection; therefore, we set the Minimum Bounding Rectangles (MBR) of OVRs as their shapes in this approach. Two OVRs will be treated as overlapped if their MBRs are overlapped. This approach is able to significantly reduce the cost of the overlap operation by simplifying boundary maintenance and avoiding real region overlapping calculation (line 5 in Algorithm 3 is replaced by line 5 in Algorithm 4); however, the approach suffers from the issue that unnecessary OVRs (false positives which are not really overlapped) would be appended to the new MOVD.

The data structure used in MBRBHandler is shown in Fig. 9. An OVR is indicated as $<MBR, pois>$, where an MBR is comprised of minimum and maximum points on the x and y axes, and $pois$ is a list of objects associated with the OVR. The MBRBHandler is described in Algorithm 4. In particular, the new MOVD ($Result$) is initialized to be an empty set in Algorithm 2. When a start event occurs, the MBRBHandler only detects whether two MBRs are overlapped. If this is the case, the MBRs are overlapped and the objects associated with the two OVRs are merged. The new OVR is appended to $Result$. The final branch remains unchanged.

Compared to the RRB approach, the complexity of region overlapping θ decreases in constant time, but the size of output I increases, the performance impact of which is difficult to evaluate. The upper bound of I is n^2; therefore, the complexity of the MBRB approach becomes $O(n^2)$ in the worst case.

It is worth noting that the MBRB solution is correct because the results of the MBRB approach are a "superset" of the results of the RRB approach. First, given any element

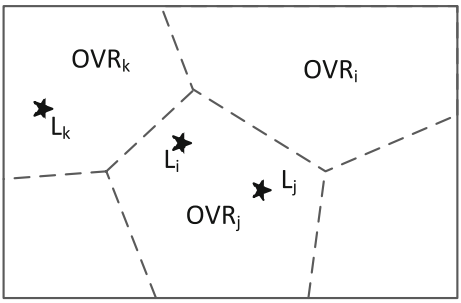


Fig. 10 Optimal locations

$< region, pois >$ generated by the RRB approach, there must be an element $< mbr, pois >$ in the result set of the MBRB solution, where mbr contains all locations in $region$. The MBRB solution does not discard any location from the search space of the RRB solution. Second, the MBRB solution preserves all object groups ($pois$) generated by the RRB solution. If two regions are overlapped, their MBRs must overlap each other, and their associated object group is output by the MBRB solution.

Moreover, the basic principle of our solutions is that the search space is decomposed into a number of OVRs, in which a locally optimal location is found by *Optimizer*; however, the shapes of OVRs are not calculated in the MBRB approach. How does the MBRB solution determine an optimal location in an OVR?

The MBRB solution does not limit the locally optimal location in a particular OVR. Instead, we look for it in the entire search space. As shown in Fig. 10, if an optimal location L_k is found in OVR_k, L_k will undoubtedly be appended to the candidate list. If the optimal location L_i is outside of OVR_i, according to Theorem 8, L_i must be located in another OVR, for example OVR_j, which must have an optimal location L_j. L_j must be identical or better than L_i. Appending both of them to the candidate list does not change the global optimum since only the best one will be returned as the query result. Thus, appending L_i to the candidate list does not change the global optimum.

6.5 A cost-bound approach in optimizer

An optimal location q that minimizes $MWGD(q, \mathbb{E}, \varsigma^t, \sigma)$ is found in the third step of the proposed framework. The framework does not specify a weight function for type weight calculation; however, we mainly focus on a multiplicatively-based weight function, which is one of the practical methods used in real applications. For example, the residential location selection problem displayed in Fig. 1 uses a multiplicatively-based weight function. If other weight functions are required in queries, a specific algorithm in the *Optimizer* module is needed (See Fig. 6). The proposed cost-bound approach that utilizes the Fermat-Weber techniques can be used for cases of additively-based and multiplicatively-based weight functions, because the case of the additively-based weight function is a simplified case of the multiplicatively-based weight function, in which all type weights are fixed at 1.

If the type and object weight functions can be combined and represented by a multiplicatively-based weight function (e.g., applying multiplicatively-based weight functions to both type and object weights), the problem of finding an optimal location in each OVR in Euclidean space is converted into a typical Fermat-Weber problem in a d-dimensional space. The objects associated with OVRs are the points in the Fermat-Weber problem. The weights of the points are specified by the type weight function ς^t. The

object weights are integrated into the distance from locations to the points. As mentioned in Section 3.2, the problem has been solved theoretically. The optimal location in three-point cases and multiple-collinear-point cases can be found in constant and linear time, respectively. An approximate iterative approach has been proposed for other cases [48].

In the RRB and MBRB approaches, we observe that a large number of OVRs will be created by *MOVD Overlapper* (see Theorem 7). The number of the Fermat-Weber problems increases rapidly when the number of objects grows. A basic approach is to sequentially calculate the optimal locations of these Fermat-Weber problems and select the best one as the query result; however, applying the iterative method to the Fermat-Weber problems is very expensive. Therefore, we propose a cost-bound approach in which an optimal cost is set as a global lower bound. During the processing of a Fermat-Weber problem, a local lower bound of the cost in each iteration will be calculated. If the local lower bound is greater than the global lower bound, no matter how many iterations will be processed, its local optimal cost cannot be better than the global lower bound. Thus the following iterations can be avoided, even though the stopping condition has not been satisfied.

Algorithm 5 CostBoundApproach($\mathbb{E}$, ς^t, σ, γ)

1: $Cbound = \infty, l = < 0, 0 >$
2: **for** $G_i \in \mathbb{E}$ **do**
3: Initialize l_i to the center of G_i
4: **if** $|G_i| = 3$ or G_i is a collinear case **then**
5: Calculate the optimal location l_i of G_i
6: **else**
7: Let $G_i =< p_1^u, p_2^s, ..., p_n^v >$
8: Calculate the optimal location l' of $< p_1^u, p_2^s >$
9: **if** WGD($l', \{p_1^u, p_2^s\}, \varsigma^t, \sigma$) $> Cbound$ **then**
10: Continue
11: **end if**
12: **repeat**
13: l_i = f (l_i, G_i) /*Iterating, see Eq. 4*/
14: $Lbound$ = lb(l_i) /* see Eq. 7 */
15: **until** γ is satisfied or $Lbound \geqslant Cbound$
16: **end if**
17: $Cost$ = WGD($l_i, G_i, \varsigma^t, \sigma$)
18: **if** $Cbound > Cost$ **then**
19: $Cbound = Cost$
20: $l = l_i$
21: **end if**
22: **end for**
23: **return** l

As shown in Algorithm 5, the proposed cost-bound approach receives a set of object groups $\mathbb{E} = \{G_1, ..., G_n\}$, a type weight function ς^t, object weight functions σ, and a stopping condition γ. The distance from a location to points is calculated by their Euclidean distance and σ. The number of points in the Fermat-Weber problems ($|G_i|$, $1 \leq i \leq n$) is unnecessarily fixed. In particular, the global lower bound, $Cbound$, is initialized to infinity (in line 1) and reduced to the minimum cost of the optimal location found so far (in line 19). The

algorithm sequentially checks the Fermat-Weber problems, each of which have a local optimal location found in lines 4-16. In the branch of the iterative method inside the loop, an optimal location of the first two points in G_i is first detected in lines 8-11. If a better result of G_i potentially exists, a local lower bound is calculated in each iteration in line 14. If the local lower bound is greater than $Cbound$, the iteration will stop in line 15. The complexity of Algorithm 5 is $O(\mu \times |\mathbb{E}|)$, where μ denotes the average number of iterations processed for Fermat-Weber problems. The Cost-bound approach can also be used in the SSC solution.

6.6 Correctness of RRB and MBRB solutions

Theorem 1 *The proposed RRB and MBRB solutions work correctly to MOLQ queries.*

The correctness of RRB and MBRB solutions are supported by the following points. 1. RRB and MBRB can produce MOVD based on the input datasets because the MOVD overlapping operator $\oplus$ is closed under the MOVD space (See Theorem 19). Two or more MOVD can be overlapped by using $\oplus$. 2. Any MOVD fully covers the entire search space (See theorem 8). RRB and MBRB consider all possible location in the search space during the query evaluation. 3. For a specific OVR, any Fermat-Weber method is used for locating an optimal location as a sub-optimal location; and the global optimal result is returned by comparing all of these sub-optimal locations.

In the worst case, the computational complexity of RRB and MBRB solutions is $O(n^k)$, where n denotes the number of objects in each type, and k denotes the number of object types(See Theorem 7).

7 MOLQ updating algorithms

In this section, we first present a baseline approach to address the Multi-Criteria Optimal Location Query updating problem by incrementally updating Voronoi diagrams. The query result can be obtained by feeding the newly generated Voronoi diagrams into MOVD-based solutions. Then, we develop an MOVD-based incremental updating model in Section 7.2. Differing from the MOVD model that studies MOVD generation methods and properties of the MOVD overlapping operation, the proposed MOVD updating model explores updating operations over MOVDs when there are changes in input object sets. Finally, due to high cost of the MOVD overlapping operation, we propose an advanced solution that incrementally updates MOVDs in Section 7.3. Theoretically, all types of MOVDs can be processed by the proposed object insertion and deletion algorithms; however, we mainly investigate ordinary MOVDs generated by ordinary Voronoi diagrams for simplicity and better illustration in this paper. Our methods can be extended to other types of MOVDs. After a new MOVD is generated, the global optimum can be found by comparing the local optimal locations in updated OVRs with the last query result.

7.1 Voronoi-diagram-based incremental updating approach

Similarly with the MOVD-based solutions, an intuitive approach that utilizes the *MOVD Overlapper* and *Optimizer* can be used to address the MOLQ updating problem. In particular, the approach updates Voronoi diagrams by applying changes to the Voronoi diagrams of initial object sets [21–23, 34, 40]. These Voronoi diagrams are assumed to be preserved as

intermediate results when MOLQ was evaluated using MOVD-based solutions [56]. Then, *MOVD Overlapper* is used to overlap the updated Voronoi diagrams. Either the Real Region as Boundary (RRB) or Minimum Bounding Rectangle as Boundary (MBRB) solution can be employed as an MOVD overlapping approach. Last, *Optimizer* outputs the global optical location as the query result by utilizing Fermat-Weber techniques to sequentially scan all OVRs in the newly generated MOVD.

However, the approach suffers from high cost of MOVD overlapping operations (in *MOVD Overlapper*). First, the cost of MOVD generation is high; the computational complexity of the RRB approach depends on the complexity of Voronoi diagrams and the number of Voronoi diagrams overlapped. Second, although the MBRB approach can avoid real region calculation during the overlapping process, a large number of false positives are produced as a side effect, which may incur significant overhead in both the overlapping process over the next Voronoi diagram and *Optimizer*. The false positives are the regions that cannot have any global optimal location, but output as candidates to the next step. Third, in the final step, *Optimizer* sequentially scans all the regions in the MOVD produced by *MOVD Overlapper*, and selects the best one as the query result. But, in some cases, there are only a limited number of objects updated; most of the local optimal locations are not changed in the new query. Therefore, to reduce the cost of query re-evaluation, we proposed an efficient MOLQ updating solution that updates the MOVD without MOVD overlapping calculation.

7.2 MOVD incremental updating model

Before introducing our advanced solution, we developed an MOVD incremental updating model by investigating MOVDs over two binary operators. The model not only demonstrates the properties of MOVD updating operations, but also provides theoretical basis for our proposed solution.

7.2.1 MOVD updating space

Given a family of object sets $U = \{U_1, ..., U_n\}$ (assume there are n object types in the universal object sets), the MOVD updating space is

$$U(MOVD) = \{MOVD(\mathbb{E}) \mid \mathbb{E} = \{P_1, ..., P_n\}, P_1 \subseteq U_1, ..., P_n \subseteq U_n\} \tag{23}$$

$U(MOVD)$ includes all possible MOVDs generated by any combination of objects in U. For simplicity, $MOVD(\{P_i\})$ is a simple form of $MOVD(\mathbb{E})$ if $\forall P_j \in \mathbb{E}$, $P_i \neq P_j$, $P_j = \emptyset$. By the definition of the MOVD model, $MOVD(\{\emptyset\}) = \mathbb{R}$ (the entire space).

Theorem 2 *The number of MOVDs existing in $U(MOVD)$ is*

$$|U(MOVD)| = \prod_{i=1}^{n} \left(\sum_{j=0}^{|U_i|} \binom{|U_i|}{j} \right) \tag{24}$$

Since the MOVD is unique with a given family of object sets, the number of MOVDs in the updating space is equal to the number of possible object combinations existing in U. Given an object set $P_i \in U_i$, $\sum_{j=0}^{|U_i|} \binom{|U_i|}{j}$ specifies the total number of ways of picking j objects from U_i. If $j = 0$, then $P_i = \emptyset$. If U contains n object sets, then, $|U(MOVD)|$ is equal to the product of the number of object combinations in each object set.

7.2.2 Object insertion and deletion operations over MOVDs

We define two binary operators $\dot{+}$ and $\dot{-}$ for object insertion and deletion operations over MOVDs, respectively. The binary operators receive an MOVD and an object sets inserted or deleted to the MOVD, and return a new MOVD. $\dot{+}$ is an inverse operation of $\dot{-}$. An MOVD can be either the left operand or the right operand of the two operators; however, for simplicity and better explanation, we assume that an MOVD is always used as the left operand, and $\dot{+}$ and $\dot{-}$ are left associative in this paper. $\dot{+}$ and $\dot{-}$ can be defined as follows.

$$\begin{aligned} MOVD(\mathbb{E}) \dot{+} Q &= MOVD(\mathbb{E} \boxplus Q) \\ MOVD(\mathbb{E}) \dot{-} Q &= MOVD(\mathbb{E} \boxminus Q) \end{aligned} \tag{25}$$

Theorem 3 *Changing the order of $\dot{+}$ and $\dot{-}$ does not change the result if updated object sets do not have common objects. Moreover, first performing set operations on updated object sets could reduce the cost of MOVD updating operations by minimizing the size of the operands.*

$$MOVD(\mathbb{E}) \dot{+} Q_i \dot{-} Q_j = MOVD(\mathbb{E}) \dot{-} Q_j \dot{+} Q_i \quad if\ Q_i \cap Q_j = \varnothing \tag{26}$$

$$MOVD(\mathbb{E}) \dot{+} Q_i \dot{-} Q_j = MOVD(\mathbb{E}) \dot{+} (Q_i \backslash Q_j) \dot{-} (Q_j \backslash Q_i) \tag{27}$$

Proof If there exists an object p in $Q_i \cap Q_j$, either side of Eq. 26 is invalid. If $p \in \mathbb{E}$, we cannot perform $MOVD(\mathbb{E}) \dot{+} Q_i$; if $p \notin \mathbb{E}$, $\dot{-}$ is invalid.

