Thèse présentée pour obtenir le grade de:

Docteur de l'Université Paris-Sud XI

spécialité:

Informatique

par:

Luidnel Maignan

Titre:

Points, Distances, et Automates Cellulaires: Algorithmique Géométrique et Spatiale

Soutenue le 8 décembre 2010 devant le jury composé de:

Gruau	Frédéric	Encadrant de thèse
Eisenbeis	Christine	Directrice de thèse
Najman	Laurent	Président
Boussinot	Frédéric	Rapporteur
Giavitto	Jean-louis	Rapporteur
Beauquier	Joffroy	Examinateur
Yunes	Jean-Baptiste	Examinateur

Abstract

Spatial computing aims at providing a scalable framework where computation is distributed on a uniform computing medium and communication happen locally between nearest neighbors. We study the particular framework of cellular automata, using a regular grid and synchronous update. As a first step towards generic computation, we propose to develop primitives allowing to structure the medium around a set of particles. We consider three problems of geometrical nature: moving the particles on the grid in order to uniformize the density, constructing their convex hull, constructing a connected proximity graph establishing connection between nearest particles. The last two problems are considered for multidimensional grid while uniformization is solved specifically for the one dimensional grid.

The work approach is to consider the metric space underlying the cellular automata topology and construct generic mathematical object based solely on this metric. As a result, the algorithms derived from the properties of those objects, generalize over arbitrary regular grid. We implemented the usual ones, including hexagonal, 4 neighbors, and 8 neighbors square grid.

All the solutions are based on the same basic component: the distance field, which associates to each site of the space its distance to the nearest particle. While the distance values are not bounded, it is shown that the difference between the values of neighboring sites is bounded, enabling encoding of the gradient into a finite state field. Our algorithms are expressed in terms of movements according to such gradient, and also detecting patterns in the gradient, and can thus be encoded in finite state of automata, using only a dozen of state.

Contents

List of Figures

Chapter 1

Introduction

Contents

Shortly stated, this document is about *programming cellular automata* from the perspective of *spatial computing*. This introduction is mainly about spatial computing. It also summarizes the content of the document and contrast it with other works and approaches in the domains of spatial computing and cellular automata.

Section 1.1 explains what spatial computing is and provides with its practical and fundamental rationales. It shows that the programming of *efficient massively distributed systems* necessarily involves *geometric considerations*. This comes from the fact that the processing elements of the distributed system have a spatial arrangement in the physical world and the fact that the communication time between two processing elements is physically related to the distance between them. This latter property, called *locality*, relates communication time and distance in a way that in turn relates efficiency with geometry. This consideration affects the programming of these systems and specific computing models are thus required, typical examples being cellular automata and amorphous computers and they will be presented.

Section 1.2 gives some background on spatial computer programming. In addition to the geometric considerations, this programming is also affected by

the particular nature of this geometry. Indeed, classical geometry and physics are expressed continuously, in the mathematical sense, while the set of processing elements positions forms a discrete set of points. More precisely, classical geometry is Euclidean and some works consider this discrete set of points as an approximation of this Euclidean original physical space. In contrast, I advocate that forgetting this Euclidean geometry to concentrate solely on the new geometrical space made of this discrete set of processing elements and its intrinsic properties allows more precise results to be derived. This shall be achieved by placing the properties of the intrinsic distance of this new geometrical space at the center of the framework, the rationale of this thesis being the following:

As geometry is deeply based on the notion of distance, basing spatial algorithms on the intrinsic distance corresponding to the considered spatial system allows them to be expressed in a more generic and reusable way.

This thesis is exemplified in the remaining chapters in roughly three stages and Section 1.3 describes their contents and their organizations. Still, let us have a quick look at them here before diving into the subject of Spatial Computing.

At the first stage, a spatial computing framework is introduced with a highlight on the distance information at the local level. This framework is based on the cellular automata framework, one of the simplest and oldest model of fine-grained massively distributed systems. Its simple design allows both to focus on the locality property and to ease the derivation of formal results. At the second stage, a building block algorithm is described. It is called *distance field* and it provides distance information at a global level. Its computation requires a small and constant number of states that does not depend on the size of space.

At the third and final stage, three challenging problems are solved by translating them entirely in terms of distances. All of them consider a set of particles, corresponding to agents in a space or tasks in a distributed systems. We either want these particles to move in order to match some geometrical criterion, as it is the case for density uniformization problem, or want to build some geometrical structure, the convex hull and a special kind of communication graph in our case. Each problem is then iteratively transformed into an algorithm that uses distance fields and detection of distance patterns. The resulting algorithms only requires a small and constant number of states, thanks to the distance fields, and works directly for many different cellular space (square, hexagonal, tridimensional and so on), thanks to the distance-only approach.

1.1 Spatial Computing Framework

Spatial computing is primarily about parallel processing. As already said, it focuses on systems being massively distributed, fine-grained and having the locality property. Rather than an entirely new model, it consists of a class of well-established and cutting-edge models, from cellular automata and its generalization to amorphous computers. These models are used for a large number of applications from physics simulation, wireless communication, sensor networks to high performance computing for instance. Spatial computing was the subject of many workshops, the first one in 2006 at Dagstuhl [2]. An annual workshop

also takes place in the IEEE Conference on Self-Adaptative and Self-Organizing Systems from 2008.

In this section, we begin by giving the rationale of spatial computers from a practical point of view, considering spatial computers as a generalization of many massively distributed architectures. Then we provide with a fundamental rationale by considering spatial computers as universal implementation of classical physics features. Both of these considerations lead to the same definition and explain the large variety of works in the domain.

1.1.1 From Massively Distributed Architectures ...

Parallelism is nowadays present in all levels of computer science. Computer architectures incorporate parallelism at the instruction level, with pipelines for example, and by incorporating many cores in a single processor. Also, it is often the case that computers have many processors and that many computers are used together to accomplish a task. If one looks at what is inside a processor, many electronic components corresponding to billions of transistors are combined together and work in parallel.

Beside computers, other kinds of computing platforms exist. Some of them are based on electronics, such as Field-Programmable Gate Arrays architectures, better known as FPGA, and sensor networks. Others consider unconventional computing medium such as DNA computing [48] and other biological systems based on bacteria for instance; or nanotechnology with nano-tubes [40, 22]; or chemistry where the reactions between molecules are used in a computational way [7].

Even if the natures of the processing elements of these different systems are different (electronic, biological, chemical, etc), these systems share many properties. First, they have a huge number of fine-grain processing elements working in parallel. They are programmed at the level of their processing elements but each processing element alone is useless, only the combination of all of them is meaningful. Another fundamental common property of these systems is that their processing elements can only communicate with their neighboring processing elements. Non-neighbor elements need many steps to exchange information, the communication time between two elements being therefore proportional to their distance in a corresponding communication graph. This is called the *locality property* or *locality constraints*.

The only platform that seems not to share these properties is computers. Of course, when many of them is considered in grids of processors and grids of computers, these properties are respected. These properties are also present at the transistor level, whose appropriate complexity model is the classical VLSI model [30] which gives a central position to the locality property. But, single processors are traditionally considered using the so-called Von Neumann architecture model. However, the situation has been changing in the last years as more performance was required, and their model evolves continuously and tends to incorporate these properties more and more.

Indeed, the Von Neumann architecture considers a sequence of instructions executed one at a time and modifying a unique global memory, with constant speed of access to the memory content. This can clearly not scale to arbitrary high memory size without a big performance penalty or violating the locality property. However, this model has been continuously modified to optimize performances and current processors execute many instructions in parallel and have many levels of caches. The interesting thing is that most of these performance-guided modifications have been possible because of the *locality principle*, also known as *locality of reference* and including spatial, temporal, and other kind of locality [16]. Also, processors have more and more cores of smaller size, which shows that the original low level distribution and locality of the transistors tend to be restored.

This gradual incorporation of the locality property is not surprising, as it is a physical constraints, and computers are only physical devices. Also, any parallelism is necessarily inherited from the parallelism of the physical world. Let us go further in this fundamental argument.

1.1.2 From Classical Physics and Universality ...

A parallel can be drawn between the Turing machines and the universal Turing machines on one side, and the physical world and spatial computers on the other side. In fact, we can rationalize the definition of spatial computers as a pseudo-universal physical world. This parallel is particularly enlightening to understand the core motivation of all spatial computing models. Note that the term "universal" often covers different related meaning. In this section, it is used with the following sense: an instance of a model is universal if it can simulate any other instance of the model while conserving its complexity. That is to say that we do not only mean universality in terms of computability, as it is often the case, but also in terms of complexity.

To make the parallel more precise, let us make a point by point comparison. On the side of Turing machines and universal Turing machines, we have the following:

- The model of Turing machines describes how a given machine, described by a finite state automaton, evolves from an initial state, i.e. an initial automaton state and an initial tape.
- The model of Turing machines is powerful because any computable function can be computed by a Turing machines. In general, a given Turing machine only implements one computable function.
- There are Turing machines that can simulate any other Turing machine while conserving its same complexity in space and time. They are called universal Turing machines.

On the side of physical world and spatial computers, we have the followings:

- The physical world can be thought as a set of rules that describes how a given mechanism, described in physical terms, evolves from an initial state.

- The physical world is the most powerful framework simply because anything is necessarily implemented in the physical world. In general, one consider one mechanism, or machine, for a particular purpose.

- There are machines that can simulate physics, and therefore any other mechanism or machine while conserving its complexity in space and time. They are called spatial computers.

