ELSEVIER

Parallel Computing 28 (2002) 1477–1500

PARALLEL COMPUTING

www.elsevier.com/locate/parco

Probabilistic methods for centroidal Voronoi tessellations and their parallel implementations

Lili Ju [a,1], Qiang Du [b,2], Max Gunzburger [a,*,3]

[a] *Department of Mathematics, Iowa State University, 400 Carver Hall, Ames, IA 50011-2064, USA*
[b] *Department of Mathematics, Pennsylvania State University, University Park, PA 16802, USA*

Received 5 May 2001; received in revised form 16 November 2001; accepted 30 November 2001

Abstract

Centroidal Voronoi tessellations (CVTs) are Voronoi tessellations of a region such that the generating points of the tessellations are also the centroids of the corresponding Voronoi cells. In this paper, some probabilistic methods for determining CVTs and their parallel implementations on distributed memory systems are presented. By using multi-sampling in a new probabilistic algorithm we introduce, more accurate and efficient approximations of CVTs are obtained without the need to explicit construct Voronoi diagrams. The new algorithm lends itself well to parallelization, i.e., near prefect linear speed up in the number of processors is achieved. The results of computational experiments performed on a CRAY T3E−600 system are provided which illustrate the superior sequential and parallel performance of the new algorithm when compared to existing algorithms. In particular, for the same amount of work, the new algorithms produce significantly more accurate CVTs.

Keywords: Centroidal Voronoi tessellations; Probabilistic algorithms; Parallel implementations; Rejection method

* Corresponding author. Tel.: +1-515-294-1752; fax: +1-515-295-5454.
E-mail addresses: jllwyb@iastate.edu (L. Ju), qdu@math.psu.edu (Q. Du), gunzburg@iastate.edu (M. Gunzburger).

[1] Supported in part by Silicon Graphics Inc. and the National Science Foundation under grant number CCR-9988303.
[2] Supported by the National Science Foundation under grant number CCR-9988303 and by grants from the HKRGC and HKUST.
[3] Supported by the National Science Foundation under grant number CCR-9988303.

0167-8191/02/$ - see front matter
PII: S0167-8191(02)00151-5

1. Introduction

Given a set of points (called generators) and a distance function, Voronoi tessellations are subdivisions of another set of points into subsets such that the points in each subset are closest, with respect to the given distance function, to one of the generators than to any of the other generators. Given a density function, one can also determine the center of mass of each of the subsets making up the Voronoi tessellation. In general, the location of the generators of the Voronoi tessellation do not coincide with the centers of mass of the resulting Voronoi subsets. Centroidal Voronoi tessellations (CVTs) are special Voronoi tessellations of a region such that the generating points of the tessellations are also the centers of mass of the corresponding Voronoi cells.

CVTs are useful in very diverse applications, including, among others, data compression, clustering analysis, cell biology, territorial behavior of animals, optimal allocation of resources, grid generation, and meshless computing (see, e.g., [4–6,14]). As a result, algorithms for the efficient and accurate construction of CVTs are of substantial interest. Existing algorithms for this purpose fall into two classes: deterministic and probabilistic. In both cases, the algorithms are iterative in character, i.e., they involve the repeated updating of the positions of a set of points until they converge to the positions of the generators of a CVT. Known deterministic algorithms are efficient from the standpoint of the number of iterations, but are very inefficient with respect to the cost per iteration. In general, they require the explicit construction of Voronoi tessellations and the approximate evaluation of multi-dimensional integrals. The situation is reversed for known probabilistic algorithms. The cost per iteration is minute, with neither the construction of Voronoi tessellations nor the approximate evaluation of multi-dimensional integrals being required. However, the number of iterations required is usually huge. Furthermore, due to the large amount of inter-processor communications necessary, existing probabilistic algorithms are not amenable to parallel implementations.

In this paper, a new probabilistic algorithm for the construction of CVTs is developed. The new algorithm retains much of the cost-per-iteration efficiency of existing probabilistic algorithms; in particular, neither the construction of Voronoi tessellations nor the approximate evaluation of multi-dimensional integrals are required. However, compared to existing algorithms, the new algorithm requires much fewer iterations for convergence and involves a total work that, for the same quality of results, is much lower. Furthermore, the new algorithm lends itself well to parallel implementations.

The plan of the rest of the paper is as follows. In Section 2, CVTs are precisely defined and their characterization as minimizers of an "energy" functional is described. In Section 3, we briefly discuss the generation of random numbers according to a non-uniform probability density function; this is a necessary ingredient in the definition of probabilistic algorithms. Then, in Section 4, we discuss known and new sequential probabilistic methods for determining CVTs and compare their performance. In Section 5, we consider parallel implementations of the methods examined in Section 4 and study their performance through some parallel computational

experiments using message passing interface (MPI) [13]. Finally, concluding remarks are given in Section 6.

2. Centroidal Voronoi tessellations

Let $|\cdot|$ denote the Euclidean norm in $\mathbb{R}^N$. Given an open set $\Omega \subset \mathbb{R}^N$ and a set of points $\{\mathbf{z}_i\}_{i=1}^k$ belonging to $\overline{\Omega}$, let

$$V_i = \{\mathbf{x} \in \Omega \mid |\mathbf{x} - \mathbf{z}_i| < |\mathbf{x} - \mathbf{z}_j| \quad \text{for } j = 1, \ldots, k, \ \ j \neq i\} \quad i = 1, \ldots, k. \tag{1}$$

Clearly, we have

$$V_i \cap V_j = \emptyset \quad \text{for } i \neq j \quad \text{and} \quad \cup_{i=1}^k \overline{V}_i = \overline{\Omega}.$$

The set $\{V_i\}_{i=1}^k$ is referred to as a *Voronoi tessellation* or *Voronoi diagram* of Ω, the members of the set $\{\mathbf{z}_i\}_{i=1}^k$ are referred to as *generating points* or *generators*, and each V_i is referred to as the *Voronoi region* or *Voronoi cell* corresponding to $\mathbf{z}_i$. It is also well known that the Voronoi regions are polyhedra and they are very useful in a number of applications (see, e.g., [14]).

Given a density function $\rho(\mathbf{x}) \geqslant 0$ defined on $\overline{\Omega}$, then for each Voronoi region V_i, we define the *mass centroid* of *center of mass* $\mathbf{z}_i^*$ of V_i by

$$\mathbf{z}_i^* = \frac{\int_{V_i} \mathbf{x}\rho(\mathbf{x})\,\mathrm{d}\mathbf{x}}{\int_{V_i} \rho(\mathbf{x})\,\mathrm{d}\mathbf{x}} \quad \text{for } i = 1, \ldots, k. \tag{2}$$

We call the tessellation defined by (1) a CVT if and only if

$$\mathbf{z}_i = \mathbf{z}_i^* \quad \text{for } i = 1, \ldots, k,$$

i.e., the points $\mathbf{z}_i$'s which serve as the generators associated with the Voronoi regions V_i's are the mass centroids of those regions. Such tessellations are of use in very diverse applications, including data compression, clustering analysis, cell biology, territorial behavior of animals, optimal allocation of resources, grid generation, and meshless computing (see, e.g., [4–6,14]).

An arbitrary choice of generating points $\{\mathbf{z}_i\}_{i=1}^k$ in a region are not, in general, also the mass centroids of the corresponding Voronoi regions. As a result, one is left with the following construction problem: *given a region $\Omega \subset \mathbb{R}^N$, a positive integer k, and a density function $\rho(\mathbf{x})$ defined for $\mathbf{x} \in \overline{\Omega}$, determine a k-point CVT of Ω with respect to the given density function*. Note that, in general, CVTs of a given set are not uniquely defined (see [4]).

Let

$$\mathscr{K}(\{\mathbf{z}_i\}_{i=1}^k, \{V_i\}_{i=1}^k) = \sum_{i=1}^k \int_{V_i} \rho(\mathbf{x})|\mathbf{x} - \mathbf{z}_i|^2\,\mathrm{d}\mathbf{x}, \tag{3}$$

where $\{V_i\}_{i=1}^k$ is a tessellation of Ω and $\{\mathbf{z}_i\}_{i=1}^k$ are points in $\overline{\Omega}$. We refer to $\mathscr{K}$ as the *energy*; in the statistics literature it is called the *variance* or *cost*. It was proved that a necessary condition for $\mathscr{K}$ to be minimized is that $\{\mathbf{z}_i, V_i\}_{i=1}^k$ is a CVT of Ω (see [4]).

Since, in general practice, CVTs can only be approximately constructed, the energy is often used to monitor the quality of the results.

3. Determining random numbers from non-uniform densities

Random sampling points corresponding to a non-uniform density function may be determined from uniformly distributed points in a variety of ways. For example, in one dimension, given the interval (a,b) and the density function $\rho(x)$ defined on $[a,b]$, a random point x in $[a,b]$ may be determined from

$$\frac{\int_a^x \rho(s)\,\mathrm{d}s}{\int_a^b \rho(s)\,\mathrm{d}s} = X,$$

where X is a point in $[0,1]$ sampled according to a uniform distribution. If many points need be sampled, this classical procedure is very computationally intensive, especially in higher dimensions, due to the need to do repeated numerical integrations. It is also difficult to apply to complex geometries.

An alternate, completely probabilistic procedure is the *rejection method* [16]. In this procedure, a random point x in $[a,b]$ is determined by first sampling two random points X' and U with constant density in $[0,1]$ and then setting $X = a + (b-a)X'$. If $U < \rho(X)/\widehat{\rho}$, where $\widehat{\rho} = \max_{x\in[a,b]} \rho(x)$, we let $x = X$; otherwise, we begin again. Although we may need to call the random number generator many times (the number of times depends on the density function $\rho(x)$), the total computation time is in general trivial compared to the classical procedure using numerical integrations.