If $Q_i \cap Q_j = \emptyset$, let $Q_i = \{q_i^1, ..., q_i^m\}$ and $Q_j = \{q_j^1, ..., q_j^n\}$, by the definitions of object insertion and deletion operations in Eqs. 12 and 14 (and the properties of union and complement operations over sets), we can get

$$\begin{aligned} MOVD(\mathbb{E}) \dot{+} Q_i \dot{-} Q_j &= MOVD(\mathbb{E}) \boxplus \{q_i^1\} \boxplus ... \boxplus \{q_i^m\} \boxminus \{q_j^1\} \boxminus ... \boxminus \{q_j^n\} \\ &= MOVD(\mathbb{E}) \boxminus \{q_j^1\} \boxminus ... \boxminus \{q_j^n\} \boxplus \{q_i^1\} \boxplus ... \boxplus \{q_i^m\} \\ &= MOVD(\mathbb{E}) \dot{-} Q_j \dot{+} Q_i \end{aligned} \tag{28}$$

If $\exists p \in Q_i \cap Q_j$, then

$$MOVD(\mathbb{E}) \dot{+} Q_i \dot{-} Q_j = MOVD(\mathbb{E}) \dot{+} (Q_i \backslash \{p\}) \dot{-} (Q_j \backslash \{p\}) \tag{29}$$

So, Eq. 27 can be proven by applying all objects in $Q_i \cap Q_j$ to Eq. 29. Equation 27 is useful to eliminate shared objects in $Q_i \cap Q_j$ before updating MOVDs. □

Theorem 4 *Sequentially inserting/deleting object sets into/from an MOVD is equivalent to updating them at one time if there is not any object shared by any two object sets.*

$$\begin{aligned} MOVD(\mathbb{E}) \dot{+} Q_1 \dot{+} ... \dot{+} Q_n &= MOVD(\mathbb{E}) \dot{+} (Q_1 \cup ... \cup Q_n) \\ MOVD(\mathbb{E}) \dot{-} Q_1 \dot{-} ... \dot{-} Q_n &= MOVD(\mathbb{E}) \dot{-} (Q_1 \cup ... \cup Q_n) \\ & \quad if\ Q_i \cap Q_j = \varnothing,\ 1 \leqslant i, j \leqslant n,\ i \neq j \end{aligned} \tag{30}$$

This can be easily proven by using the definitions of object insertion and deletion operations in Eqs. 12 and 14 (and the properties of union and complement operations over sets).

Theorem 5 *Closure: the universal MOVD updating space is closed under operators $\dot{+}$ and $\dot{-}$.*

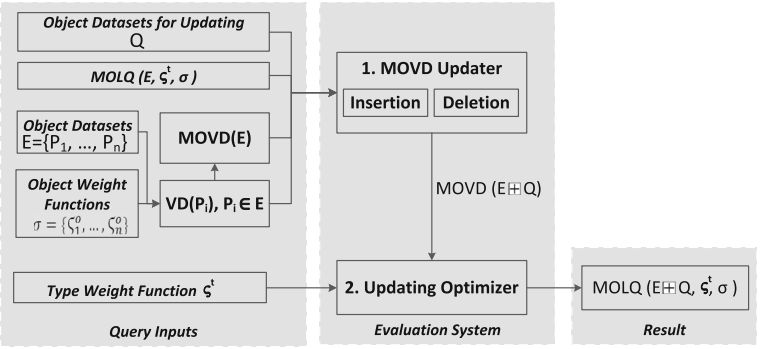


Fig. 11 Framework of MOVD-based incremental updating approach

By definition, the results returned by $\dot{+}$ and $\dot{-}$ are still MOVDs, which are elements of $|U(MOVD)|$. Thus, $|U(MOVD)|$ is closed under $\dot{+}$ and $\dot{-}$.

By combining with the MOVD overlapping operator $\oplus$, the order of the MOVD updating and overlapping operations is not important, because if $MOVD(E_i)$ and $MOVD(E_j)$ are overlapped first, the changes applied to $MOVD(E_i \cup E_j)$ include the changes to both $MOVD(E_i)$ and $MOVD(E_j)$.

Theorem 6 *The relation to the MOVD overlapping operator* $\oplus$. *Let* $Q = \{q_1, ..., q_n\}$, *then, by using the properties of union and complement operations over sets, we can get*

$$\begin{aligned}(MOVD(E_i) \dot{+} Q) \oplus MOVD(E_j) &= (MOVD(E_i \boxplus Q)) \oplus MOVD(E_j) \\ &= (MOVD(E_i \boxplus \{q_1\} ... \boxplus \{q_n\})) \oplus MOVD(E_j) \\ &= MOVD((E_i \boxplus \{q_1\} ... \boxplus \{q_n\}) \cup E_j) \\ &= MOVD((E_i \cup E_j) \boxplus \{q_1\} ... \boxplus \{q_n\}) \\ &= (MOVD(E_i) \oplus MOVD(E_j)) \dot{+} Q\end{aligned} \tag{31}$$

$$\begin{aligned}(MOVD(E_i) \dot{-} Q) \oplus MOVD(E_j) &= (MOVD(E_i \boxminus Q)) \oplus MOVD(E_j) \\ &= (MOVD(E_i \boxminus \{q_1\} ... \boxminus \{q_n\})) \oplus MOVD(E_j) \\ &= MOVD((E_i \boxminus \{q_1\} ... \boxminus \{q_n\}) \cup E_j) \\ &= MOVD((E_i \cup E_j) \boxminus \{q_1\} ... \boxminus \{q_n\}) \\ &= (MOVD(E_i) \oplus MOVD(E_j)) \dot{-} Q\end{aligned} \tag{32}$$

7.3 MOVD-based incremental updating approach

We now propose an advanced MOVD-based incremental updating approach in this subsection. We will first describe the framework of the solution, and then illustrate the object insertion and deletion algorithms. We assume that the MOVD of input object sets $\mathbb{E}$ is preserved by MOVD-based solutions in the process of query evaluation last time. Thus, $MOVD(\mathbb{E})$ and Voronoi diagrams of objects in each type are additional inputs to our proposed solution.

7.3.1 Framework of MOVD-based incremental updating approach

As defined in Section 4.2, the MOLQ updating problem receives a family of object sets $\mathbb{E}$, object weight functions σ, a type weight function ς^t, a set of objects Q needed to update, and the query result, $MOLQ(\mathbb{E}, \varsigma^t, \sigma)$, as inputs, and returns an optimal location after Q has been applied to $\mathbb{E}$. In addition, $MOVD(\mathbb{E})$ and basic Voronoi diagrams $VD(P_i)$, $P_i \in \mathbb{E}$, are additional inputs of our approach. As shown in Fig. 11, we evaluate the MOLQ updating problem in two steps: (1) *MOVD Updater* applies the changes to $MOVD(\mathbb{E})$ by utilizing the MOVD incremental updating model; (2) *Updating Optimizer* sequentially scans OVRs in the new MOVD, and returns the global optimum as the query result. The cost-bound approach that finds the global optimal location by scanning all OVRs in the MOVD can also be used in *Updating Optimizer*; however, unnecessary scans in unchanged OVRs can be avoided in the following two cases: (1) $MOLQ(\mathbb{E} \boxplus Q, \varsigma^t, \sigma) = l'$ (or $MOLQ(\mathbb{E} \boxminus Q, \varsigma^t, \sigma) = l'$) if l' is the best local optimal location in updated OVRs and l' is better than $MOLQ(\mathbb{E}, \varsigma^t, \sigma)$, because in this case, any location in unchanged OVRs cannot be better than l'; (2) Let ovr be the OVR where the query result $MOLQ(\mathbb{E}, \varsigma^t, \sigma)$ locates, if there is no change to ovr (Q does not contain any object in the object group of ovr or that of ovr's neighbor OVRs), then $MOVD(\mathbb{E} \boxplus Q, \varsigma^t, \sigma)$ (or $MOVD(\mathbb{E} \boxminus Q, \varsigma^t, \sigma)$) is either $MOLQ(\mathbb{E}, \varsigma^t, \sigma)$ or the best local optimal location in updated OVRs. In this case, ovr is not changed, $MOLQ(\mathbb{E}, \varsigma^t, \sigma)$ is the optimal location in unchanged OVRs. Thus, the global optimum is the better one between the best location in unchanged OVRs and the one in updated OVRs. In other cases, *Updating Optimizer* has to scan all the OVRs to find the global optimal location.

7.3.2 Object insertion algorithm

Inserting a new object to an MOVD may include adding new OVRs to the MOVD, updating the boundaries of existing OVRs, and removing OVRs from the MOVD. Figure 12 shows an example, in which an object q is inserted into an $MOVD(\{P, Q\})$. We assume that q is in the same type with objects in Q. The green polygon indicates the dominance region of q after q is inserted into $MOVD(\{Q\})$ (which is equivalent to the Voronoi diagram of Q). To efficiently update the MOVD, the fundamental idea of our MOVD insertion algorithm is that we only update the OVRs that overlap the dominance region of q ($Dom(q)$). In particular, all OVRs in the input MOVD can be categorized into three groups: (1) there is

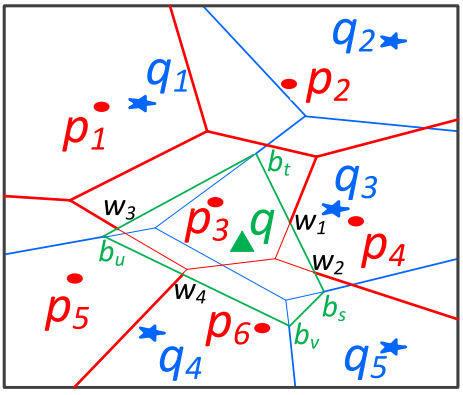


Fig. 12 An example of MOVD insertion operation

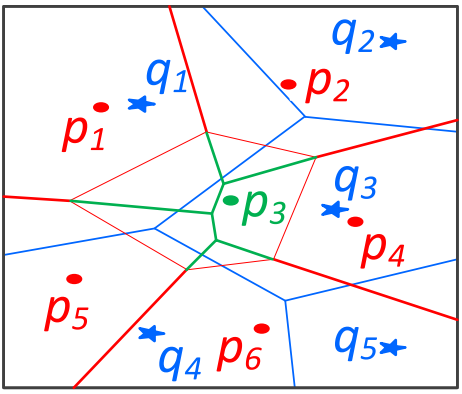


Fig. 13 An example of MOVD deletion operation

no change in the OVRs outside $Dom(q)$ (e.g., $OVR_{<p_1,q_1>}$ remains unchanged during the insertion since it does not overlap the green polygon, $Dom(q)$, in Fig. 12); (2) the OVRs (e.g., $OVR_{<p_3,q_4>}$) inside $Dom(q)$ are removed from the MOVD, because the entire space in the OVRs is dominated by the new object q; (3) boundaries of the OVRs that overlap $Dom(q)$ are updated. When a new object q is inserted into $MOVD(Q)$, the dominance regions of q's neighbor generators are updated. For example, $OVR_{<p_4,q_3>}$ is decomposed into two sub-regions by the line segment from w_1 to w_2. The sub-region inside $Dom(q)$ becomes a new $OVR_{<p_4,q>}$ inserted into the input MOVD, and the other sub-region is $OVR_{<p_4,q_3>}$ with a new boundary.

The details of our MOVD insertion algorithm are presented in Algorithm 6. The algorithm receives an $MOVD(\mathbb{E})$ and a new object q as inputs, and it returns a new $MOVD(\mathbb{E} \boxplus \{q\})$. The new MOVD is abstracted by M in the algorithm. Initially, we find $MOVD(\{Q\})$ and the nearest generator $q_i \in Q$ from q. Then, we create the bisector line of q and q_i (B_{qq_i}), which intersects with $Dom(q_i)$ at two points b_s, b_t. Take Fig. 12 for example, the new object q locates in $Dom(q_3)$, and B_{qq_3} indicates a new boundary of $Dom(q_3)$. Moreover, B_{qq_3} also intersects with $MOVD(\mathbb{E})$ at other points (e.g., B_{qq_3} intersects with the boundary (red line segments) of $Dom(p_4)$ at two points w_1 and w_2). These two points help us to identify OVRs, the boundaries of which have to be updated due to the change of $Dom(q_3)$. With the first bisector line B_{qq_i}, we start generating the dominance region of q by visiting its neighbor generators and creating bisector lines between q and the generators. For example, we iterate the four line segments of the green polygon ($Dom(q)$) in counter clockwise order. Since b_t is on the bisector line of q_3 and q_1, we can easily find q_1 as a new neighbor generator of q, and the bisector line from b_t to b_u. We continue the process until reaching b_s, one end of the first bisector line (from line 5 to 21). In the while loop from line 12 to 21, b always points to one end of the new bisector line (at line 11 and 20), and the loop is terminated when b is equal to b_s at line 12. When each bisector line is visited, we also mark the intersection points with $MOVD(\mathbb{E})$ (e.g., w_1 and w_2) at line 8 and 17. Then, we use these intersection points to update the boundaries of OVRs overlapping with $Dom(q)$ in function OVR_Update at line 22. In OVR_Update, each overlapped OVR is decomposed into two sub-regions ($OVR_{<p_4,q_3>}$ is decomposed into two sub-regions by line from w_1 to w_2); the one inside $Dom(q)$ is appended to an OVR set $OVRs\_In\_Dom\_P$. The details of OVR_Update will be described in Algorithm 7. Third, all the OVRs in $Dom(q)$ are removed from M (at line 25). These OVRs are also kept in $OVRs\_In\_Dom\_P$ (at line 26) because they contain the region of overlapping multiple dominance regions. If these

OVRs are discarded, after obtaining $Dom(q)$, we have to sequentially overlap $Dom(q)$ on the dominance regions of objects in other types, the cost of which could be high. In addition, it is worth noting that the union of the OVRs in $OVRs\_In\_Dom\_P$ is equal to $Dom(q)$. Finally, in the for loop from line 29 to 32, we iterate all OVRs in $OVRs\_In\_Dom\_P$, and replace old objects in object groups of the OVRs with q. These OVRs may share object groups, because they might be separated by the dominance regions of two generators in the old $MOVD(\{Q\})$, but they become sub-regions in $Dom(q)$ after q is inserted. Thus, we check the OVRs in $OVRs\_In\_Dom\_P$, and merge them if necessary (at line 31). After the check, all OVRs are added to M as new OVRs (at line 33). More analysis of the object insertion operation to an MOVD will be provided in Appendix A.4.