To put it concisely, in the same way as a universal Turing machine is an implementation in the Turing machine framework of the Turing machine framework, a spatial computer is merely an implementation in the physical world of the physical world. Of course, this is only a hypothetical statement, since one should prove that all physical phenomenon can be simulated on a spatial computer. However, spatial computer seems to implement enough of the physical features to simulate many physical phenomenon and also any possible computing models that fits with the classical physics constraints.

In fact, spatial computers are currently the best platform for fine-grained physical simulation, and many physical phenomenon have been successfully reproduced, as exemplified in the next paragraph. Indeed, as just said, spatial computers really looks like universal physical world in some important aspect. Said differently, spatial computers are "programmable spaces". Indeed, physics is such that computations happen everywhere and at every time in the same manner in the physical space. Comparatively, spatial computers consider virtually as many processing elements that there are points in the space. Also, each processing elements only has a very small amount of memory, so that the memory capacity and the processing power are both uniformly distributed in the space. This respect the fact that a single point is meaningless in physics, but only a volume of space needs to be considered. Finally, communications between processing elements are local, to respect the locality of the physical world. In terms of communication links, these links needs all to be shorter than a given bounding length.

While only these few features of physics are implemented in spatial computers, a wide range of works using cellular automata, lattice gas, amorphous computer and other spatial computers shows that physics phenomena can by simulated [13, 26, 42]. Just to cite one famous example, gases thermodynamic physics has been perfectly reproduced with an hexagonal cellular automata called FHP [18, 25]. On an amorphous computer, wave propagation has also been precisely reproduced [42]. A simulation of electronic circuit on cellular automata is WireWorld, created by Brian Silverman and popularize in [17]. All the electronic components and the logical gates required to build a computer can be simulated, illustrating that a spatial computer can implement any electronic computer.

1.1.3 . . . to Spatial Computers

It is now clear that the spatial computing framework arises from a kind of convergence between practical needs and fundamental properties. Since spatial computers have been defined in two different ways in the previous sections, a

summary definition might clarify what spatial computers are: a spatial computer is made of an infinite set of processing elements distributed uniformly in a space, each processing elements having a small bounded memory and being able to communicate only with the neighboring processing elements.

As already said, there are many examples of spatial computers and we have already cited a few. The most classical ones are cellular automata, where the processing elements are organized as a crystalline spatial structure, with synchronicity of processings and communications. Another example is amorphous computers [3, 15], where the processing elements are randomly and uniformly distributed in the space, and no synchronicity is assumed. As shown in the previous sections, both of these frameworks are useful for modeling and simulating massively distributed systems, such as grids of processors or sensor networks, and simulating physical phenomena.

1.2 Programming Spatial Computers

We described the spatial computing framework and why we consider their programming. We now describe the programming itself. Programming spatial computers is very different from common programming, even distributed. As already said, a processing element alone cannot do anything, so that any computation has to happen "in space", forgetting thus pure "in time" computations, dictated by the one-at-a-time approach of the Von Neumann architecture. In some aspect, programming a spatial computer is like writing "physical laws" even if the goal is not to obtain a physical simulation. The goal is usually expressed in terms of a global behavior while the systems is programmed by local rules executed by the individual processing elements. This global/local mapping is generally difficult to obtain, since spatial computers belongs to the class of complex systems.

There are many results in the sometime long history of models corresponding to spatial computers. Platforms have been developed to write the local rules in an appropriate manner and simulates their execution. To name a recent platform, the MGS [21, 20] project allows to apply transformation corresponding to local rewriting rules to any abstract space, hence allowing to simulate many spatial computers. Thus, models based on chemical reactions, tackled by the specific platform Gamma [11], or based on encapsulated membranes, such as the P-systems [41] and BioAmbients [43], can be considered in the same platform. MGS transformation can also modify the space, which eases the expression of embryogenic and morphogenic phenomena models. Another project called Proto proposes an abstraction of the discreteness of the system space and consider operators to account of local communication, the processing process, and the composition of information as if the space was continuous in space and time. Many other platforms exist, but we use none of them in this document. Instead, we simply describe the framework with all the details and provide the algorithms using a mathematical rather than a programming formulation.

On the algorithmic part, many results have been obtained involving geometrical structures that can be used for their own sake or to achieve a more general goal. In cellular automata, examples are the computation of the Voronoï diagram [4] of a set of points or the computation their convex hull. The Voronoï diagram partitions the space and also implicitly establishes a network by using the Delaunay triangulation and the convex hull wraps the set points and materializes the privileged region of communication between the points. However, the provided solutions are only valid for a particular grid. The solutions presented in this document work for many grids, even tridimensional.

On amorphous computers [3], algorithms have been design that compute a set of coordinates for each processing element, manipulates particles that emit gradients and move along them [15, 38], and these algorithms are made composable by specific languages such as the Growing Point Language [15] or Origami Language [38]. However, most of these results produces static patterns and circuits. Also, the memory required by these algorithms is often dependent on the size of the space and the number of considered particles. While we concentrate on regular spaces, the techniques we will present also rely a lot on the notion of gradient, but in a dynamic and finite-state way. Moreover these techniques are simple and can be used on amorphous computers.

On the composition part, this document presents buildings bolcks that are combined together to solve bigger problems. Proto platform and many works on amorphous computers consider the processing element discrete space as an approximation of the original physical space. The results of their algorithms are therefore compared with results on continuous space and errors are calculated [10]. This is a requirement for problems where the original space in the center of interest, such as application where some agent needs to be guided in the real environment. However, applications in computer architecture or wireless communication may not consider the original space at all but only need to optimize some communication time for example. For these applications, considering the building blocks having errors is a problem as the error is likely to grow as many blocks are combined together. We choose another approach that consider the discrete programmable space directly, as if the original space does not exist by any other mean than by the properties conserved by the discrete programmable space. The choice of approach depends on which space is the main focus of the algorithm.

Note that the results presented in this document were almost all initially designed by thinking in terms of Euclidean space, and by managing the errors and problems due to the discretization afterward. But as more precision were necessary to improve the predictability of the system, or simply to obtain generic mathematical results, the Euclidean-based approach has been replaced by the generic cellular metric space approach. Coming back to the basics, every geometrical properties are consequences of the properties of the considered space metric. The chosen approach is, thus, to consider directly the metric of cellular spaces in order to obtain results that naturally match the structure of the space, improving also composability and modularity.

1.3 Points, Distances and Cellular Automata

Early, and sometimes very early versions of chapters 3, 4, 5 and 6 has already been published as four different articles [33, 34, 36, 35]. Apart from results improvements, one big difference between the chapters and their associated articles is that they are now expressed more details with the unifying framework described in chapter 2 and refined in chapter 3. Let us describe more precisely the content of each chapter.

In Chapter 2, we describe the cellular automata framework, justifying all of its aspects, but insisting on the fact that no direction information needs to be consider in our framework. Instead of the identification of north, south, east and west for example, we rather consider that the only information available to each cell, or site, of the space is the local graph linking its neighbors. This information correspond to whether two neighbors are connected or not, and is presented as a local distance function giving the length of the shortest path link the two neighbors. Using this approach, we do not restrict ourself to a specific cellular space but consider all common regular cellular spaces and their counterparts of higher dimensionality.

In Chapter 3, we build what we call the distance field. It is a cellular rule which, for a given set of moving particles, associates to each site of the space its distance to the nearest observable particle. It is constructed by composing the local distance functions together, thus provides a distance information at a more global level. Not that even if there are many particles, only one distance value is associated to each site. But distances values can be arbitrary high in general. We therefore show that when a field is Lipschitz continously, and the absolute of the values that compose it are not used, it is possible to construct an equivalent field using a finite and constant number of state. For the case of the distance field, this condition is fulfilled when the particles move at a bounded speed less than 1, i.e. when the particles movements are themselve Lipschitz continous. This is the key building block that allows us to solve all the problems in the following chapters by a finite-state cellular automata that completely respect all of the constraints of the cellular automata framework. The end of Chapter 3 presents a basic use of the distance field, namely movements of a boolean field according to a distance field. The following chapters are build on top of Chapter 2 and 3.

In Chapter 4, we solve a first problem using the building blocks defined previously. We consider any unidimensional cellular spaces of bounded size on which a boolean configuration indicate the presence of particle in some site of the space. From the distributed algorithmics point of view, what we want to achieve is a load-balancing algorithm for these task-particles. From a physical point of view, this corresponds to the design of a repulsion mechanism between the particles. From the geometric point of view, this may be call the density uniformisation problem. In the chapter, the problem is formally defined and then decomposed using two distance fields generated by two different kind of particles. The particles of each kind use the distance field of the particles of the other kind to run away from them. The final solution is therefore a composition of the distance field and particle movement building blocks and is expressed

a system of equation. While the system description is simple, the dynamic of the system itself exhibits physical-like characteristic: we can observe that the updates act like quantity of movement signals travelling through the space and causing particles to move. The cellular automata converges as its global energy decrease, either when the signals leave the spaces on the border, or when they meet signals of opposite sign.

In Chapter 5, infinite multidimensional cellular spaces are considered with the problem of the construction of the convex hull of a set of particles. In terms of distributed algorithmics, the goal is to determine which sites belong to the privileged area of communication the set of task-particle. More clearly, given a set of particle we want to detect the sites of the space that belong to the convex hull of the particle. Because of the difference because Euclidean spaces and cellular spaces, the result is not the classical Euclidean convex hull. Previous works have described this difference by identifying a set of allowed angles in the construction of the convex hull. However the associated solutions are also very limited: they are only valid for an already connected set of particles, and consider only the 8-neighbor cellular space in most cases. Our approach has been rather to characterize the diffenrencies between Euclidean and cellular convex hulls by a simple change of metric, leads to an equivalent but intrinsic formulation of the cellular convex hull. The associated algorithm handle arbitrary distant set of particle for any cellular space of arbitrary dimesionnality (2D or 3D for example). Again, the solution is entirely expressed in terms of distance and by a system of equations using one distance field, a rule detecting pattern of the distance field, and movement from the detected pattern to the particles. In fact, this solution underlies the construction of a graph, and its precise characterization is required to complete the definition of the system of arbitrary spaces.

