Integer Gradient for Cellular Automata: Principle and Examples

Luidnel Maignan, Frédéric Gruau

Integer Gradient for Cellular Automata: Principle and Examples

Luidnel Maignan* and Frédéric Gruau*†‡§
* INRIA Futurs Saclay - 4, rue Jacques Monod, 91893 ORSAY Cedex, France
† LRI - Université Paris-Sud 11, bâtiment 490, 91405 Orsay Cedex, France
‡ LIRMM - 31 rue Ada, 34000 Montpellier, France
§ UWE, Frenchay Campus, Coldharbour Lane, Bristol BS16 1QY, UK

***Abstract*—When programming a spatial computing medium such as a cellular automaton, the hop count distance to some set of sources (particles) is an often used information. In particular, we consider the case where the sources themselves are moving. Due to the locality of communication, only an estimation of the distance can be made at each time step, and each location. When no assumption is made on the size of the medium, that distance takes its values in $\mathbb{N}$, which is not desirable, because it does not lead to finite state. This paper shows how to use the modulo operation to project that set of $\mathbb{N}$-fields into a finite set of $\mathbb{Z}/n\mathbb{Z}$-fields. Using the modulo stored at each site, we show that we are still able to compute the local differential of the original field, allowing to manipulate the former as a directional gradient. It allows us to evaluate the direction of the nearest source, provided the sources move at bounded speed, less than one site per time unit. This information can be used to solve several problems of spatial nature. In the particular case of cellular automata, we present two rules using the modulo representation of distance field: Voronoi Diagram and Convex Hull. The two rules applies for moving sources, although at the moment, the convex hull has been implemented only for static sources.**

I. Introduction

Spatial computing considers a computing medium made of many elementary processing elements with local connections and homogeneously distributed in space. When programming the medium, a common ingredient is to evaluate the distance to some source objects. Using this information, it is possible to move an agent away from or towards the sources [1], compute the Voronoi Diagram [2][3], or the Convex Hull [4] among many examples. In all of those examples, the actual value of the distances does not matter; only the differential of the distances is actually used.

Depending on the framework, various kind of rules can be used to compute the distances. A classical way to compute them is to diffuse a gradient of chemical, and use the chemical density as a distance information [1]. In this case, the distances take their values in $\mathbb{R}$. In cases where the space can be described by a graph, the hop count distance can also be used. Between two nodes, it is defined as the number of edges of the shortest path connecting them. Then the distance is a value in $\mathbb{N}$.

In this article, we consider a spatial computing medium described by a bounded degree graph of nodes which are processing elements operating synchronously, in parallel. The hop count distance is computed to the nearest source, at any location. Also, the distance is used in a way such that only its differential really matters. We call the *distance field* the values associated to each node at each time. Nodes can host a point source which can dynamically move from one node to a neighbor one. A distance field is such that, the distance stored in one node is an estimation of the distance to the nearest node hosting a source at that particular time. The slower the movement of sources is, the more precise the estimation becomes. When (and if) the sources stop moving, the estimated distances should converge to the exact distances values.

We show that, if the differential is the only information that is exploited from the distance field, it is possible to transform the original distance field defined on $\mathbb{N}$ by a field defined on finite state set producing exactly the same behavior. We choose the word *output* to describe this *differential*, because we organize the rules managing a processing element into several layers, where the layer responsible for computing the distance field *outputs* this differential to some higher level layer which is using it. We describe two cellular automata rules that only use the differential of the distance field. The first one is the Dynamic Voronoi Diagram of a set of moving sources, and the second is the Convex Hull of a set of static sources.

II. Infinite Distance Field with Finite State

This section is organized as follows. We start off by giving two rules that produce $\mathbb{N}$-valued distance fields with somewhat different properties. Then we underline the fact that distance fields are often used as directional gradients. We then define reasonable properties that needs to be satisfied in order to transform the $\mathbb{N}$-distance fields into $\mathbb{Z}/n\mathbb{Z}$-distance fields for some bounded n, providing the way to produce exactly the same output from the new field.