The rejection method can be extended to higher dimensions. For example, for two-dimensional domains, one can use the following completely probabilistic procedure (in the sense that no deterministic integrations are involved) for generating random points. Given the domain $\Omega \subset \mathbb{R}^2$, we determine and enclosing rectangle $D = [a,b] \times [c,d]$, i.e., a rectangle D such that $\overline{\Omega} \subset D$ and whose sides are parallel to the coordinate axes. We are also given the density function $\rho(x,y)$ defined on $\overline{\Omega}$; we set $\widehat{\rho} = \max_{(x,y)\in\overline{\Omega}} \rho(x,y)$. Then, a random point $(x,y) \in \overline{\Omega}$ is determined as follows. First, we sample a random point X' with constant density in $[0,1]$ and set $X = a + (b-a)X'$. Then, we sample a second random point Y' with constant density in $[0,1]$ and set $Y = c + (d-c)Y'$. If $(X,Y) \notin \overline{\Omega}$, then begin again. Otherwise, we next sample a random point U with constant density in $[0,1]$. If $U < \rho(X,Y)/\widehat{\rho}$, set $(x,y) = (X,Y)$; otherwise, start again. (The rejection of points in D which are outside of $\overline{\Omega}$ may also be effected by setting $\rho(x,y) = 0$ in $D \setminus \overline{\Omega}$.) This two-dimensional sampling algorithm can be obviously extended to higher dimensions.

In the sequel, we will refer to the rejection method as a *Monte Carlo method*; other sampling methods may be substituted without substantially affecting the relative efficiencies of the algorithms we consider. Sampling methods such as the Monte Carlo method play a crucial role in probabilistic algorithms for determining CVTs. Not only can Monte Carlo methods be used to generate initial conditions for the latter type of algorithms, but they will be used to generate the random sampling points

needed during each iteration of those algorithms. In addition, Monte Carlo methods will provide results for comparisons to CVTs.

4. Methods for determining centroidal Voronoi tessellations

In this section, we discuss sequential methods for determining CVTs. The first two methods, one probabilistic and one deterministic, are well known in the literature. The third method is a new probabilistic method which may be viewed as a generalization of both known methods.

4.1. MacQueen's method

First, we introduce *MacQueen's method* [12], a very elegant random sequential sampling method which divides the samples into k sets or clusters by taking means of sampling points.

Algorithm 1. (*MacQueen's method*): Given a region Ω, a density function $\rho(\mathbf{x})$ defined for all $\mathbf{x} \in \overline{\Omega}$, and a positive integer k,

0. Choose an initial set of k points $\{\mathbf{z}_i\}_{i=1}^k$ in Ω, e.g., by using a Monte Carlo method; set $j_i = 1$ for $i = 1, \ldots, k$;
1. Determine a point $\mathbf{y}$ in Ω at random, e.g., by a Monte Carlo method, according to the probability density function $\rho(\mathbf{x})$;
2. Find a $\mathbf{z}_{i^*}$ among $\{\mathbf{z}_i\}_{i=1}^k$ that is the closest to $\mathbf{y}$;
3. Set
$$\mathbf{z}_{i^*} \leftarrow \frac{j_{i^*}\mathbf{z}_{i^*} + \mathbf{y}}{j_{i^*} + 1} \quad \text{and} \quad j_{i^*} \leftarrow j_{i^*} + 1;$$
the new $\mathbf{z}_{i^*}$, along with the unchanged $\{\mathbf{z}_j\}, j \neq i^*$, form the new set of points $\{\mathbf{z}_i\}_{i=1}^k$;
4. If the new points meet some convergence criterion, terminate; otherwise, return to step 1.

In all our numerical experiments for MacQueens's method and other iterative methods, we will terminate at a fixed number of iterations. In practice, other stopping criteria, e.g., a sufficiently small difference in successive energies or in the successive positions of the generators, may be used instead.

Clearly, at any stage of MacQueen's method, each point $\mathbf{z}_i$, $i = 1, \ldots, k$, is the mean of all sampling points that have been found to be closest to the point $\mathbf{z}_i$ that was in effect when each point was sampled. Thus, this subset of all sampling points forms a cluster and the point $\mathbf{z}_i$ is the mean of its cluster.

The almost sure convergence of the energy for the random MacQueen's method has been proved, although not in all cases does the method converge to a CVT (see [7,12]). As is the case for most probabilistic methods, Monte Carlo methods play a key role in MacQueen's method. Of course, many variations of MacQueen's

method have been developed, such as using bootstrapping techniques [8,15]. Here, we focus on Algorithm 1 since this is the method we will generalize.

One-dimensional numerical experiments for Algorithm 1 (MacQueen's method) were performed on a CRAY T3E–600 [3] located at the CRAY Inc. facility in Eagan, Minnesota. In all cases, 128 generators in the interval $(-1,1)$ were used. Three different density functions were considered: a constant density function; e^{-10x^2} which has a huge variation, i.e., its values range from 1 to e^{-10}; and $e^{-20x^2} + 0.05\sin^2(\pi x)$ which has a large peak at the center of the interval but also varies with small amplitude away from the peak. Since MacQueen's method is a probabilistic one, four experiments for each situation were performed. Different seeds (time-based) were used to initialize the random number generator for each experiment. Since only one closest point search and only one update of the search space are needed at each iteration, there is not much motivation for efficiently determining closest neighbors. Thus, no special algorithm was used for the one-dimensional experiments. The results of the one-dimensional computational experiments for Algorithm 1 (MacQueen's method) are given in Table 1 and Fig. 1. The data given in Table 1 are the average of the energies and the maximum running times (in seconds) over the four realizations vs. the

Table 1
Maximum running times and average energies over four realizations of MacQueen's method vs. number of iterations for three density functions

No. of iterations	1		e^{-10x^2}		$e^{-20x^2} + (1/20)\sin^2(\pi x)$	
	Average energy	Maximum time (s)	Average energy	Maximum time (s)	Average energy	Maximum time (s)
0	22.514E−5	0.00	8.211E−5	0.00	9.323E−5	0.00
200,000	5.888E−5	2.93	2.285E−5	3.62	2.802E−5	3.83
800,000	5.658E−5	11.59	2.159E−5	13.94	2.702E−5	16.48
3,200,000	5.475E−5	46.34	2.059E−5	55.21	2.622E−5	63.44

Algorithm 1 (MacQueen's method); 128 generators on $(-1,1)$.

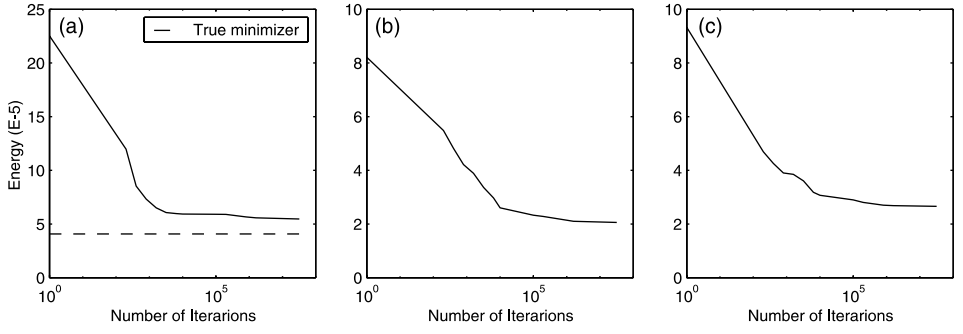

Fig. 1. Energy vs. number of iterations for MacQueen's method: (a) $\rho(x) = 1$; (b) $\rho(x) = e^{-10x^2}$; (c) $\rho(x) = e^{-20x^2} + 0.05\sin^2(\pi x)$.

number of iterations of MacQueen's method. In Fig. 1, the energy vs. the number of iterations is displayed for the three choices for the density function.

Table 1 and Fig. 1 illustrate the slow convergence of MacQueen's method; the energy is reduced very little during each step of the algorithm and, even after a large number of iterations, one does not achieve a good approximation. For a constant density function, the energy of the true minimizer for 128 generators in $(-1,1)$ can be computed exactly and is found to be 4.069E−5. We see from Table 1 and Fig. 1 for a constant density function that even after 3,200,000 iterations, we are not very close to the minimum energy; the lowest energy achieved after that many steps is over 15% larger than the energy of the minimizer. (This does not imply that we are necessarily "far away" from the optimal set of points; we will return to this point in Section 5.3.) One reason for the inefficiency of MacQueen's algorithm is that only a single random sampling point is used and only a single point $\mathbf{z}_i$ is updated at each iteration. We also notice that, using the rejection method for generating non-uniformly distributed sampling points, the running times for the non-constant density functions are only slightly higher than the corresponding running times for the constant density function.

One observation that can be gleaned from a detailed examination of the results of the computational experiments (although not definitively from the data given in the tables) is that the energy of the final set of points is closely correlated to the energy of the corresponding initial set of points, i.e., in general, the smaller the initial energy, the smaller the final energy.

4.2. Lloyd's method

In the discussion in Section 4.3 about the new probabilistic algorithm, we will need to refer to a deterministic algorithm for determining CVTs that is known in some circles as *Lloyd's method* [4,10,11] and which is the obvious iteration between computing Voronoi diagrams and mass centroids.

Algorithm 2. (*Lloyd's method*): Given a region Ω, a density function $\rho(\mathbf{x})$ defined for all $\mathbf{x} \in \overline{\Omega}$, and a positive integer k,

0. Select an initial set of k points $\{\mathbf{z}_i\}_{i=1}^k$, e.g., by using a Monte Carlo method;
1. Construct the Voronoi sets $\{V_i\}_{i=1}^k$ associated with $\{\mathbf{z}_i\}_{i=1}^k$;
2. Determine the mass centroids of the Voronoi sets $\{V_i\}_{i=1}^k$; these centroids form the new set of points $\{\mathbf{z}_i\}_{i=1}^k$;
3. If the new points meet some convergence criterion, terminate; otherwise, return to step 1.

Generally, compared to MacQueen's method, Lloyd's method requires substantially fewer iterations but has a much higher cost per iteration. In step 1 of the Lloyd's method, a Voronoi tessellation must be explicitly constructed and in step 2, numerical integrations over polyhedral domains must be employed to calculate centroids. A virtue of MacQueen's method is that it does not require the

construction of Voronoi tessellations. Furthermore, if a completely probabilistic procedure such as the rejection method is used to determine random sampling points, then MacQueen's method also does not involve any numerical integrations.