Chapter 6 therefore focuses on this proximity graph is terms of geometry, or energy-minimizing communication graph in terms of distributed algorithmic. An analysis shows that this graphs is analoguous to the Gabriel graph, a subgraph of the Delaunay graph defined on for Euclidean spaces. However, the exact definition given for the Euclidean spaces does not conserve the good properties of the structure when applied to cellular spaces. So a mathematical work is done to generalize Gabriel graphs to arbitrary metric spaces in a way that conserves its properties. Of course, when applied to Euclidean spaces, this generalization gives back the original Gabriel graph definition. When applied to cellular space metric, a derivation is possible down to a system of equation describing a finite-state cellular automata rule building the metric Gabriel graph for arbitrary set of particle and arbitrary dimensionnality of the space. Here again, the system of equations using one distance field, a rule detecting pattern of the distance field, and movement from the detected pattern to the particles, but in a more general way.

In Chapter 7, we conclude by giving the list of things that remains to do on each of these algorihtms, with some sketch on how they can be handled. We also give a look at other problems that can be expressed in this framework, problems of geometric or non-geometric formulation. We finally provides with directions that we think needs to be taken to extend this work and framework.

Chapter 2

Space, Time, and Cellular Automata

Contents

This chapter is a presentation of the cellular automata framework, but not only. The approach chosen is to start from the basics and separate each consideration. In this way, the results are clearer and their generality and modularity are more easily visible. As the concepts are presented, the useful vocabulary related to them is also given. Now, let us introduce the most basic concepts manipulated in our framework: time, space, and finiteness.

2.1 Determinism, and Dynamical Systems

The work presented here mostly consider purely *deterministic* systems. Determinism is the concept that the same causes lead to the same consequences, and that the future is entirely determined by the present. In mathematics, such systems involving time and determinism in this way are called *dynamical systems*. Formally:

Definition 2.1.1. *A dynamical system is a tuple* (T, M, Φ) *where* T *represents the possible intervals of time,* M *is the set of all possible system states, and* Φ *is the evolution function* $T \times M \to M$*. This evolution function is such that* $\Phi_0(m) = m$ *and* $\Phi_{t_1+t_2}(m) = \Phi_{t_2}(\Phi_{t_1}(m))$*.*

In this definition, the *evolution function* Φ tells for each possible initial *system state* m, what is the system state $\Phi_t(m)$ reached by the system after an interval t of time. The determinism is expressed in the properties of the evolution function: whatever system state $m_1 = \Phi_{t_1}(m_0)$ is reached after an interval t_1 of time, the system state $\Phi_{t_1+t_2}(m_0)$ reached t_2 latter only depends on m_1, and is obtained by applying the same evolution function Φ for the same amount of time t_2.

However, the structure of the time itself is not enforced by the definition. While the time can be circular, or dependent on the initial system state – which is not taken into account in this definition – most dynamical systems consider either $T = \mathbb{R}^+$ or $T = \mathbb{N}$. These choices correspond respectively to *continuous-time* and *discrete-time* dynamical systems. Physics usually consider the former one, and computer science the latter which is simpler. Indeed, any evolution function $\Phi_t \colon M \to M$ for $t \in \mathbb{N}$ of a discrete-time dynamical system can be obtain by a composition of Φ_1, i.e. $\Phi_t = (\Phi_1)^t$. The function Φ_1 is therefore given the special name of *transition function*:

Definition 2.1.2. *The* transition function *of a discrete-time dynamical system* $(T = \mathbb{N}, M, \Phi)$ *is the function* $\Phi_1 \colon M \to M$ *that associates to every system state* $m \in M$ *a next system state* $\Phi_1(m)$.

2.2 Space, Time, and Locality

Many dynamical systems describe evolutions that actually occur in a *space*. In this case, the system states are decomposed into a spatially extended object, the space being a projection of how the parts of the system states interact with each other. Therefore, the notion of space is tightly entangled with the notion of *locality*: the space is endowed with a *metric* representing the time taken by each pair of parts of a system states to interact with each other.

More concretely, the space is a metric space (S, d), where S is the set of points, and $d \colon S^2 \to \mathbb{R}$ the metric or distance function. For a dynamical system (T, M, Φ) considering this space, a system state m associates to each point x a state or *value* $m(x)$ taken from a set V. Therefore we have $M = V^S$. System states are usually called *configurations*, the word *state* being used for the elements of V, if not otherwise stated.

By locality, the result at any point x of an evolution of t time units can only take into account values of m in a bounded region $B(x, rt)$, whose radius is proportional to the evolution time. But only those values need to be considered, and not the position of x, because we want the same evolution laws to be applied everywhere.

This implies that the space looks the same from all of its points. In particular, for any rt, any point x has the same local space S_{rt}, i.e. $B(x, rt) \equiv S_{rt}$. Therefore, the evolution function Φ applies the same local evolution functions $\varphi_t \colon V^{S_{rt}} \times S \to V$ at every point of the space:

$$\Phi_t(m)(x) = \varphi_t(m \restriction B(x, rt)). \tag{2.1}$$

where $f\restriction A$ denotes the restriction of the function f to A.

As previously, we specially consider $t = 1$ for discrete-time dynamical systems, and introduce vocabulary for this case. The factor r is called the *neighborhood radius*, and $B(x, r)$ is called the *neighborhood* of x. A *local configuration* correspond the values of the configuration in the neighborhood of a point. Finally the local evolution function for $t = 1$ is called *local transition function*:

Definition 2.2.1. *The* local transition function *of a spatial discrete-time dynamical system* $(T = \mathbb{N}, M = V^S, \Phi)$ *is the function* $\phi : V^{S_r} \to V$ *such that* $\phi = \varphi_1$*, i.e.* $\Phi_1(m)(x) = \phi(m\restriction B(x, r))$.

2.3 Finiteness, and Cellular Automata

Cellular Automata [45, 13] are discrete-time spatial dynamical systems where the local transition functions are in fact finite state automata. This finiteness requirement allows, in particular, exact simulations of the system by computers, or even to consider it as a spatially extended computer architecture. In term of spatial dynamical system, this finiteness implies that both inputs set and outputs set of the the local transition function $\phi : V^{S_r} \to V$ are finite. Therefore, the state space V is finite, and so is V^{S_r}, causing the space S to be discrete.

Because of its discreteness and uniformity, the space can be considered as a vertex-transitive metric graph. The vertex of this graph, usually called *cells* or *sites*, are the points of the space, called *cellular space*, and the metric is the usual graph metric. This latter assigns unitary length to edges, the distance between two sites being the length of the shortest path joining them in the graph. Let us note that the word *neighborhood*, defined earlier, can also be used to designate the neighbors of a site x in the graph, i.e. $N(x) = B(x, 1)\backslash\{x\}$. But the meaning will be clear from the context.

On top of being vertex-transitive graphs, cellular spaces are usually considered as being lattices. The most commonly used bi-dimensional cellular spaces are the square grids, with 4 or 8 neighbors by sites, and the hexagonal grids, having 6 neighbors by site. Moreover, local transition functions commonly have directional information in addition to the distances. Indeed, the classical framework considers the neighboring sites of any site to be labeled as North or East for example.

In our framework, we also consider the three classical grids, but as a sample to obtain general algorithms that should be therefore easier to apply to other spaces. Also, we consider no directional information and only manipulate distances, thus restricting ourselves to rotation-invariant cellular automata. Depending on the context, an emphasis will be put on the sites or the edges. We therefore use the different representations shown in Fig. 2.1.

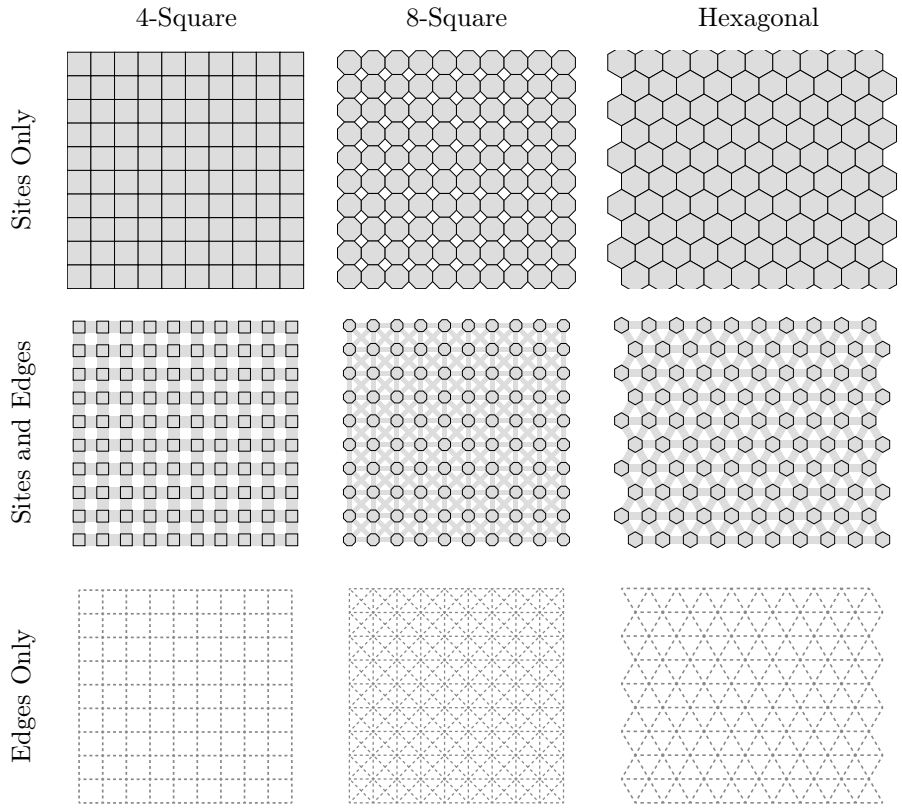


Figure 2.1: Grids used in this article: the polygons (squares, octagons and hexagons) correspond to the sites, and the lines to the edges.

