Geographic Quorum System Approximations

Paz Carmi* Shlomi Dolev† Sariel Har-Peled‡ Matthew J. Katz§
Michael Segal¶

November 4, 2004

Abstract

Quorum systems are a mechanism for obtaining fault-tolerance and efficient distributed systems. We consider *geographic quorum systems*; a geographic quorum system is a partition of a set $\mathcal{X}$ of sites in the plane (representing servers) into quorums (i.e. clusters) of size k. The distance between a point p and a cluster $\mathcal{C}$ is the Euclidean distance between p and the site in $\mathcal{C}$ that is the farthest from p.

We present a near linear time constant-factor approximation algorithm for partitioning $\mathcal{X}$ into clusters, such that, the maximal distance between a point in the underlying region and its closest cluster is minimized. Next, we describe a data-structure for answering (approximately) nearest-neighbor queries on such a clustering.

Finally, we study the problem of partitioning into clusters with an additional load-balancing requirement.

1 Introduction

A *quorum system* is an important abstraction used in distributed systems for achieving *fault-tolerance*, *availability* and *load balancing*, see [11, 13, 15, 12, 16, 5]. A quorum system is a collection of possibly intersecting subsets of servers, called here *quorums* or *clusters*. The

*Department of Computer Science, Ben-Gurion University of the Negev, Beer-Sheva 84105, Israel, carmip@cs.bgu.ac.il. Partially supported by grant no. 2000160 from the U.S.-Israel Binational Science Foundation, and by a Kreitman Foundation doctoral fellowship.

†Department of Computer Science, Ben-Gurion University of the Negev, Beer-Sheva 84105, Israel, dolev@cs.bgu.ac.il. Partially supported by IBM faculty award, NSF grant, the Israeli Ministry of Defense, the Israeli Ministry of Industry and Trade (STRIMM consortium), and Rita Altura trust chair in computer sciences.

‡Department of Computer Science, DCL 2111; University of Illinois; 1304 West Springfield Ave.; Urbana, IL 61801; USA; http://www.uiuc.edu/~sariel/; sariel@uiuc.edu. Work on this paper was partially supported by a NSF CAREER award CCR-0132901.

§Department of Computer Science, Ben-Gurion University of the Negev, Beer-Sheva 84105, Israel, matya@cs.bgu.ac.il. Partially supported by grant no. 2000160 from the U.S.-Israel Binational Science Foundation, and by the MAGNET program of the Israel Ministry of Industry and Trade (LSRT consortium).

¶Department of Communication Systems Engineering, Ben-Gurion University of the Negev, Beer-Sheva 84105, Israel, segal@cse.bgu.ac.il.

fault tolerance or *availability* of a quorum system is related to the number of server failures that it can tolerate. E.g., there must remain at least one cluster without faulty servers, for each type of operation. The *load* of a quorum system is usually determined by the number of clusters a server belongs to. We suggest an additional new measure for quorum systems, namely the *communication cost* of accessing a cluster, which is an important aspect in the design of distributed quorum systems. In the context of mobile computing, the new communication cost measure raises several problems in computational geometry, that are of significant interest independent of the quorum system application.

The settings considered here are related to subdivisions of the plane [5], such that the cluster that is used by a wandering client is determined by the the region in which the client is currently located. A *cluster* in our problems always consists of exactly k servers, and a server can only belong to one cluster. We seek an optimal partition of the servers into clusters according to a certain cost measure. For example, the partition should imply low energy consumption of a mobile wandering ad-hoc host, see [14, 6, 5]. Roughly speaking we would like to have a (geographically) close enough cluster for every point in the plane.

Once a partition into clusters is defined, a mobile host can write a value to the servers of a near-by cluster, knowing that a later read operation must access a representative of each cluster and thus, access a server in the particular cluster that the mobile host used for the write operation. The question of whether to use a near-by cluster for writing and an intersecting set of servers for reading, or vice versa, may depend on the frequency of the write and read operations (see [5]).

Our results. In this paper we introduce several new problems concerning geographic quorum systems. Besides their obvious relevance to the study of quorum systems, these problems are also interesting as purely geometric clustering problems. The first problem deals with the issue of constructing a good quorum system for a given set of servers and a service region R.

Geographic quorum partition. The task we address is to partition the set of n servers (represented by a set $\mathcal{X}$ of n points in the plane) into quorums (i.e., clusters), each of size k, such that, the cost associated with the partitioning is minimal. The cost $\mu(\mathcal{Q})$ associated with a partitioning $\mathcal{Q}$ is $\max_{p \in R} \mathbf{d}_{\mathcal{Q}}(p)$, where $\mathbf{d}_{\mathcal{Q}}(p)$ is the distance between p and the cluster of $\mathcal{Q}$ that is the closest to p, and the distance between p and a cluster $\mathcal{C}$ of $\mathcal{Q}$ is the *maximum* (Euclidean) distance between p and a point in $\mathcal{C}$ (see Figure 1). (This definition of the distance between a point and a cluster follows from the need to access each of the servers in a near-by cluster in a write operation, and from the independence of the servers in a cluster. It is the client's responsibility to access each of the servers in the cluster; the client cannot rely on possible connections between the members of a cluster.)

We present a 3-approximation for this problem. More precisely, we construct a quorum system $\mathcal{Q}$ (i.e., a partitioning as above), in $O(n(k + \log^2 n))$ time, such that, for *any* point $p \in R$, we have $\mathbf{d}_{\mathcal{Q}}(p) \leq 3\mathbf{d}_{\mathcal{Q}'}(p)$, where $\mathcal{Q}'$ is any quorum partition of $\mathcal{X}$. In particular, we have $\mu(\mathcal{Q}) \leq 3\mu(\text{opt})$, where opt is an optimal quorum system. If one is satisfied with a $(3 + \varepsilon)$-approximation, the running time improves to $O\big(n \log^2 n + n/(k\varepsilon^3) \log^2(1/\varepsilon)\big)$.

