Mathematical and Computer Modelling, vol. 26, no. 4, 1997, pp. 71-78.

Fine-Grain Discrete Voronoi Diagram Algorithms in L_1 and L_∞ Norms

William A. Maniatty and Boleslaw K. Szymanski
Computer Science Department
Rensselaer Polytechnic Institute
Troy, NY 12180

November 25, 2002

Abstract

The well known Voronoi diagram problem partitions a space containing a finite set of points, P, in such a way that each partition contains a single point in P and the subspace for which it is the nearest point from the set. Adamatzky defined the *discrete Voronoi diagram* (DVD) problem as finding the Voronoi diagram in a discrete lattice. Adamatzky provides some efficient algorithms for computing DVDs on fine grained synchronous single instruction multiple data (SIMD) mesh architectures when either the L_1 or the L_∞ distance metric is used. This paper presents improvements to Adamatzky's algorithms that ensure their correctness without increasing their complexity.

1 Introduction

In the well known Voronoi diagram problem [5, 6], given are a space and a finite set of points, P, selected from it. The goal is to partition the space in such a way that each partition contains a single point in P and the subspace for which it is the nearest point from the set. It is often assumed that the space is continuous. Sequential algorithms for two-dimensional Voronoi Diagrams in the general case have complexity in $\Omega(|P|\log|P|)$. Chew and Fortune [3] recently demonstrated that if P is pre-sorted and a convex triangular distance function is used, the Voronoi diagram can be sequentially computed with $O(|P|\log\log|P|)$ complexity. Parallel Voronoi diagram algorithms have been designed for a variety of architectures [2]. Jeong and Lee [4] show that for a two-dimensional Euclidean space with N points in P, the Voronoi diagram can be computed in $O(\sqrt{N})$ message passing operations and computation steps on a $\sqrt{N}\times\sqrt{N}$ processor mesh. Adamatzky [1] defines the *discrete Voronoi diagram* (DVD) problem in which a Voronoi diagram is computed in d-dimensional discretized space. The algorithms solving this problem for the L_1 and L_∞ distance metrics are also presented in [1]. This paper describes improvements to these algorithms which ensure the quality and correctness of results generated.

2 Summary of Adamatzky's Algorithms

For a given lattice, L, with $P \subset L$ being a given set of points, Adamatzky [1] defines the DVD of P as a partitioning of L into disjoint sub-lattices, called Discrete Voronoi Partitions (DVP). The DVP about point $p \in P$, includes all points in L which are closer to p than any other point in P. Letting $d(x,y)$ represent the distance between points $x, q \in L$, $\mathrm{DVP}(p) = \{x \in L, \forall q \in P : d(x,p) \le d(x,q)\}$[1] and $\mathrm{DVD}(P) = \bigcup_{p\in P} \mathrm{DVP}(p)$. Given a set of two points, $P = \{p,q\}$, the *bisector*, $B(p,q)$, is the set of points (nearly) equidistant from p,q which partition the lattice into the sub-lattices $\mathrm{DVP}(p), \mathrm{DVP}(q)$. Adamatzky restricts his work to the L_1 and L_∞ norms, and defines bisectors as $B(p,q) = \{x \in L : |d(p,x) - d(q,x)| \le 1\}$. He describes a series of rules for establishing the shape and position of bisectors (for the sake of space, we are not citing these rules here, they can be found in [1]).

The algorithm works as follows:

[1] any ties between $p, q \in L$ such that $d(x,p) = d(x,q)$ are broken arbitrarily, i.e. at most $x \in DVP(p)$ or $x \in DVP(q)$, but not both.