Algorithm 6 MOVD_Insert($MOVD(\mathbb{E}), q$)

1: $M = MOVD(\mathbb{E})$
2: $OVRs\_In\_Dom\_P = \varnothing$
3: Find $MOVD(\{Q\})$ where Q contains objects in the same type with q
4: Let $q_i \in Q$ be the nearest generator from q
5: Create the bisector line $B(q, q_i)$;
6: Let b_s and b_t be the two points where $B(q, q_i)$ intersects the boundary of $Dom(q_i)$
7: **if** $B(q, q_i)$ intersects the boundary of OVRs $\in MOVD(\mathbb{E})$ **then**
8: Mark the intersection points on the line segment $L_{b_s b_t}$
9: **end if**
10: Add $L_{b_s b_t}$ to $Dom(q)$
11: $b = b_t$
12: **while** $(b \neq b_s)$ **do**
13: Let q_j be the generator where $Dom(q_j)$ intersects $B(q, q_i)$ at b_t
14: Create the bisector line $B(q, q_j)$;
15: Let b_t and b_u be the two points where $B(q, q_j)$ intersects with the boundary of $Dom(p_j)$
16: **if** $B(q, q_j)$ intersects the boundary of OVRs $\in MOVD(\mathbb{E})$ **then**
17: Mark the intersection points on the line segment $L_{b_t b_u}$
18: **end if**
19: Add $L_{b_t b_u}$ to $Dom(q)$
20: $b = b_u, b_t = b_u, q_i = q_j$
21: **end while**
22: OVR_Update(M, $Dom(q)$, $OVRs\_In\_Dom\_P$)
23: **for** $\forall\ ovr \in MOVD(\mathbb{E})$ **do**
24: **if** ovr is inside $Dom(q)$ **then**
25: Remove ovr from M
26: Append ovr to $OVRs\_In\_Dom\_P$
27: **end if**
28: **end for**
29: **for** $\forall\ ovr \in OVRs\_In\_Dom\_P$ **do**
30: Replace $q_k \in Q$ with q in the object group of ovr
31: Merge ovrs if their object groups are the same
32: **end for**
33: Append all OVRs in $OVRs\_In\_Dom\_P$ to M
34: **return** M

The boundaries of OVRs that overlap with $Dom(q)$ (except for OVRs inside $Dom(q)$) are updated in OVR_Update in Algorithm 7. In particular, the intersection points and two ends of bisector lines on the boundary of $Dom(q)$ had been marked when $Dom(q)$ was created in Algorithm 6. OVR_Update iterates every two consecutive marked points in the counter clockwise order. For each iteration, let m_i and m_j be two marked points, and l be the line segment from m_i to m_j on the boundary of $Dom(q)$, any point d on l must be in the OVR that overlaps $Dom(q)$. So, we use d to find the OVR (at line 5 and 6), and update its boundary with l (at line 8). In addition, the old OVR is decomposed into two sub-regions. The one outside $Dom(q)$ becomes the new region of the OVR; the other region inside $Dom(q)$ is appended to $OVRs\_In\_Dom\_P$ (at line 9). For example, $OVR_{<p_3,q_1>}$ is divided by the line from b_t and w_3. The sub-region above the line becomes the new region of $OVR_{<p_3,q_1>}$, and the other sub-region is inserted into $OVRs\_In\_Dom\_P$.

Algorithm 7 OVR_Update(M, $Dom(q)$, $OVRs\_In\_Dom\_P$)

```
1:  /* Marked points include the intersection points and two ends of bisector lines */
2:  for every two consecutive marked points on the boundary of Dom(q) in counter
    clockwise order do
3:      Let m_i and m_j be two consecutive marked points
4:      Let l be the line segment from m_i to m_j on the boundary of Dom(q)
5:      Pick a random point d on l
6:      Find ovr ∈ M where d ∈ ovr
7:      Use l to decompose ovr into two sub-regions
8:      Update the boundary of the sub-region outside Dom(q) with l
9:      Append the other sub-region inside Dom(q) to OVRs_In_Dom_P
10: end for
11: return
```

7.3.3 Object deletion algorithm

Similarly with the object insertion operation, deleting an object from an MOVD may include adding, deleting, and updating OVRs. For example, we assume that p_3 is deleted from $MOVD(\{P, Q\})$, which is shown in Fig. 13. In the deletion process, the OVRs inside $Dom(p_3)$ are first removed, because all these OVRs are sub-regions of $Dom(p_3)$, which is removed in the updated MOVD. Then, to fill the space of $Dom(p_3)$ in the new MOVD, the neighbor OVRs of $Dom(p_3)$ (e.g., $OVR_{<p_1,q_1>}$ and $OVR_{<p_2,q_1>}$) are visited, and their dominance regions are updated. For every OVR inside $Dom(p_3)$, we also calculate the overlapping regions of the OVR and the boundary of updated $MOVD(\{P\})$, and insert the overlapping regions to the MOVD as new OVRs. In the region overlapping calculation, an OVR could be decomposed into multiple OVRs, because $Dom(p_3)$ will be decomposed into a number of sub-regions by re-calculating the dominance regions of p_3's neighbor generators in P.

The deletion algorithm is described in Algorithm 8. The algorithm receives an $MOVD(\mathbb{E})$ and a deleted object q as inputs, and returns $MOVD(\mathbb{E} \boxminus \{q\})$. The result is abstracted by M in the algorithm. In particular, we first iterate all OVRs inside $Dom(q)$, and delete them from M (at line 5). Before appending the OVRs to a temporary set $Removed\_OVRs$, q is removed from object groups of the OVRs (at line 6). Then, we incrementally update $MOVD(\{Q\})$ by using a Voronoi diagram updating method (at line 9) [35]. The dominance regions of q's neighbor generators are updated in the process. We

also append the updated OVRs in $MOVD(\{Q\})$ to a temporary set $Updated\_OVRs$. In the for loop from line 11 to 17, the overlapping regions of the updated dominance regions and OVRs in $Removed\_OVRs$ are calculated. If any overlapping region is not an empty set, we create a new OVR for the region; the object group of the new OVR is the union of object groups of the two input OVRs (line 12 to 16). Finally, we merge two OVRs in M if their object groups are the same. It is worth noting that only updated OVRs and their neighbor OVRs are visited in the merging process, because other OVRs are not changed. Take an $MOVD(\{P, Q\})$ in Fig. 13 for example, we assume p_3 is deleted from the MOVD. First, $OVR_{<p_3,q_1>}$, $OVR_{<p_3,q_3>}$, and $OVR_{<p_3,q_4>}$ are removed from the MOVD, and kept in $Removed\_OVRs$. Then, $MOVD(\{P\})$ is incrementally updated; green line segments inside $Dom(p_3)$ indicate new boundaries of dominance regions of p_3's neighbor generators (p_1, p_2, p_4, p_5, and p_6). Since $OVR_{<p_3,q_4>}$ is further decomposed into two sub-regions by the new $Dom(p_5)$ and $Dom(p_6)$; we calculate the new overlapping regions at line 13, and append the new OVRs to M at line 15. Object groups of the new OVRs are also updated by replacing p_3 with the corresponding generators. Last, we merge the new OVRs to their neighbor OVRs if their object groups are the same. A sub-region of $OVR_{<p_3,q_4>}$ is merged with $OVR_{<p_5,q_4>}$ after p_3 is deleted.

Algorithm 8 MOVD_Delete($MOVD(\mathbb{E}), q$)

1: $M = MOVD(\mathbb{E})$
2: $Removed\_OVRs = \emptyset$, $New\_OVRs = \emptyset$, $Updated\_OVRs = \emptyset$
3: Q be the object set containing q
4: **for** $\forall$ ovr inside $Dom(q)$ **do**
5: Remove ovr from M
6: Remove q from the object group of ovr
7: Append ovr to $Removed\_OVRs$
8: **end for**
9: Incrementally update $MOVD(\{Q\})$
10: Let $Updated\_OVRs$ to a set of updated OVRs in the updating process at line 9
11: **for** $\forall ovr_1 \in Updated\_OVRs, ovr_2 \in Removed\_OVRs$ **do**
12: **if** $ovr_1 \cap ovr_2 \neq \emptyset$ **then**
13: region = $ovr_1 \cap ovr_2$
14: $os = ovr_1.Objects \cup ovr_2.Objects$
15: Append a new OVR $< region, os >$ to M
16: **end if**
17: **end for**
18: **for** updated OVRs in M **do**
19: Merge two OVRs if their object groups are the same.
20: **end for**
21: **return** M

As we expect, the proposed object insertion and deletion algorithms only process the OVRs that require changes. In general, given an object q inserted into or deleted from an $MOVD(\mathbb{E})$, there are two steps in the two proposed algorithms. We first update $MOVD(\{Q\})$ with q and then apply the changes to $MOVD(\mathbb{E})$ by using the new dominance regions in $MOVD(\{Q\})$. The computational complexity of the process depends on a number of factors, such as the location of q, the size and object distribution of Q, and the OVR density and distribution of $MOVD(\mathbb{E})$. To simplify our analysis, we use I to

denote the average number of OVRs (in $MOVD(\mathbb{E})$), which overlap with the updated dominance regions in $MOVD(\{Q\})$, and C to denote the average number of neighbor OVRs of a given OVR. Then, the computational complexity of the insertion and deletion algorithms are $O(C \times I)$ in average cases, because there are I OVRs inserted into, removed from, or updated in $MOVD(\mathbb{E})$, and each of them is detected to merge with their neighbor OVRs. In for loop from line 11 to 17 in $MOVD\_Delete$ algorithm, the cost of testing the overlapping region of ovr_1 and ovr_2 is the product of size of $Updated\_OVRs$ and $Removed\_OVRs$. Since $Updated\_OVRs$ and $Removed\_OVRs$ maintain the neighbor OVRs of q in $MOVD(\{Q\})$ and the OVRs inside $Dom(q)$ in $MOVD(\mathbb{E})$; the number of OVRs in $Updated\_OVR$ and $Removed\_OVRs$ can be represented by C and I respectively. Thus, the computational complexity of the for loop is also $O(C \times I)$ in average cases. Moreover, in the worst case, all OVRs in $MOVD(\{Q\})$ could be updated in the process, and the computational complexity becomes $O(C \times |MOVD(\mathbb{E})|)$, where $|MOVD(\mathbb{E})| = n^{|MOVD(\mathbb{E})|}$, n denotes the number of objects in object set Q, and $|E|$ denotes the number of object sets in E.

8 Experimental validation

In this section, we evaluate the performance of the OVD model and MOLQ solutions with real-world data sets in Euclidean spaces. We implemented our proposed SSC (See Section 6.1), RRB (See Section 6.3), MBRB (See Section 6.4), VD-IU (See Section 7.1), and MOVD-IU (See Section 7.3) in C++. Specifically, SSC is used as a baseline solution, and RRB and MBRB are two advanced MOVD-based solutions for MOLQ queries. VD-IU and MOVD-IU are a baseline solution and an advanced MOVD-based solution for MOLQ update queries. All data was loaded into the main memory (except explicitly specified) during the execution of the simulations. All the experiments were conducted on a CentOS 6.5 Linux server equipped with two Intel Xeon E5-2660 v2 2.20 GHz processors and 128 GB of memory. All results were recorded after the system model reached a steady state.

In our experiments, the data sets were downloaded from GeoNames.[1] We retrieved the largest five object types, 230,762 streams (STM), 225,553 churches (CH), 200,996 schools (SCH), 166,788 populated places (PPL), and 110,289 buildings ($BLDG$), in the United States. By default, the type weight w^t and object weight w^o are randomly generated from 0 to 10. The multiplicatively-based weight functions are used as ς^t and σ. GPC library[2] is used for polygon overlapping calculation.

8.1 MOLQ in Euclidean space

8.1.1 MOLQ evaluation in Euclidean space

We evaluate the solutions for MOLQ queries with three and four object types that are popular applications in the real world. The type weights are randomly generated from 0 to 10. We use the largest object types, $\mathbb{E} = \{STM, CH, SCH\}$ for the three-type case and $\mathbb{E} = \{STM, CH, SCH, PPL\}$ for the four-type case. The objects are randomly selected from the data sets.

[1] http://www.geonames.org/

[2] http://www.cs.man.ac.uk/~toby/gpc/

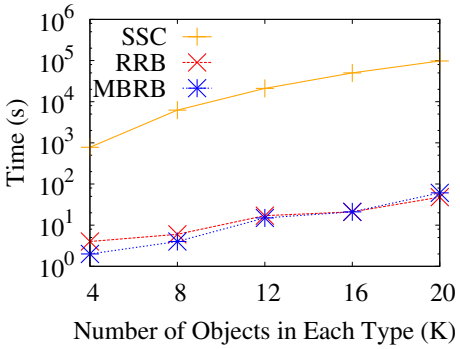


Fig. 14 MOLQ with three object types in Euclidean space

Figure 14 displays the performance of SSC, the proposed RRB and MBRB solutions. The cost-bound approach is used in all the three solutions. As Fig. 14 shows, RRB and MBRB run 24-48 times and 22-51 times faster than SSC, respectively, because they avoid a significant number of object combinations. Overlapping Voronoi diagrams is a process of filtering out combinations that cannot be the closest objects of any location. Another observation is that MBRB takes only 1/2 of the execution time of RRB over datasets of four thousand objects in each type. But when the datasets grow larger, MBRB suffers from more overhead of maintaining and processing false positives. The evidence has been shown in Figs. 17 and 20.