To facilitate this, we present a fast implementation of the incremental greedy algorithm, which repeatedly picks the smallest disc that contains k input points, and continues with

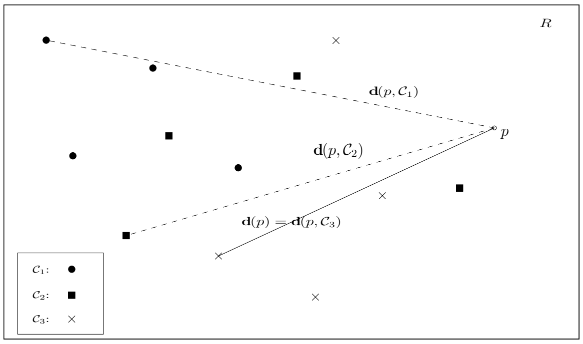

Figure 1: A partitioning into 3 clusters (c_1, c_2, c_3) of size 4, and the distance between point p and each of the clusters.

the remaining points. Our algorithm has near linear running time if one is satisfied with a $(1+\varepsilon)$-approximation of the smallest disc at each iteration, and might be of independent interest.

Nearest quorum queries. Given a quorum system $\mathcal{Q}$, we present a data-structure that answers queries of the form: which is the cluster that is closest (using the definition above) to the query location $q \in \mathbb{R}^2$. The natural data structure for such queries is the Voronoi diagram of the quorums; however, this diagram is too complex. We describe how to construct, in $O\left(n \log k + \frac{n}{k\varepsilon^2} \log \frac{1}{\varepsilon} \log \frac{n}{\varepsilon}\right)$ time, a data structure of size $O(n/(k\varepsilon^2) \log(1/\varepsilon))$ that supports such queries in $O(\log(n/\varepsilon))$ time per query, where $\varepsilon > 0$ is a prespecified parameter. The distance between q and the cluster returned is at most $(1+\varepsilon)\mathbf{d}_{\mathcal{Q}}(q)$.

Load balancing. Under the nearest quorum selection policy, some of the quorums might be over utilized, while others might be under utilized. This happens when the region in which a quorum $\mathcal{C}$ is selected (i.e., its Voronoi cell) is either especially large or especially small (assuming uniform distribution of the customers in the region R.) It is therefore natural to consider the following problem. Partition the set $\mathcal{X}$ into k-clusters, and assign to each cluster a region of responsibility, such that the load is divided evenly among the clusters, and the cost associated with the partition is minimal.

In other words we wish to partition $\mathcal{X}$ into k-clusters, to divide the region R into n/k subregions, each of area area$(R)/(n/k)$, and to assign to each cluster one of these subregions. All requests initiated by clients in a subregion are served by the cluster to which this subregion was assigned. The goal is to minimize the cost that is associated with the partition, where this cost is defined as before (except that now a point in R must access the cluster that is responsible for the region in which it lies, rather than the cluster that is closest to it). We present a constant-factor approximation algorithm for this problem.

The rest of the paper is organized as follows. The geographic quorum partition problem is studied in Section 2. A fast algorithm for computing the quorum partition is presented in Section 3. A data structure supporting approximate nearest quorum queries for a given partition is described in Section 4. Section 5 addresses the case in which a load-balanced quorum partition is required.

2 Geographic Quorum Partition

In this section we study the problem of partitioning the set of servers into clusters, each of size k, so as to minimize the cost of the partitioning. Let $\mathcal{X}$ be a set of points in the plane representing the servers. Let R be the region for which the servers are required to provide service. We assume R is a simple, connected and closed region (e.g., an axis-parallel rectangle).

The *distance* between a point p in R and a cluster $\mathcal{C} \subseteq \mathcal{X}$ is defined as the maximum (Euclidean) distance between p and a point of $\mathcal{C}$ (see Figure 1). That is

$$\mathbf{d}(p, \mathcal{C}) = \max_{c \in \mathcal{C}} \|pc\|,$$

where $\|pc\|$ is the Euclidean distance between p and c.

Let n denote the number of points in $\mathcal{X}$ and assume $n = km$, where $k \geq 2$ and $m \geq 2$ are integers. Let $\mathcal{Q}$ be a partitioning of $\mathcal{X}$ into m clusters, each of size k. The *cost* of $\mathcal{Q}$, denoted by $\mu(\mathcal{Q})$, is a positive real number that is defined as follows. For each point $p \in R$, let $\mathbf{d}_{\mathcal{Q}}(p)$ be the distance between p and the cluster of $\mathcal{Q}$ that is the closest to p (according to the distance defined above). Then $\mu(\mathcal{Q}) = \max_{p \in R} \mathbf{d}_{\mathcal{Q}}(p)$.

We wish to compute a partitioning of $\mathcal{X}$ into k-clusters (i.e., clusters of size k) with minimum cost. We define a specific partitioning (called mec) of $\mathcal{X}$ into k-clusters, and prove that it is a 3-approximation; that is, its cost is at most three times the cost of an optimal partitioning, and it can be computed in polynomial time. Actually, we prove a stronger claim stating that for *any* point $q \in R$, $\mathbf{d}_{\mathsf{mec}}(q)$ is at most 3 times $\mathbf{d}_{\mathcal{Q}}(q)$, where $\mathcal{Q}$ is any partitioning of $\mathcal{X}$ into k-clusters. We say that mec is a *3-competitive* quorum clustering of $\mathcal{X}$.