A. Computation and use of the distance field

Before going into any details of the transformation of the $\mathbb{N}$-distance field into a finite distance field, let us describe first how it is possible to locally compute the $\mathbb{N}$-distance field. Considering a predicate $source_t(i)$ which is true if there is a source in the site i at time t, the distance field d is computed by the following rule:

$$d_{t+1}(i) = \begin{cases} 0 \text{ if } source_{t+1}(i) \\ 1 + \min\{d_t(j)\}_{j \in N(i)} \text{ otherwise.} \end{cases} \tag{1}$$

This well-known rule [5][3][2] converges to a configuration where each node i of the considered graph has its $d(i)$ equals to the distance to the nearest source. Using this information, an agent moving in the graph can decide to get closer or further from the sources. In [2], the case where the sources positions evolve in time is also considered. A slightly different version of that rule is used, which ensures that the distance field always indicates the direction to nearest source for any site at any time:

$$d_{t+1}(i) = \begin{cases} 0 \text{ if } source_{t+1}(i) \\ 1 \text{ if } \neg source_{t+1}(i) \wedge source_t(i) \\ 2 + \min\{d_t(j)\}_{j \in N(i)} \text{ otherwise.} \end{cases} \tag{2}$$

In many important applications of this spatial gradient, it is often the case that only the *difference* between neighboring state values is required, not the actual absolute values of the states. Formally define:

$$\delta_t(i,j) = d_t(j) - d_t(i) \text{ for } j \in N(i) \tag{3}$$

Many applications depend just on δ, not on the values of d (except perhaps at the origin 0). The set of differences can be finite even when the set of absolute values is not. When this is the case, we will now show how to take advantage of this fact to quotient the state space into a finite set without affecting the outputs.

B. Properties of the distance field

The distance field can be seen as a layer that takes a set of sources as input and allows to compute the differential of the distance to the nearest sources as output. Whether this output is used in a dynamical or static way, some properties are required of the field in order to admit a finite state representation.

The most important property is that the difference between two neighbor sites must be bounded by some fixed constant Δ at any time. As we have already defined $\delta_t(i,j)$ in Eq. (3), the property can be written:

$$\forall i \forall j \in N(i) : \ \delta_t(i,j) \in [-\Delta, \Delta] \tag{4}$$

If it is not the case, then the set of outputted values is infinite, which prevent any finite state field to produce it.

The property (4) implies a bound on the speed of the sources in the dynamical case. Indeed, if the sources move one step each time for example, it moves as fast as the information in the field, so the sites can not update their values fast enough to preserve the Δ bound. In fact, the Δ bound is dependent on the speed of the sources when considering the two rules described above.

C. Transformation into a finite state field

If the property (4) is satisfied, then a simple modulo operator is enough to project the initial field d into a field f where the output is exactly the same:

$$f_t(v) = d_t(v) \bmod (2.\Delta + 1). \tag{5}$$

By this transformation, the $\mathbb{N}$-distance field is transformed into a $\mathbb{Z}/n\mathbb{Z}$-distance field, with $n = 2.\Delta + 1$. The former value is the size of the output set $\{-\Delta, \ldots, 0, \ldots, \Delta\}$.

When applying the modulo operator, the global order on the values is lost, i.e. some values that were less than their neighbor's value have modulo value being greater. Because the definition of distance field depends on taking a minimum, we must correct for this explicitly. The neighbors who have these "mis-ordered values are:

$$N^+(i) = \{j \in N(i) | f(j) \geq (f(i) - \Delta) \bmod n\}$$

In order to correctly transform the rules (1) and (2), the values of $N^+(i)$ must be tested by the minimum operator before the others, because, if there are any, they correspond to the true minimal values in the original field. We obtain respectively:

$$f_{t+1}(i) = \begin{cases} 0 \text{ if } source_{t+1}(i); \text{ else:} \\ 1 + \min\{f_t(j)\}_{j \in N^+(i)} \text{ if } N^+(i) \neq \emptyset \\ 1 + \min\{f_t(j)\}_{j \in N(i)} \text{ otherwise} \end{cases} \tag{1'}$$

$$f_{t+1}(i) = \begin{cases} 0 \text{ if } source_{t+1}(i); \text{ else:} \\ 1 \text{ if } source_t(i); \text{ else:} \\ 2 + \min\{f_t(j)\}_{j \in N^+(i)} \text{ if } N^+(i) \neq \emptyset \\ 2 + \min\{f_t(j)\}_{j \in N(i)} \text{ otherwise} \end{cases} \tag{2'}$$

where $\emptyset$ denotes the empty set. The output function is transformed into:

$$\delta_t(i,j) = \begin{cases} (f(j) - n) - f(i) \text{ if } j \in N^+(i) \\ f(j) - f(i) \text{ otherwise} \end{cases}$$

III. Voronoi Diagram and Convex Hull

We present two examples in the context of cellular automata, where distance fields are used to solve non-trivial problems. Once the preceding method is applied, a finite state automaton can be derived.