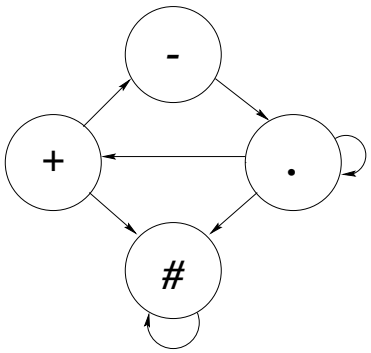

Figure 1: Local State Transitions in Adamatzky's Algorithm

Transition	Rule
$+ \rightarrow -$	$S(x)^t = 0$
$. \rightarrow .$	$S(x)^t = 0$
$. \rightarrow +$	$(S(x)^t \geq 1) \wedge (S_v(x)^t < 2) \wedge (S_h(x)^t < 2)$
$. \rightarrow \#$	$(S_v(x)^t = 2) \vee (S_h(x)^t = 2)$
$+ \rightarrow \#$	$((S(x)^t \geq 1)$

Table 1: Adamatzky's L_1 State Transition Rules

- Each vertex in the lattice at time $t \geq 0$ is in one of the following (local) states (see Figure 1):

  $+$ — the *excited* state, for a point, $p \in P$, the state of p at time $t = 0$ is $+$

  - — the *refractory* state

  $\#$ — the *bisector* state

  . — the *rest* state, for a point $x \in L - P$, the state of x at time $t = 0$ is .

  Additional information is computed about the neighborhood (stencil):

  - $S(x)^t$ — The number of (nearest) neighbors in the excited state about site x at time t.
  - $S_v(x)^t$ — The number of neighbors, to the north and south.
  - $S_h(x)^t$ — The number of neighbors to the east and west.

  The following notation is introduced here:

  - $S_d(x)^t$ - The number of neighbors to the northeast and southwest.
  - $S_t(x)^t$ - The number of neighbors to the northwest and southeast.

  Adamatzky uses a predicate equivalent to: $P(x)^t \stackrel{\text{def}}{=} (S_h(x)^t = 2) \vee (S_v(x)^t = 2) \vee (S_d(x)^t = 2) \vee (S_t(x)^t = 2)$
  Transition rules for the L_1 case are given in Table 1.
  Transition rules for the L_∞ case are listed in Table 2.

It should be noted that Adamatzky's algorithms amount to parallel (and synchronized) growth of the Breadth First Search (BFS) trees rooted at the source points, P. It requires $O(\sqrt{N})$ parallel operations and message passing steps. When two trees collide the collision region is labeled a bisector and growth in the direction of collision stops.

Transition	Rule
$+ \rightarrow -$	$S(x)^t < 4$
$. \rightarrow .$	$S(x)^t = 0$
$. \rightarrow +$	$(1 \leq S(x)^t < 4) \wedge (\neg P(x)^t)$
$. \rightarrow \#$	$(S(x)^t \geq 4) \wedge P(x)^t$
$+ \rightarrow \#$	$((S(x)^t \geq 5) \wedge (t \leq 2)) \vee ((S(x)^t \geq 4) \wedge (t > 2))$

Table 2: Adamatzky's L_∞ State Transition Rules

	0	1	2
0	+	.	.
1	.	.	.
2	.	.	+

Figure 2: A configuration in which point (1,1) has no transition defined by Adamatzky's L_1 and L_∞ algorithms. In this and following figures we present lattices with the first row of digits defining the x-coordinates of lattice points and the first column of digits defining the y-coordinates.

3 Some Direct Adjustments to the L_∞ Version of the Algorithm

Consider two points, (p_1, q_1) and (p_2, q_2), in P, such that $|p_1 - p_2| = |q_1 - q_2|$ and $|p_1 - p_2| \bmod 2 = 0$, i.e. they are separated by an odd number of vertices on the diagonal. This induces a failure of the $. \rightarrow \#$ rule. The problem is that there is no state transition rule specified for the case shown in Figure 2 (note that the site in the center of Figure 2 is a bisector). Therefore, the transition $. \rightarrow \#$ needs adjustment, a better rule might be: $((S(x)^t \geq 4) \vee P(x)^t)$, which follows from de'Morgan's theorem. Testing shows that all 4^9 permutations of states in a neighborhood are defined using the adjusted rule. However computing bisectors at the boundary may fail under this rule, since the connectivity of these vertices differs from interior vertices. One solution is to augment the lattice with a strip of vertices initialized to the rest state at each lattice boundary. Assuming the lattice has $m \times m$ elements, the augmented lattice contains $(m+1) \times (m+1)$ elements. Such augmentation does not increase $O(m)$ asymptotic computational complexity of the algorithm.