We now define the partitioning mec. Among all subsets of $\mathcal{X}$ of size k, let $\mathcal{C}$ be the subset whose smallest enclosing circle is minimal. We take $\mathcal{C}$ as our first cluster and repeat for the set $\mathcal{X} \setminus \mathcal{C}$ of remaining points, until we are left with an empty set of points.

Lemma 2.1 *The partitioning* mec *defined above is a* 3*-competitive quorum clustering of* $\mathcal{X}$*.*

Proof: Let q be a point in R and let $\mathcal{A}$ be the cluster of mec that is the closest to q. That is, $\mathbf{d}_{\mathsf{mec}}(q) = \mathbf{d}(q, \mathcal{A})$. Let $\mathcal{C}$ be the cluster of opt that is the closest to q. Then we have $\mathbf{d}(q, \mathcal{C}) \geq r_{\mathcal{C}}$, where $r_{\mathcal{C}}$ is the radius of the smallest enclosing circle of $\mathcal{C}$.

Assume that $\mathcal{C}$ is not one of the clusters of mec (since otherwise $\mathbf{d}_{\mathsf{mec}}(q)$ is at most $\mathbf{d}_{\mathrm{opt}}(q)$ and we are done). Consider the (at most k) clusters of mec that have a non-empty intersection with $\mathcal{C}$. For at least one of them, $\mathcal{A}'$, we must have $r_{\mathcal{A}'} \leq r_{\mathcal{C}}$, since otherwise, C would have been created during the construction of mec. Let x be a point in $\mathcal{C} \cap \mathcal{A}'$. Let x' be the point in $\mathcal{A}'$ that is the farthest from q. Then

$$\mathbf{d}(q, \mathcal{A}) \leq \mathbf{d}(q, \mathcal{A}') = \|qx'\|,$$

and

$$\|qx'\| \leq \|qx\| + \|xx'\| \leq \mathbf{d}(q, \mathcal{C}) + 2r_{\mathcal{A}'},$$

and since $r_{\mathcal{A}'} \leq \mathbf{d}(q, \mathcal{C})$, we obtain that

$$\mathbf{d}(q, \mathcal{A}) \leq 3\mathbf{d}(q, \mathcal{C}).$$

In particular, if q is the point that determines $\mu(\mathsf{mec})$, then we have $\mu(\mathsf{mec}) = \mathbf{d}(q, \mathcal{A}) \leq 3\mathbf{d}(q, \mathcal{C}) \leq 3\mu(\text{opt})$. ∎

The partitioning mec, for $k > 2$, can be computed by, e.g., repeatedly applying the recent algorithm of Har-Peled and Mazumdar for computing a smallest enclosing circle containing k points [10]. Their algorithm uses $O(n + k^2)$ space and its expected running time is $O(nk)$. For $k = 2$ the partitioning mec is computed by simply picking, at each iteration, the closest pair of points from the remaining points. This matching can be computed in $O(n \log n)$ time (see [3]). The following theorem summarizes our results.

Theorem 2.2 *One can compute a partitioning $\mathcal{Q}$ of $\mathcal{X}$ into k-clusters, such that, for any point $q \in R$, we have $\mathbf{d}_{\mathcal{Q}}(q) \leq 3\mathbf{d}_{\text{opt}}(q)$. For $k > 2$, $\mathcal{Q}$ can be computed in expected time $O(n^2)$ using $O(n + k^2)$ space. For $k = 2$, $\mathcal{Q}$ can be computed in time $O(n \log n)$ using linear space.*

3 A faster clustering algorithm

Let T be a compressed quadtree computed over a set $\mathcal{X}$ of n points; T can be computed in $O(n \log n)$ time [2]. We trim T by removing all nodes v, such that the parent of v has less than $c \cdot k$ points in its associated cell, where $c > 1$ is a constant to be specified shortly. In particular, a leaf of a trimmed quadtree is *legal* only if its parent contains at least $c \cdot k$ points. The trimmed quadtree is *legal* if all its leaves are legal. A point $x \in \mathcal{X}$ is stored in a leaf v of T, if $x \in r_v$, where r_v is the cell associated with v.

Our faster clustering algorithm (described below) maintains several trimmed compressed quadtrees, such that one of them fully contains the required smallest k-enclosing disc in one of its leaves. The algorithm repeatedly finds (an approximation of) this smallest disc, removes, from each of the quadtrees, the k points that are covered by this disc, and trims the quadtrees. This process ends after all points have been covered.

For now, consider a single trimmed compressed quadtree T, and let us assume that we are "lucky", and the smallest disc that contains k points is fully contained in one of the leaves of T. Clearly, every leaf v of T contains $|P_v| = O(k)$ points, and, therefore, we can in $O(|P_v|)$ time compute a constant factor approximation to the smallest disc that contains k points of P_v [10]. Applying this (almost) smallest disc computation to all the leaves of T, in advance, takes $O(n)$ time overall.

Let D_1 be the smallest k-enclosing disc that was precomputed in one of the leaves of T. By our assumption, D_1 is a constant factor approximation to the smallest disc that contains k points of $\mathcal{X}$. Denote by $\mathcal{C}_1$ the subset of k points covered by D_1, and take $\mathcal{C}_1$ to be the first cluster.