A. Discrete Dynamic Voronoi Diagram

The Voronoi Diagram of a set of sources can be defined as the set of sites that are equidistant to at least two sources. This is a very practical structure [6] which makes a direct use of a distance field.

The computation of the Voronoi Diagram of a static set of sources is described in [7], [8]. In this solution, a wave is sent by all the sources at the same time. When two waves meet, a Voronoi region boundary is materialized. This is possible because the synchronicity of the system is used to encode the distances in time, so that when two waves meet on a site, that site is equidistant to the two corresponding sources.

An interesting generalization is the dynamic computation of the Voronoi Diagram, under the motion of the sources.

We first designed a solution by sending waves periodically from all the moving sources. The detection of the Voronoi Diagram is done as described above, but materialized boundaries are removed after one period. Then the next waves re-materialize the boundaries at the correct location according to the (possibly) new positions of the sources. However, when the sources stop moving, they continue to send waves, which was unsatisfying. We were looking for a rule that could converge, i.e. when sources stop moving, the state of every site eventually fixes.

So we programmed a second solution by using the distance field to directly express the distances in space instead of in time. More precisely, by using the derivative of the distance field, equidistant points that compose the Voronoi Diagram can be seen as intersections of two circles with same radius, centered in two sources. Distance values *in* the circle are lower than the radius, while distances *out* of the circle are greater. Detection of the Voronoi Diagram is done by using the values of the distances of sites in the neighborhood of each site. Hence, almost all[1] local configurations corresponding to a Voronoi Diagram boundary can be characterized by the following local predicate:

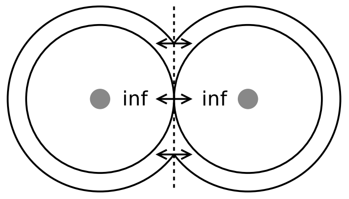
$$\begin{aligned} V_t(i) \quad = \quad & \exists j,k \in N(i)^2, \\ & axis(j,i,k) \wedge d_t(i) > \max\{d_t(x)\}_{x\in\{j,k\}}, \end{aligned} \tag{6}$$

where $axis(a,b,c)$ is a predicate that is true if the sites a, b, and c are along the same axis. Eq. (6) is valid no matter the number of dimensions of the space. Figure 1(a) shows two examples of circles with different radius, the resulting intersections, and the singular axis.

This algorithm has been implemented as an hexagonal cellular automata rule. The distances are computed using rule (2), as its additional dynamic property is required, and Figure 1(b) illustrates the results. The rule also works fine when the sources move. Indeed, the rule is a simple detection of distances pattern, so the Voronoi Diagram boundary is updated as soon as the distances are; and when the sources stop moving long enough, the boundaries stop moving as soon as the distances converge.

Finally, the algorithm would work on other graphs, as it only relies on the notion of circles, which is constructed only with the the notion of distances. However, the errors due to the discrete nature of the space present in hexagonal grid case would be worse in a non-standard graph. Also, there is no obvious relation between a arbitrary graph and the euclidean space, so things like *axis*, for example, need to be redefined. The identification of well-behaving graphs could be of great interest.

[1]Almost all, because, the discrete nature of the space requires some minor arrangement.



(a)

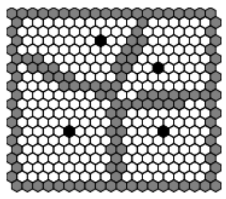

(b)

Fig. 1. (a) For every point of the Voronoi Diagram, there exists a singular axis such that the distance field is a local maximum (b) Result of the computation of the Voronoi Diagram on an hexagonal grid

B. Discrete Convex Hull

The discrete convex hull of a set of sources can be defined as the minimal connected component including all the sources and such that every shortest path between two points of the hull is also in the hull.