4 An alternative bisector approach

In [1], Adamatzky also reports a known incorrect computation of some bisector points (labeling them as ".") for the L_1 case, which is coincidentally identical in structure to the case shown in Figure 2 (for an example, see Figure 4). The problem is caused by the algorithm inability to distinguish between colliding wave fronts of +'s and an expanding wave front of +'s. Unfortunately, unlike the L_∞ case, no direct extension to the L_1 algorithm fixes this problem. This is the result of a combination of the following assumptions used Adamatzky's algorithm:

1. The bisector of the lattice consists of a set of points of the lattice. The lattice is composed of both points

$t = 0$	$t = 1$	$t = 2$	$t = 3$	$t = 4$
012	012	012	012	012
0+..	0-+.	0.-+	0..-	0...
1...	1+#+	1-#-	1.#.	1.#.
2..+	2.+-	2+-.	2-..	2...

Figure 3: Boundary conditions interfering with bisector computation in the adjusted algorithm. Points are identified by (column,row) coordinates. Points $(2, 0)$ and $(0, 2)$ should be labeled bisectors.

$t=0$	$t=1$	$t=2$	$t=3$	$t=4$
012	012	012	012	012
0+..	0-+.	0.-+	0..-	0...
1...	1+.+	1-#-	1.#.	1.#.
2..+	2.+-	2+-.	2-..	2...

Figure 4: Adamatzky's L_1 algorithm cannot label bisectors at points $(0,0)$ or $(2,2)$.

(vertices) and the edges connecting them. We consider bisectors consisting of edges of the lattice below.

2. A small state space needs to be used due to a small (constant) amount of available memory per point (assuming one processor is allocated to each point). In many fine grained SIMD mesh architectures (such as the MasPar MP-1), there is a moderate amount of available space, making it feasible to assume there is $O(\log |P|)$ available space per point.

Consider an efficient sequential algorithm that can be implemented using a forest of BFS trees rooted at points in P. The algorithm can label each point as belonging to $DVP(p)$ as it inserts points into the BFS tree of $p \in P$. This can be done in $O(N)$ operations using $O(N \log |P|)$ memory. In this case the bisectors partition the graph across the edges and not along the vertices. Direct parallelization of this algorithm on a fine grained SIMD mesh topology yields an $O(\sqrt{N})$ time complexity and $O(\log |P|)$ memory requirement per point in L. Since it is possible for a point, $x \in L - P$ to be equidistant to several points, P, the bisecting cut is not uniquely defined. We can apply an arbitrary but consistent tie breaking scheme where $u \in DVP(w)$ is resolved by selecting $w \in W$ where w has the least rank with respect to P. Labeling the points in P by their row major position in the lattice requires $O(\log N)$ bits of information (this is no worse than the worst case of Adamatzky's algorithm assuming that $|P| \in O(\log N)$). Under such labeling, the tie breaker can always prefer points above and to the left (respectively). This algorithm will correctly (and implicitly) compute a bisector in all cases, while Adamatzky's L_1 algorithm cannot do so.

Another alternative is to modify the above algorithm to label points adjacent to points labeled from two or more different sources as bisectors at the time this condition is detected. The synchronous BFS tree discipline for growing the regions ensures that the newly labeled points are equidistant. Since the source information is embedded in the label, testing for this condition is not difficult. These regions are guaranteed to be convex, and this is qualitatively similar to Adamatzky's approach. In addition, our algorithm labels the bisectors correctly.