4.3. A new probabilistic method

Although MacQueen's method is a very elegant random sampling method for determining CVTs, its slow convergence provides motivation for developing alternative methods that are more efficient. One could, of course, look for parallel implementations of MacQueen's method; we consider this approach in Section 5.1. Here, we present a new probabilistic method that can be viewed as a probabilistic version of Lloyd's method and/or a generalization of the random MacQueen's method.

Algorithm 3. Given a region Ω, a density function $\rho(\mathbf{x})$ defined for all $\mathbf{x} \in \overline{\Omega}$, and a positive integer k,

0. Choose a positive integer q and constants $\{\alpha_i, \beta_i\}_{i=1}^2$ such that $\alpha_2 > 0$, $\beta_2 > 0$, $\alpha_1 + \alpha_2 = 1$, and $\beta_1 + \beta_2 = 1$; choose an initial set of k points $\{\mathbf{z}_i\}_{i=1}^k$, e.g., by using a Monte Carlo method; set $j_i = 1$ for $i = 1, \ldots, k$;
1. Choose q points $\{\mathbf{y}_r\}_{r=1}^q$ in Ω at random, e.g., by a Monte Carlo method, according to the probability density function $\rho(\mathbf{x})$;
2. For $i = 1, \ldots, k$, gather together in the set W_i all sampling points $\mathbf{y}_r$ closest to $\mathbf{z}_i$ among $\{\mathbf{z}_i\}_{i=1}^k$ (i.e., in the Voronoi region of $\mathbf{z}_i$); if the set W_i is empty, do nothing; otherwise, compute the average $\mathbf{u}_i$ of the set W_i and set
$$\mathbf{z}_i \leftarrow \frac{(\alpha_1 j_i + \beta_1)\mathbf{z}_i + (\alpha_2 j_i + \beta_2)\mathbf{u}_i}{j_i + 1} \quad \text{and} \quad j_i \leftarrow j_i + 1;$$
the new set of $\{\mathbf{z}_i\}$, along with the unchanged $\{\mathbf{z}_j\}$ (i.e., W_j is empty), form the new set of points $\{\mathbf{z}_i\}_{i=1}^k$;
3. If the new points meet some convergence criterion, terminate; otherwise, return to step 1.

For $q = 1$, $\alpha_1 = \beta_2 = 1$, and $\alpha_2 = \beta_1 = 0$, this method reduces to MacQueen's method. If $\alpha_1 = \beta_1 = 0$, $\alpha_2 = \beta_2 = 1$, then $\mathbf{z}_i = \mathbf{u}_i$, the average of the points in the set W_i; then, since the points in W_i are randomly selected points in the Voronoi region corresponding to $\mathbf{z}_i$, one may view $\mathbf{u}_i$ as a probabilistic approximation to the centroid of V_i. Thus, for $\alpha_1 = \beta_1 = 0$, $\alpha_2 = \beta_2 = 1$, this method is *a probabilistic version of Lloyd's method*; the larger q is, the better the centroid approximations. This justifies our view of Algorithm 3 as both a probabilistic Lloyd's method and a generalization of MacQueen's method. Other choices for $\{\alpha_j, \beta_j\}_{j=1}^2$ define other methods.

At every iteration of Algorithm 3, we sample a set of q points in the region Ω and, at least if q is large, we update may, perhaps all, of the generators $\{\mathbf{z}_i\}_{i=1}^k$. Furthermore, again if q is large, it is likely that many sampling points are used to update each generator. This gives us hope that this algorithm is more efficient than

Algorithm 1 (MacQueen's method) for which only a single point is sampled and only a single generator is updated at each iteration.

The updating of the generators effected in step 3 of Algorithm 3 may be rewritten in the form

$$\mathbf{z}_i \leftarrow C_{1i}\mathbf{z}_i + C_{2i}\mathbf{u}_i, \tag{4}$$

where

$$C_{1i} = \frac{\alpha_1 j_i + \beta_1}{j_i + 1} \quad \text{and} \quad C_{2i} = \frac{\alpha_2 j_i + \beta_2}{j_i + 1}.$$

This clearly shows that the new position of each generator is a linear combination of the old position and the average of the points sampled in the Voronoi region of the old generator (see Fig. 2 for a visual description). Note that since $\alpha_1 + \alpha_2 = 1$ and $\beta_1 + \beta_2 = 1$, we have that $C_{1i} + C_{2i} = 1$. From (4) and the fact that α_1 and β_1 may be negative, one can see that under and over-relaxation updating methods can be defined (see [4,9]).

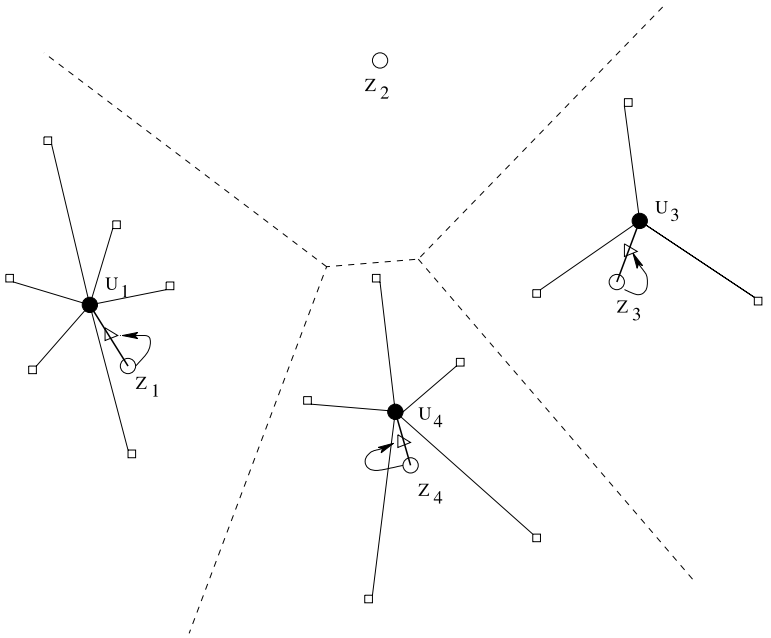

Fig. 2. A visual description of Algorithm 3. The open circles denote the current positions of the generators and the dashed lines denote their corresponding Voronoi tessellation. The small squares denote the points sampled during a single iteration of the algorithm. These are grouped according to which Voronoi cell they belong to and then all sample points in each cell are averaged to produce the filled circles. A triangle denotes the new position of a generator which is determined by taking a linear combination of the old position (the open circle) and the local sample average (the filled circle.) The Voronoi regions as depicted only for illustrative purposes; Algorithm 3 does not require the construction of the Voronoi tessellation.

Algorithm 3 represents a two parameter family of algorithms, e.g., each choice of α_2 and β_2 defines a new algorithm. (One also has to choose q, the number of points sampled at each iteration.) From (4), it is clear that Algorithm 3 will converge (in a probabilistic sense) only if $C_{1i} \to 1$ and $C_{2i} \to 0$ as the number of iterations increases. This holds for MacQueen's method for which $C_{1i} = j_i/(j_i + 1)$ and $C_{2i} = 1/(j_i + 1)$ but not for the probabilistic Lloyd's method for which $C_{1i} = 0$ and $C_{2i} = 1$. The reason why the probabilistic Lloyd's method cannot converge is that, for a fixed number of sampling points q per iteration, the average of the sampling points in the Voronoi region of a generator cannot be the true centroid of that region. However, this does not mean that methods such as the probabilistic Lloyd's method are useless. Quite the contrary, as we shall see, they can be quite effective. One need only realize that one cannot hope to get better accuracy than that allowed for by the amount of sampling done at each iteration. In turn, this means that such methods require "large" q in order to be very effective.

We now report on the results of some one-dimensional computational experiments with 128 generators in the interval $(-1, 1)$. Since we want q to be large, it is useful to do the closest point searches efficiently. In the one-dimensional setting, these searches were done by first quickly sorting the set of k points $\{z_i\}_{i=1}^{k}$, then, for each y_r, $r = 1, \ldots, q$, using binary searching to find the z_{i_r} in the set $\{z_i\}_{i=1}^{k}$ which is closest to y_r. For the results for Algorithm 3 given in Table 2, we chose five values for q (500, 1000, 2000, 8000, and 16,000), two sets of parameters $\{\alpha_i, \beta_i\}_{i=1}^{2}$ ($\alpha_1 = \beta_1 = 0, \alpha_2 = \beta_2 = 1$ and $\alpha_1 = \alpha_2 = \beta_1 = \beta_2 = 0.5$), and the same three density

Table 2
Maximum running times and average energies over four realizations of Algorithm 3 for a fixed number (3,200,000) of random points sampled

$\rho(x)$	q	No. of iterations	$\alpha_1 = 0$, $\alpha_2 = 1$, $\beta_1 = 0$, $\beta_2 = 1$		$\alpha_1 = 1/2$, $\alpha_2 = 1/2$ $\beta_1 = 1/2$, $\beta_2 = 1/2$	
			Average energy	Maximum time (s)	Average energy	Maximum time (s)
1	500	6400	5.112E−5	36.01	4.488E−5	38.03
	1000	3200	4.472E−5	38.78	4.299E−5	39.23
	2000	1600	4.343E−5	40.07	4.341E−5	40.64
	8000	400	4.267E−5	36.72	4.322E−5	36.14
	16,000	200	4.424E−5	37.36	4.366E−5	37.97
e^{-10x^2}	500	6400	0.606E−5	43.99	0.603E−5	45.37
	1000	3200	0.593E−5	43.15	0.533E−5	44.25
	2000	1600	0.582E−5	45.60	0.634E−5	45.38
	8000	400	0.720E−5	45.91	0.827E−5	43.46
	16,000	200	0.933E−5	44.84	0.916E−5	43.75
$e^{-20x^2} + \frac{1}{20}\sin^2(\pi x)$	500	6400	0.605E−5	54.88	0.594E−5	55.38
	1000	3200	0.593E−5	54.65	0.533E−5	56.71
	2000	1600	0.590E−5	53.87	0.709E−5	54.74
	8000	400	0.822E−5	54.24	1.013E−5	54.30
	16,000	200	1.153E−5	54.21	1.319E−5	55.69

Algorithm 3; 128 generators on $(-1, 1)$; 3,200,000 sampling points.

functions as were used for Table 1. For each value of q, the number of random points sampled per iteration, the number of iterations of Algorithm 3 performed was chosen so that the total number or random points sampled (other than for initialization) was 3,200,000. For reference, the initial energies for the runs reported in Table 2 are in the range [1.598E−4, 3.113E−4] for $\rho = 1$, [4.363E−5, 1.482E−4] for $\rho = e^{-10x^2}$, and [4.682E−5, 1.679E−4] for $\rho = e^{-20x^2} + (1/20)\sin^2(\pi x)$. Comparing with Table 1 for the random MacQueen's method, we see that for the same density function and the same number (3,200,000) of random sampling points, the running times and the energies for Algorithm 3 are substantially lower than for Algorithm 1. The reason the running times are lower even though the number of sampling points are the same is that it is difficult to do efficient searching with MacQueen's algorithm.