There exists a cellular automata rule that solves this problem for a connected set of sources [4]. Its rule is defined for the Von Neumann neighborhood as:

$$c_{t+1}(i) = \begin{cases} 1 \text{ if } \sum\{c_t(j)\}_{j\in N(i)} \geq 4 \\ c_t(i) \text{ otherwise.} \end{cases} \tag{7}$$

The initial configuration encodes the source presence in a site i as $c_0(i) = 1$, and the rule produces the convex hull according to our definition only if we consider the Euclidean length of the paths as distances. To our knowledge, there also exist algorithms that work on sets of non-connected sources, but all of them make some assumption of a previous knowledge about an upper-bound on the distance between the sources [9], [10], [11].

We will sketch here an algorithm that computes the convex hull without any previous knowledge. Indeed, as the definition of the convex hull involves shortest paths, we will see that it can be constructed from distance fields, which are the distances to the nearest source, i.e. the length of the shortest paths to the set of sources. As this result has not been published yet, only the insights will be given.

Our algorithm consists of the parallel composition of three stages:

1) Detection of *middle points*
2) Propagation to neighbors of lower distance value
3) Convexification

The two first stages alone connect arbitrarily distant sources without adding any point outside of the convex hull. As the connectivity problem is solved, using a rule related to Eq. (7) as third stage produces the convex hull. Let us define and describe the three stages more carefully:

1) Middle points: "Middle points" are defined as the points on a shortest path between two sources, and equidistant to them. Since they lie along a shortest path between two points of the convex hull, they are in.

When two sources are sufficiently close compared to others sources, their middle points can be detected by only using the

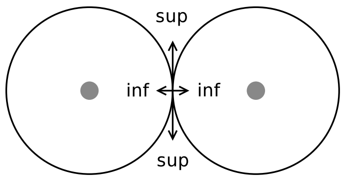


Fig. 2. Detection of middle points. The two circles are tangent, they correspond to all points that have the same distance as the middle point. The two axis are the maximum axis (inf-equal-inf) and the minimum axis (sup-equal-sup).

distance field from the set of sources. Indeed, middle points are the intersection of two tangent circles centered on the two sources and having the same radius. Figure 2 shows that relying of the same property of the circles used previously to define Eq. (6), the middle points can be detected using two singular axis. Thus, the rule is related to the following local predicate:

$$\begin{aligned} M_t(i) \quad = \quad & \exists j,k,l,m \in N(i)^4, \\ & axis(j,i,k) \wedge d_t(i) > \max\{d_t(x)\}_{x\in\{j,k\}} \wedge \\ & axis(l,i,m) \wedge d_t(i) < \min\{d_t(x)\}_{x\in\{l,m\}}, \end{aligned}$$

2) Propagation and Delaunay-like Structure: During the propagation stage, the distant sources get interconnected. When taking two sources into account, the propagation draws every shortest paths from one source to the other. All these paths go through the middle points, and those middle points have the maximal distance value. Hence, it is enough to progress backwards from the middle points by detecting points that have a lower-valued neighbor:

$$P_t(i) = \exists j \in N(i),\, c_t(j) = 1 \wedge d_t(j) > d_t(i)$$

This makes the middle points propagate towards its two corresponding sources without adding points not belonging to the convex hull. In fact, going from a value to a lower value corresponds always to a shortest path from each site to its nearest source. The result is a connected set of sites connecting closer sources in a Delaunay-like fashion.

3) Convexifying Rule: The two previous stages transform a non-connected set of sources into a connected one by adding sites that lie in the convex hull. The last thing to do is to adapt the predicate used in the rule (7), that is able to transform any connected set into a convex one, for the hexagonal case with hop count distance:

$$C_{t+1}(i) = \sum\{c_t(j)\}_{j\in N(i)} \geq 3$$

Now that all the stages are defined, let us define their parallel composition:

$$c_{t+1}(i) = \begin{cases} 1 \text{ if } M_t(i) \vee P_t(i) \vee C_t(i) \\ c_t(i) \text{ otherwise.} \end{cases}$$

Starting from a configuration where only the sites i containing a source are such that $c_0(i) = 1$, this rule constructs the convex hull of arbitrarily distant sources. It converges in the following sense: if we wait sufficiently long, any arbitrary large area will stabilize. Thanks to the distance field modulo representation, this algorithm also requires only a finite number of state, as one can check that every defined predicates can be rewritten in order to only use differences. Figure 3 shows an execution trajectory where the three stages have been applied one by one, and three snapshots of the execution trajectory of the parallel composition of the three stages.