2.4 System [De]Composition and Modularity

The definition of cellular automata, and dynamical systems in general, actually describe fully deterministic, and therefore, fully determined and closed systems. This can be tackled directly in many situations, especially in many cellular automata work where only binary states $V = \mathbf{2}$ are considered. However, systems involving complex states are usually more easily expressed as a composition of many sub-systems.

At the dynamical system level, while the whole system evolution is entirely determined by the initial system state, this is not the case for the sub-systems which are in fact open systems, since they continuously interact with each other. Formally, the evolution function $\Phi\colon T \times M \to M$ of a dynamical system can be viewed, in a curryfied form, as a function $\Phi\colon M \to (T \to M)$ that associates an evolution to any initial system state. However, the evolution of an open system may be determined by something else than its initial system state. One can even consider "reactive" systems that do not have any internal system state. Therefore, we will consider that open systems belongs to the class $X \to (T \to M)$, for an arbitrary X. These systems, either closed or open, can be considered as parametrized *evolutions*.

Definition 2.4.1. *An evolution $e\colon T \to M$ associates a system state $e_t \in M$ to all instant of time $t \in T$.*

One thing should be mention at this point. As a system is decomposed into many sub-systems, its system states M are also decomposed into many sub-states $M_1, \ldots, M_n$. However, it does not mean that $M = M_1 \times \ldots \times M_n$, since some combination of sub-state may be impossible to reach. The same thing is true for systems constructed from many sub-systems, while restricting the possible initial system state. This remark takes its full importance when properties like finiteness have to be verified and when counting the number of states of the system.

For the particular case of spatial dynamical systems, all these considerations apply with the additional concepts of spaces and locality. In this case, the evolutions associate to each instant of time a configuration $m \in V^S$. In this case, we call it *field*.

Definition 2.4.2. *A field $f\colon T \times S \to V$ associates a state $f_t(x) \in V$ to all points x of the space S, at all instant $t \in T$ of time.*

A spatial dynamical system can therefore be defined as a parametrized field respecting locality and determinism, the parameter being the initial configuration of course. But since other kind of parameters can be considered, we introduce the notation $F[p_1, \ldots, p_n]_t(x)$ that denotes the value associated at site x and time t by parametrized field F when the parameters are set of p_1, ..., and p_n. Using this notation, a closed system can, for example, be described as a composition of system of the form, where $p_{ab} \in \{\mathrm{f}_1, \ldots, \mathrm{f}_k\} \cup \{c_1, \ldots, c_{n_0}\}$, c_i

is a constant or a parameter of the system:

$$\left\{\begin{array}{lcl} \mathrm{f}_1 & = & F_1[p_{11},\ldots,p_{1n_1}] \\ \mathrm{f}_2 & = & F_2[p_{21},\ldots,p_{2n_2}] \\ \vdots & \vdots & \vdots \\ \mathrm{f}_k & = & F_k[p_{k1},\ldots,p_{kn_k}] \end{array}\right.$$

Instances of such systems can be found in pages 48, 73, and 86. They combine distance fields $D[\bullet]$, given in Eq. (3.8) (page 33), and the moving-particles boolean field $M'[\bullet]$, given in Eq. (3.12) (page 42), with other specific fields.

As said for dynamical system, the set of configuration M can be smaller than the product of sub-system set of configuration $M_1 \times \ldots \times M_k$. Thus, the possible local configuration V^{S_r} can also be smaller than $V_1^{S_r} \times \ldots \times V_k^{S_r}$. Consequently, so can be the set of states V regarding $V_1 \times \ldots \times V_k$. These facts are illustrated in the next chapter on the basic components used by cellular automata considered in all the other chapters. As it is explained, we can even compose infinite sets of states and obtain, for example, a set of 10 states.

Chapter 3

Distance Fields and Gradients

Contents

Parts of the content of this chapter has been published [34] (mainly Sect 3.3), and in [33] (early version of Eq (3.8) and the associated comments and figures).

Starting with a cellular space, thought as a metric space, there are two basic components that naturally comes into considerations: Boolean fields and distance fields. We will consider both of them in this chapter. Boolean fields will be treated mostly as dynamic sets of particles. From these particles, a distance field can be computed, associating to each site of the space an integer value corresponding to its distance to the nearest particle. This field therefore allows some other set of particles to move accordingly to the first set of particles, and so on.

The second important aspect described in this chapter is that, while using distance fields of arbitrary high values, we identify conditions that ensure that

the final resulting system can be reduced to a cellular automata, i.e. that the set of states is finite. This is done by considering the gradient of distance fields as the useful information.

3.1 Classical Definition and Computation

Let us begin by the basics, and derive all the results from them. Putting aside locality and dynamicity, the distance field is reduced to the simple concept of distance map of a set of points.

Definition 3.1.1. *Given a set of points P, the distance map D_P associates to each point x of the space its distance $d(x, y)$ to the closest point y of the set P.*

$$D_P(x) = d(P, x) = \min\{\, d(x, y) \mid y \in P \,\}. \tag{3.1}$$

Because this structure computes values that depend on arbitrary distant data, the locality principle prevents it to be computed instantaneously. However, the following well-know recursive relation [49, 32, 33] can easily by used to obtain an evolution that converges to the distance map. The neighborhood radius is denoted as r.

$$D_P(x) = \begin{cases} 0 \text{ if } x \in P \text{ else:} \\ \min\{\, d(x, y) + D_P(y) \mid y \in B(x, r) \backslash \{x\} \,\}. \end{cases} \tag{3.2}$$

In fact, this relation is only true for spaces where the distance between two points also corresponds to the length of shortest paths linking them, namely length metric spaces, also called path metric spaces. Anyway, we only consider such spaces. This evolution is obtained by unrolling the recursion on the time axis, thus recovering the locality and the dynamicity:

$$D[P]_{t+1}(x) = \begin{cases} 0 \text{ if } x \in P_t \text{ else:} \\ \min\{\, d(x, y) + D[P]_t(y) \mid y \in B(x, r) \backslash \{x\} \,\}. \end{cases} \tag{3.3}$$

But for the moment, we consider the set of points or particles P to be static. If we assume that all particles appears at time $t = 0$, and that they are static, i.e. $P_t = P_0$ for any $t > 0$, a coherent initial configuration for the distance field is:

$$D[P]_0(x) = \begin{cases} 0 \text{ if } x \in P_0 \\ \infty \text{ otherwise.} \end{cases} \tag{3.4}$$

Figure 3.1 shows the evolution of the resulting system with neighborhood radius $r = 1$ and a single particle. The sites with infinite value are represented by empty sites. It can be observed that each site changes its infinite value to the correct one as soon as it notices the presence of the particle. The locality principle expressed in Eq. (2.1) is completely respected, expressed in the particular case of static particles as:

$$D[P]_t(x) = D_{P_0 \cap B(x, r.t)}(x)$$

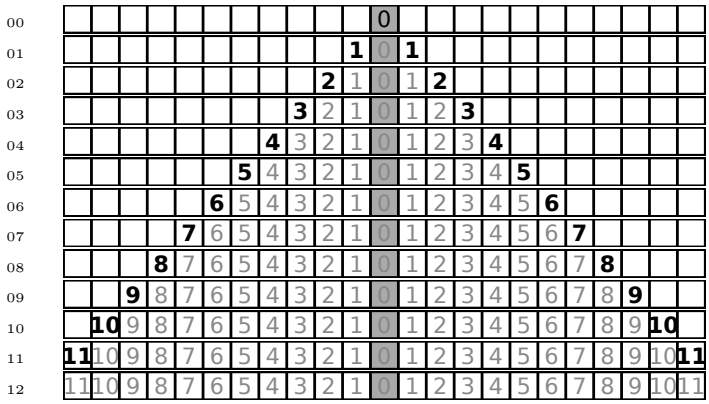


Figure 3.1: Evolution of Eq. (3.3) and Eq. (3.4)