It is also interesting to examine the effect of changing the number of points sampled at each iteration. In Table 3, we give the average running times and average energies over four realizations of Algorithm 3 for a fixed number of iterations but for different numbers of points sampled during each iteration. The ranges of the initial energies are again as given above. We can conclude from the table that q, the number of points sampled at each iteration, does not need to be too large. For example, in our experiments for 128 generators and 400 iterations, $q = 8000$ seems to be good enough; choosing $q = 16,000$ effects almost no improvement, i.e., almost no further reduction in the energy, but requires much higher running times. We note that q should scale with the number of generators. Also, from Table 3, we see that, as expected, the running times scale linearly with the total number of points sampled.

Table 3
Average running times and average energies over four realizations of Algorithm 3 for a fixed number (400) of iterations

$\rho(x)$	q	Total points sampled	$\alpha_1 = 0$, $\alpha_2 = 1$, $\beta_1 = 0$, $\beta_2 = 1$		$\alpha_1 = 1/2$, $\alpha_2 = 1/2$, $\beta_1 = 1/2$, $\beta_2 = 1/2$	
			Average energy	Average time (s)	Average energy	Average time (s)
1	500	200,000	5.156E−5	2.36	4.678E−5	2.38
	1000	400,000	4.748E−5	4.66	4.433E−5	4.69
	2000	800,000	4.397E−5	9.87	4.542E−5	9.30
	8000	3,200,000	4.267E−5	36.28	4.322E−5	35.69
	16,000	6,400,000	4.282E−5	72.64	4.257E−5	73.54
e^{-10x^2}	500	200,000	0.978E−5	2.83	1.028E−5	2.93
	1000	400,000	1.003E−5	5.41	1.077E−5	5.70
	2000	800,000	0.921E−5	10.81	0.926E−5	11.16
	8000	3,200,000	0.720E−5	43.22	0.827E−5	42.23
	16,000	6,400,000	0.755E−5	85.04	0.786E−5	84.45
$e^{-20x^2} + \frac{1}{20}\sin^2(\pi x)$	500	200,000	1.313E−5	3.48	1.405E−5	3.51
	1000	400,000	1.244E−5	6.84	1.410E−5	6.96
	2000	800,000	0.958E−5	13.30	1.213E−5	13.71
	8000	3,200,000	0.822E−5	53.06	1.013E−5	52.79
	16,000	6,400,000	0.841E−5	104.61	1.017E−5	106.73

Algorithm 3; 128 generators on $(-1, 1)$; 400 iterations.

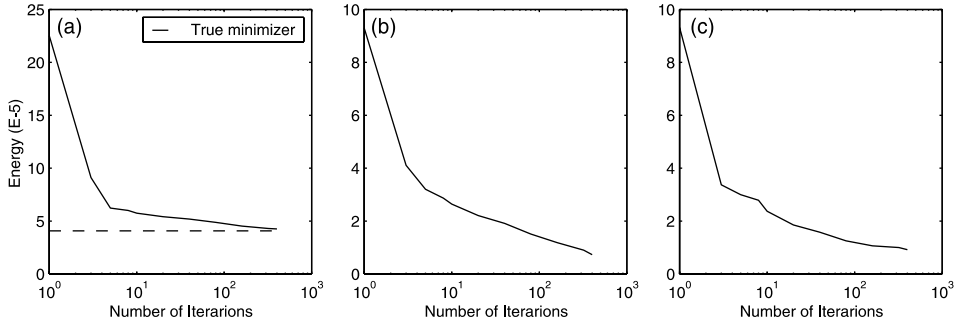

Fig. 3. Energy vs. number of iterations for Algorithm 3 with $q = 8000$ and $\{\alpha_1 = 0,\ \alpha_2 = 1,\ \beta_1 = 0,\ \beta_2 = 1\}$: (a) $\rho(x) = 1$; (b) $\rho(x) = \mathrm{e}^{-10x^2}$; (c) $\rho(x) = \mathrm{e}^{-20x^2} + 0.05\sin^2(\pi x)$.

Fig. 3 provides a display of the energy versus number of iterations for Algorithm 3 with $q = 8000$ and $\{\alpha_1 = 0, \alpha_2 = 1, \beta_1 = 0, \beta_2 = 1\}$. Comparing with Fig. 1, we see that Algorithm 3 converges much faster, i.e., in much fewer iterations, than does Algorithm 1. Of course, the work per iteration is higher for Algorithm 3 but, as can be seen from Tables 1–3, for the same amount of total work, Algorithm 3 produces much better results than does Algorithm 1. It also should be pointed out that when the variations of the density functions become larger and more complex, more iterations are needed to obtain good results.

5. Parallel implementations

We now consider parallel implementations on distributed memory systems of the sequential probabilistic algorithms discussed in Section 4. For convenience of description, given two positive integers k and p and a non-negative integer i such as $p \leqslant k$ and $i < p$, we define

$$M(i,p,k) = \begin{cases} \left[\frac{k}{p}\right] & \text{if } i < k \bmod(p) \\ \left[\frac{k}{p}\right] + 1 & \text{otherwise,} \end{cases} \qquad G(0,p,k) = 0, \quad \text{and} \quad G(i,p,k) = \sum_{j=0}^{i-1} M(j,p,k),$$

where $[\cdot]$ denotes the greatest integer function. We begin with parallelizations of Algorithm 1, the random MacQueen's method.

5.1. Parallel MacQueen's methods

An obvious parallelization of MacQueen's method is given by the following algorithm.

Algorithm 4. (*A parallel MacQueen's method*): Given a region Ω, a density function $\rho(\mathbf{x})$ defined for all $\mathbf{x} \in \overline{\Omega}$, a positive integer k, and p processors with rank $= 0, 1, \ldots, p-1$;

0. For rank $= 0, \ldots, p-1$, set $m_{\text{rank}} = M(\text{rank}, p, k)$; then, each processor independently chooses its own initial set of m_{rank} points $\{\mathbf{z}_i^{\text{rank}}\}_{i=1,}^{m_{\text{rank}}}$, e.g., by using a Monte Carlo method; set $j_i^{\text{rank}} = 1$ for $i = 1, \ldots, m_{\text{rank}}$;
1. Processor 0 selects a $\mathbf{y} \in \Omega$ at random, e.g., by a Monte Carlo method, according to the probability density function $\rho(\mathbf{x})$ and then broadcasts it to all other processors;
2. Each processor finds the $\mathbf{z}_{i^*}^{\text{rank}}$ which is the closest to $\mathbf{y}$ in $\{\mathbf{z}_i^{\text{rank}}\}_{i=1}^{m_{\text{rank}}}$ respectively; determine the corresponding distance by $d_{i^*}^{\text{rank}}$;
3. Compare them by communication to obtain the d^*, where d^* is the minimum among the $d_{i^*}^{\text{rank}}$ of all processors;
4. On each processor, if $d_{i^*}^{\text{rank}} \neq d^*$, do nothing; otherwise, set

$$\mathbf{z}_{i^*}^{\text{rank}} \leftarrow \frac{j_{i^*}^{\text{rank}} \mathbf{z}_{i^*}^{\text{rank}} + \mathbf{y}}{j_{i^*}^{\text{rank}} + 1} \quad \text{and} \quad j_{i^*}^{\text{rank}} \leftarrow j_{i^*}^{\text{rank}} + 1;$$

   this new $\mathbf{z}_{i^*}^{\text{rank}}$, along with the unchanged $\mathbf{z}_i^{\text{rank}}$, form the new set of points $\{\mathbf{z}_i^{\text{rank}}\}_{i=1}^{m_{\text{rank}}}$;
5. If the new points meet some convergence criterion, terminate; otherwise, return to step 1.

In practice, Algorithm 4 is not a good one because in step 3 it requires the global transmission of a small message, i.e., a single floating point number. It is wasteful with regard to network bandwidth and this kind of communication is also inefficient because the hardware start-up time is much larger (generally by a factor of about 1000) than the communication time to send one floating point number through the network. Thus, we need to make some modifications to improve the communication efficiency. We accomplish this in the following algorithm by updating multiple points at each iteration; the algorithm is clearly based on MacQueen's method, but it also has some differences with the direct parallel implementation of the latter method.

Algorithm 5. (*A modified parallel MacQueen's method*): Given a region Ω, a density function $\rho(\mathbf{x})$ defined for all $\mathbf{x} \in \overline{\Omega}$, a positive integer k, and p processors with rank $= 0, 1, \ldots, p-1$;

0. Choose a positive integer s; for rank $= 0, \ldots, p-1$, set $m_{\text{rank}} = M(\text{rank}, p, k)$; then, each processor independently chooses its own initial set of m_{rank} points $\{\mathbf{z}_i^{\text{rank}}\}_{i=1}^{m_{\text{rank}}}$, e.g., by using a Monte Carlo method; set $j_i^{\text{rank}} = 1$ for $i = 1, \ldots, m_{\text{rank}}$;
1. Each processor independently selects s points in Ω at random, according to the probability density function $\rho(\mathbf{x})$; combine them together by communication to form a set of sampling points $\{\mathbf{y}_r\}_{r=1}^{q}$ $(q = sp)$;
2. On each processor, for $r = 1, \ldots, q$, find a $\mathbf{z}_{i_r}^{\text{rank}}$ among $\{\mathbf{z}_i^{\text{rank}}\}_{i=1}^{m_{\text{rank}}}$ that is closest to $\mathbf{y}_r$; determine the corresponding distance by d_r^{rank};

3. Compare them by communication to obtain the set $\{d_r\}_{r=1}^{q}$, where d_r is the minimum among d_r^{rank} on all processors;
4. For $r = 1, \ldots, q$, on each processor, if $d_r^{\text{rank}} \neq d_r$, do nothing; otherwise, set

$$\mathbf{z}_{i_r}^{\text{rank}} \leftarrow \frac{j_{i_r}^{\text{rank}} \mathbf{z}_{i_r}^{\text{rank}} + \mathbf{y}_r}{j_{i_r}^{\text{rank}} + 1} \quad \text{and} \quad j_{i_r}^{\text{rank}} \leftarrow j_{i_r}^{\text{rank}} + 1;$$

this new set of $\{\mathbf{z}_{i_r}^{\text{rank}}\}$, along with the unchanged $\mathbf{z}_j^{\text{rank}}$ $(j \neq i_r)$, form the new set of points $\{\mathbf{z}_i^{\text{rank}}\}_{i=1}^{m_{\text{rank}}}$;
5. If the new points meet some convergence criterion, terminate; otherwise, return to step 1.