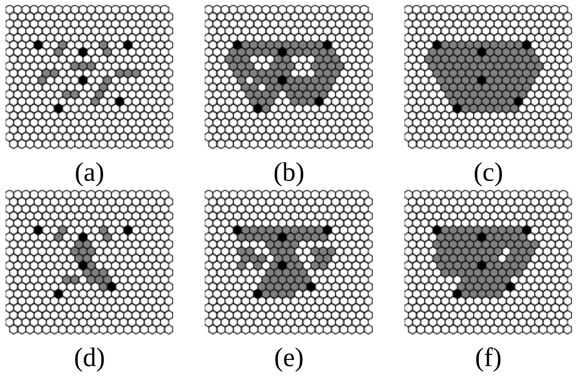


Fig. 3. (a) Middles detection (b) Delaunay-like structure (c) Convex Hull (d,e,f) intermediate configurations of the parallel composition of the three

IV. Conclusion

In this article, we shown how to transform an unbounded distance field into a bounded-valued field. We show that this is possible only is the differences between adjacent sites is bounded, and we gives the relation that link that bound and the number of state of the transformed field. In spite of the fact that every thing is done only for the distance field, the described transformation may be usable for other integer fields satisfying the bounded difference requirement. We have also describes two algorithms using the intuitive language of integers: the Dynamic Voronoi Diagram and the Convex Hull. For both examples, it is clear that the distance is only used for its differential, allowing the transformation to be used in order to obtain cellular automata rules. Those algorithms have been programmed and tested in the case of synchronous cellular automata, using a square and hexagonal lattices. Future work includes expending this result to quasi regular architectures and quasi synchronous update, such as Amorphous Computers [12].

Acknowledgment

We acknowledge Andrew Adamatzky who pointed us to the Convex Hull problem and provide his knowledge on existing results and challenges on the subject, Christine Eisenbeis for her advices, and Sonia Dahdouh for her careful support, proof-reading and advice. We would like to thank the reviewers of the article also, as they greatly contribute to improve its quality by their comments, and Daniel Yamins for his corrections and suggestions. We acknowledge support from EPRC grant EP/F003811/1 on general purpose spatial computation.

References

[1] D. Coore, "Botanical computing: a developmental approach to generating interconnect topologies on an amorphous computer," Ph.D. dissertation, MIT, 1999.

[2] L. Maignan and F. Gruau, "A 1D cellular automata that moves particles until regular spatial placement," 2008.

[3] L. Maignan, "Uniformisation de la répartition spatiale d'un ensemble de points par diagramme de voronoï: Implémentation sur automate cellulaire," Ph.D. dissertation, Universite Paris XI, 2007.

[4] A. Adamatzky, *Identification of Cellular Automata*. Taylor & Francis, 1994.

[5] D. Yamins, "A theory of local-to-global algorithms for one-dimensional spatial multi-agent systems," Ph.D. dissertation, Harvard University Cambridge, Massachusetts, 2007.

[6] F. Aurenhammer, "Voronoi diagramsa survey of a fundamental geometric data structure," *ACM Comput. Surv.*, vol. 23, no. 3, pp. 345–405, 1991.

[7] A. Adamatzky, "Voronoi-like partition of lattice in cellular automata," *Mathematical and Computer Modelling*, vol. 23, pp. 51–66(16), February 1996.

[8] M. W.A. and S. B.K., "Fine-grain discrete Voronoi diagram algorithms in L_1 and L_∞ norms," *Mathematical and Computer Modelling*, vol. 26, pp. 71–78(8), August 1997.

[9] R. Jarvis, "On the identification of the convex hull of a finite set of points in the plane," *Information Processing Letters*, vol. 2, pp. 18–21, 1973.

[10] F. P. Preparata and S. J. Hong, "Convex hulls of finite sets of points in two and three dimensions," *Commun. ACM*, vol. 20, no. 2, pp. 87–93, 1977.

[11] F. P. Preparata, "An optimal real-time algorithm for planar convex hulls," *Commun. ACM*, vol. 22, no. 7, pp. 402–405, 1979.

[12] H. Abelson, D.Allen, D. Coore, C. Hanson, G. Homsy, J. T. F. Knight, R. Nagpal, E. Rauch, G. J. Sussman, and R. Weiss, "Amorphous computing," *Commun. ACM*, vol. 43, no. 5, pp. 74–82, 2000.