We continue with the remaining points $\mathcal{X}_2 = \mathcal{X} \setminus \mathcal{C}_1$. To this end, we update T by deleting from it the points of $\mathcal{C}_1$, and performing trimming if needed, so that T becomes legal again. Since all the points of $\mathcal{C}_1$ lie in a single leaf of T, all we need to do is to climb up in T, till we reach a node of T that its parent has at least ck points in its cell. Since every time we move one level up in T, the cardinality of the set in the associated cell increases by at least one, it follows that we need to climb at most $O(k)$ levels to reach this node w (clearly, we need to trim all the children of w from T, as they are not legal (i.e., their parents contain too few points of $\mathcal{X}_2$).

To facilitate this, we will maintain an external data-structure of orthogonal range searching with deletions. Such that, at every node, we simply perform a range searching query to decide how many points are in the cell associated with this node. This takes $O(\log^2 n)$ per query and update [1]. Thus, after $O(k \log^2 n)$ work, we found the node w and trimmed its children.

If we are lucky again, and the smallest disc containing k points of $\mathcal{X}_2$ is completely contained in the cell of one of the leaves of T, then we can again extract a constant approximation to it as described above, and repeat. Thus, assuming (unreasonably) that we are lucky throughout the execution of the algorithm, we get a clustering which is constant factor competitive in $O(n \log^2 n)$ time.

Let us consider now the case where the current disc D_i (which is, say, an α-approximation to the smallest disc covering k points of $\mathcal{X}_i$) and the associated cluster $\mathcal{C}_i$ are given to us, and we would like to update T by removing the points of $\mathcal{C}_i$ from it.

Observe that D_i intersects at most a constant number of leaves of T. Indeed, let L be the set of leaves of T that intersect D_i. Since the parent of each node in L contains at least ck points of $\mathcal{X}_i$, it follows that its side length must be at least $\sqrt{2c}\,\mathrm{radius}(D_i)/\alpha$, otherwise D_i would not be an α-approximation to the smallest disc that covers k points of $\mathcal{X}_i$ (indeed, such a node v contains a disc of radius $\mathrm{SideLen}(r_v)/\sqrt{2c}$ that contains k points assuming $c = i^2$ where i is an integer). Thus, let L' be the set of parents of nodes of L. Clearly, the regions that correspond to cells of nodes of L' are disjoint, and it follows that $|L'| = O(1)$ as they are all relatively large compared to D_i, and they all intersect D_i. Since every node of L' has a constant number of children, it follows that $|L| = O(1)$.

Thus, applying the algorithm described above for updating T to each leaf in L, implies that given $\mathcal{C}_i$ and D_i, we can update T to be the (trimmed) compressed quadtree of $\mathcal{X}_{i+1}$ in $O(k \log^2 n)$ time. Thus, we can maintain a trimmed quadtree throughout the clustering extraction process in $O(n \log^2 n)$ time.

We are still left with the task of guaranteeing that luck is on our side. A natural approach would be to use several quadtrees, and to observe that it is sufficient to be lucky in one of them. In particular, one can generate such a set of compressed quadtrees by using random translations for the location of the starting bounding square of the quadtree. But in fact, no randomization is necessary, as a deterministic scheme is known. Indeed, assume that the point set is contained in the cube $C = [0, 1/\sqrt{d}]^d$ (this can be easily guaranteed by scaling the point set), and that the square used to compute the quadtree is $S = [-1, 1]^d$, then the analysis of Chan [4, Lemma 3.3] implies the following:

Lemma 3.1 *Assume d is even, and let $v^i = \left(\frac{i}{d+1}, \ldots, \frac{i}{d+1}\right)$ for $i = 0, \ldots, d$, $\mathcal{X}$ be a set of n points in $\mathbb{R}^d$ contained inside $C = [0, 1/\sqrt{d}]^d$, and $T_0, \ldots, T_d$ be the quadtrees of $\mathcal{X}$ having $v^0 + [-1, 1]^d, \ldots, v^d + [-1, 1]^d$ as the starting region for their root, respectively.*

Then, for any point $p \in C$, and $r \le 1$, such that $\mathrm{ball}(p, r)$ contains a point of $\mathcal{X}$, there exists a cell R in one of those $d+1$ quadtrees, such that $\mathrm{ball}(p, r) \subseteq R$, and furthermore, the side length of R is bounded by $4(d+1)r$.

This lemma implies that we can maintain 3 quadtrees (since $d = 2$ in our setting), and we are guaranteed to be lucky in one of them. Indeed, if the disc D_i, of radius r_i, is an α-

approximation to the smallest disc that contains k points of $\mathcal{X}_i$, then by a packing argument, the cell that contains D_i in its interior, contains at most $\lceil (4(d+1)r_i)^2/(r_i/\alpha)^2 \rceil\, k = O(k)$ points of $\mathcal{X}_i$ (otherwise, there would be a disc of radius smaller than r_i/α that contains k points of $\mathcal{X}_i$). Thus, setting $c = \lceil 16(d+1)^2\alpha^2 \rceil$, we are guaranteed to be lucky at one of the 3 trimmed quadtrees that we are maintaining. Indeed, by Lemma 3.1, the required disc is contained inside one of the nodes of the (original) quadtrees we maintain, and this node is not too big, and does not contain too many points. As such, it is going to be a leaf in the corresponding trimmed quadtree.

Note, that at every stage, the algorithm outputs a disc which is a constant factor approximation to the optimal disc containing k points, for the uncovered points. It is easy to check, that this property is sufficient to guarantee a constant factor approximation, by plugging it into the proof of Lemma 2.1. We omit the tedious but straightforward details.