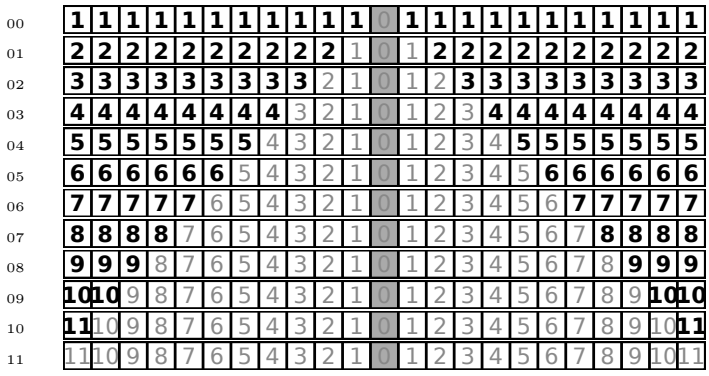
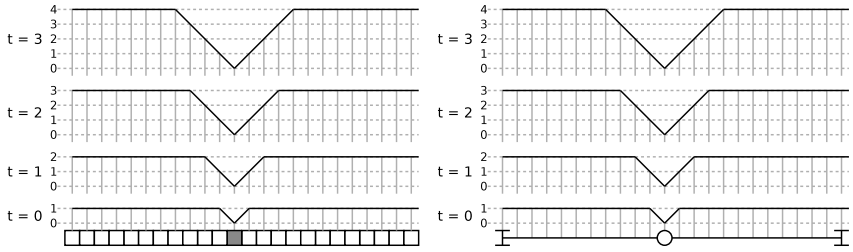
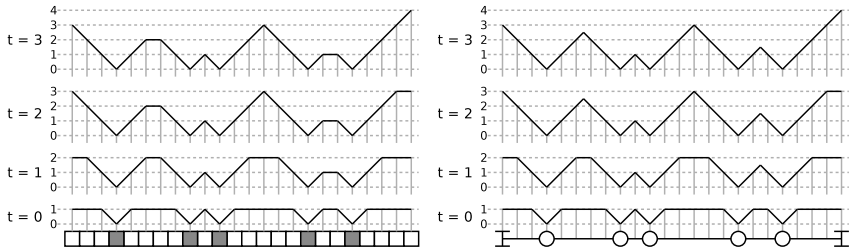


Figure 3.2: Evolution of Eq. (3.3) and Eq. (3.5)



(a) Discrete space (b) Continuous space

Figure 3.3: Evolution of Eq. (3.3) for a single static point



(a) Discrete space (b) Continuous space

Figure 3.4: Evolution of Eq. (3.3) for many static points

Another possibility is to assume that the space was full of particles, and that they all disappeared at time $t = 0$, except the set P. In this case, the initial configuration is:

$$D[P]_0(x) = \begin{cases} 0 \text{ if } x \in P_0 \\ 1 \text{ otherwise.} \end{cases} \tag{3.5}$$

Figure 3.2 shows the resulting evolution for this initial configuration. Here, each site increments its value like a counter until the correct value is reached. This is again coherent with the locality principle. This time, each site updates its value as it notices the absence of particles further and further away. We prefer to delay the corresponding formal expression to the next section.

With no surprise, everything is so simple in this case that putting side by side the considered discrete evolution with a continuous one, as done in Fig. 3.3, only shows that the discrete case corresponds to snapshots of the continuous case at integral position and time. As shows in Fig. 3.4, considering many static particles allows, however, to observe some little differences at some local maxima. Those differences do not appear on the sites though, but on virtual points between them. Considering these points can be useful, but again, let us skip this for the moment.

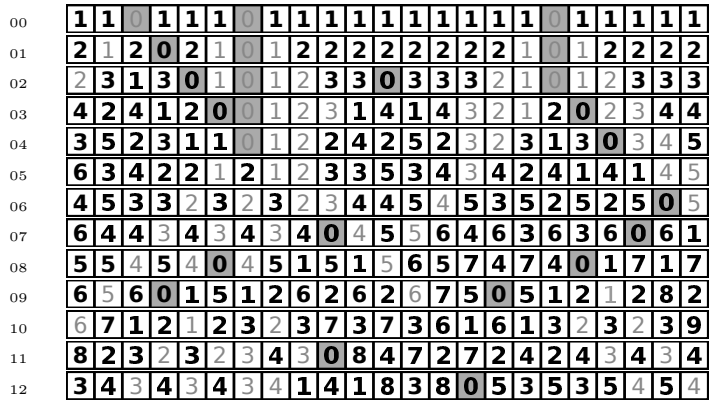


Figure 3.5: Evolution of Eq. (3.3) on an arbitrary dynamic set

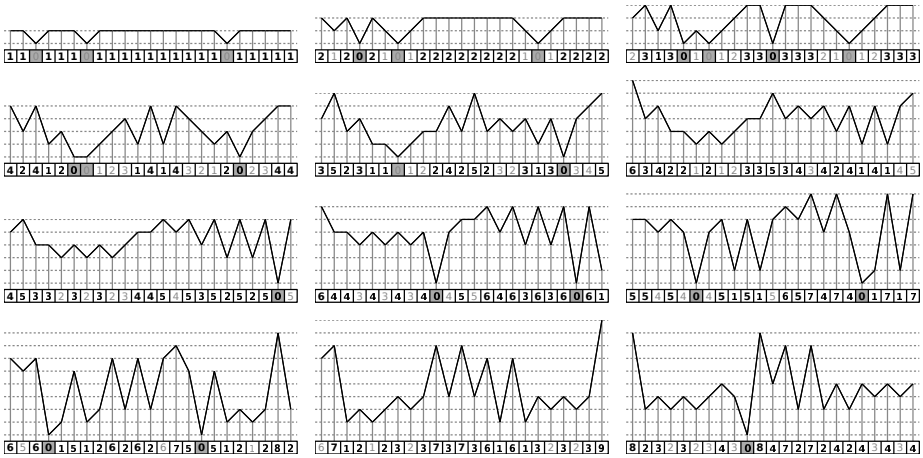


Figure 3.6: Evolution of Eq. (3.3) on an arbitrary dynamic set

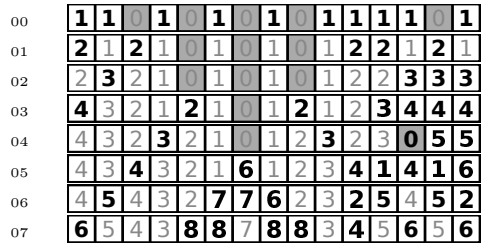


Figure 3.7: Perfect evolution with Eq. (3.7)

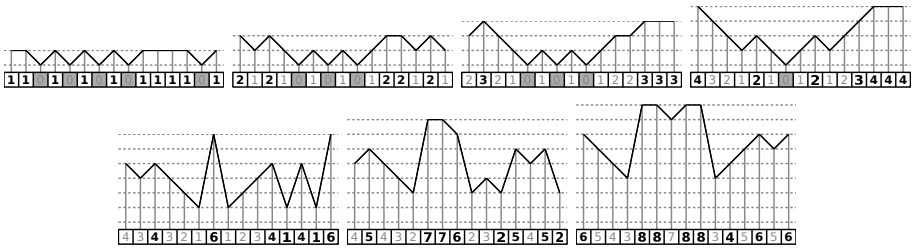


Figure 3.8: Perfect evolution with Eq. (3.7)

3.2 Definition for Continuous Sets of Particles

Let us now consider the case of dynamic sets of particles, i.e. sets of particles that change as time goes. As one may expect, using Eq. (3.3) produces many incoherent values in general, as shown in Fig. 3.5. By incoherent values, we mean values that do not correspond to any distance, even when taking the locality into account. For example, we can see that when the left particle at time 4 disappears, the null value is replaced by the value 2 at time 5. This value is incoherent, since there is no particle at distance 2 at time $3 = 5-2$. In fact, this value should be 6, since we consider that the initial configuration corresponds to Eq. (3.5), which means that all sites have a particle at time -1. So the nearest particles which the site does not know about the deletion are at distance 6, at time $-1 = 5 - 6$.

Indeed, a site x should ideally takes into account the last state it is able to observe for all the sites. For a site y, this last state observed by site x at time t correspond to time $t' = t - d(x, y)$, by direct application of locality. This gives an equivalent of light cone in physics, that we call information cone. Applying this to know which set of points $P_t^*(x)$ is observed by site x at time t, we have:

$$P_t^*(x) = \bigcup_{t' \leq t} \{\, y \in P_t \mid d(x, y) = t - t' \,\}. \tag{3.6}$$

A perfect evolution should then associates to each site x at time t, the value the site would have in the distance map corresponding to the observed set $P_t^*(x)$. Formally:

$$D[P]_t(x) = D_{P_t^*(x)}(x) = \min\{\, d(x, y) \mid y \in P_t^*(x) \,\}. \tag{3.7}$$

To fix ideas, a perfect evolution is shown in Fig 3.7 with dynamic set showing explicitly the differences. Indeed, we see in this evolution that sites values may jump from low to high values in some cases of many-particles deletion as described before. This is the case for each particle deletion occurring at time 5.

However, implementing this perfect evolution function requires an infinite number of states in general. In the one-dimensional case, it would then be possible to memorize two values, one corresponding to the left nearest particle, and the other to the right one, but this can not be used for higher dimensional spaces.

Since it is not possible to have a perfectly coherent distance field for arbitrary dynamic sets, we need to identify which are the dynamic sets allowing a coherent distance field to be computed. We will see that with a minor modification, we can perfectly handle the dynamic sets that respect, in some sense, the locality of the space: continuous dynamic sets.