We expect that this modified parallel MacQueen's algorithm will be more efficient than Algorithm 4. Note that here $q = sp$ should not be large; thus, there is no need to do fast closest point searches because the preparation job (such as building the data structure) prior to any fast search also needs much time and therefore there is a trade off. We coded Algorithm 5 using MPI [13] and ran the code using up to 16 processors on the CRAY T3E−600; the results of some computational experiments are given in Table 4 for 128 generators on $(-1, 1)$ and the same three density functions used previously. In the table, p is the number of processors and s is the number of random points sampled by each processor at each iteration. Different values of p and s were used such that $q = ps = 160$ in all cases, i.e., the number of points sampled at each iteration over all processors is fixed. With 20,000 iterations being carried, the total number of points sampled in each case was 3,200,000.

Comparing Table 4 for the modified parallel MacQueen's method with Table 1 for the sequential MacQueen's method ($p = 1$), we see that for 3,200,000 random sampling points that the average energies obtained by the two methods are very similar, i.e., independent of the number of processors. However, the running times are very much lower for the parallel method. This is seen more clearly in Fig. 4 which gives, for each of the three density function, the speed up obtained by parallelization. (The average time of all similar experiments was used to compute the

Table 4
Maximum running times and average energies over four realizations of Algorithm 5 for a fixed number of iterations (20,000) and a fixed number (3,200,000) of random points sampled

p	s	Density function					
		1		e^{-10x^2}		$e^{-20x^2} + (1/20)\sin^2(\pi x)$	
		Average energy	Maximum time	Average energy	Maximum time	Average energy	Maximum time
1	160	5.563E−5	39.94	2.221E−5	47.91	2.812E−5	57.73
2	80	5.421E−5	26.22	2.172E−5	30.30	2.789E−5	34.88
4	40	4.976E−5	16.24	1.914E−5	18.39	2.755E−5	20.94
8	20	5.112E−5	12.17	1.837E−5	13.42	2.710E−5	14.86
16	10	5.259E−5	12.63	1.875E−5	13.30	2.715E−5	14.24

Algorithm 5; 128 generators on (−1,1); 20,000 iterations; 3,200,000 sampling points.

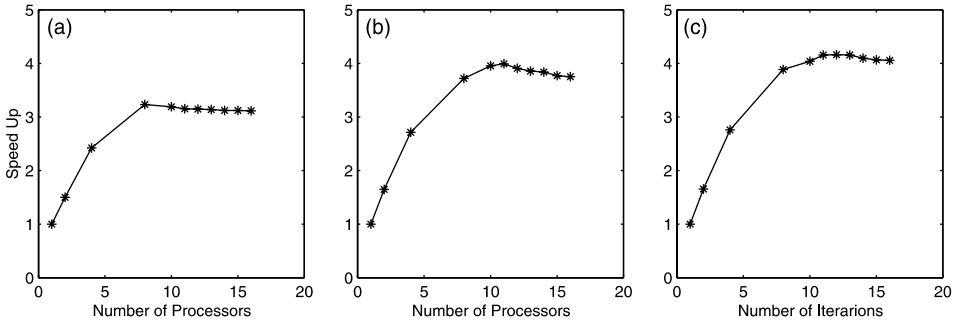

Fig. 4. Speed up of Algorithm 5 with $q = ps = 160$: (a) $\rho(x) = 1$; (b) $\rho(x) = e^{-10x^2}$; (c) $\rho(x) = e^{-20x^2} + 0.05\sin^2(\pi x)$.

parallel speed ups and the results for $p = 10, \ldots, 15$ are also included in this figure although they were not given in Table 4.) Note that the speed up appears to become sublinear as the number of processors increases, and it becomes even worse as p goes from 10 to 16 because the loss caused by the cost for communication exceeds the gain obtained from the distribution of computations. It is also worth noting that the speed up of Algorithm 5 depends strongly on the number of sampling points q used at each iteration: the larger q is, the better the speed. However, q should not be too large in this modified parallel MacQueen's method as was pointed out in the last paragraph.

5.2. Parallelization of the new algorithm

A direct parallel implementation of Algorithm 3 is given in the following algorithm.

Algorithm 6. (*A parallel version of Algorithm* 3): Given a region Ω, a density function $\rho(\mathbf{x})$ defined for all $\mathbf{x} \in \overline{\Omega}$, a positive integer k, and p processors with rank $= 0, 1, \ldots, p-1$;

0. Choose a positive integer s and constants $\{\alpha_i, \beta_i\}_{i=1}^{2}$ such that $\alpha_1 > 0$, $\beta_1 > 0$, $\alpha_1 + \alpha_2 = 1$ and $\beta_1 + \beta_2 = 1$; for rank $= 0, \ldots, p-1$, set $m_{\text{rank}} = M(\text{rank}, p, k)$, $g_{\text{rank}} = G(\text{rank}, p, k)$ and $g_p = G(p, p, k)$; then, each processor independently chooses its own initial set of m_{rank} points $\{\mathbf{z}_i^{\text{rank}}\}_{i=1}^{m_{\text{rank}}}$, e.g., by using a Monte Carlo method;
1. Combine them together by communication to form a complete initial set of k points $\{\mathbf{z}_i\}_{i=1}^{k}$; set $j_i = 1$ for $i = 1, \ldots, k$;
2. Each processor selects its own set of s points $\{\mathbf{y}_r^{\text{rank}}\}_{r=1}^{s}$ in Ω at random according to the probability density function $\rho(\mathbf{x})$;
3. On each processor, for each generator $\mathbf{z}_i$, gather together all sampling points $\mathbf{y}_r^{\text{rank}}$ closest to $\mathbf{z}_i$, (i.e., in the Voronoi region of $\mathbf{z}_i$) in the set W_i^{rank} respectively, and denote the cardinality of W_i^{rank} by w_i^{rank};

4. If the set W_i^{rank} is empty, set $\mathbf{v}_i^{\text{rank}} = \mathbf{0}$; otherwise, determine the sum $\mathbf{v}_i^{\text{rank}}$ of the members of the set W_i^{rank};
5. On each processor, by communication, obtain the set $\{\mathbf{v}_i\}_{i=g_{\text{rank}}+1}^{g_{\text{rank}+1}}$ respectively, where $\mathbf{v}_i$ is the sum of $\mathbf{v}_i^{\text{rank}}$ over all processors, and the corresponding set $\{w_i\}_{i=g_{\text{rank}}+1}^{g_{\text{rank}+1}}$, where w_i is the sum of w_i^{rank} over all processors.
6. On each processor, for $i = g_{\text{rank}} + 1, \ldots, g_{\text{rank}+1}$, if $w_i \neq 0$, compute the average $\mathbf{u}_i = \frac{1}{w_i}\mathbf{v}_i$, and then set

$$\mathbf{z}_i \leftarrow \frac{(\alpha_1 j_i + \beta_1)\mathbf{z}_i + (\alpha_2 j_i + \beta_2)\mathbf{u}_i}{j_i + 1} \quad \text{and} \quad j_i \leftarrow j_i + 1;$$

   otherwise, do nothing; by communication, this new set of $\mathbf{z}_i$, along with the unchanged $\mathbf{z}_j$ (i.e., $w_j = 0$), form the new set of points $\{\mathbf{z}_i\}_{i=1}^{k}$;
7. If the new points meet some convergence criterion, terminate; otherwise, return to step 1.

Note that, in step 3 of the above algorithm, searching for closest points on each processor is required.

However, sequential fast searching algorithms can still be directly applied since the searching is local, i.e., totally independent between processors, and s is generally a large number in order to make the algorithm effective as was mentioned in Section 4.3, i.e., $q = ps$. We coded Algorithm 6, also using MPI, and ran the code using up to 16 processors of a CRAY T3E–600; the results of some computational experiments are given in Table 5 for 128 generators on $(-1, 1)$ and the same three density functions used previously. The same two sets of parameters $\{\alpha_i, \beta_i\}_{i=1}^{2}$ used for Tables 2

Table 5
Average running times and average energies over four realizations of Algorithm 6 for a fixed number (400) of iterations and a fixed number (3,200,000) of random points sampled

Parameters values	p	s	Density function					
			1		e^{-10x^2}		$e^{-20x^2} + 0.05\sin^2(\pi x)$	
			Average energy	Average time	Average energy	Average time	Average energy	Average time
$\alpha_1 = 0, \alpha_2 = 1, \beta_1 = 0, \beta_2 = 1$	1	8000	4.274E−5	38.85	0.706E−5	44.82	0.809E−5	55.69
	2	4000	4.235E−5	19.77	0.751E−5	21.59	0.906E−5	26.26
	4	2000	4.234E−5	9.63	0.722E−5	10.86	0.846E−5	13.62
	8	1000	4.159E−5	4.99	0.795E−5	5.80	0.840E−5	6.96
	16	500	4.155E−5	2.70	0.753E−5	2.97	0.879E−5	3.64
$\alpha_1 = \alpha_2 = 1/2, \beta_1 = \beta_2 = 1/2$	1	8000	4.413E−5	38.84	0.901E−5	44.37	0.977E−5	56.76
	2	4000	4.337E−5	19.76	0.897E−5	21.88	1.085E−5	27.40
	4	2000	4.328E−5	9.52	0.851E−5	10.90	1.085E−5	13.43
	8	1000	4.228E−5	4.95	0.885E−5	5.79	1.015E−5	6.98
	16	500	4.150E−5	2.67	0.812E−5	3.10	1.075E−5	3.64

Algorithm 6; 128 generators on $(-1, 1)$; 400 iterations; 3,200,000 sampling points.

and 3 are used for Table 5. Different values of p and s were used such that $q = ps = 8000$, i.e., the number of points sampled at each iteration over all processors is fixed; 400 iterations were carried out so that the total number of points sampled in each case was again 3,200,000.