Thus, we just proved the following:

Lemma 3.2 *Given a set $\mathcal{X}$ of n points in the plane and a parameter k, one can compute, in $O(n \log^2 n)$ time, an $O(1)$-competitive quorum clustering of $\mathcal{X}$.*

Computing exactly the smallest enclosing disc of k points, given $O(k)$ points, takes $O(k^2)$ time [10]. Also, given an $\varepsilon > 0$, one can compute a $(1+\varepsilon)$-approximation to the smallest disc containing k points, when given $O(k)$ points, in $O\big(k + (1/\varepsilon^3)\log^2(1/\varepsilon)\big)$ time [10]. Plugging this into the above algorithm we get the following:

Theorem 3.3 *Given a set $\mathcal{X}$ of n points in the plane and parameters k and $\varepsilon > 0$, one can compute:*

1. *In $O(n(k + \log^2 n))$ time, a 3-competitive quorum clustering of $\mathcal{X}$.*
2. *In $O\big(n \log^2 n + \frac{n}{k\varepsilon^3} \log^2 \frac{1}{\varepsilon}\big)$ time, a $(3+\varepsilon)$-competitive quorum clustering of $\mathcal{X}$.*

Note, that the result of Theorem 3.3, can be further improved by using a better auxiliary data-structure for orthogonal range searching, or avoiding it all together. Since this only results in a minor improvement in the running time of the algorithm and substantially complicates the description of the algorithm, we did not pursue this minor improvement.

4 Nearest Quorum Queries

Assume we already have a partitioning $\mathcal{Q}$ of a point set $\mathcal{X}$ into k-clusters (i.e., quorums). We now consider the following problem. Construct a data structure, so that given a query point $q \in \mathbb{R}^2$ (representing an ad-hoc client) one can quickly find the cluster (i.e., quorum) $\mathcal{C}^*$ of $\mathcal{Q}$ that is closest to q. (Recall that the distance $\mathbf{d}(q, \mathcal{C})$ between q and a cluster $\mathcal{C}$ is the maximum (Euclidean) distance between q and a point of $\mathcal{C}$.)

Given a prescribed parameter $\varepsilon > 0$, we show in this section, how to preprocess $\mathcal{Q}$, such that given any query point q in the plane, one can quickly decide what is (approximately) the nearest cluster of $\mathcal{Q}$ to q.

Assume that $\mathcal{Q}$ partitions $\mathcal{X}$ into the clusters $\mathcal{C}_1, \ldots, \mathcal{C}_m$, and recall that $\mathbf{d}_{\mathcal{Q}}(q) = \min_{\mathcal{C}_i \in \mathcal{Q}} \mathbf{d}(q, \mathcal{C}_i)$. Thus, the function $\mathbf{d}_{\mathcal{Q}}(q)$ can be interpreted as a minimization diagram of the maximization diagrams of the clusters. This min of max structure makes finding the nearest neighbor cumbersome, and our first task is to replace the maximization by minimization. Namely, we want to replace the furthest neighbor Voronoi diagram of each cluster by a "regular" minimization Voronoi diagram.

Definition 4.1 A pair $\mathcal{S} = (S, w)$ is a *weighted set of points* if $S = \{p_1, \ldots, p_m\}$ is a finite set of points in $\mathbb{R}^d$, and $w(\cdot)$ is a function assigning non-negative weights to the points of S. We define the distance of a point p from the point p_i to be $V_{(p_i,w(p_i))}(p) = \|pp_i\| + w(p_i)$. We define $V_{\mathcal{S}}(p) = \min_{i=1}^{m} V_{(p_i,w(p_i))}(p)$. The function $V_{\mathcal{S}}(p)$ induces a natural subdivision $\mathcal{V}_{\mathcal{S}}$ of $\mathbb{R}^d$ into cells, known as the (additive) *weighted Voronoi diagram* of $\mathcal{S}$, such that the ith cell is the locus of all points closest to p_i in this distance function. As is well known, in the planar case, $\mathcal{V}_{\mathcal{S}}$ has complexity $O(m)$, and it can be computed in $O(m \log m)$ time (see [8]).

Lemma 4.2 ([9]) *Any furthest neighbor Voronoi diagram of points in $\mathbb{R}^d$, can be ε-approximated by a (nearest neighbor) weighted Voronoi diagram, having $O((1/\varepsilon^d)\log(1/\varepsilon))$ sites.*

We shortly outline how to compute this approximation, for a point set U, and its furthest neighbor Voronoi diagram $\mathcal{F}_U$: (i) Find a point x for which the value of the furthest Voronoi diagram of U is minimal. This can be easily done by computing the smallest disc that contains U. This takes linear time using low dimensional linear programming techniques [17]. (ii) Let $l = \mathcal{F}_U(x)$ be the distance of the furthest neighbor of x in U. Given l and x, the locations of the approximation sites are now determined, and let S be this set of $O(1/\varepsilon^2 \log 1/\varepsilon)$ sites. Since this is just an exponential grid, this can be done in linear time in the output size; namely, $O(1/\varepsilon^2 \log 1/\varepsilon)$ time. (iii) Next, we compute the furthest neighbor Voronoi diagram of U, and for each point s of S, we compute its weight $\mathcal{F}_U(s)$, by performing a point-location query in the furthest neighbor Voronoi diagram of $\mathcal{F}_U$ (i.e., $w(s)$ is set to $\mathcal{F}_U(s)$). Computing the Voronoi diagram of $\mathcal{F}_U$ takes $O(|U| \log |U|)$ time, and performing a point-location query takes $O(\log |U|)$ time. Thus, this takes $O(|U| \log |U| + \varepsilon^{-2}(\log U) \log 1/\varepsilon)$

The resulting weighted additive Voronoi diagram (S, w) is the required approximation. We summarize this in the following lemma:

Lemma 4.3 *Given a cluster C of k points in the plane and a parameter $\varepsilon > 0$, one can compute, in $O(k \log k + (\log k)/\varepsilon^2 \log 1/\varepsilon)$ time, an additive weighted Voronoi diagram $\mathcal{V}_C$, with $O(1/\varepsilon^2 \log(1/\varepsilon))$ sites, such that $\mathcal{V}_C$ is a $(1+\varepsilon)$ approximation to the furthest neighbor Voronoi diagram $\mathcal{F}_C$. Formally, for any point $x \in \mathbb{R}^2$, we have $\mathcal{F}_C(x) \le \mathcal{V}_C(x) \le (1 + \varepsilon)\mathcal{F}_C(x)$.*

Let us apply Lemma 4.3 to each cluster $\mathcal{C}_i$ in $\mathcal{Q}$, and let $\mathcal{V}_i$ be the resulting weighted additive Voronoi diagram and $\mathcal{S}_i$ be the weighted set that induces it, for $i = 1, \ldots, m$. Clearly, an approximate nearest neighbor query $q \in \mathbb{R}^2$ in the quorum clustering $\mathcal{Q}$, can be resolved by finding the i that realizes

$$\min_i \mathcal{V}_i(q) = \min_i \min_{(s,w)\in\mathcal{S}_i} (\|sq\| + w) = \min_{(s,w)\in\bigcup_i S_i} (\|sq\| + w),$$

but this is exactly the value of the weighted additive Voronoi diagram of $\mathcal{S} = \cup_i \mathcal{S}_i$ at the point q. Thus, if we precompute the diagram $\mathcal{V}_{\mathcal{S}}$, then answering approximate nearest neighbor queries on $\mathcal{Q}$, is equivalent to performing a point-location query in $\mathcal{V}_{\mathcal{S}}$. We summarize the result:

Theorem 4.4 *Let $\varepsilon > 0$ be a parameter, and $\mathcal{Q}$ be a given quorum clustering of a set $\mathcal{X}$ of n points in the plane, where each cluster is of cardinality k. One can preprocess $\mathcal{Q}$ in $O\big(n \log k + \frac{n}{k\varepsilon^2} \log \frac{1}{\varepsilon} \log \frac{n}{\varepsilon}\big)$ time and $O(n/(k\varepsilon^2) \log(1/\varepsilon))$ space, such that given a query point q, one can decide in $O(\log(n/\varepsilon))$ time what is the $(1+\varepsilon)$-approximate nearest neighbor cluster to q among the clusters of $\mathcal{Q}$.*

Proof: We just verify the time and space bounds. Computing the approximation to each cluster takes $O(k \log k + (\log k)/\varepsilon^2 \log 1/\varepsilon)$ time. Once this approximation is computed, we have to compute the additive Voronoi diagram of all the approximations together. Since there are $O(n/(k\varepsilon^2) \log(1/\varepsilon))$ sites, this takes $O\big(\frac{n}{k\varepsilon^2} \log \frac{1}{\varepsilon} \log \frac{n}{k\varepsilon}\big)$ time. Thus, the overall running time is

$$O\left(\frac{n}{k}\left(k \log k + \frac{\log k}{\varepsilon^2} \log \frac{1}{\varepsilon}\right) + \frac{n}{k\varepsilon^2} \log \frac{1}{\varepsilon} \log \frac{n}{k\varepsilon}\right) = O\left(n \log k + \frac{n}{k\varepsilon^2} \log \frac{1}{\varepsilon} \log \frac{n}{\varepsilon}\right).$$

As for the space requirement, we build a point-location data structure for $O(n/(k\varepsilon^2)$ $\log(1/\varepsilon))$ sites, and this is the amount of space needed for such a data structure. ■

5 Load Balancing

In this section we add a load balancing requirement. That is, we would like to partition the set $\mathcal{X}$ into k-clusters and to assign to each cluster a region of responsibility, such that the load is divided evenly among the clusters, and the cost associated with the partition is minimal. In other words we wish to partition $\mathcal{X}$ into k-clusters, to divide the region R into m subregions, each of area area$(R)/m$, and to assign to each cluster one of these subregions (where each subregion is assigned to exactly one cluster). All requests initiated by clients in a subregion are served by the cluster to which this subregion was assigned. The goal is to minimize the cost that is associated with the partition.

Let us re-formulate the problem more precisely. We assume that the number m of clusters is equal to l^2, for some integer $l \geq 2$. We also assume (for convenience only) that the underlying region R is a square. (This assumption is not necessary; all subsequent results hold for any rectangle R that can be divided into m squares of equal size. Thus the aspect ratio of R can be as large as m.) We study the following problem. Partition $\mathcal{X}$ into m clusters $\mathcal{C}_1, \ldots, \mathcal{C}_m$, each of size k, and divide R into m regions $R_1, \ldots, R_m$, each of area area$(R)/m$, and associate region R_i with cluster $\mathcal{C}_i$, such that the cost of the partition is minimal.

The cost of the partition is defined as before, except that now a point in R must access the cluster that is responsible for the region in which it lies, rather than access the cluster that is closest to it. That is, the cost of the partition is $max_i\, \mu(\mathcal{C}_i)$, where $\mu(\mathcal{C}_i)$ is the maximum distance between a point in R_i and a point in $\mathcal{C}_i$. We also define the cost $\mu(x)$ of $x \in \mathcal{C}_i$ as the maximum distance between a point in R_i and x, and notice that $\mu(\mathcal{C}_i) = max_{x \in \mathcal{C}_i}\, \mu(x)$.