Indeed, we have seen that particles deletions causes coherence problem. Therefore, we consider dynamic sets such that, if a particle is at site x at time t, then at time $t+1$, there must be a particle in $B(x, 1)$. Also, to prevent the space to be empty of any particle, and thus prevent infinite values, we also require a particle to be in $B(x, 1)$ at time $t - 1$. By transitivity, this requirement can be expressed as:

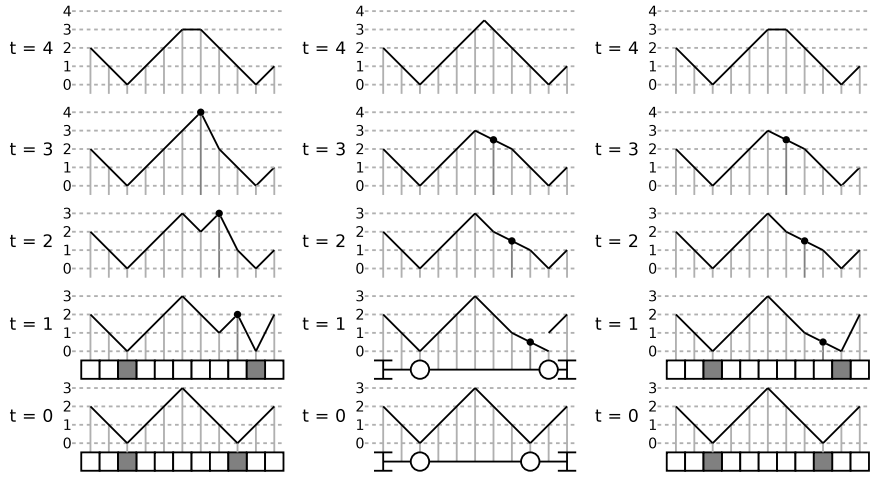
Definition 3.2.1. *A dynamic set P of particle is continuous if and only if it is not empty and for any particle $x \in P_t$, we have at any time t', at least one particle y such that $d(x, y) \leq |t - t'|$.*

This requirement basically means that the dynamic set corresponds to a set of moving particles, where the particles can split or merge, but never appear nor disappear. They also have to move at speed 0 or 1, being thus bound by their information cone. In this way, all the distances should evolve in a smooth and tractable way. Let us look at the evolution of Eq. (3.3) on a simple continuous set.

Fig 3.9(a) and 3.9(b) show such an evolution, for Eq. (3.3) and for the ideal continuous case respectively. As initial conditions, two distinct particles are considered as being static before $t = 0$ so that the distance field has locally converged. Then, from time $t = 0$ to $t = 1$, the right particle moves one step to the right. At time $t = 1$ and $t = 2$, we can see that Eq. (3.3) produces a spurious local maximum not present in the continuous space-time evolution. The reason is that for the discrete version, the particle simply disappears and reappears.

We therefore modify the rule to take into account the continuity of the dynamic set. Indeed, if a particle is not visible anymore, then it has moved, with speed 1, to at least one of the neighboring sites. Thus, we can conclude that this case always corresponds to a distance value of 0.5 as it is the case for the continuous case. This gives the following rule:

$$D[P]_{t+1}(x) = \begin{cases} 0 \text{ if } x \in P_{t+1} \text{ else:} \\ 0.5 \text{ if } x \in P_t \text{ else:} \\ \min\{1 + D[P]_t(y) \mid y \in B(x, 1) \backslash \{x\}\}. \end{cases} \tag{3.8}$$



(a) Discrete space: Eq. (3.3) (b) Continuous space (c) Discrete space: Eq. (3.8)

Figure 3.9: Evolution of distance fields with many moving points

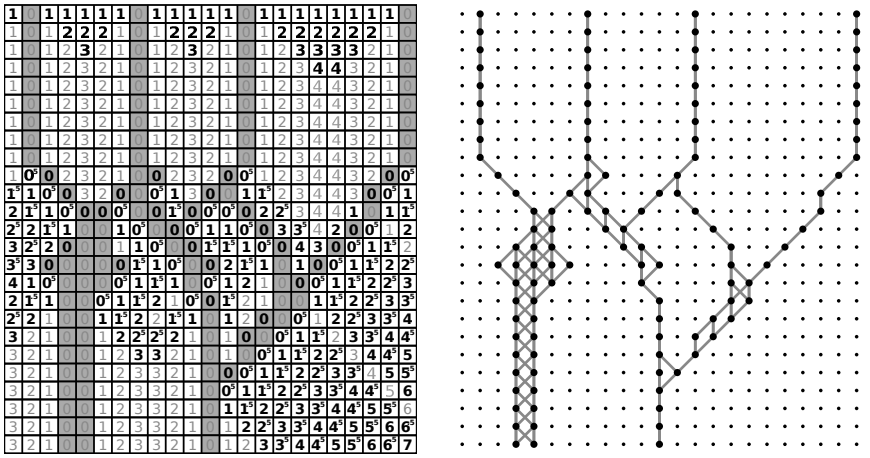
With this correction, we obtain the evolution shown in Fig 3.9(c), which only differs with the continuous ideal evolution in the local maxima as previously. Let us consider a more complex continuous dynamic set, as in Fig 3.10(a). In fact, this dynamic set can be straightforwardly interpolated into continuous space-time trajectories (Fig 3.10(b)). We can therefore check that the values are coherent. Also evolution of Eq. (3.8) corresponds exactly to snapshots of the corresponding continuous space-time evolution at integral position and time. Actually, this equation applied on continuous dynamic sets verifies the locality as expressed in Eq. (3.7).

The last thing to note can be shown on Fig 3.11, corresponding to the same evolution as Fig 3.10(a). We can see that the distance field is more organized compared to the previous evolutions. In fact, between every two particles, the distance fields first strictly increase to a local maximum and then strictly decrease to the next particle. Thus, any site can use the difference between its distance value and the values of its neighboring site to determine the direction to the nearest observed particle(s).

3.3 Finiteness, Gradients, and Continuity

While the distance field rule (3.8) takes locality into account, it does not address finiteness, which prevent it to be directly usable in a pure cellular automaton. In fact, it is not possible to address it in general, since the space is infinite and so arbitrary dynamic set will produce arbitrary high distance values. Thus, we need to restrict the class of considered dynamic sets and/or to represent only a finite part of distance field information in order to have a finite number of possible states.

An easy case is for example to only consider finite cellular space having a



(a) Evolution of Eq. (3.8) (b) Interpolated dynamic set

Figure 3.10: The values of (a) corresponds to the interpolated dynamic set

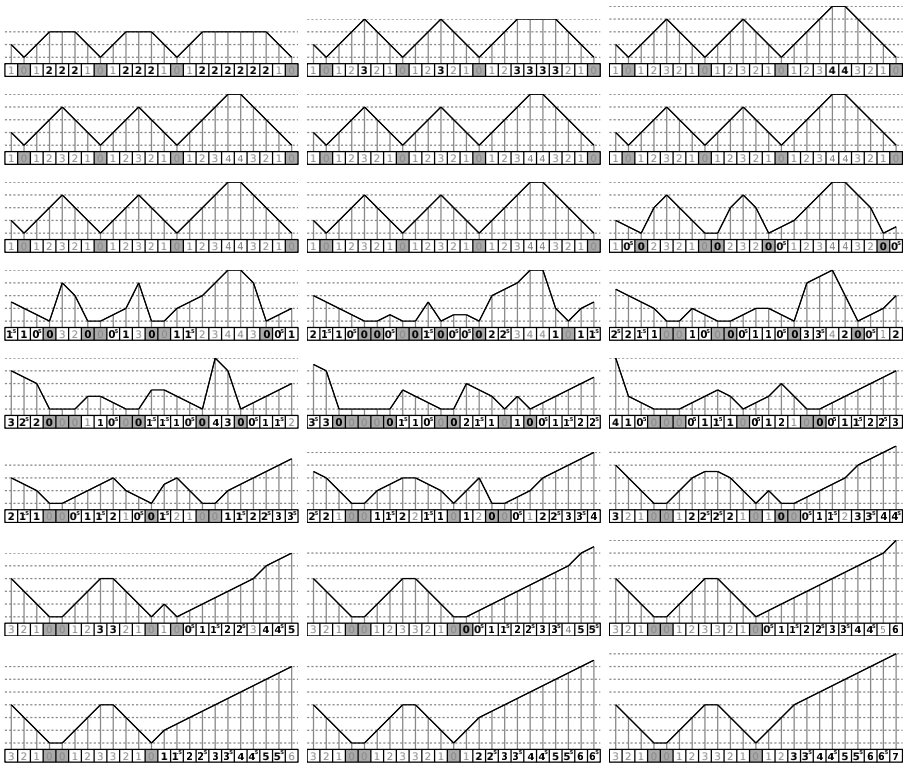


Figure 3.11: Evolution of Eq. (3.8)

known diameter, or dynamic sets that fill the infinite space sufficiently enough to bound the distances. In this case, only a finite number of distances are actually generated by the distance field, that can therefore be entirely represented with a finite number of states.

We tackle here the more interesting case where no bound on the distance field values is assumed, but where the gradient of the distance field is the only part of the information that is used by the cellular automaton.