In the query with four object types, only approximate results can be provided by the three approaches. The error bound ϵ is set to be 0.001. Figure 15 shows the execution time of the three solutions, in which the RRB solution has the best performance (14 times faster than SSC and 150% faster than MBRB, on average). Although the execution time of overlapping process in the MBRB approach is slightly shorter than RRB as shown in Fig. 20b, the overhead of maintaining false positives makes MBRB expensive in both overlapping the next Voronoi diagram and Fermat-Weber calculation (finding an optimal location in each OVRs). In the general cases, the overlapping process takes nearly 90% of execution time in the query evaluation.

8.1.2 Cost-bound approach evaluation in Euclidean space

We evaluate the basic (Original) and cost-bound (CB) approaches by varying the number of Fermat-Weber problems and the error bound ϵ. The basic approach sequentially calculates

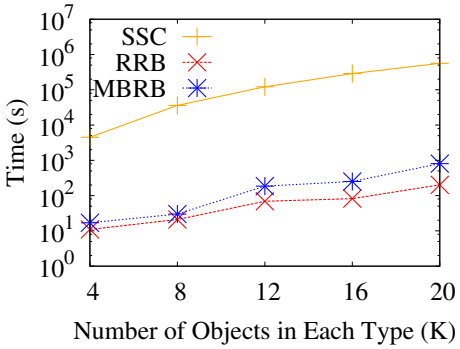


Fig. 15 MOLQ with four object types in Euclidean space

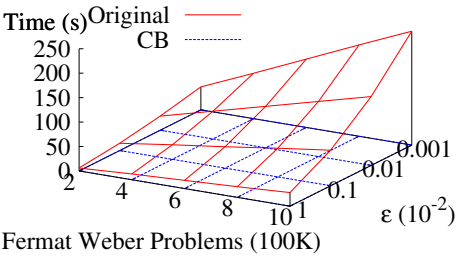


Fig. 16 CB approach evaluation

the optimum locations of all Fermat-Weber problems, and selects the best location for the result. The number of points in each Fermat-Weber problem is fixed to 5. The coordinates and type weights (from 0 to 10) of points are randomly generated. The iterative method for a Fermat-Weber problem will stop when the deviation from the optimal cost is less than the error bound ϵ (see Section 3.2) [38].

Figure 16 displays the execution time of the two approaches. As either the problem size increases or ϵ decreases, the execution time of both approaches rises. Obviously, the growth rate of the original approach is higher than the cost-bound approach because a significant number of unnecessary iterations can be avoided by setting a cost bound, which makes the cost-bound approach more efficient (60 to 517 times faster than the original approach), even though it has to pay extra overhead on lower bound calculation in each iteration.

8.1.3 Overlapping two Voronoi diagrams in Euclidean space

Two overlap approaches, RRB and MBRB, on two ordinary Voronoi diagrams are evaluated with various data set sizes. The Voronoi diagrams are generated by two object sets, which are randomly selected from STM and CH. Their sizes are indicated by the x and y axes in Figs. 17, 18 and 19.

From Fig. 17, we observe that the execution time of the overlapping process in MBRB is shorter than that of RRB. In particular, the speedup of MBRB ranges from 3 times in two 10K data sets to 7.9 times in two 160K data sets. The reason is that the regions of OVRs generated by RRB are determined by real region overlapping calculation (polygon overlapping calculation in this experiment). The complexity of overlapping two polygons is proportional to the number of vertices in the polygons, which is more expensive than the MBR detection (rectangle overlapping calculation) that can be completed in constant time in MBRB. Also, Fig. 18 shows the evidence that due to replacing real regions of OVRs with their MBRs, MBRB generates around 322% more OVRs, on average, than RRB. Two OVRs that are not really overlapped with each other may be determined to be overlapped

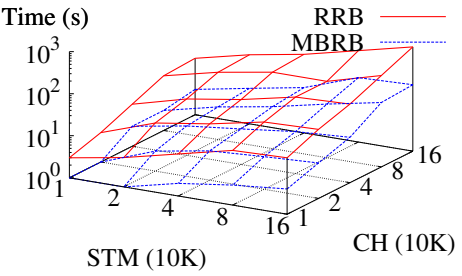


Fig. 17 Execution time

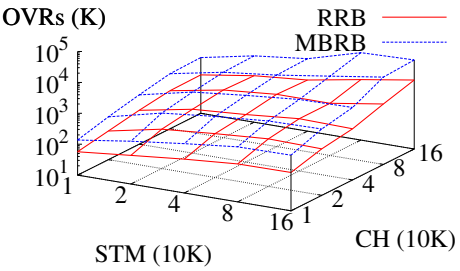


Fig. 18 Number of OVRs

by the MBR detection. However, Fig. 19 shows that the memory consumption of MBRB is very close to that of RRB. Although MBRB generates more OVRs, the regions (MBRs) of which can be represented by two points, all vertices of polygons have to be recorded in RRB. According to Fig. 19, the total number of points managed by the MBRB approach is close to RRB.

8.1.4 Overlapping multiple Voronoi diagrams in Euclidean space

In this experiment, we examine the overlap operation by varying the number of Voronoi diagrams. These Voronoi diagrams are generated by objects randomly selected from $\mathbb{E} = \{STM, CH, SCH, PPL, BLDG\}$. For object type selection, we follow the sequence in $\mathbb{E}$ (i.e., $\mathbb{E} = \{STM, CH\}$ for the two-type case, $\mathbb{E} = \{STM, CH, SCH\}$ for the three-type case, and so on). In addition to performance evaluation, we explore the availability of the overlap operation, which is described by the maximum size of objects in a particular number of object types that can be processed with around 1 GBytes memory. All data is assumed to be loaded into the main memory.

Figure 20a demonstrates the availability of the overlap operation by varying the number of object types. When the number of object types increases, the maximum numbers of objects in both RRB and MBRB approaches drop rapidly, from 10^5 objects in two types to 10^4 objects in four types. The more Voronoi diagrams overlap, the more OVRs are generated, which requires more memory. Moreover, the dropping rate of the MBRB approach is higher than RRB because the MBRB approach consumes more memory when the number of object types is greater than three, as shown in Fig. 20d.

Figure 20b, c, and display corresponding execution time, the number of OVRs, and memory consumption of both approaches with parameters that lie on the availability lines (in Fig. 20a). Due to different data sizes and the number of object types configured in the two groups of evaluation, we conduct another group of experiments that evaluate the

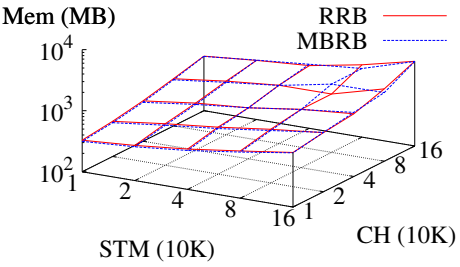


Fig. 19 Memory consumption

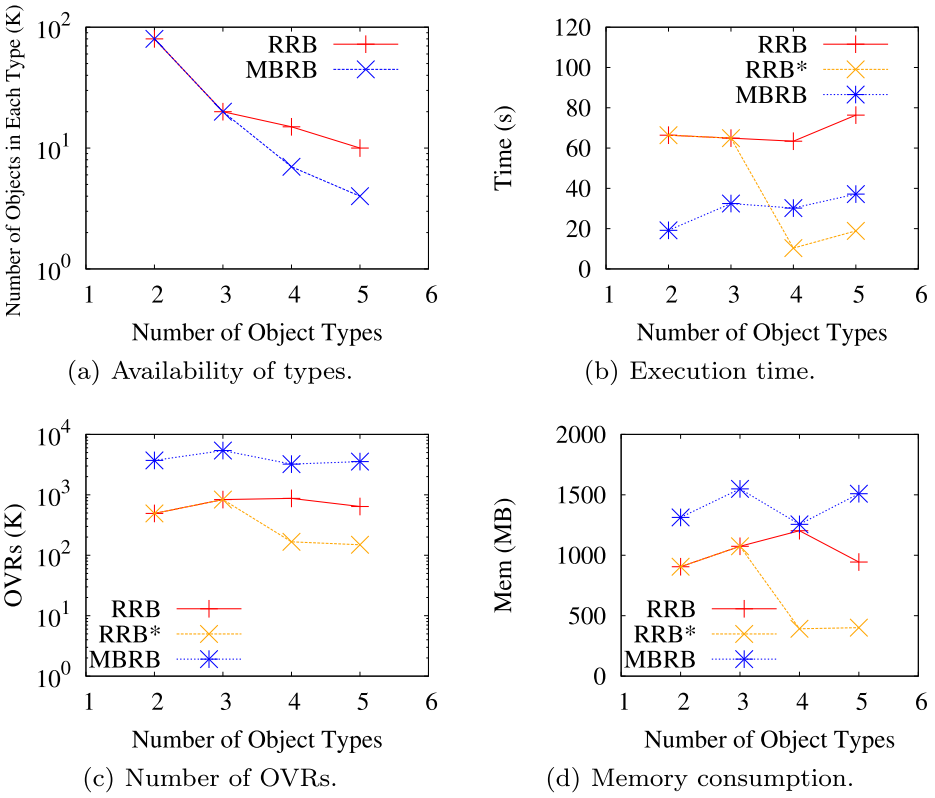


Fig. 20 Varying number of object types in Euclidean space

RRB approach with the parameters used in MBRB evaluation for fair comparison. The experimental results are highlighted by RRB*.

As we expect, the MBRB approach always produces relatively larger set of OVRs than RRB*, as shown in Fig. 20c. The more OVRs generated by MBRB than RRB* increase from 4.4 times in two object types, 19 times in four object types, to 24 times in five object types, since the false positives in the overlapping process will be fed in the next overlapping process, which generates more false positives. Moreover, a turning point in terms of execution time is observed between 3 and 4 in Fig. 20b because the computation complexity induced by a surprisingly large number of OVRs dominates the entire process in the MBRB approach, which has a greater impact than the benefits obtained from the region overlapping calculation. The drop of the execution time of RRB* indicates that RRB will outperform MBRB with the same parameters if objects in four types are considered in the query.

8.1.5 Scalability with object types

To evaluate the scalability of RRB and MBRB solutions for MOLQ over datasets of multiple object types, we develop disk-based RRB and MBRB approaches, in which OVRs in large MOVDs are sequentially read from or written to disk in the MOVD overlapping operation. We use the largest ten object types from our datasets in the experiments.

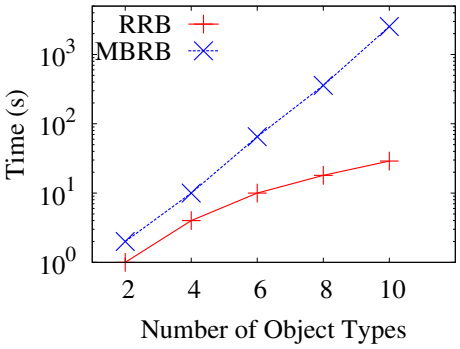


Fig. 21 MOLQ with varied object types

Figure 21 shows the execution time of MOLQ queries by using the disk-based RRB and MBRB solutions. The number of object types varies from 2 to 10, and the number of objects in each object type is fixed at 1000. As more types of objects are considered in the query, disk-based MBRB solution takes much longer for the query than RRB solution. An interesting observation is that MBRB outperforms RRB over datasets of two or three object types if OVRs are kept in memory; but the disk-based MBRB is twice as slow as the disk-based RRB. The reason is that MBRB suffers from extra I/O cost of reading and writing false positives. Moreover, we also conduct a group of experiments, in which we fix the number of object types at five and vary the number of objects in each object type from 0.5 K to 2.5 K. The execution time of disk-based MBRB and RRB is displayed in Fig. 22. RRB also outperforms MBRB and the difference becomes larger as data cardinality increases.

8.1.6 Summary

We have evaluated the performance of the RRB and MBRB solutions in Euclidean space. Which method performs better depends on the number of object types and the size of object set in each type. The MBRB solution outperforms the RRB solution in queries with relatively small types of objects ($\leq$ 3) and small number of objects ($\leq 10^4$) in each type, because there are only a limited number of false positives generated during the overlapping operation by MBRB. The computational cost on the false positives does not dominate the entire query evaluation. But, if there are more than three object types, and each data set contains ten thousands of objects, RRB becomes a better solution in terms of the execution time and memory consumption. The growth rate of the execution time of the RRB solution

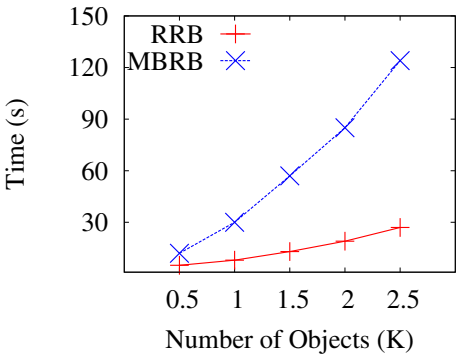


Fig. 22 MOLQ with varied object cardinality in each object type

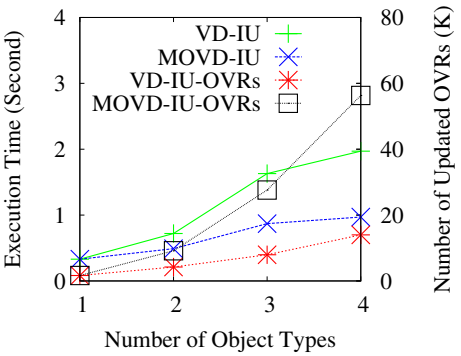


Fig. 23 Update 100 STM objects by varying object types

is much lower than the ones of the MBRB solution when receiving larger data sets. Moreover, if there are more than five object types in input datasets, disk-based RRB solution is a better solution, which suffers from less cost in both overlapping computation and loading OVRs from disk.