We first describe a simple algorithm that computes a load-balanced partition; we call the partition computed by the algorithm GRID-MIN-MAX. We then prove that GRID-MIN-MAX is a $(1+\frac{3}{2}\sqrt{2\pi})$-approximation, meaning that the ratio between the cost of GRID-MIN-MAX and the cost of an optimal load-balanced partition is at most $(1+\frac{3}{2}\sqrt{2\pi})$. The running time of our algorithm is $O(n^{1.5}\log n)$.

The algorithm divides R into $m = l^2$ squares of equal size. Let $\mathcal{S}$ denote the set of squares. Let o_i be the center point of square $\sigma_i \in \mathcal{S}$, for $i = 1,\dots,m$, and let Φ be the multiset in which each center point o_i occurs exactly k times. Let $G = (\mathcal{X}, \Phi; E)$ be the complete bipartite graph with vertex sets $\mathcal{X}$ and Φ. We associate weights with the edges in E; the weight of an edge (x_i, o_j), for $x_i \in \mathcal{X}$ and $o_j \in \Phi$, is simply the Euclidean distance between the two points. We now compute a bottleneck matching in G, i.e., a matching in which the weight of the heaviest edge is minimal. This matching defines the partition into clusters. All points matched to o_1 consist of the first cluster whose region of responsibility is the square σ_1, etc. Using the algorithm of Efrat et al.[7] this can be done in $O(n^{1.5}\log n)$ time.

5.1 GRID-MIN-MAX is a constant-factor approximation

Let opt1 denote an optimal partition (where region R_i is assigned to cluster $\mathcal{C}_i$). We prove that the partition that was obtained (i.e., GRID-MIN-MAX) is a constant-factor approximation of opt1. We first obtain from opt1 a new partition, grid1, that also uses the squares in $\mathcal{S}$, and show that grid1 is a $(1+\sqrt{2\pi})$-approximation of opt1. We then use this intermediate result in order to show that GRID-MIN-MAX is a $(1+\frac{3}{2}\sqrt{2\pi})$-approximation of opt1.

Define a bipartite graph $G = (\mathcal{Q}, \mathcal{S}; E)$, where $\mathcal{Q} = \{\mathcal{C}_1,\dots,\mathcal{C}_m\}$ is the set of clusters of opt1, and there is an edge between $\mathcal{C}_i \in \mathcal{Q}$ and $\sigma_j \in \mathcal{S}$ if $R_i \cap \sigma_j \neq \emptyset$. Hall's matching theorem [18] gives a necessary and sufficient condition for G to contain a perfect matching. According to Hall's matching theorem, G contains a perfect matching if and only if for any subset $\mathcal{Q}'$ of $\mathcal{Q}$ we have $|N(\mathcal{Q}')| \geq |\mathcal{Q}'|$, where $N(\mathcal{Q}')$ is the set of squares in $\mathcal{S}$ that are connected by an edge to a cluster in $\mathcal{Q}'$. However, this condition trivially holds in our case, since we need at least $|\mathcal{Q}'|$ squares of $\mathcal{S}$ in order to cover a region of area $|\mathcal{Q}'|\text{area}(R)/m$. We thus associate the squares in $\mathcal{S}$ with the clusters in $\mathcal{Q}$ to obtain the partition grid1 by computing any perfect matching in G.

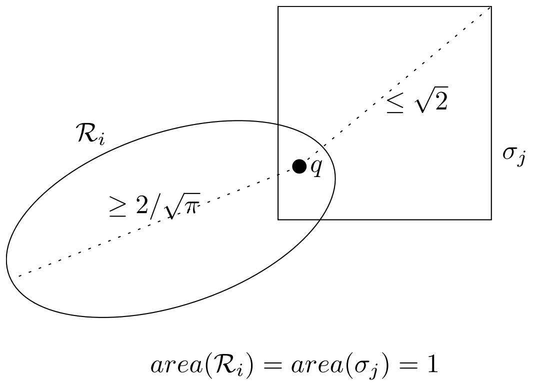

Figure 2: grid1 is a constant-factor approximation.

We show that grid1 is a $(1+\sqrt{2\pi})$-approximation of opt1. Let $\mathcal{C}'$ be a cluster of grid1 such that the cost of grid1 is $\mu(\mathcal{C}')$, and let $x' \in \mathcal{C}'$ be a point of $\mathcal{C}'$ such that $\mu(\mathcal{C}') = \mu(x')$.

It is enough to show that

$$\frac{\mu_{\mathsf{grid1}}(x')}{\mu_{\mathsf{opt1}}(x')} \le 1 + \sqrt{2\pi}\ ,$$

since the cost of opt1 is at least $\mu_{\mathsf{opt1}}(x')$.

We show that the inequality above is true for any $x \in \mathcal{X}$ and therefore also for x'. Let $x \in \mathcal{X}$ and let $\sigma_j \in \mathcal{S}$ be the square that is assigned to x's cluster in grid1. Recall that the region assigned to x's cluster in opt1 is R_i (assuming $x \in \mathcal{C}_i$). Let $q \in R_i \cap \sigma_j$. Assume w.l.o.g. that area$(R_i) =$ area$(\sigma_j) = 1$, then the diameter of R_i is at least $2/\sqrt{\pi}$ (see Figure 2). Therefore, $\mu_{\mathsf{opt1}}(x)$ is, on the one hand, at least $1/\sqrt{\pi}$, and, on the other hand, at least $\|xq\|$. As to $\mu_{\mathsf{grid1}}(x)$ we have $\mu_{\mathsf{grid1}}(x) \le \|xq\| + \sqrt{2}$. Now, if $\|xq\| \le 1/\sqrt{\pi}$, then using the first inequality for $\mu_{\mathsf{opt1}}(x)$ we obtain that

$$\frac{\mu_{\mathsf{grid1}}(x)}{\mu_{\mathsf{opt1}}(x)} \le \frac{\|xq\| + \sqrt{2}}{1/\sqrt{\pi}} \le 1 + \sqrt{2\pi},$$

and, if $\|xq\| > 1/\sqrt{\pi}$, then using the second inequality for $\mu_{\mathsf{opt1}}(x)$ we obtain that

$$\frac{\mu_{\mathsf{grid1}}(x)}{\mu_{\mathsf{opt1}}(x)} \le \frac{\|xq\| + \sqrt{2}}{\|xq\|} \le 1 + \sqrt{2\pi}.$$