3.3.1 Distance field gradient

Indeed, in cases where we only need to get some directional information from the distance field, only the *difference* between neighboring sites values is required, not the actual absolute values of the sites. We call the collection of these differences *gradient* of the distance field. Formally, we denote the gradient by $\Delta D[P] : T \times S^2 \to \mathbb{R}$ and define it for discrete spaces as:

$$\Delta D[P]_t(x, y) = D[P]_t(x) - D_t[P](y) \quad \text{for } d(x, y) \leq 1 \tag{3.9}$$

The set of differences can be finite even when the set of absolute values is not. Then, the goal is to identify classes of dynamics sets having a finite gradient, and a distance field finite representation that conserves this gradient. The most important property is that the difference between two neighboring sites must be bounded by some fixed constant K at any time.

$$\forall (t, x, y) \in T \times S^2; \quad |\Delta D[P]_t(x, y)| \leq K \quad \text{for } d(x, y) \leq 1. \tag{3.10}$$

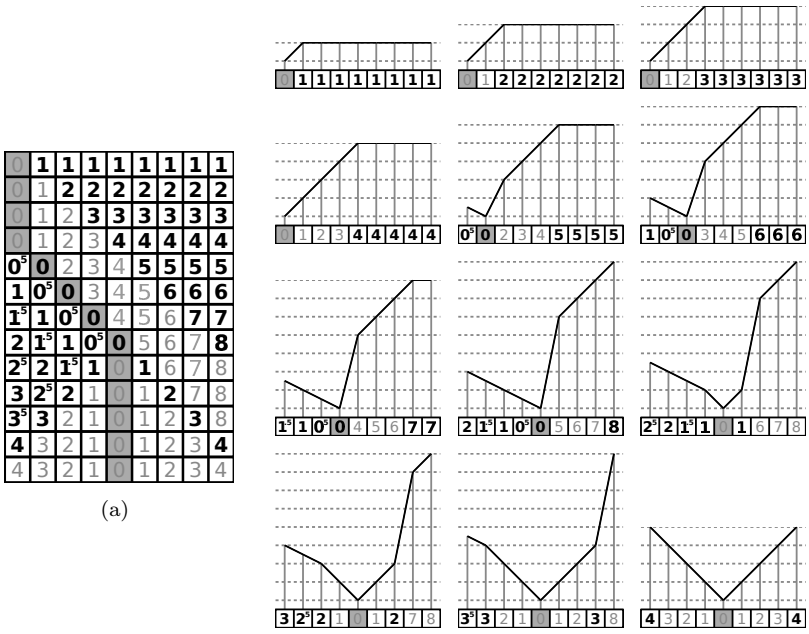
When considering the effects of this condition on arbitrary distant sites x and y, we obtain the condition $|\Delta D[P]_t(x, y)| \leq K.d(x, y)$. This corresponds to the so-called *K-Lipschitz continuity* in mathematics. In the next sub-section, we show that it is possible to have bounded gradient and identify a corresponding class of dynamics sets.

Definition 3.3.1. *A function f from a metric space (A, d_A) to another (B, d_B) is* K-Lipschitz continuous *if and only if $d_B(f(x), f(y)) \leq K.d_A(x, y)$.*

3.3.2 Dynamic sets of bounded gradient

First things to note is that any fixpoint distance field has bounded gradient, with $K = 1$. However, bigger differences actually appear because of particles movements. Indeed, Eq. (3.8) assigns a null distance whenever a particle is present. This assignment is the crucial part since nothing prevent a dynamic set to assign a null distance beside any arbitrary high distance value. This fact is obvious for arbitrary dynamic set, but is also true for continuous ones when a particle move in the same direction for many times (see Fig 3.12).

A way to prove it formally is to prove that the min part of the equation does not create differences larger than the differences already present in the neighborhood of a considered site, as done below:



(a)

(b)

Figure 3.12: Evolution of Eq. (3.8) with many moves in the same direction

Proposition 3.3.2. *The μ operator that associates to a map f another map $\mu(f)$ such that $\mu(f)(x) = \min\{f(y) \mid y \in B(x,r)\backslash\{x\}\}$ preserves K-Lipschitz continuity.*

Proof. Let f be a K-Lipschitz continuous map, x and y two points such that, and $y \in B(x,r)$. We can consider $d(x,y) = r$ without loss of generality. By definition of K-Lipschitz continuity, there is

$$|f(x) - f(y)| \leq K.d(x,y) = K.r.$$

Since $\mu(f)(x)$ takes the minimum value in $\{f(z) \mid z \in B(x,r)\backslash\{x\}\}$, then it is majored by $f(y)$ and minored by the minimal possible values of the z's, namely $f(x) - K.r$. The same holds for $\mu(f)(y)$:

$$f(x) - K.r \leq \mu(f)(x) \leq f(y),$$

$$f(y) - K.r \leq \mu(f)(y) \leq f(x).$$

By considering the $\mu(f)(x)$ inequality $\pm K.r$, we obtain:

$$\begin{aligned} f(x) - 2K.r \leq &\quad \mu(f)(x) - K.r &&\leq f(y) - K.r, \\ f(x) \leq &\quad \mu(f)(x) + K.r &&\leq f(y) + K.r, \end{aligned}$$

Combining these two last inequality with the $\mu(f)(y)$ one, we obtain:

$$\mu(f)(x) - K.r \leq \mu(f)(y) \leq \mu(f)(x) + K.r,$$

By transitivity across the space, this result extends to any pair of points:

$$|\mu(f)(x) - \mu(f)(y)| \leq K.d(x,y) \quad \forall\{x,y\} \subset S.$$

□

Now it is established that the null distance assignment is the only source of possible K-discontinuity. The requirement is therefore that a site x can receive a particle only if any neighboring site y has a low distance value $D[P](y) \leq K$. Looking at Fig. 3.12, we can see that the bigger difference appears because the middle site with value 4 does not receive any information before being actually set of zero.

A simple solution is to increase the distance field neighborhood radius to 2. In this case, the distance values are always updated fast enough. But let us skip this one and only consider the case where the neighborhood is one for every fields.

With a single particle, it is then clear that there is a problem if the particle moves at the speed of the information. The solution is, therefore, as simple as bounding the speed of the particle. But since the particles trajectory is a composition of null speed move and speed 1 move, we can only bound the number of consecutive moves in the same direction (Fig. 3.3.2). We summarize this in the following formal statements:

Definition 3.3.3. *In continuous space, a dynamic set of particles P is K-continuous if for any $x \in P_t$, there exists, for any t', some $y \in P_{t'}$ such that $d(x, y) \leq K.|t - t'|$.*

Definition 3.3.4. *In discrete space, a dynamic set of particles P is K-continuous if for any $x \in P_t$, there exists, for any t', some $y \in P_{t'}$ such that $d(x, y) \leq \lceil K.|t - t'| \rceil$.*

Proposition 3.3.5. *If a dynamic set of particle P is $\frac{K-1}{K}$-continuous, then its distance field $D[P]$ is K-Lipschitz continuous.*

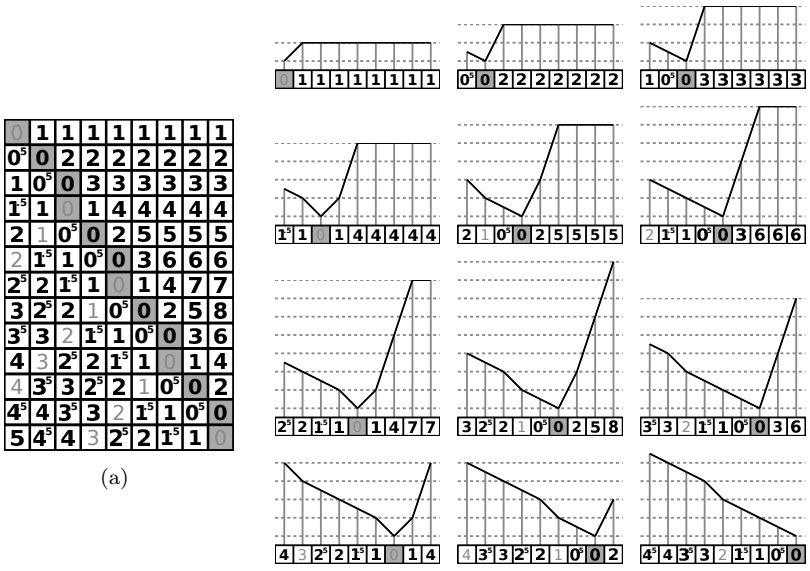
With many particles, the proximity of particles may bound the distance values, allowing some particles to move a higher number of consecutive time without causing any K-discontinuity. We come back to this in Sect. 3.5, but for the moment, the point is made: having a bounded gradient is possible without reducing the class of considered dynamic set of particles to useless cases. So let us now find a way to use the gradient finiteness to obtain a finite field.

3.4 Continuous Fields Representation

In this section, we show that if the gradient is bounded (Eq. (3.10)), then the modulo of the distances are enough to compute this gradient. For readability, let us denote by $\hat{f}$ the modulo of function f. So the modulo operator projects the initial field $D[P] \colon T \times S \to \mathbb{N}$ into a field $\widehat{D}[P] \colon T \times S \to \mathbb{Z}/n\mathbb{Z}$:

$$\widehat{D}[P]_t(x) = D[P]_t(x) \bmod n \text{ for all } x \in S$$

with $n = 2K + 1$, while conserving the gradient information $\Delta D[P]$. More generally, this is true for any K-continuous function $f \colon S \to \mathbb{N}$ and its associated gradient Δf. Let us examine what are the effects of the modulo transformation, and describe how to retrieve the gradient Δf back from the transformed function $\widehat{f}$.



(a) (b)

Figure 3.13: Evolution of Eq. (3.8) with bounded number of consecutive moves in the same direction

Figure 3.14: Effects of the modulo operator on the order

3.4.1 Preservation of gradient by the modulo

Let x be any point of the space. We know that for any $y \in B(x,1)$, $f(y) \in [f(x)-K, f(x)+K]$. Since the value $n = 2K+1$ used to apply the modulo is actually the size of the set $\{f(x)-K, \ldots, f(x)+K\}$, the modulo operator acts as a bijection from $[f(x)-K, f(x)+K]$ to $[0, 2K+1]$. This ensures that all information $f \restriction B(x,1)$ contained in the neighborhood of x is conserved, including the local gradient.