8.2 MOLQ updating in Euclidean space

We also evaluate the performance of our MOVD-based Incremental Updating (MOVD-IU) approach over various data sets. We implemented our proposed algorithms and the Voronoi-diagram-based Incremental Updating solution (VD-IU) in C++ [26, 55, 56]. VD-IU uses RRB solution for fair comparison, because the RRB solution and MOVD-IU produce real regions of OVRs in MOVDs. All experimental results are presented in figures with double y axes. MOVD-IU and VD-IU indicate the execution time of MOVD-IU and VD-IU in the experiments, and MOVD-IU-OVRs and VD-IU-OVRs display the number of OVRs updated by the two methods.

8.2.1 MOLQ updating evaluation in Euclidean space

We first evaluate the performance of updating 100 objects over MOLQ by using VD-IU and MOVD-IU. The initial datasets have two, three, or four object types in the experiments, and the number of objects in each type is fixed at 1000. An object update is completed by an object deletion and an object insertion in both methods. The 100 updated objects are

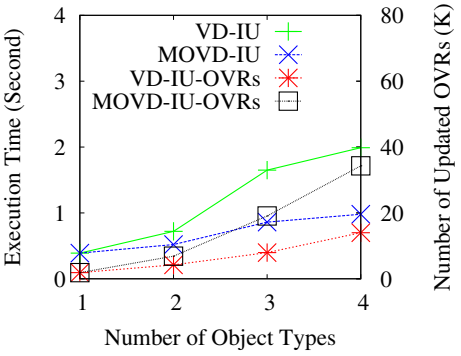


Fig. 24 Randomly update 100 objects by varying object types

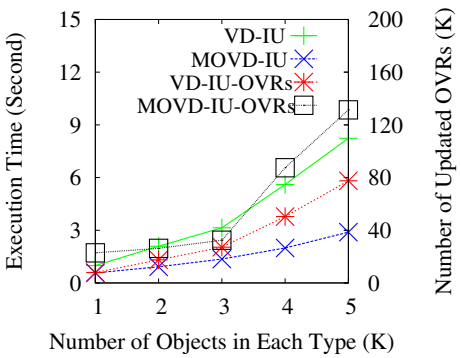


Fig. 25 Insert 100 STM objects by varying dataset cardinality

randomly selected from either STM object set or all object sets, the results of which are displayed in Figs. 23 and 24, respectively.

A similar pattern is observed from the two figures. As more object types are considered in MOLQ, updating objects takes longer because more OVRs are generated and maintained by the two methods. If the number of objects in each type is fixed, the area of the dominance regions of objects is also fixed. Thus, with higher density of OVRs in an MOVD, more OVRs should be, in general, updated in an object insertion or deletion process. The evidence is shown by VD-IU-OVRs and MOVD-IU-OVRs in the figures. Moreover, another observation is that the difference in the execution time of the two methods becomes larger as more object types are considered in the query. Even though MOVD-IU updates more OVRs than VD-IU, MOVD-IU still run, on average, 32% faster than MOVD-IU because MOVD-IU avoids OVR overlapping operations in object insertion. The boundaries of OVRs are updated when the dominance regions of inserted objects are re-calculated by using Voronoi diagram updating algorithms. In object deletion, the number of times overlapping operations are performed is linear to the number of updated OVRs in MOVD-IU; while VD-IU (RRB is used) is output-sentitive, which may suffer from quadratic cost at the worst case.

8.2.2 Effect of dataset cardinality

In this subsection, we focus primarily on the MOLQ evaluation over object insertion and deletion operations by varying dataset cardinality. We select STM, SCH and PPL objects; the number of objects in each type varies from 1000 to 5000. There are 100 objects randomly

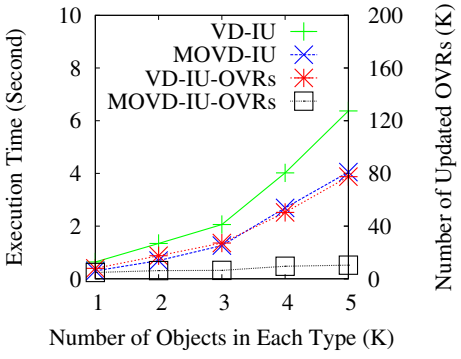


Fig. 26 Delete 100 STM objects by varying dataset cardinality

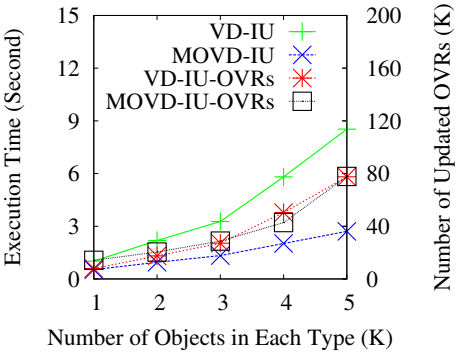


Fig. 27 Randomly insert 100 objects by varying dataset cardinality

inserted into or deleted from STM datasets in each group of experiments. The experimental results are displayed in Figs. 25 and 26, respectively.

As we expect, VD-IU and MOVD-IU take longer to answer MOLQ with 100 objects inserted/deleted over larger datasets. The execution time of VD-IU grows 2.14 times, on average, faster than MOVD-IU due to large number of OVRs generated by RRB approach (used by VD-IU) and high cost of polygon overlapping operations. MOVD-IU does not perform polygon overlapping operations and only merges OVRs if they are in the dominance region of inserted objects and share object groups. On the other hand, there are more OVRs updated by MOVD-IU in object insertion because the growth rate of the OVR density is higher than the falling rate of area of dominance regions of inserted objects as datasets grow.

A similar patten is also observed in Figs. 27 and 28, which display the experimental results of randomly inserting/deleting 100 objects into/from three datasets. The types of updated objects are randomly selected from STM, SCH, and PPL datasets. With larger datasets, MOVD-IU saves more time (around 47%) than VD-IU. Moreover, there is no difference between object insertion and deletion in VD-IU, which always overlaps Voronoi diagrams after objects are updated. However, it is interesting that the growth of execution time of object insertion is slower than that of object deletion in MOVD-IU, but the object insertion suffers from updating more OVRs than object deletion. The reason is that MOVD-IU may perform overlapping calculation when objects are deleted, and the number of times the overlapping calculation is performed goes up proportionally to the number of updated OVRs.

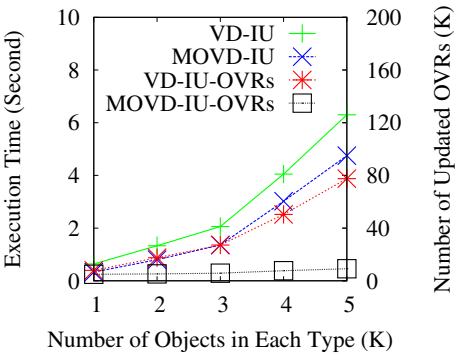


Fig. 28 Randomly delete 100 objects by varying dataset cardinality

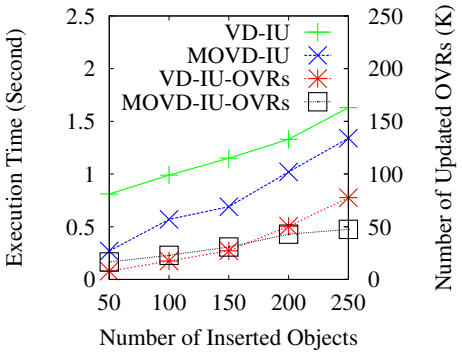


Fig. 29 Insert STM objects by varying the number of inserted objects

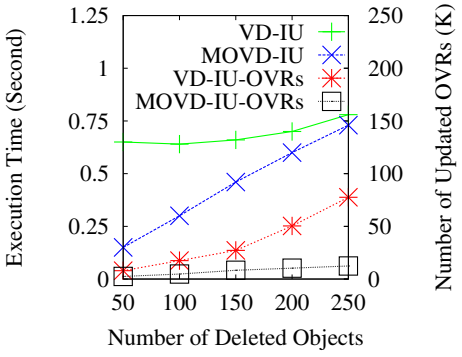


Fig. 30 Delete STM objects by varying the number of deleted objects

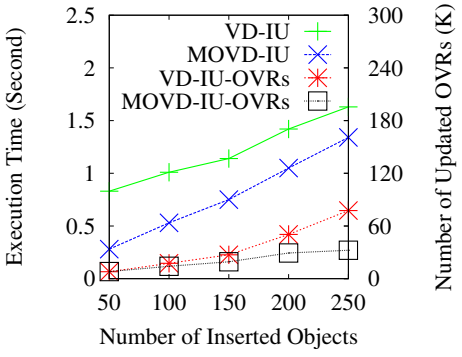


Fig. 31 Randomly insert objects by varying the number of inserted objects

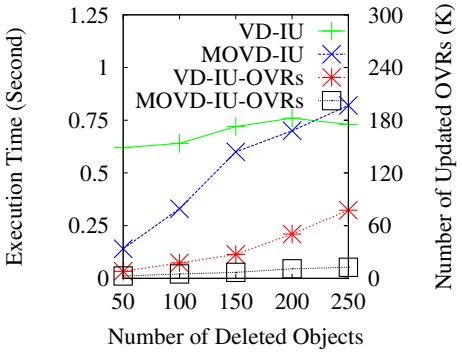


Fig. 32 Randomly delete objects by varying the number of deleted objects

8.2.3 Effect of number of updated objects

We also evaluate the efficiency of MOVD-IU and VD-IU to update MOLQ by varying the number of objects inserted into or deleted from initial datasets. We select STM, SCH and PPL objects; the number of objects in each type is fixed at 1000. The number of updated objects is varied from 50 (5% of objects in a type) to 250 (25%). From Figs. 29 and 30, both methods need more time for MOLQ updating as there are more objects inserted or deleted. In the updating process, VD-IU maintains Voronoi diagrams of objects and generates an MOVD after all objects have been updated; while MOVD-IU updates the MOVD in each object change. Because the cost of updating Voronoi diagrams is lower than that of updating MOVD, the benefit of using MOVD-IU will be offset by updating relatively large object sets.

Figures 31 and 32 shows the execution time and the number of updated OVRs when objects are randomly inserted into or deleted from all three datasets. An interesting observation is that VD-IU outperforms MOVD-IU in terms of execution time when more than 250 objects are deleted from initial datasets. Although the updated OVRs by MOVD-IU is much less than the ones by VD-IU, MOVD-IU has to update OVRs whenever an object is inserted or deleted, but VD-IU only computes the MOVD once.

9 Conclusion

In this research, we formulate a novel optimal location selection problem. In addition to the Sequential Scan Combinations (SSC) method, we propose a Minimum Overlapped Voronoi Diagram (MOVD) based Real Region as Boundary (RRB) approach that efficiently answers the query. To minimize the costs of region overlapping, we propose the Minimum Bounding Rectangle as Boundary (MBRB) approach that uses Minimum Bounding Rectangles (MBRs) as the boundaries of Overlapped Voronoi Regions (OVRs), since overlapping two rectangles is much cheaper than overlapping two arbitrary regions. We also proposed advanced solutions to efficiently address the novel query in Euclidean space. We demonstrated the excellent performance of the proposed approaches through extensive simulations.

Moreover, we extend the original Multi-criteria Optimal Location Query (MOLQ) to MOLQ updating query. After providing a formal definition of MOLQ updating problem, we proposed an MOVD-based updating model and an advanced solution that incrementally updates objects to the MOVD generated in the last query evaluation. We further analyze the computational complexity of the object insertion and deletion algorithms over ordinary and general MOVDs. The efficiency of the proposed approach has been evaluated over real-world datasets.

For the future work, we plan to extend our solutions to MOLQ on road networks.

Acknowledgments This research has been funded in part by the U.S. National Science Foundation grants IIS-1618669 (III) and ACI-1642133 (CICI).

Appendix

A.1 More OVD/MOVD properties

A number of OVD/MOVD properties can be derived from the OVD/MOVD definitions. These properties are the basis of the OVD/MOVD model utilized in our MOVD-based solution.

Theorem 7 $|MOVD(\mathbb{E})| \leq |OVD(\mathbb{E})| = \prod_{P_i \in \mathbb{E}} |P_i|$.

By Eq. 19, OVRs are generated by a combination of selected Voronoi regions. The number of OVRs in $OVD(\mathbb{E})$ is the product of the number of Voronoi regions in these Voronoi diagrams. Because all the possible empty sets have been removed, the size of $MOVD(\mathbb{E})$ is less than or equal to $OVD(\mathbb{E})$.

Theorem 8 *Any MOVD fully covers the entire search space* $\mathbb{R}$.

$$\bigcup_{OVR_j \in MOVD(\mathbb{E})} OVR_j = \mathbb{R} \tag{33}$$

Proof This theorem can be proved by contradiction. Assume that $\exists q \in \mathbb{R}$, but $q \notin MOVD(\mathbb{E})$. Then, by the property of Voronoi diagram, $\forall P_i \in \mathbb{E}, \exists p_i \in P_i, q \in Dom(p_i)$. By the definition of MOVD (19), $q \in MOVD(E)$, which contradicts with the assumption that $q \notin MOVD(\mathbb{E})$. This completes the proof. □

A.2 Algebraic structure of MOVD

After theoretically introducing the OVD/MOVD model, we will mainly focus on the overlap operation. We create an algebraic structure of MOVD by exploring MOVD space under the overlap operation and discuss its properties. The implementation details of the operation will be presented in Section 6.