5 L_1 norm algorithm with a constant number of states

Recall that the approach suggested by Adamatzky could not label some bisectors correctly [1] (see Section 4) due to its small state space and the limited connectivity of the lattice. It should also be noted that Adamatzky's assumption of a constant amount of memory per processor is sometimes justified, i.e. for a hardware implementation using a systolic array. In this section both of these issues are addressed by an algorithm that requires a (small) constant number of states for computing the DVD for the L_1 norm, while preserving the $O(\sqrt{N})$ computational complexity.

Our algorithm works by enhancing the state space with directional information embedded in the wave fronts which it generates. The extends state space is as follows:

- D — The *Down* state replicating itself down and to the right from it current position.
- B — The *Border point* state, used for the end points of currently known diagonal bisectors.
- L — The *Left* state always moving down and to the left.
- R — The *Right* state traveling up and to the right.
- U — The *Up* state replicating itself up and to the left.

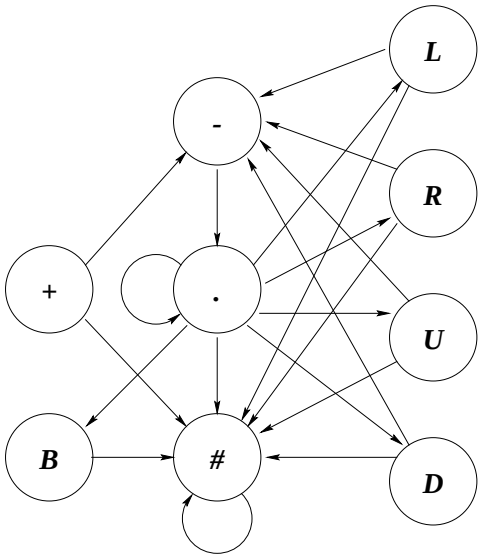

Figure 5: State table for new algorithm using L_1 distance metric

$t=0$	$t=1$	$t=2$	$t=3$	$t=4$	$t=5$	$t=6$
01234	01234	01234	01234	01234	01234	01234
0.....	0.....	0..U..	0..U-R.	0U-.-R	0-...-	0.....
1.....	1..U..	1.U-R.	1.U-.-R	1-...-	1.....	1.....
2..+..	2.L-R.	2L-.-R	2-...-	2.....	2.....	2.....
3.....	3..D..	3.L-D.	3.L-.-D	3-...-	3.....	3.....
4.....	4.....	4..D..	4..L-D.	4L-.-D	4-...-	4.....

Figure 6: Wave front propagation for L_1 directional algorithm

The state transition diagram is given in Figure 5. The transition rules are given in Table 3 in which the nearest north, east, south and west neighbors of each point in the lattice are referred to as n, e, s, w, respectively.

To avoid complicated boundary conditions, this algorithm employs an augmented lattice, as described in Section 3, which does not increase the asymptotic complexity of the algorithm.

The transitions in this algorithm seem more complex than Adamatzky's but they can be derived via case analysis, and are symmetric which makes them more readily understood. Figure 6 shows the propagation of a wave front in the absence of collisions.

To illustrate reasons for (and prove correctness of) the transition rules, we will discuss here the propagation of the D direction wave. By symmetry of the rules, the same proof applies to the other directions. Symmetry also limits the cases that we must consider to just three: in which the wave originating points are (i) colinear on the line parallel to one of the axes, (ii) colinear on the diagonal of the axes, and (iii) neither (i) or (ii).

Consider first the case (i), i.e., two points (p_1, q) and (p_2, q) with the same horizontal coordinate q. Consider first the case that $|p_1 - p_2| = 1$, then they are adjacent, and should be labeled bisectors[2], as indicated in the $(e = +)$ and $(w = +)$ terms of the $+ \rightarrow \#$ rule. When the points are not adjacent then the distance between them, $|p_1 - p_2| > 1$, is either even (see Figure 7) or odd (see Figure 8). In the former case, the condition for the points $((p_1 + p_2)/2, y)$ in the $. \rightarrow D$ rule will not have the $s = .$ term in the predicate satisfied at one instance, triggering the $. \rightarrow \#$ transition for those points. When $|p_1 - p_2|$ is odd, then $D \rightarrow -$ transition rule will not apply to points $((p_1 + p_2 + 1)/2, y)$ because of $s = .$ term, so a bisector label $\#$ will be generated for all of such points

[2]We exactly follow Adamatzky's definition of a bisector. In some applications, it might be useful not to label points in P as bisectors. Edge based partitioning does not need to make such a distinction.