We conclude that in both cases the ratio $\mu_{\mathsf{grid1}}(x)/\mu_{\mathsf{opt1}}(x) \le 1 + \sqrt{2\pi}$, and therefore grid1 is a $(1 + \sqrt{2\pi})$-approximation of opt1.

We use this intermediate result to show that GRID-MIN-MAX is a constant-factor approximation of opt1. Indeed the cost of GRID-MIN-MAX is at most the length of the longest edge in the bottleneck matching (between $\mathcal{X}$ and Φ) that defined GRID-MIN-MAX plus $\sqrt{2}/2$. But the latter value does not exceed the cost of grid1 + $\sqrt{2}/2$. (Since otherwise the matching between $\mathcal{X}$ and Φ induced by grid1 would be better than the bottleneck matching defining GRID-MIN-MAX.) We conclude that $\mu(\text{GRID-MIN-MAX}) \le \mu(\mathsf{grid1}) + \sqrt{2}/2$. Therefore

$$\frac{\mu(\text{GRID-MIN-MAX})}{\mu(\mathsf{opt1})} \le \frac{\mu(\mathsf{grid1}) + \sqrt{2}/2}{\mu(\mathsf{opt1})} \le 1 + \sqrt{2\pi} + \frac{\sqrt{2}/2}{\mu(\mathsf{opt1})} \le 1 + \frac{3}{2}\sqrt{2\pi}\ .$$

The following theorem summarizes the main result of this section.

Theorem 5.1 *A load-balanced partition that is a $(1 + \frac{3}{2}\sqrt{2\pi})$-approximation of* opt1 *can be computed in $O(n^{1.5} \log n)$ time.*

References

[1] P. K. Agarwal. Range searching. In J. E. Goodman and J. O'Rourke, editors, *Handbook of Discrete and Computational Geometry*, chapter 31, pages 575–598. CRC Press LLC, Boca Raton, FL, 1997.

[2] S. Arya, D. M. Mount, N. S. Netanyahu, R. Silverman, and A. Y. Wu. An optimal algorithm for approximate nearest neighbor searching in fixed dimensions. *J. Assoc. Comput. Mach.*, 45(6), 1998.

[3] S. N. Bespamyatnikh. An optimal algorithm for closest pair maintenance. In *Proc. 11th Annu. ACM Sympos. Comput. Geom.*, pages 152–161, 1995.

[4] T. M. Chan. Approximate nearest neighbor queries revisited. *Discrete Comput. Geom.*, 20:359–373, 1998.

[5] S. Dolev, S. Gilbert, N. A. Lynch, A. Shvartsman, and J. L. Welch. Geoquorums: Implementing atomic memory in ad hoc networks. In *Proc. of the 17th Int. Symp. Dist. Comp.*, 2003.

[6] S. Dolev, E. Schiller, and J. Welch. Walk for self-stabilizing group communication in ad-hoc networks. In *Proc. 21st IEEE Symp. Reliable Dist. Sys.*, pages 70–79, 2002.

[7] A. Efrat, A. Itai, and M. J. Katz. Geometry helps in bottleneck matching and related problems. *Algorithmica*, 31(1):1–28, 2001.

[8] S. J. Fortune. A sweepline algorithm for Voronoi diagrams. *Algorithmica*, 2:153–174, 1987.

[9] S. Har-Peled. Constructing approximate shortest path maps in three dimensions. *SIAM J. Comput.*, 28(4):1182–1197, 1999.

[10] S. Har-Peled and S. Mazumdar. Fast algorithms for computing the smallest k-enclosing disc. In *Proc. 11th Annu. European Sympos. Algorithms*, volume 2832 of *Lect. Notes in Comp. Sci.*, pages 278–288. Springer-Verlag, 2003.

[11] M. P. Herlihy. Dynamic quorum adjustment for partitioned data. *ACM Transactions on Database Systems*, 12(2):170–194, June 1987.

[12] D. Malkhi and M. Reiter. Byzantine quorum systems. *Distributed Computing*, 11(4):203–213, 1998.

[13] M. Naor and A. Wool. The load, capacity, and availability of quorum systems. *SIAM J. Comput.*, 27(2):423–447, 1998.

[14] J. C. Navas and T. Imielinski. GeoCast – geographic addressing and routing. In *Mobile Computing and Networking*, pages 66–76, 1997.

[15] D. Peleg and A. Wool. The availability of quorum systems. *Information and Computation*, 123(2):210–223, 1995.

[16] I. Stojmenovic and P. E. V. Pena. A scalable quorum based location update scheme for routing in ad hoc wireless networks. Technical report, Computer Science, SITE, University of Ottawa, December 1999. TR-99-11.

[17] E. Welzl. Smallest enclosing disks (balls and ellipsoids). In H. Maurer, editor, *New Results and New Trends in Computer Science*, volume 555 of *Lect. Notes in Comp. Sci.*, pages 359–370. Springer-Verlag, 1991.

[18] D. B. West. *Intorudction to Graph Theory.* Prentice Hall, 2ed edition, 2001.