A.2.1 MOVD space

MOVD space is a universal set of MOVDs that are fed into and produced by the overlap operation. Given a universal set of object sets $\mathbb{E} = \{P_1, ..., P_n\}$, the universal set of MOVD($\mathbb{E}$) is defined as

$$U(MOVD(\mathbb{E})) = \{MOVD(E_i) \mid E_i \subseteq \mathbb{E}\} \tag{34}$$

Take $\mathbb{E} = \{P_1, P_2\}$ for example, E_i is a subset of $\mathbb{E}$. E_i could be $\emptyset$, $\{P_1\}$, $\{P_2\}$, or $\{P_1, P_2\}$. So, the universal space of MOVDs of $\mathbb{E}$ is $\{MOVD(\emptyset), MOVD(\{P_1\}), MOVD(\{P_2\}), MOVD(\{P_1, P_2\})\}$.

Theorem 9 *The overlapping area of two different OVRs is a subset of their common boundaries.*

Proof By Eq. 19, an OVR is the overlapping region of $\{Dom(p_1^u), ..., Dom(p_n^v)\}$. If we have two OVRs from an OVD such that $OVR = \bigcap_{p_i^s \in \{p_1^u, ..., p_n^v\}} Dom(p_i^s)$, and $OVR' = \bigcap_{p_i^{s'} \in \{p_1^{u'}, ..., p_n^{v'}\}} Dom(p_i^{s'})$, then the overlapping area of OVR and OVR' is

$$\begin{aligned} &OVR \bigcap OVR' \\ &= \Big(\bigcap_{p_i^s \in \{p_1^u, ..., p_n^v\}} Dom(p_i^s)\Big) \bigcap \Big(\bigcap_{p_i^{s'} \in \{p_1^{u'}, ..., p_n^{v'}\}} Dom(p_i^{s'})\Big) \\ &= \bigcap_{i=1}^{n} \left(Dom(p_i^s) \bigcap Dom(p_i^{s'})\right) \end{aligned} \tag{35}$$

According to the properties of Voronoi diagrams, $Dom(p_i^s) \cap Dom(p_i^{s'})$, where $p_i^s \neq p_i^{s'}$, p_i^s, $p_i^{s'} \in P_i$, is either their common boundaries or an empty set. Moreover, if OVR and OVR' are different, there must exist a p_i^s and $p_i^{s'}$ that are different. The boundaries of an OVR are comprised of the boundaries of corresponding Voronoi regions. Hence, the overlapping region of OVR and OVR' is a subset of their common boundaries. □

Theorem 10 *When $\mathbb{E}$ is made up of only one object set $\mathbb{E} = \{P\}$, then*

$$MOVD(\mathbb{E}) = OVD(\mathbb{E}) = VD(P) \tag{36}$$

Proof This theorem is straightforward. If $\mathbb{E}$ has only one object set P, there is no other Voronoi diagram overlapped on $VD(P)$. Obviously $VD(P)$ does not have any empty regions. $OVD(\mathbb{E})$ and $MOVD(\mathbb{E})$ are identical to $VD(P)$. This theorem not only states an extreme case of definitions, but also highlights basic units in the OVD/MOVD model. All OVDs are generated from these building blocks. □

Theorem 11 *Given a type weight function ς^t, object weight functions $\sigma = \{\varsigma_1^o, ..., \varsigma_n^o\}$, and a point q in OVR $(p_1^u, ..., p_n^v)$, the total weighted distance from q to the corresponding object group $G = \{p_1^u, ..., p_n^v\}$ is the minimum weighted distance from q to all object combinations G', where $G' \in P_1 \times ... \times P_n$.*

$$\begin{aligned} WGD\ (q, G, \varsigma^t, \sigma) = \min(\\ \{\ WGD(q, G', \varsigma^t, \sigma) \mid G' \in P_1 \times ... \times P_n\ \}\ ) \end{aligned} \tag{37}$$

Proof If $VD(P_1)$ is generated by P_1 and the weight function $\varsigma_1^o \in \sigma$, a point q in $OVR(p_1^u, ..., p_n^v)$ must fall in $Dom(p_1^u)$ of $VD(P_1)$ so that p_1^u is the closest point in P_1 to q. $WD(q, p_1^u, \varsigma^t, \varsigma_1^o)$ is the minimum weighted distance from q to any points in P_1. We can get the same result in other sets $P_i \in \mathbb{E}$. After summing them up, we obtain Theorem 11 that $WGD(q, G, \varsigma^t, \sigma)$ has the minimum distance. □

Theorem 12 *An OVR may have one or more generators, or may not have any generators.*

Proof The overlapping operation decomposes Voronoi cells into smaller subregions. A generator can only fall in one subregion, and other subregions do not have any generators. It is possible that two generators from two Voronoi diagrams may be in the same subregion. For example, in Fig. 5c, p_1 and q_1 are in the top-left subregion, and the doubly shaded subregion does not have any generators. □

Theorem 13 *$|MOVD(\mathbb{E})|$ is bigger than or equal to $|VD(P_i)|$, where $P_i \in \mathbb{E}$.*

$$|MOVD(\mathbb{E})| \geq |VD(P_i)| \tag{38}$$

Proof Overlapping two Voronoi diagrams is a process in which one Voronoi diagram is decomposed by another Voronoi diagram. Each Voronoi region is divided into a number of subregions, unless two Voronoi regions from different VDs are exactly the same, or one region contains the other. In these extreme cases, the Voronoi region remains unchanged. Thus, after overlapping Voronoi diagrams, the number of overlapping regions in an MOVD is either greater than or equal to the basic Voronoi diagrams. □

Theorem 14 *The number of MOVDs existing in the universal space is as follows:*

$$|U(MOVD(\mathbb{E}))| = \sum_{i=0}^{|\mathbb{E}|} \binom{|\mathbb{E}|}{i} \tag{39}$$

Proof By definition, MOVD space consists of a number of MOVDs, each of which is generated by a subset of $\mathbb{E}$; thus the number of MOVDs in the space equals the number of subsets in $\mathbb{E}$, which is presented as Eq. 39. The case that i equals 0 indicates a special subset, the empty set, defined in Eq. 21. □

A.3 ⊕ operation properties

By properties of the union operation on sets, we can obtain the following three laws.

Theorem 15 *Idempotent Law*

$$MOVD(E_i) \oplus MOVD(E_i) = MOVD(E_i) \tag{40}$$

Proof By the idempotent law of union operation on sets, $MOVD(E_i \bigcup E_i)$ is equal to $MOVD(E_i)$. Thus, overlaying two identical MOVDs produces the MOVD itself. □

Theorem 16 *Commutative Law*

$$\begin{aligned} MOVD(E_i) \oplus MOVD(E_j) = \\ MOVD(E_j) \oplus MOVD(E_i) \end{aligned} \tag{41}$$

Proof By the commutative law of union operation on sets, $MOVD(E_i \bigcup E_j)$ is equal to $MOVD(E_j \bigcup E_i)$. Thus, swapping two operands does not change the result. □

Theorem 17 *Associative Law*

$$\begin{aligned} (MOVD(E_i) \oplus MOVD(E_j)) \oplus MOVD(E_k) = \\ MOVD(E_i) \oplus (MOVD(E_j) \oplus MOVD(E_k)) \end{aligned} \tag{42}$$

Proof By the associative law of union operation on sets, $MOVD((E_i \bigcup E_j) \bigcup E_k)$ is equal to $MOVD(E_i \bigcup (E_j \bigcup E_k))$. Thus, ⊕ operation can be performed in any order without ambiguity. □

Corollary 1 *MOVD(E_i), where $E_i \subseteq \mathbb{E}$, is unique.*

Proof According to the commutative and associative laws of operation ⊕, the order of overlapping Voronoi diagrams does not cause the result to change. Thus MOVD(E_i) is unique. □

Theorem 18 $MOVD(\emptyset)$ *is an identity element.*

Proof $MOVD(\emptyset)$ equals $\{\mathbb{R}\}$ such that it leaves MOVDs unchanged under operation $\oplus$. The following equation can be easily proved by the definition of $\oplus$.

$$MOVD(E_i) \oplus MOVD(\emptyset) = MOVD(E_i) \tag{43}$$

□

Theorem 19 *Closure: the universal MOVD space of* $\mathbb{E}$ *is closed under operation* $\oplus$.

Proof By definition, given any MOVD(E_i) and MOVD(E_j), where $E_i, E_j \subseteq \mathbb{E}$, the result of overlapping them is MOVD($E_i \cup E_j$). $E_i \cup E_j$ is still a subset of $\mathbb{E}$, so the result is an element of $U(MOVD(\mathbb{E}))$. □

Definition 1 Sequential Overlap Operations

$$\begin{aligned} \sum_{i=1}^{n} MOVD(E_i) &= MOVD(E_1) \oplus ... \oplus MOVD(E_n) \\ &= MOVD(\bigcup_{i=1}^{n} E_i) \end{aligned} \tag{44}$$

Definition 2 Partial Order
If $MOVD(E_i) = MOVD(E_j) \oplus MOVD(E_k)$ then,

$$\begin{aligned} MOVD(E_i) &\succcurlyeq MOVD(E_j) \\ MOVD(E_i) &\succcurlyeq MOVD(E_k) \end{aligned} \tag{45}$$

The partial order definition formalizes a comparison model for evaluating how much information MOVDs maintain. In Eq. 45, $MOVD(E_i)$ is generated by $MOVD(E_j)$ and $MOVD(E_k)$. $MOVD(E_i)$ has more information (i.e., objects) than either $MOVD(E_j)$ or $MOVD(E_k)$. We use $\succcurlyeq$ to denote the relationship.

Theorem 20 $MOVD(E_i) \oplus MOVD(E_j) = MOVD(E_i)$ *if* $MOVD(E_i) \succcurlyeq MOVD(E_j)$.

Proof The following equation proves Theorem 20 by applying the partial order definition that decomposes $MOVD(E_i)$ into $MOVD(E_j)$ and $MOVD(E_k)$, and the commutative and idempotent laws of operation $\oplus$. □

$$\begin{aligned} MO\ VD(E_i) &\oplus MOVD(E_j) \\ &= MOVD(E_j) \oplus MOVD(E_k) \oplus MOVD(E_j) \\ &= MOVD(E_j) \oplus MOVD(E_j) \oplus MOVD(E_k) \\ &= MOVD(E_j) \oplus MOVD(E_k) \\ &= MOVD(E_i) \end{aligned} \tag{46}$$

A.4 Analysis of object insertion operation to an MOVD

Since $MOVD(\mathbb{E})$ would be more complex than $MOVD(\{Q\})$, the proposed algorithms would be dominated by updating $MOVD(\mathbb{E})$ rather than updating $MOVD(\{Q\})$ in most cases. To accurately estimate the cost of MOVD updating algorithms, we further explore

the object insertion operation over ordinary MOVDs (generated from ordinary Voronoi diagrams) after $MOVD(\{Q\})$ has been updated. An alternative OVR merging method can be used in the insertion algorithm, the computational complexity of which is bounded at $3 \times I$ (See Corollary 3). In addition, we also find that the computational complexity of the object deletion algorithm over general MOVDs can be bounded at $O(I)$, where I denotes the number of OVRs in the dominance region of the deleted object, because every updated OVR is only needed to merge with at most one of its neighbor OVRs (See Theorem 22).

In this section, all the following theoretical analysis and examples are presented in a 2-dimensional space; however, they can be extended to higher-dimensional spaces in similar methods. We use $\mathcal{S}$ to denote a subdivision of a search space $\mathbb{R}$. We assume that $\mathcal{S}$ can be abstracted by a set of vertices and line segments. $\mathcal{V}$, $\mathcal{E}$, and $\mathcal{F}$ are used to denote the sets of vertices, edges, and faces in $\mathcal{S}$. $\mathcal{V}^*$ and $\mathcal{E}^*$ indicate vertices and edges of $\mathcal{S}$ in a convex set $\mathcal{C}$. $\mathcal{F}^*$ contains faces of $\mathcal{S}$ overlapping $\mathcal{C}$.

Lemma 1 *Given a convex set $\mathcal{C}$ in a search space and a subdivision $\mathcal{S}$ of the space, let $\mathcal{W}^*$ be the set of intersection points of $\mathcal{S}$ and $\mathcal{C}$, $\mathcal{V}^*$ be the vertices of $\mathcal{S}$ in $\mathcal{C}$ ($\mathcal{V}^* = \{v \in \mathcal{V} \mid v$ is inside $\mathcal{C}\}$), and $\mathcal{F}^*$ be the set of faces overlapped with $\mathcal{C}$ ($\mathcal{F}^* = \{f \in \mathcal{F} \mid f$ overlaps $\mathcal{C}\}$), if the degree of all vertices of $\mathcal{S}$ is 3 ($deg(v) = 3, v \in \mathcal{V}$), then $|\mathcal{W}^*| + |\mathcal{V}^*| \leq 2(|\mathcal{F}^*| - 1)$.*

Lemma 1 cannot be applied to non-convex sets because a line of the subdivision may intersect with the boundary of a non-convex set at two or more points. $|\mathcal{W}^*|$ may go up to infinity in extreme cases, and the upper bound of $|\mathcal{W}^*| + |\mathcal{V}^*|$ does not exist. Figure 33 shows an example of Lemma 1, in which the search space is decomposed by an ordinary Voronoi diagram. We assume that the degree of all vertices in the figure is 3. The figure also shows a convex polygon $\mathcal{C}$ as an example of convex sets, which intersects with the subdivision at $\mathcal{W}^* = \{w_1, w_2, w_3, w_4\}$. v_1 and v_2 are two vertices inside $\mathcal{C}$ ($\mathcal{V}^* = \{v_1, v_2\}$). The faces overlapping $\mathcal{C}$ are $\mathcal{F}^* = \{Dom(p_2), Dom(p_3), Dom(p_5), Dom(p_6)\}$.