Transition	Rule
$+ \to \#$	$(n = +) \vee (e = +) \vee (s = +) \vee (w = +)$
$+ \to -$	$(n \neq +) \wedge (e \neq +) \wedge (s \neq +) \wedge (w \neq +)$
$- \to .$	always
$\# \to \#$	always
$B \to \#$	always
$. \to .$	$(n \in \{., -\}) \wedge (s \in \{., -\}) \wedge (e \in \{., -\}) \wedge (w \in \{., -\})$
$. \to B$	$((n = w = .) \wedge (e \in \{L, +\}) \wedge (s = \{U, +\})) \vee ((n = e = .) \wedge (w \in \{R, +\}) \wedge (s \in \{U, +\})) \vee$ $((s = w = .) \wedge (e \in \{L, +\}) \wedge (n \in \{D, +\})) \vee ((s = e = .) \wedge (w \in \{R, +\}) \wedge (n \in \{D, +\}))$
$. \to L$	$[(e \in \{+, U\}) \vee ((e = L) \wedge (n \neq D)) \vee ((e = D) \wedge (n \in \{L, B\}))] \wedge [(s = .) \vee ((s = B) \wedge (e = U))] \wedge (w = .)$
$. \to R$	$[(w \in \{+, D\}) \vee ((w = R) \wedge (s \neq U)) \vee ((w = U) \wedge (s \in \{R, B\}))] \wedge [(n = .) \vee ((n = B) \wedge (w = D))] \wedge (e = .)$
$. \to U$	$[(s \in \{+, R\}) \vee ((s = U) \wedge (e \neq L)) \vee ((s = L) \wedge (e \in \{U, B\}))] \wedge [(w = .) \vee ((w = B) \wedge (s = R))] \wedge (n = .)$
$. \to D$	$[(n \in \{+, L\}) \vee ((n = D) \wedge (w \neq R)) \vee ((n = R) \wedge (w \in \{D, B\}))] \wedge [(e = .) \vee ((e = B) \wedge (n = L))] \wedge (s = .)$
$. \to \#$	$\neg [(. \to .) \vee (. \to B) \vee (. \to L) \vee (. \to R) \vee (. \to U) \vee (. \to B)]$
$L \to -$	$(w = .) \wedge (s \in \{., \#\}) \wedge (n \notin \{L, D\})$
$L \to \#$	$\neg(L \to -)$
$R \to -$	$(e = .) \wedge (n \in \{., \#\}) \wedge (s \notin \{R, U\})$
$R \to \#$	$\neg(R \to -)$
$U \to -$	$(n = .) \wedge (w \in \{., \#\}) \wedge (e \notin \{L, U\})$
$U \to \#$	$\neg(U \to -)$
$D \to -$	$(s = .) \wedge (e \in \{., \#\}) \wedge (w \notin \{R, D\})$
$D \to \#$	$\neg(D \to -)$

Table 3: Transition rules for the directional L_1 Algorithm, boundary conditions omitted

$t = 0$	$t = 1$	$t = 2$	$t = 3$	$t = 4$	$t = 5$
01234	01234	01234	01234	01234	01234
0.....	0..U..	0.U-R.	0U-.-R	0-...-	0.....
1..+..	1.L-R.	1L-.-R	1-...-	1.....	1.....
2.....	2..#..	2.###.	2#####	2#####	2#####
3..+..	3.L-R.	3L-.-R	3-...-	3.....	3.....
4.....	4..D..	4.L-D.	4L-.-D	4-...-	4.....