Comparing Table 4 for the modified parallel MacQueen's method with Table 5 for the parallel version of the new algorithm shows the clear superiority of the latter; for the same number of sampling points and for the same number of processors, the new method results in a substantial lower energy and a substantial reduction in the running times. This is especially true as the number of processors increases.

Comparing Table 5 for the parallel version of the new algorithm with Table 2 for the sequential version $(p = 1)$ shows that for the same number of sampling points, the new algorithm results in lower energies and requires much lower running times. The parallel speed up for each of the three density functions is given in Fig. 5. Note that for Algorithm 6 the speed up in all cases is almost perfectly linear in the number of processors, and there are even some superlinear speed ups for $p = 2$ which are probably mainly caused by the cache.

5.3. Two-dimensional computational experiments

We also carried out some experiments with Algorithms 5 and 6 for 256 generators on the two-dimensional domains $\Omega = (-1, 1)^2$ and $\Omega = \{(x, y)|(x^2 + y^2 < 1\}$ using 32 processors on a CRAY T3E−600. There are many algorithms to efficiently compute closest neighbors using some data structures such as KD tree or binary search tree (see, e.g., [1,2,17]) for centroidal Voronoi generators in two dimensions; however, this is not our focus here so we did not employ them.

For Algorithm 6, the experiments use the same two sets of parameters $\{\alpha_i, \beta_i\}_{i=1}^2$ used for the one-dimensional experiments. For both Algorithms 5 and 6, the density functions used were similar to those used in the one-dimensional experiments. Specifically, for $\Omega = (-1, 1)^2$, the density functions used were a constant, $e^{-10(x^2+y^2)}$, and $e^{-20(x^2+y^2)} + 0.05\sin^2(\pi x)\sin^2(\pi y)$; for $\Omega = \{(x, y)|(x^2 + y^2 < 1\}$, the density functions used were a constant and $e^{-5(1-x^2-y^2)}$. For each iteration of Algorithm 5, 10 random

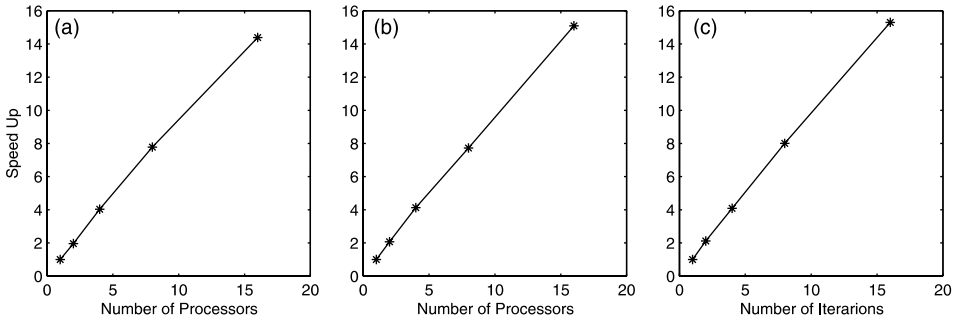

Fig. 5. Speed up of Algorithm 6 with $q = ps = 8000$: (a) $\rho(x) = 1$; (b) $\rho(x) = e^{-10x^2}$; (c) $\rho(x) = e^{-20x^2} + 0.05\sin^2(\pi x)$.

points were sampled per processor; a total of 30,000 iterations were performed. For each iteration of Algorithm 6, 375 random points were sampled per processor and a total of 800 iterations were performed. Thus, for each experiment, a total of

Table 6
Maximum running times for four realizations of Algorithms 5 and 6 for a fixed number (16) of processors and a fixed number (9.6×10^6) of random points sampled

Two-dimensional simulations
Algorithm 5
Algorithm 6a ($\alpha_1 = 0$, $\alpha_2 = 1$, $\beta_1 = 0$, $\beta_2 = 1$)
Algorithm 6b ($\alpha_1 = \alpha_2 = \beta_1 = \beta_2 = 1/2$)
32 processors, 256 generators, 9,600,000 sampling points

Density	Algorithm	s	Iterations	Time
$\Omega = (-1,1)^2$				
1	5	10	30,000	867.3
	6a	375	800	329.6
	6b	375	800	329.1
$e^{-10(x^2+y^2)}$	5	10	30,000	1012.6
	6a	375	800	436.7
	6b	375	800	435.9
$e^{-20(x^2+y^2)} + \frac{1}{20}\sin^2(\pi x)\sin^2(\pi y)$	5	10	30,000	1246.2
	6a	375	800	680.4
	6b	375	800	684.1
$\Omega = \{(x,y) \mid (x^2+y^2) < 1\}$				
1	5	10	30,000	869.5
	6a	375	800	332.2
	6b	375	800	331.2
$e^{-5(1-x^2-y^2)}$	5	10	30,000	953.1
	6a	375	800	381.8
	6b	375	800	380.9

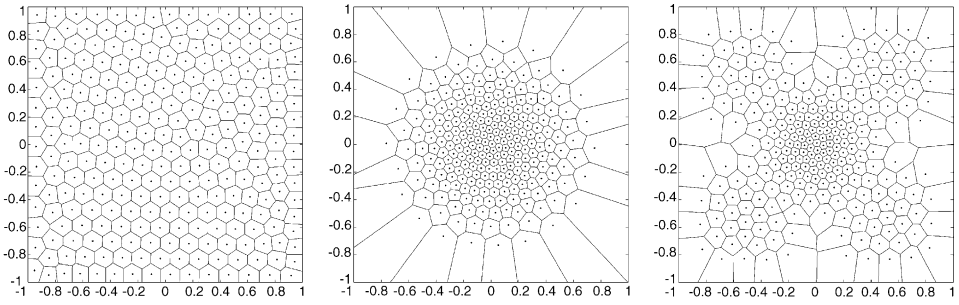

Fig. 6. Two-dimensional centroidal Voronoi diagrams for 256 generators in $\Omega = (-1,1)^2$ found by the deterministic Lloyd's method (Algorithm 2); left: $\rho(x,y) = 1$; middle: $\rho(x,y) = e^{-10(x^2+y^2)}$; right: $\rho(x,y) = e^{-20(x^2+y^2)} + 0.05\sin^2(\pi x)\sin^2(\pi y)$.

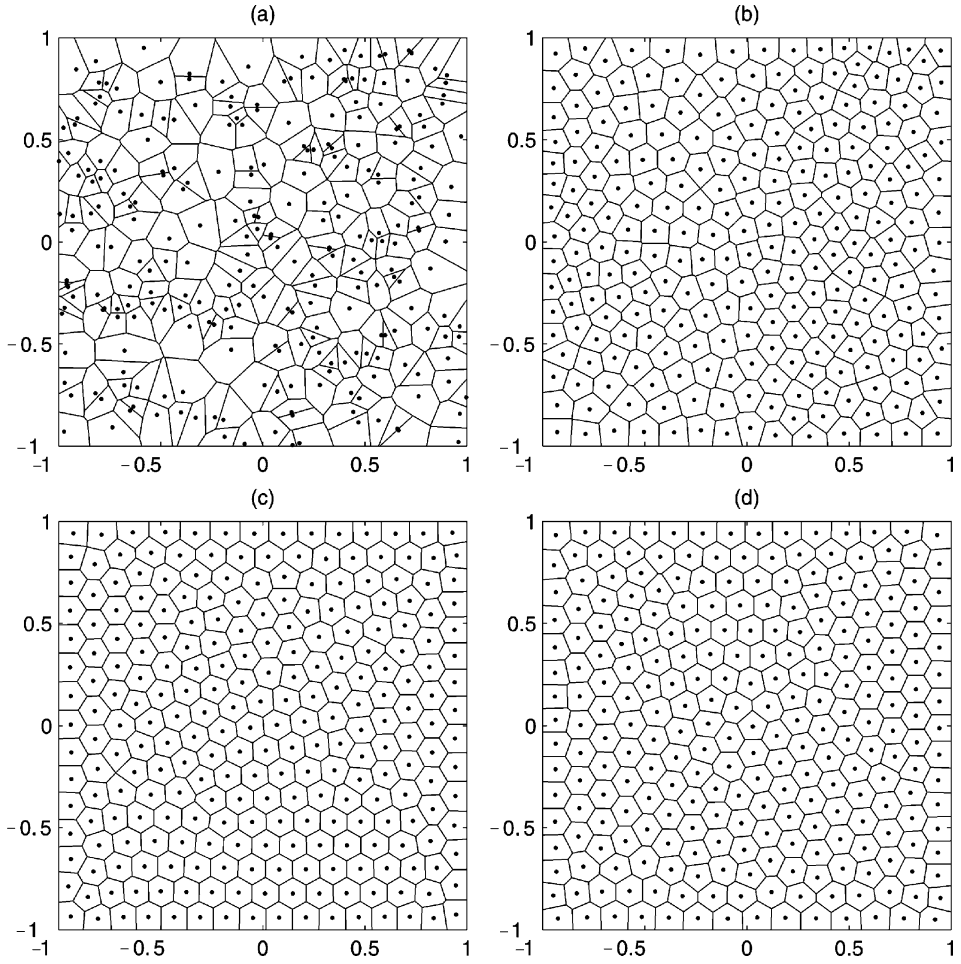

Fig. 7. Two-dimensional Voronoi diagrams for 256 generators in $\Omega = (-1,1)^2$ with the density function $\rho(x,y) = 1$: (a) Monte Carlo simulation; (b) approximate centroidal Voronoi diagram using Algorithm 5; (c) approximate centroidal Voronoi diagram using Algorithm 6 with $\alpha_1 = \beta_1 = 0$ and $\alpha_2 = \beta_2 = 1$; (d) approximate centroidal Voronoi diagram using Algorithm 6 with $\alpha_1 = \alpha_2 = \beta_1 = \beta_2 = 0.5$; 9.6×10^6 sampling points were used for (b), (c), and (d).