Proof We prove this lemma using Euler's formula, which states that $v - e + f = 2$ is always held for any connected planar graph with v vertices, e edges, and f faces. As shown in Fig. 34, we create a space $\mathbb{R}'$, which is limited by the boundary of the convex polygon $\mathcal{C}$. All edges inside the polygon are edges in the space $\mathbb{R}'$. v_1 and v_2 are two inner vertices in $\mathbb{R}'$,

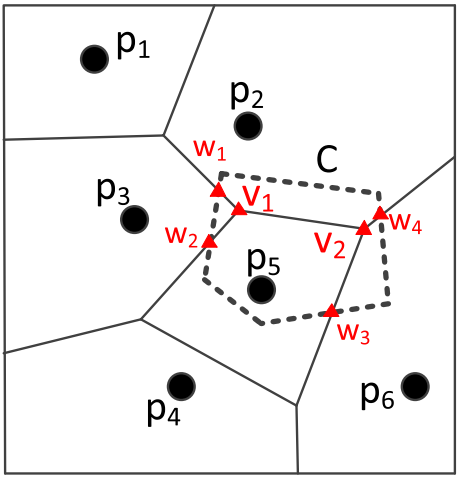


Fig. 33 An example of Lemma 1

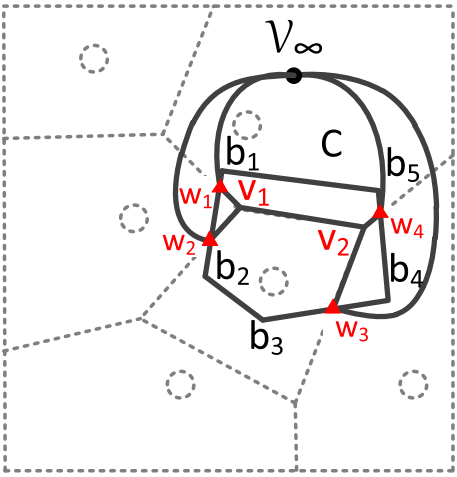


Fig. 34 The convex polygon $\mathcal{C}$ is converted to a planar graph

and w_1, w_2, w_3, and w_4 are four vertices on the boundary of $\mathbb{R}'$. Euler's formula cannot be applied directly to the subdivision in $\mathbb{R}'$ due to the existence of half-infinite edges [3]. Thus, we create an extra vertex v_∞, which connects to the boundary points $\mathcal{W}^*$, to convert the subdivision to a connected planar graph. Since each edge connects two vertices, the number of edges is equal to the half of the total degree of all vertices in $\mathbb{R}'$ (the number of edges is $\frac{2|\mathcal{W}^*|+3|\mathcal{V}^*|+deg(v_\infty)}{2}$). Then, according to Euler's formula,

$$(|\mathcal{W}^*| + |\mathcal{V}^*| + 1) - \frac{2|\mathcal{W}^*|+3|\mathcal{V}^*|+deg(v_\infty)}{2} + |\mathcal{F}^*| = 2 \tag{47}$$

where "1" indicates the vertice v_∞ and $deg(v_\infty) = |\mathcal{W}^*|$. Finally, we obtain $|\mathcal{W}^*|+|\mathcal{V}^*| = 2(|\mathcal{F}^*| - 1)$ by simplifying (47). Moreover, the convex set $\mathcal{C}$ may intersect with the subdivision at vertex v_2 instead of w_4, and there is one less boundary point in this case. To take the case into account, v_2 is considered as an inner vertex because its degree is 3; while the degree of boundary points is always 2. Therefore, $2(|\mathcal{F}^*| - 1)$ is an upper bound of $|\mathcal{W}^*| + |\mathcal{V}^*|$, and this concludes the proof. □

Take Fig. 33 for example, there are four boundary points ($\mathcal{W}^* = \{w_1, w_2, w_3, w_4\}$), two inner vertices ($\mathcal{V}^* = \{v_1, v_2\}$), and four faces ($\mathcal{F}^* = \{Dom(p_2), Dom(p_3), Dom(p_5), Dom(p_6)\}$) overlapping with $\mathcal{C}$. Thus, Lemma 1 is held in this example.

Theorem 21 *Given a subdivision $\mathcal{S}$ in a search space, a convex set $\mathcal{C}$, and a space $\mathbb{R}'$ in $\mathcal{C}$, the degree of vertices in the subdivision would range from 3 to k. Let $\mathcal{V}_i^*$ denote the set of vertices with degree i inside $\mathcal{C}$, where $3 \leq i \leq k$ ($|\mathcal{V}^*| = \sum_{i=3}^{k} |\mathcal{V}_i^*|$), and $\mathcal{F}^*$ be the set of faces overlapping $\mathcal{C}$, then*

$$|\mathcal{W}^*| + |\mathcal{V}^*| \leq 2(|\mathcal{F}^*| - 1) - \sum_{i=3}^{k}((i-3) \times |\mathcal{V}_i^*|) \tag{48}$$

Proof Similarly with the proof with Lemma 1, Euler's formula can be applied to the cases presented in Theorem 21 by adding v_∞. The difference is that the degree of vertices inside $\mathcal{C}$ may be greater than 3; but the degree of boundary points in $\mathcal{W}^*$ are still fixed at 2, and $deg(v_\infty) = |\mathcal{W}^*|$ is also held. Thus, the total degree of vertices and boundary points is

$2|\mathcal{W}^*| + deg(v_\infty) + \sum_{i=3}^{k}(i \times |\mathcal{V}_i^*|)$, and the number of edges in $\mathbb{R}'$ is the half of the total degree. According to Euler's formula, we get

$$(|\mathcal{W}^*| + |\mathcal{V}^*| + 1) - \frac{2|\mathcal{W}^*| + deg(v_\infty) + \sum_{i=3}^{k}(i \times |\mathcal{V}_i^*|)}{2} + |\mathcal{F}^*| = 2 \tag{49}$$

Moreover, there might be less boundary points in $\mathcal{W}^*$ in the cases when $\mathcal{C}$ intersects with $\mathcal{S}$ at inner vertices. Thus, after simplifying (49), we get (48), and this concludes Theorem 21. □

Theorem 21 studies the relation between vertices and faces in the overlapping region of a convex set and a subdivision. In Corollary 2, we will explore a specific case that a convex polygon overlaps a subdivision, and analyze the upper bound of the number of vertices and edges in the overlapping region.

Corollary 2 *Given a subdivision $\mathcal{S}$ of a search space, in which the degree of vertices ranges from 3 to k, let $\mathcal{F}^*$ be the set of faces overlapping a convex polygon $\mathcal{C}$ of m vertices, then the convex polygon contains at most $m + 2(|\mathcal{F}^*| - 1)$ vertices and $m + 3(|\mathcal{F}^*| - 1)$ edges.*

Proof Similarly, we first create a vertex v_∞ to convert the subdivision in the convex polygon to a connected planar graph. v_∞ is connected to all boundary points in $\mathcal{W}^*$ in the convex polygon, shown in Fig. 34. According to Theorem 21, we can get an upper bound of $|\mathcal{W}^*| + |\mathcal{V}^*|$ in Eq. 48. Then, we create a new graph that consists of the inner vertices in $\mathcal{V}^*$, inner edges, boundary points in $\mathcal{W}^*$, vertices of $\mathcal{C}$, and the boundary of $\mathcal{C}$. v_∞ is not included in the graph. For example, the new graph in Fig. 34 is made up of all vertices and edges in $\mathcal{C}$ and on the boundary of C. The degree of inner vertices ranges from 3 to k; the degree of boundary points is always 3 in the graph (w_1 connects to w_2, v_1, and b_1), and the degree of vertices of the polygon ($\{b_1, b_2, b_3, b_4, b_5\}$) is 2. Then, the total degree of all vertices in the new graph is $2m + 3|\mathcal{W}^*| + \sum_{i=3}^{k}(i \times |\mathcal{V}_i^*|)$. Thus, the number of edges in the graph is

$$\begin{aligned}
|e| &= \frac{2m + 3|\mathcal{W}^*| + \sum_{i=3}^{k}(i \times |\mathcal{V}_i^*|)}{2} \\
&= m + \frac{3|\mathcal{V}^*| + 3|\mathcal{W}^*| + \sum_{i=3}^{k}((i-3) \times |\mathcal{V}_i^*|)}{2} \\
&\leq m + \frac{3 \times [2(|\mathcal{F}^*| - 1) - \sum_{i=3}^{k}((i-3) \times |\mathcal{V}_i^*|)] + \sum_{i=3}^{k}((i-3) \times |\mathcal{V}_i^*|)}{2} \\
&= m + 3(|\mathcal{F}^*| - 1) - \sum_{i=3}^{k}((i-3) \times |\mathcal{V}_i^*|) \\
&\leq m + 3(|\mathcal{F}^*| - 1)
\end{aligned} \tag{50}$$

By Theorem 21, the total number of vertices in $\mathcal{C}$ is

$$|\mathcal{W}^*| + |\mathcal{V}^*| + m \leq m + 2(|\mathcal{F}^*| - 1) - \sum_{i=3}^{k}((i-3) \times |\mathcal{V}_i^*|) \leq m + 2(|\mathcal{F}^*| - 1) \tag{51}$$

Equations 50 and 51 conclude Corollary 2. Equations 50 and 51 are also held in cases if any point in $\mathcal{W}^*$ happens to be a vertex of $\mathcal{C}$, because the two upper bounds do not change in the cases. □

Since each boundary point breaks a line on the boundary of $\mathcal{C}$ into two line segments, and the number of sides of $\mathcal{C}$ is equal to the number of vertices of $\mathcal{C}$, so the number of line

segments on the boundary of $\mathcal{C}$ is $m + |\mathcal{W}^*|$, and the number of inner edges inside $\mathcal{C}$ (the line segments on the boundary are excluded) is

$$|e| - (m + |\mathcal{W}^*|) \leq 3(|\mathcal{F}^*| - 1) - \sum_{i=3}^{k}((i-3) \times |\mathcal{V}_i^*|) - |\mathcal{W}^*| \tag{52}$$

If Corollary 2 is applied to the proposed MOVD updating operations, we can get the following corollary, in which MOVDs are considered as examples of subdivisions in a search space and the dominance region of an updated object is a convex polygon in ordinary MOVDs.

Corollary 3 *Given an $MOVD(\mathbb{E})$, a Voronoi diagram $VD(Q)$ ($Q \in \mathbb{E}$), and an object q that is needed to insert into or delete from $MOVD(\mathbb{E})$, let $Dom(q)$ be the dominance region of q after the insertion or before the deletion, then $Dom(q)$ contains at most*

$$\begin{aligned} &m + 2(|\mathcal{F}^*| - 1) - (\sum_{i=3}^{\infty}(i-3) \times |\mathcal{V}_i^*|) \quad \textit{vertices, and} \\ &m + 3(|\mathcal{F}^*| - 1) - (\sum_{i=3}^{\infty}(i-3) \times |\mathcal{V}_i^*|) \quad \textit{edges} \end{aligned} \tag{53}$$

where m is the number of vertices of $Dom(q)$, $\mathcal{F}^*$ denotes the set of OVRs overlapping $Dom(q)$ in $MOVD(\mathbb{E})$, and $\mathcal{V}_i^*$ indicates the set of vertices with degree i inside $Dom(q)$. The upper bound of the number of inner edges in $Dom(q)$ (derived from Eq. 52*) is*

$$3(|\mathcal{F}^*| - 1) - \sum_{i=3}^{k}((i-3) \times |\mathcal{V}_i^*|) - |\mathcal{W}^*| < 3|\mathcal{F}^*| \tag{54}$$

In addition to the analysis of the computational complexity of the proposed MOVD updating algorithms in Sections 7.3.2 and 7.3.3, Corollary 3 provides us with an upper bound of the computational complexity of ordinary MOVD updating. In the object insertion operation, the dominance region of an inserted object q and its neighbor OVRs are found by detecting the boundary points. The cost of the detection process is proportional to the total number of the boundary points and the number of vertices of $Dom(q)$. When merging OVRs in the final step, we have analyzed the cost of one possible method that checks every pair of neighbor OVRs. The cost of the method is $O(C \times I)$, where C denotes the average number of neighbor OVRs of a given OVR, and I denotes the average number of OVRs in $Dom(q)$. However, an alternative OVR merging method (for ordinary MOVDs) is iterating all the edges in $Dom(q)$. Two OVRs will be merged if they share an edge and their group objects. Thus, the cost of the second OVR merging method is proportional to the number of inner edges in $Dom(q)$, the upper bound of which is $3 \times I$ specified in Eq. 54.

If an object p is deleted from an MOVD, the computational complexity of the object deletion algorithm is also bounded by $O(C \times I)$ at average cases, because every OVR inside $Dom(q)$ is detected to merge with its neighbor OVRs outside $Dom(q)$. If two OVRs have the same object group, they are merged in the deletion operation (See line 19 in Algorithm 8). However, we observe that, for any given OVR inside $Dom(q)$, there is at most one

neighbor OVR outside $Dom(q)$ needed to merge with the given OVR. All other neighbor OVRs inside $Dom(p)$ cannot share object groups with the given OVR, because they have been associated with different objects before deletion.

Lemma 2 *Given an $MOVD(E)$, where $E = \{P_1, ..., P_n\}$, and an object $p_i^k \in P_i$, $P_i \in E$, if p_i^k is deleted from $MOVD(E)$, then $\forall$ ovr inside $Dom(p_i^k)$, there is at most one OVR needed to merge with ovr at line 19 in Algorithm 8.*

Proof When the algorithm reaches line 19, given two updated OVRs, ovr and ovr', there are two possible cases: (1) both ovr and ovr' are in $Dom(p_i^k)$. In this case, ovr and ovr' are not merged, because they cannot share an object group. By the definition of MOVD, ovr and ovr' are two OVRs before deletion; there must exist an object type j ($j \neq i$); p_j^s in the object group of ovr is different from p_j^t in the object group of ovr'. Moreover, since there is not any object in P_j removed during the deletion operation, p_j^s and p_j^t are still in the object groups of ovr and ovr' after deletion. Thus, ovr and ovr' cannot be merged in the deletion operation. (2) if ovr is inside $Dom(p_i^k)$ but ovr' is outside $Dom(p_i^k)$, Theorem 2 is also held, and the proof is by contradiction as follows. We assume that ovr is needed to merge with two OVRs, ovr' and ovr'', outside $Dom(p_i^k)$. Apparently, object groups of the three OVRs are the same after deletion. The deleted object p_i^k is not in the object group of ovr' or ovr'', and object groups of ovr' and ovr'' do not change during the deletion. However, ovr' and ovr'' are two OVRs outside $Dom(p_i^k)$ before deletion; by the definition of MOVD, the object group of ovr' must be different from that of ovr'', which contradicts with our assumption. Thus, there is at most one OVR needed to merge with ovr, and this concludes the proof. □

Moreover, from line 11 to 17 in Algorithm 8, we calcuate the boundaries of OVRs after q is removed from $MOVD(\{Q\})$. The neighbor OVRs of q in $MOVD(\{Q\})$ are kept in $Updated\_OVRs$ and all OVRs of $MOVD(E)$ inside $Dom(q)$ are in $Removed\_OVRs$. Calculating the overlapping region of each pair of these OVRs are not necessary. The overlapping relation among these OVRs can be established when $MOVD(\{Q\})$ is updated in line 9 and 10, because the new boundaries of OVRs in $Updated\_OVRs$ will go inside the OVRs in $Removed\_OVRs$ if they overlap with each other. Thus, we can obtain Lemma 3.