Figure 7: Bisecting (p_1, q) and (p_2, q) when $|p_1 - p_2| \bmod 2 = 0$

$t=0$	$t=1$	$t=2$	$t=3$	$t=4$
01234	01234	01234	01234	01234
0..+..	0.L-R.	0L-.-R	0-...-	0.....
1.....	1..D..	1.L#D.	1L###D	1#####
2.....	2..U..	2.U#R.	2U###R	2#####
3..+..	3.L-R.	3L-.-R	3-...-	3.....

Figure 8: Bisecting (p_1, q) and (p_2, q) when $|p_1 - p_2| \bmod 2 = 1$

$t=0$	$t=1$	$t=2$	$t=0$
012345	012345	012345	012345
0.....	0....U.	0.UU-R.	0.U##.-R
1....+.	1..UL-R	1.U##.-	1U-##..-
2..+...	2.L-RD.	2L-.##D	2-..##-D
3.....	3..D...	3.L-DD.	3L-.##D.

Figure 9: Bisecting (p_1, q_1) and (p_2, q_2) when $(|p_1 - p_2| \neq |q_1 - q_2|) \wedge (p_1 \neq p_2) \wedge (q_1 \neq q_2)$

immediately after they are labeled D. Similarly, points $((p_1 + p_2 - 1)/2, y)$ will be marked as a part of a bisector after first being labeled with R's.

Next, consider case (iii), in which points (p_1, q_1) and (p_2, q_2) are placed in the lattice in such a way that $|p_1 - p_2| \neq |q_1 - q_2|$, $p_1 \neq p_2, q_1 \neq q_2$, (see Figure 9) . In such a case $D \rightarrow -$ transition rule is not applicable to any Ds in one or two bisecting columns, because for one of them (column 3 in Figure 9) $e \in \{., \#\}$ term evaluates to false, and for another (if it is different from the previous one, column 4 in Figure 9) because $w \notin \{R, D\}$ term. The rotated and symmetric cases as well as transitions from other labels to # can be analyzed in a similar way.

Finally, consider the case (ii) of points (p_1, q_1) and (p_2, q_2) for which $|p_1 - p_2| = |q_1 - q_2|$, which is the most complicated case, as seen in Figure 10 and Figure 11. Consider the case where $|p_1 - p_2| > 1$ (see Figure 10), the ends of the diagonal bisector must be labeled differently than the interior points. If at $t = 3$ point $(1, 4)$ were labeled # instead of B, then, at $t = 4$, the L waves from the two initial sources would merge, making it impossible to distinguish them from a wave front generated from a single source. The end point of the D wave at point $(1, 3)$ at $t = 3$, is an example of a necessary precondition, and is uniquely recognized by the term $(s = w = .) \wedge (e \in \{L, +\}) \wedge (n \in \{D, +\})$, which is satisfied in the $. \rightarrow B$ rule but not in the $. \rightarrow \#$ rule, which has the negation embedded in it (symmetric rules apply in all directions). The end marked B at point $(1, 4)$ at $t = 3$ is used at $t = 4$ to induce a $. \rightarrow D$ transition at point $(0, 4)$ using the rule's $((e = B) \wedge (n = L))$ term. Once the D label has been strategically introduced, at $t = 5$, the end point $(0, 5)$ can be correctly labeled, allowing the bisector to propagate properly. Care needs to be taken to ensure that the D label at point $(0, 4)$ goes to $-$ at time $t = 5$, which motivated the presence of the # label in the term $e \in \{., \#\}$ of the transition $D \rightarrow -$.

$t=0$	$t=1$	$t=2$	$t=3$	$t=4$	$t=5$	$t=6$
012345	012345	012345	012345	012345	012345	012345
0.....	0.U....	0U-R...	0-.-R..	0...-R.	0....-B	0.....#
1.+....	1L-R...	1-.-R..	1...-B.	1....#U	1....#-	1....#.
2.....	2.D....	2L-D.U.	2-.-#-R	2...#.-	2...#..	2...#..
3.....	3....U.	3.D.U-R	3L-#-.-	3-.#...	3..#...	3..#...
4....+.	4...L-R	4..L-.-	4.B-...	4D#....	4-#....	4.#....
5.....	5....D.	5...L-D	5..L-.-	5.L-...	5B-....	5#.....