9.6×10^6 random points were sampled. Thus, the sequential ($p = 1$) running times for the two algorithms should be about the same. The parallel running times are reported on in Table 6. Again, s denotes the number of random points sampled by each processor at each iteration. We easily see that for all three density functions and for the same number of total sampling points, the running times for the new method (Algorithm 6) are substantially lower than for Algorithm 5, a parallel MacQueen's algorithm. Also, there is little difference between the running times for each of the two sets of parameters $\{\alpha_i, \beta_i\}_{i=1}^2$ tested for Algorithm 6.

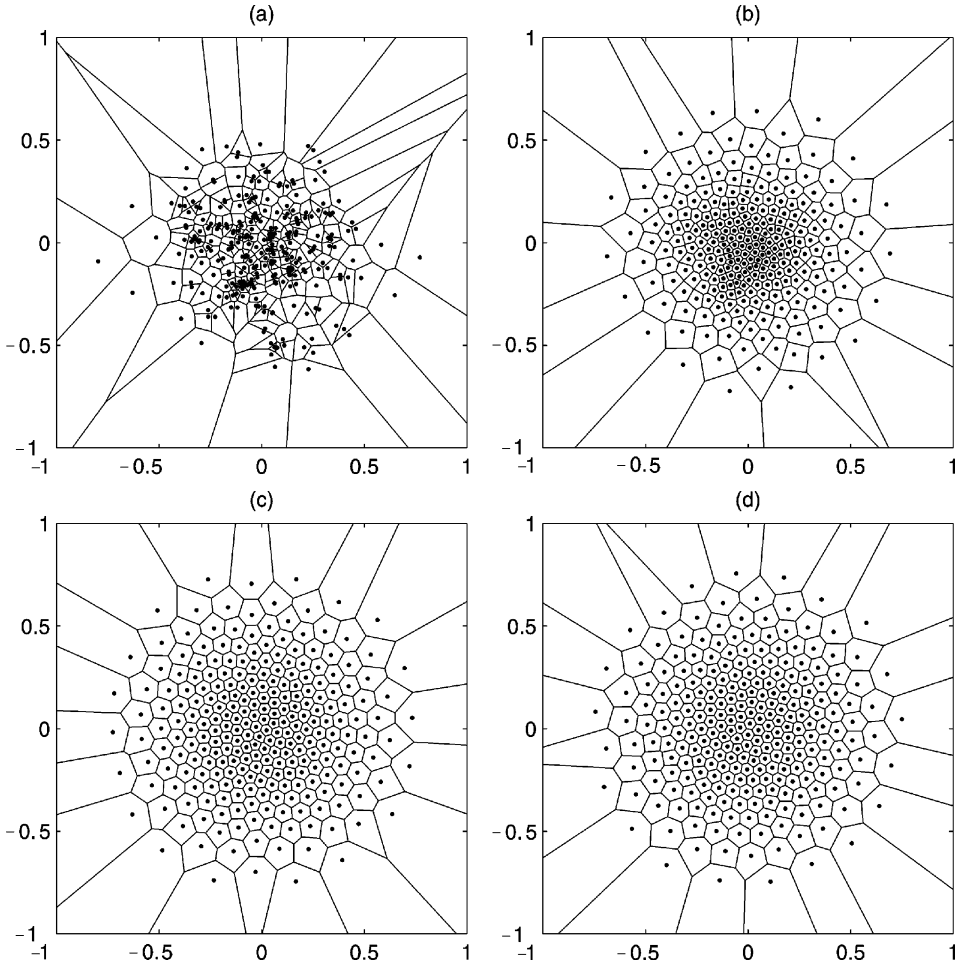

Fig. 8. Two-dimensional Voronoi diagrams for 256 generators in $\Omega = (-1,1)^2$ with the density function $\rho(x,y) = \mathrm{e}^{-10(x^2+y^2)}$: (a) Monte Carlo simulation; (b) approximate centroidal Voronoi diagram using Algorithm 5; (c) approximate centroidal Voronoi diagram using Algorithm 6 with $\alpha_1 = \beta_1 = 0$ and $\alpha_2 = \beta_2 = 1$; (d) approximate centroidal Voronoi diagram using Algorithm 6 with $\alpha_1 = \alpha_2 = \beta_1 = \beta_2 = 0.5$; 9.6×10^6 sampling points were used for (b), (c), and (d).

Naturally, one would like to know what is the quality of the centroidal Voronoi generators determined by the experiments with Algorithms 5 and 6. This is especially important in view of the observation made previously for the one-dimensional experiments with a constant density for which, even after 3,200,000 points were sampled, the energies were not always close to the energy of the true minimizer. Here, in Figs. 6–9, we give visual evidence of the relative quality of solutions found using Algorithms 5 and 6 with 9.6×10^6 sampling points in $\Omega = (-1,1)^2$. A baseline to compare with for all three density functions is provided by (very expensive) converged (with respect to the energies) deterministic solutions obtained using the deterministic

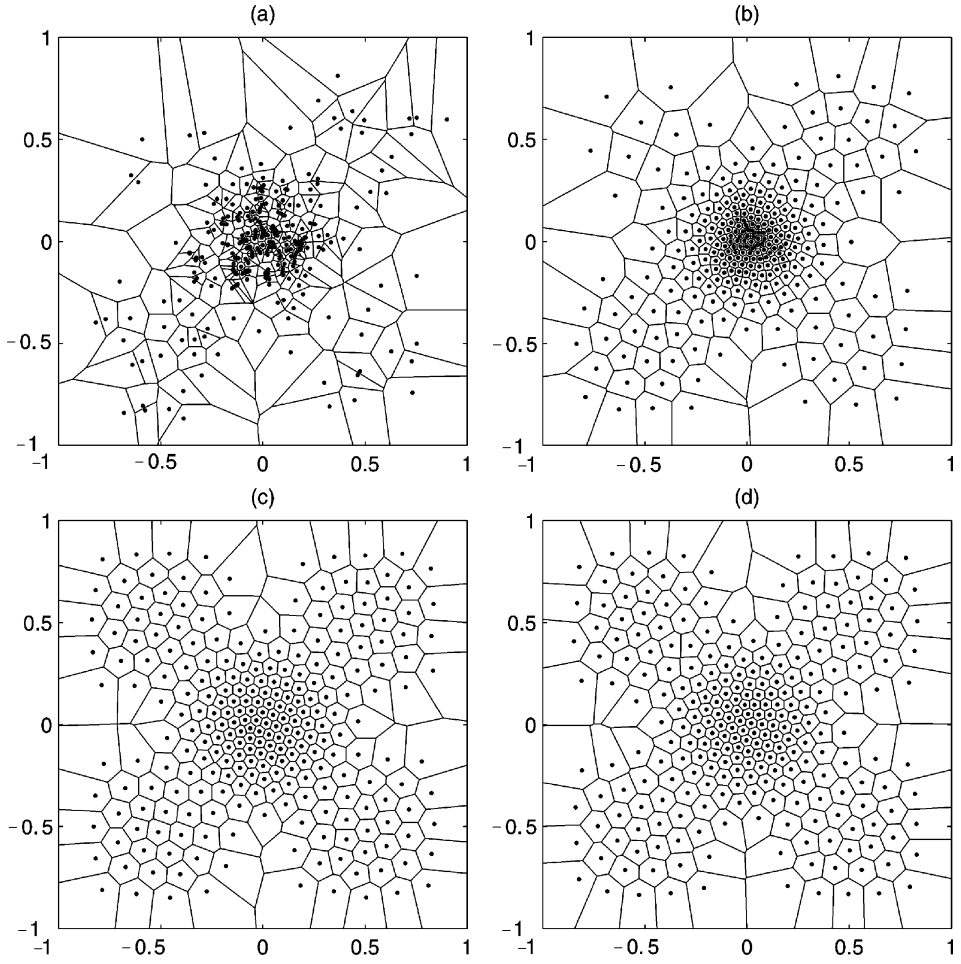

Fig. 9. Two-dimensional Voronoi diagrams for 256 generators in $\Omega = (-1,1)^2$ with the density function $\rho(x,y) = \mathrm{e}^{-20(x^2+y^2)} + 0.05\sin^2(\pi x)\sin^2(\pi y)$: (a) Monte Carlo simulation; (b) approximate centroidal Voronoi diagram using Algorithm 5; (c) approximate centroidal Voronoi diagram using Algorithm 6 with $\alpha_1 = \beta_1 = 0$ and $\alpha_2 = \beta_2 = 1$; (d) approximate centroidal Voronoi diagram using Algorithm 6 with $\alpha_1 = \alpha_2 = \beta_1 = \beta_2 = 0.5$; 9.6×10^6 sampling points were used for (b), (c), and (d).

Lloyd's method (Algorithm 2). Then, comparing Fig. 6 with Figs. 7–9 shows that, for the same total number of sampling points, Algorithm 6 with either choice for the parameters $\{\alpha_i, \beta_i\}_{i=1}^2$ provides substantially superior results than does Algorithm 5. Of course, the latter algorithm provides vastly better results than does just a single random sampling of points according the density function. Thus, it seems that Algorithm 6 is superior to Algorithm 5 with respect to both speed and accuracy. We also provide, in Figs. 10 and 11, some centroidal Voronoi diagrams found using Algorithms 5 and 6 with 9.6×10^6 sampling points in the circle $\Omega = \{(x,y)|(x^2+y^2 < 1\}$. The results again show that Algorithm 6 performs much better than Algorithm 5.