Lemma 3 *Given an $MOVD(E)$ and an object p_i^k deleted from $MOVD(E)$, the number of times the overlapping region is calculated at line 12 in Algorithm 8 is $|MOVD(E)|$ in the worst case, where $|MOVD(E)|$ denotes the number of OVRs in $MOVD(E)$.*

With Lemma 2 and 3, we get Theorem 22 as follows.

Theorem 22 *Given an $MOVD(E)$ and an object p_i^k deleted from $MOVD(E)$, the computational complexity of Algorithm 8 is $|MOVD(E)|$ in the worst case, where $|MOVD(E)| = n^t$, n denotes the number of objects in each type and t denotes the number of object types.*

References

1. Aurenhammer F (1991) Voronoi diagrams–a survey of a fundamental geometric data structure. ACM Comput Surv 23(3):345–405
2. Aurenhammer F, Klein R, Lee D-T (2013) Voronoi diagrams and delaunay triangulations. World Scientific Publishing Co Inc
3. Bajaj CL (1988) The algebraic degree of geometric optimization problems. Discret Comput Geom 3:177–191
4. Boissonnat J-D, Delage C (2005) Convex Hull and Voronoi diagram of additively weighted points. In: ESA, pp 367–378
5. Chandrasekaran R, Tamir A (1990) Algebraic optimization: the Fermat-Weber location problem. Math Program 46:219–224
6. Cheema MA, Zhang W, Lin X, Zhang Y, Li X (2012) Continuous reverse k nearest neighbors queries in euclidean space and in spatial networks. VLDB J 21(1):69–95
7. Chen Z, Liu Y, Wong RC-W, Xiong J, Mai G, Long C (2014) Efficient algorithms for optimal location queries in road networks. In: SIGMOD conference, pp 123–134
8. Farhana M, Choudhury J, Culpepper S, Sellis T, Cao X (2016) Maximizing bichromatic reverse spatial and textual K nearest neighbor queries. PVLDB 9(6):456–467
9. de Berg M, Cheong O, van Kreveld M, Mark O (2008) Computational geometry: algorithms and applications, 3rd edn. Springer
10. Demiryurek U, Shahabi C (2012) Indexing network Voronoi diagrams. In: The 17th International conference on database systems for advanced applications, DASFAA, pp 526–543
11. Devillers O (2002) On deletion in Delaunay triangulations. Int J Comput Geom Appl 12(03):193–205
12. Dinis J, Mamede M (2011) Updates on Voronoi Diagrams. In: ISVD, pp 192–199
13. Dong P (2008) Generating and updating multiplicatively weighted Voronoi diagrams for point, line and polygon features in GIS. Comput Geosci 34(4):411–421
14. Du Y, Zhang D, Xia T (2005) The optimal-location query. In: SSTD, pp 163–180
15. Finke U, Hinrichs KH (1995) Overlaying simply connected planar subdivisions in linear time. In: Proceedings of the eleventh annual symposium on computational geometry. ACM, pp 119–126
16. Fortune S (1986) A sweepline algorithm for Voronoi diagrams. In: Proceedings of the second annual symposium on computational geometry. ACM, pp 313–322
17. Fortune S (1992) Numerical stability of algorithms for 2D delaunay triangulations. In: Proceedings of the eighth annual symposium on computational geometry, pp 83–92
18. Gao Y, Zheng B, Chen G, Li Q (2009) Optimal-location-selection query processing in spatial databases. IEEE Trans Knowl Data Eng 21(8):1162–1177
19. Ghaemi P, Shahabi K, Wilson JP, Banaei-Kashani F (2012) Continuous maximal reverse nearest neighbor query on spatial networks. In: ACM SIGSPATIAL, pp 61–70
20. Green PJ, Sibson R (1978) Computing Dirichlet tessellations in the plane. Comput J 21(2):168–173
21. Green PJ, Sibson R (1978) Computing Dirichlet tessellations in the plane. Comput J 21(2):168–173
22. Guibas L, Stolfi J (1985) Primitives for the manipulation of general subdivisions and the computation of Voronoi. ACM Trans Graph (TOG) 4(2):74–123
23. Guibas LJ, Knuth DE, Sharir M (1992) Randomized incremental construction of Delaunay and Voronoi diagrams. Algorithmica 7(1):381–413
24. Guibas LJ, Stolfi J (1985) Primitives for the manipulation of general subdivisions and computation of Voronoi diagrams. ACM Trans Graph 4(2):74–123
25. Haldane JBS (1948) Note on the median of a multivariate distribution. Biometrika 35:414–415
26. Harn P-W, Ji Z, Sun M-T, Ku W-S (2016) A framework for updating multi-criteria optimal location query. In: ACM SIGSPATIAL
27. Jalal G, Krarup J (2003) Geometrical solution to the fermat problem with arbitrary weights. Annals OR 123(1–4):67–104
28. Karavelas MI, Yvinec M (2002) Dynamic additively weighted Voronoi diagrams in 2D. In: ESA, pp 586–598
29. Korn F, Muthukrishnan S (2000) Influence sets based on reverse nearest neighbor queries. In: SIGMOD conference, pp 201–212
30. Korn F, Muthukrishnan S, Srivastava D (2002) Reverse nearest neighbor aggregates over data streams. In: VLDB, pp 814–825

31. Liu R, Fu AW-C, Chen Z, Huang S, Liu Y (2016) Finding multiple new optimal locations in a road network. In: ACM SIGSPATIAL
32. Mostafavi MA, Gold C, Dakowicz M (2003) Delete and insert operations in voronoi/delaunay methods and applications. Comput Geosci 29(4):523–530
33. Mu L (2004) Polygon characterization with the multiplicatively weighted Voronoi diagram. Prof Geogr 56(2):223–239
34. Ohya T, Iri M, Murota K (1984) Improvements of the incremental method for the Voronoi diagram with computational comparison of various algorithms. J Oper Res Soc Japan 27(4):306–336
35. Okabe A, Boots B, Sugihara K, Chiu SN (2000) Spatial tessellations: concepts and applications of Voronoi diagrams. Probability and statistics, 2nd edn. Wiley, NYC
36. Pagliara F, Preston J, David S (2010) Residential location choice: models and applications. Springer
37. Qi J, Zhang R, Kulik L, Lin D, Xue Y (2012) The min-dist location selection query. In: ICDE
38. Morris JG, Love RF, Wesolowsky GO (1988) Facilities location models and methods
39. Stanoi I, Riedewald M, Agrawal D, El Abbadi A (2001) Discovery of influence sets in frequently updated databases. In: VLDB, pp 99–108
40. Sugihara K, Iri M (1992) Construction of the Voronoi diagram for'one million'generators in single-precision arithmetic. Proc IEEE 80(9):1471–1484
41. Tao Y, Papadias D, Lian X (2004) Reverse kNN search in arbitrary dimensionality. In: VLDB, pp 744–755
42. Tao Y, Papadias D, Lian X, Xiao X (2007) Multidimensional reverse k NN search. VLDB J 16(3):293–316
43. Tao Y, Yiu ML, Mamoulis N (2006) Reverse nearest neighbor search in metric spaces. IEEE Trans Knowl Data Eng 18(9):1239–1252
44. Üster H, Love RF (2002) A generalization of the rectangular bounding method for continuous location models. Comput Math Appl 44(1–2):181–191
45. Vardi Y, Zhang C-H (2001) A modified Weiszfeld algorithm for the Fermat-Weber location problem. Math Program 90:559–566
46. Verkhovsky BS, Polyakov YS (2003) Feedback algorithm for the single-facility minisum problem. Ann Europ Acad Sci 1:127–136
47. Weisbrod G, Ben-Akiva M, Lerman S (1980) Tradeoffs in residential location decisions: transportation versus other factors. Transp Polic Decis-Making, 1(1)
48. Weiszfeld E, Plastria F (2009) On the point for which the sum of the distances to n given points is minimum. Annals OR 167(1):7–41
49. Xia T, Zhang D, Kanoulas E, Du Y (2005) On computing top-t most influential spatial sites. In: VLDB, pp 946–957
50. Xiao X, Yao B, Li F (2011) Optimal location queries in road network databases. In: ICDE, pp 804–815
51. Yang C, Lin K-I (2001) An index structure for efficient reverse nearest neighbor queries. In: ICDE, pp 485–492
52. Yao B, Xiao X, Li F, Wu Y (2014) Dynamic monitoring of optimal locations in road network databases. In: VLDB, pp 697–720
53. Yiu ML, Papadias D, Mamoulis N, Tao Y (2006) Reverse nearest neighbors in large graphs. IEEE Trans Knowl Data Eng 18(4):540–553
54. Zhang D, Du Y, Xia T, Tao Y (2006) Progressive computation of the min-dist optimal-location query. In: VLDB, pp 643–654
55. Ji Z, Ku W-S, Jiang X, Qin X, Sun M-T, Lu H (2015) A framework for multi-criteria optimal location selection. In: ACM SIGSPATIAL
56. Ji Z, Ku W-S, Sun M-T, Qin X, Lu H (2014) Multi-criteria optimal location query with overlapping Voronoi diagrams. In: EDBT, pp 391–402

Ji Zhang is a Ph.D student in the Department of Computer Science and Software Engineering at Auburn University. He received the B.S. and M.S. in Computer Science from Huazhong University of Science and Technology (HUST), Wuhan, China in 2004 and 2007. He worked as a software engineer in Huawei Technologies from 2007 to 2010. His research interests include file and storage systems, parallel and distributed systems, and geographic information systems.

Po-Wei Harn is currently an engineer at Institute for Information Industry, Taiwan. He received his B.S. degree in Mathematics in 2013 and M.S. degree in Computer Science and Information Engineering in 2015, both from National Central University, Taiwan. His research interest is in spatial algorithm design and analysis.

Wei-Shinn Ku received his Ph.D. degree in Computer Science from the University of Southern California (USC) in 2007. He also obtained both the M.S. degree in Computer Science and the M.S. degree in Electrical Engineering from USC in 2003 and 2006, respectively. He is a professor with the Department of Computer Science and Software Engineering at Auburn University. His research interests include data management systems, mobile computing, and cybersecurity. He has published more than 100 research papers in refereed international journals and conference proceedings. He is a senior member of the IEEE and a member of the ACM SIGSPATIAL.

Min-Te Sun received his B.S. degree in Mathematics from National Taiwan University in 1991, the M.S. degree in Computer Science from Indiana University in 1995, and the Ph.D. degree in Computer and Information Science from the Ohio State University in 2002. Since 2008, he has been with the Department of Computer Science and Information Engineering at National Central University, Taiwan. His research interests include distributed algorithm design and wireless network protocol development.

Xiao Qin received the B.S. and M.S. degrees in Computer Science from the Huazhong University of Science and Technology, Wuhan, China, and the Ph.D. degree in Computer Science from the University of Nebraska-Lincoln, Lincoln, in 1992, 1999, and 2004, respectively. For three years, he was an Assistant Professor with the New Mexico Institute of Mining and Technology, Socorro, NM. Currently, he is an Associate Professor with the Department of Computer Science and Software Engineering, Auburn University, Auburn, AL. His research interests include parallel and distributed systems, storage systems, fault tolerance, real-time systems, and performance evaluation.

Hua Lu is an associate professor in the Department of Computer Science, Aalborg University, Denmark. He received the BSc and MSc degrees from Peking University, China, and the PhD degree in computer science from National University of Singapore. His research interests include database and data management, geographic information systems, and mobile computing. He has served as PC cochair or vice chair for ISA 2011, MUE 2011 and MDM 2012, demo chair for SSDBM 2014, and PhD forum cochair for MDM 2016.

Xunfei Jiang is an Assistant Professor in the Department of Computer Science at Earlham College. She received the B.S. and M.S. degrees in Computer Science from Huazhong University of Science and Technology (HUST), China, in 2004 and 2007. Then she joined Digital Video Networks Co., Ltd. in 2007 and Cisco Systems (Shanghai) Video Technology Co., Ltd. in 2010. She received the Ph.D. degree in the Department of Computer Science and Software Engineering at Auburn University in 2014. Her research interests include parallel and distributed systems, energy-efficient storage systems, thermal modeling, thermal-aware task scheduling and data placement, and hybrid data storage systems.

Affiliations

Ji Zhang[1] · Po-Wei Harn[2] · Wei-Shinn Ku[1] · Min-Te Sun[3] · Xiao Qin[1] · Hua Lu[4] · Xunfei Jiang[5]

Ji Zhang
jizhang@auburn.edu

Po-Wei Harn
poweiharn@iii.org.tw

Wei-Shinn Ku
weishinn@auburn.edu

Min-Te Sun
msun@csie.ncu.edu.tw

Xiao Qin
xqin@auburn.edu

Hua Lu
luhua@cs.aau.dk

1 Department of Computer Science and Software Engineering, Auburn University, Auburn, AL 36849, USA

2 Institute for Information Industry, Taipei, 106, Taiwan

3 Department of Computer Science and Information Engineering, National Central University, Taoyuan, 320, Taiwan

4 Department of Computer Science, Aalborg University, Aalborg, Denmark

5 Department of Computer Science, Earlham College, Richmond, IN 47374, USA