Figure 10: Bisecting (p_1, q_1) and (p_2, q_2) when $|p_1 - p_2| = |q_1 - q_2| > 1$

$t=0$	$t=1$	$t=2$	$t=3$	$t=4$	$t=5$
012345	012345	012345	012345	012345	012345
0.....	0.....	0...U..	0..U-R.	0.L-.-R	0B-...-
1.....	1...U..	1..L-R.	1.B-.-R	1U#...-	1-#....
2...+..	2..B-R.	2.U#.-R	2U-#..-	2-.#...	2..#...
3..+...	3.L-B..	3L-.#D.	3-..#-D	3...#.-	3...#..
4.....	4..D...	4.L-R..	4L-.-B.	4-...#D	4....#-
5.....	5.....	5..D...	5.L-D..	5L-.-R.	5-...-B

Figure 11: Bisecting (p_1, q_1) and (p_2, q_2) when $|p_1 - p_2| = |q_1 - q_2| = 1$

When $|p_1 - p_2| = |q_1 - q_2| = 1$ (as shown in Figure 11) then the algorithm initiates the diagonal bisector using a symmetric technique (note that there are no interior points for this diagonal bisector). The end point at $(3,3)$ is correctly recognized at $t = 1$ by the term $(s = w = .) \wedge (e \in \{L, +\}) \wedge (n \in \{D, +\})$, satisfied in the $. \rightarrow B$ rule but not in the $. \rightarrow D$ rule. It is necessary to propagate the diagonal bisector by generating a D at point $(4,3)$, to the right of a B at $t = 2$, motivating the $(n = R) \wedge (w \in \{D, B\})$ term in the $. \rightarrow D$ transition.

6 Conclusion

This paper presents improvements to Adamatzky's DVD algorithms. The direct extensions given in Section 3 for the L_∞ case correct the algorithm without impacting the complexity of the algorithm. The extensions given in Section 4 provide two alternative approaches. One algorithm bisects the lattice along its edges rather than along its points (vertices) with an additional $O(\log |P|)$ memory cost per point in the lattice. The other algorithm bisects the lattice along its vertices (like Adamatzky's) with an additional $O(\log |P|)$ memory cost per point in the lattice, but can correctly label all cases (which Adamatzky's algorithm cannot). Finally, the algorithm given in Section 5 uses a constant number of states and bisects the lattice along the vertices for L_1 norm, finding correct bisectors in cases in which the more limited algorithm proposed by Adamatzky fails.

Acknowledgments

The authors would like to thank A. Adamatzky for reviewing some of our analysis and A. Shapira for stimulating conversations about the problem. This work was partially supported by the NSF grant BIR-9320264, and fellowship from IBM Corporation. The contents of this entry does not necessarily reflect the position or policy of the U.S. Government—no official endorsement should be inferred or implied.

References

[1] A. I. Adamatzky. Voronoi-like partition of lattice in cellular automata. *Mathematical and Computer Modelling*, 23:51–66, February 1996.

[2] S. G. Akl and K. A. Lyons. *Parallel Computational Geometry.* Prentice-Hall, Englewood Cliffs, NJ USA, 1993.

[3] L. P. Chew and S. Fortune. Sorting helps for Voronoi diagrams. *Algorithmica*, 18:217–218, 1997.

[4] C. S. Jeong and D. T. Lee. Parallel geometric algorithms on a mesh connected computer. *Algorithmica*, 5:155–177, 1990.

[5] Ketan Mulmuley. *Computational Geometry an Introduction through Randomized Algorithms.* Prentice Hall, Englewood Cliffs, NJ USA, first edition, 1994.

[6] F. P. Preparata and M. I. Shamos. *Computational Geometry: an Introduction.* Springer-Verlag, New York City, NY USA, 1985.