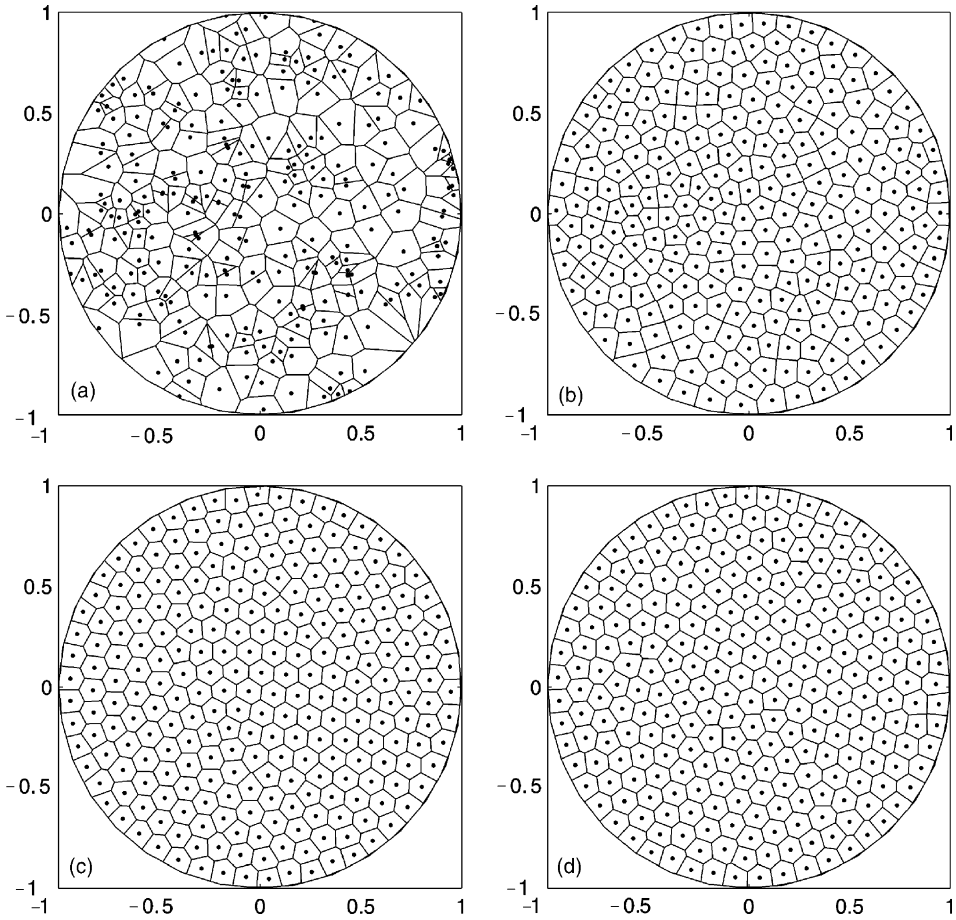

Fig. 10. Two-dimensional Voronoi diagrams for 256 generators in $\Omega = \{(x,y)|(x^2+y^2) < 1\}$ with the density function $\rho(x,y) = 1$: (a) Monte Carlo simulation; (b) approximate centroidal Voronoi diagram using Algorithm 5; (c) approximate centroidal Voronoi diagram using Algorithm 6 with $\alpha_1 = \beta_1 = 0$ and $\alpha_2 = \beta_2 = 1$; (d) approximate centroidal Voronoi diagram using Algorithm 6 with $\alpha_1 = \alpha_2 = \beta_1 = \beta_2 = 0.5$; 9.6×10^6 sampling points were used for (b), (c), and (d).

6. Concluding remarks

We examined several probabilistic algorithms and their parallel implementations for determining CVTs. From the results of some computational experiments, it seems that a new probabilistic algorithm we introduced performs much better than the random MacQueen's method and is more suitable for parallel computation. There are a number of issues that need further study. These include, for example, improving the convergence rate of MacQueen's method, finding better parallel implementations, optimal choices for s (or equivalently $q = ps$), and optimal choices of

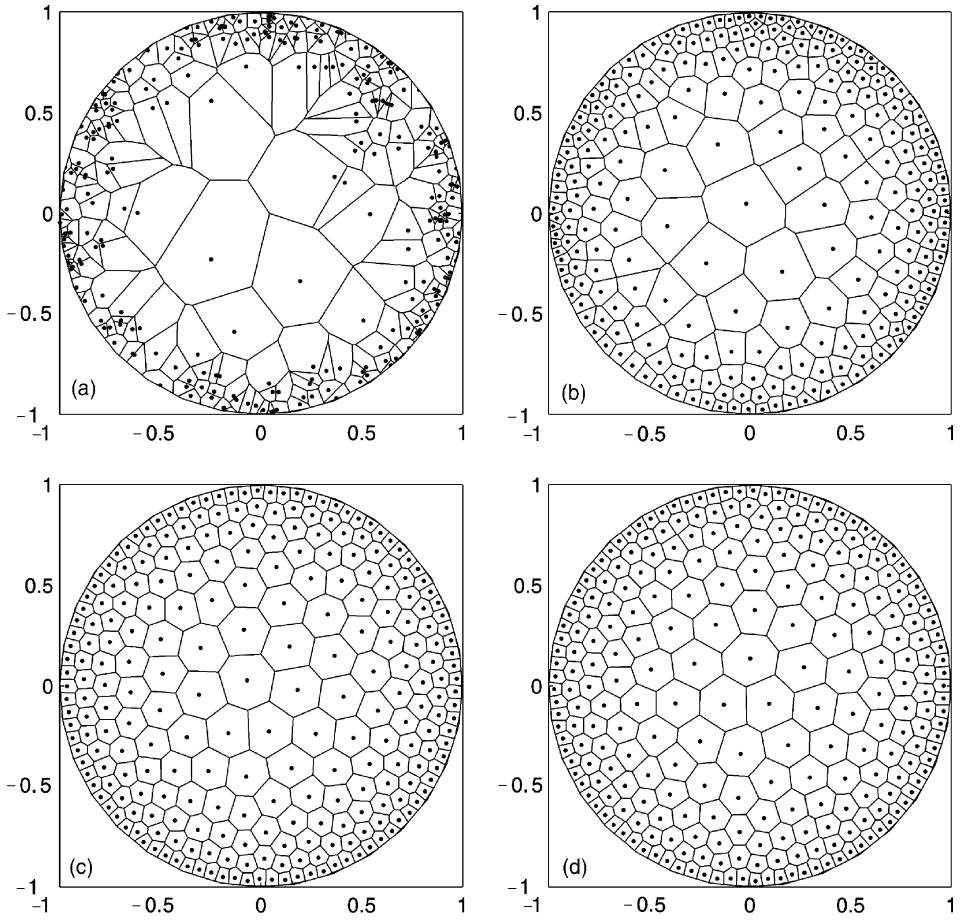

Fig. 11. Two-dimensional Voronoi diagrams for 256 generators in $\Omega = \{(x,y)|(x^2+y^2) < 1\}$ with the density function $\rho(x,y) = \mathrm{e}^{-5(1-x^2-y^2)}$: (a) Monte Carlo simulation; (b) approximate centroidal Voronoi diagram using Algorithm 5; (c) approximate centroidal Voronoi diagram using Algorithm 6 with $\alpha_1 = \beta_1 = 0$ and $\alpha_2 = \beta_2 = 1$; (d) approximate centroidal Voronoi diagram using Algorithm 6 with $\alpha_1 = \alpha_2 = \beta_1 = \beta_2 = 0.5$; 9.6×10^6 sampling points were used for (b), (c), and (d).

the parameters $\{\alpha_i, \beta_i\}_{i=1}^2$ for Algorithms 3 and 6. Of course, theoretical convergence analyses for Algorithms 3, 5, and 6 would also be desirable.

Acknowledgements

We thank CRAY Inc. for allowing us to use their CRAY T3E−600 computer located in Eagan, MN. The authors thank Glenn Luecke for some helpful suggestions concerning earlier versions of the paper.

References

[1] S. Arya, D. Mount, N. Netanyahu, R. Silverman, A. Wu, An optimal algorithm for approximate nearest neighbor searching, J. ACM 45 (1998) 891–923.
[2] J. Bentley, Multidimensional binary search trees used for associative searching, Comm. ACM 18 (1975) 509–517.
[3] CRAY Inc. Web Site, http://www.cray.com.
[4] Q. Du, V. Faber, M. Gunzburger, Centroidal Voronoi tessellations: applications and algorithms, SIAM Rev. 41 (1999) 637–676.
[5] Q. Du, M. Gunzburger, Grid generation and optimization based on centroidal Voronoi tessellations, Appl. Math. Comput., 2001, in press.
[6] Q. Du, M. Gunzburger, L.-L. Ju, Meshfree, probabilistic determination of point sets and support regions for meshless computing, Comput. Meths. Appl. Mech. Engrg. 191 (2002) 1349–1366.
[7] J. Hartigan, Clustering Algorithms, Wiley-Interscience, New York, 1975.
[8] M. Jhum, Bootstrapping k-means clustering, J. Jpn. Soc. Comput. Stat. 3 (1990) 1–14.
[9] D. Lee, S. Baek, K. Sung, Modified k-means algorithm for vector quantizer design, IEEE Signal Proc. Lett. 4 (1997) 2–4.
[10] Y. Linde, A. Buzo, R. Gray, An algorithm for vector quantizer design, IEEE Trans. Comm. 28 (1980) 84–95.
[11] S. Lloyd, Least square quantization in PCM, IEEE Trans. Infor. Theory 28 (1982) 129–137.
[12] J. MacQueen, Some methods for classification and analysis of multivariate observations, in: L. Le Cam, J. Neyman (Eds.), Proceedings of the Fifth Berkeley Symposium on Mathematical Statistics and Probability, vol. I, University of California, 1967, pp. 281–297.
[13] M. Snir, S. Otto, S. Huss-Lederman, D. Walker, J. Dongarra, MPI: The Complete Reference, MIT, 1996.
[14] A. Okabe, B. Boots, K. Sugihara, S. Chiu, Spatial Tessellations: Concepts and Applications of Voronoi Diagrams, Wiley, Chichester, 2000.
[15] D. Pollard, Strong consistency of k-means clustering, Ann. Stat. 10 (1982) 135–140.
[16] S. Ross, A First Course in Probability, fifth ed., Prentice Hall, Englewood Cliffs, 1998.
[17] H. Samet, The Design and Analysis of Spatial Data Structures, Addison Wesley, Reading, 1990.