If we denote this local bijection as $\mathrm{mod}_u \colon [u-K, u+K] \to [0,n]$, we need to find its inverse function mod_u^{-1}. Figure 3.14 shows that the transformation $\mathrm{mod}_{f(x)}$ cuts the interval $[u-K, u+K]$ into two parts at some multiple of n and switches their relative position while conserving the order inside each interval. Indeed, the modulo maps all multiples $(a-1).n$, $a.n$, $(a+1).n$ to zero, thus translating the intervals $[(a-1).n, a.n[$, $[a.n, (a+1).n[$, and $[(a+1).n, (a+2).n[$

with different amounts $-(a-1).n$, $-a.n$, $-(a+1).n$ respectively.

$$\mathrm{mod}_u(v) = -a.n + \begin{cases} v - n \text{ if } v > a.n + 2K, \\ v + n \text{ if } v < a.n, \\ v \text{ otherwise.} \end{cases}$$

Because we only want to recover the gradient, and not the absolute value, we can safely ignore the $-a.n$ translation. Thus, we define the operator dom_u that translates any value $\widehat{v}$ to another value having the same modulo, but being in the required bound, i.e. $\mathrm{dom}_{\widehat{u}}(\widehat{v}) \equiv v \pmod{n}$ and $\mathrm{dom}_u(\widehat{v}) \in [\widehat{u}-K, \widehat{u}+K]$. Since $\widehat{u}, \widehat{v} \in [0, n]$ then:

$$\mathrm{dom}_u(\widehat{v}) = \begin{cases} \widehat{v} - n \text{ if } \widehat{v} > \widehat{u} + K, \\ \widehat{v} + n \text{ if } \widehat{v} < \widehat{u} - K, \\ \widehat{v} \text{ otherwise.} \end{cases}$$

It is clear that $\mathrm{dom}_u(\mathrm{mod}_u(v)) = \widehat{u} - u = -a.n$ and $\mathrm{mod}_u^{-1} = \mathrm{dom}_u + a.n$. Therefore, any information computed from $f \restriction B(x,1)$, which is taken from an infinite set, gives the same results when computed on $\mathrm{dom}_{\widehat{f}(x)} \circ \widehat{f} \restriction B(x,1)$, which belongs to a finite set if the computation does not depend on the absolute values. For the local gradient in particular, we have:

$$\begin{aligned} \Delta f(x,y) &= \mathrm{dom}_{\widehat{f}(x)}(\widehat{f}(y)) - \mathrm{dom}_{\widehat{f}(x)}(\widehat{f}(x)) \\ &= \mathrm{dom}_{\widehat{f}(x)}(\widehat{f}(y)) - \widehat{f}(x) \end{aligned}$$

3.4.2 Determinism of the modulo field

The preservation of the gradient by the modulo for any K-Lipschitz field has no implications on the number of states in the general case. Indeed, the modulo is applied on each configuration, but the transition from a configuration to next one is defined on the original field. The question is thus to determine whether the next modulo configuration can be computed from the previous modulo configuration or not, i.e. whether the modulo field is locally deterministic or not.

To have a locally deterministic modulo field, we need the modulo of the next configuration not to depend on the absolute values of the previous configuration. Let us check this for Eq. (3.8). In fact, it is pretty easy to see that this is the case. Indeed, the first two cases of this equation gives 0 and 0.5, whose modulo are themselves and only depend on P. In the last case, the modulo of the min only depends on the modulo of the minimal value of the considered set, and the identification of that minimal value only depends on the order of the element in the set. So the modulo of the transition does not depend on any absolute value.

Armed with this result, we can transform Eq. (3.8) in the following equation

$$\widehat{D}[P]_{t+1}(x) = \begin{cases} 0 \text{ if } x \in P_{t+1} \text{ else:} \\ 0.5 \text{ if } x \in P_t \text{ else:} \\ 1 \widehat{+} \min\{\, \mathrm{dom}_{\widehat{D}[P]_t(x)}(\widehat{D}[P]_t(y)) \mid y \in B(x,1) \backslash \{x\} \,\}. \end{cases}$$

which corresponds to the local transition function of the modulo distance field. However, one should not forget that the bound on the gradient depends on set of particles P. In the next section, we consider the design of moving particles respecting the different conditions seen until now. When such sets of moving particles are combined with Eq. (3.8), the result is therefore always a finite state system, as it will be the case in all the following chapters.

For the number of states needed for the modulo distance field, one should take into account that if a particle moves, then there will be some 0.5 values in the evolution. Since the interval $[-K, +K]$ contains the possible values $\{-K, -K + 0.5, \ldots, K - 0.5, K\}$, which gives $\{2K, -2K + 1, \ldots, 2K - 1, 2K\}$ when multiplying by 2, we thus have that the number of states is $4K + 1$. But if no 0.5 values are needed, the number of states is $2K + 1$.

3.5 Particles Movement According to a Field

Until now, we have studied the computation of distance fields of some external set of moving particles. In this section, we consider particles whose movements are typically determined by an external field. We tackle this by considering that the particle go through all edges "open for traffic" by some external parameter, this parameter being typically a field. This allows us to improve the modularity, so that considering a different movement does not imply to study all the details again.

We also consider the case where a distance field is computed from the particles. In this case, we show how to ensure the K-Lipschitz continuity of the distance field in order to obtain a finite number of states.

The last thing to note is that the movements described here are simply those that arose naturally in the different study made for this document. No claim of full generality is made.

3.5.1 Movements through open edges

In order to factorize some consideration, we consider that the particles do not move directly according to a field, but according to a function that tells for each oriented edge, if the particle goes through the edge or not. Let us denote this function as $O\colon S^2 \times T \to \mathbf{2}$, and read $O_t(x, y)$ as "the oriented edges (x, y) is open at time t". Figure 3.15 shows the openness possibilities for an edge $\{x, y\}$, and their respective graphical representations.

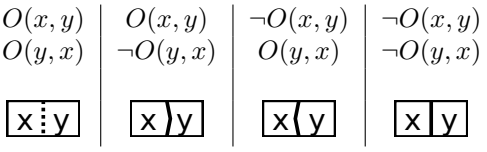
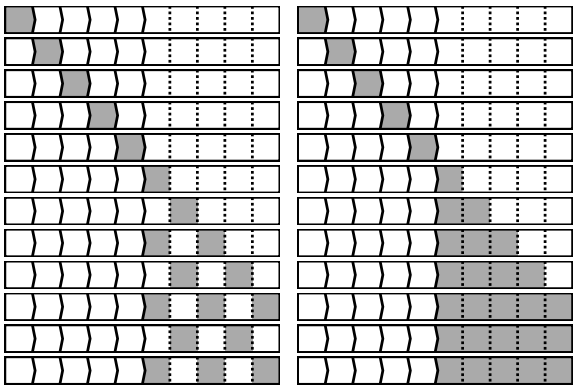


Figure 3.15: Representation of edges openness



(a) Moving with rule M (b) Moving with rule M'

Figure 3.16: Movement through doubly open edges

For our study, we only consider particles that move whenever they can, but never disappear. Let us express this in terms of site gaining, loosing or keeping its particle. In order to prevent a particle to disappear, if a site has a particle, but no edge open toward any neighboring sites, it simply keeps its particle. We formalize this into the following predicate:

$$K[m,O]_t(x) = m_t(x) \ \wedge \ \neg\exists y \in N(x), O_t(x,y)$$

In all other cases, the site looses its particle, being sure that a neighboring site will gain the particle. So from the point of view of this neighboring site, it gains a particle because there is an edge open toward it, and a particle at the other extremity of this edge

$$R[m,O]_t(x) = \exists y \in N(x), m_t(y) \ \wedge \ O_t(y,x)$$

Composing these two predicates, and a Boolean field $in\colon S \times T \to \mathbf{2}$ indicating particles insertion, we obtain the following local transition function $M[in,O]$:

$$m_{t+1}(x) = M[in,O]_{t+1}(x) = in_{t+1}(x) \ \vee \ R[m,O]_t(x) \ \vee \ K[m,O]_t(x) \quad (3.11)$$

This rule has the correct behavior whenever no edges is doubly open, for strictly increasing movement for example. In the case of doubly open edges, a blinking is generated, which is not what we want in our case (Fig. 3.16(a)). We prevent this blinking by requiring site having a doubly open edge to keep their particles:

$$K'[m,O]_t(x) = m_t(x) \ \wedge \ \exists y \in N(x), O_t(x,y) \ \wedge \ O_t(y,x)$$

Adding this to the local transition function $M[in,O]$, we obtain $M'[in,O]$:

$$m_{t+1}(x) = M'[in,O]_{t+1}(x) = M[in,O]_{t+1}(x) \ \vee \ K'[m,O]_t(x) \quad (3.12)$$

3.5.2 Movements according fields

As the movement evolution has been described, let us examine its behavior when used to obtain a movement according to an integer field. Let us consider only

the unidimensional cellular spaces for the moment. In this case, one usually considers the movements towards the local maxima or minima of the field. So we consider the following edge openness function:

$$\begin{array}{lcl} \mathrm{Dir}[f,<]_t(x,y) & = & f_t(x) < f_t(y) \\ \mathrm{Dir}[f,\leq]_t(x,y) & = & f_t(x) \leq f_t(y) \\ \mathrm{Dir}[f,\geq]_t(x,y) & = & f_t(x) \geq f_t(y) \\ \mathrm{Dir}[f,>]_t(x,y) & = & f_t(x) > f_t(y) \end{array} \tag{3.13}$$

3.5.3 Movements of bounded distance field gradient

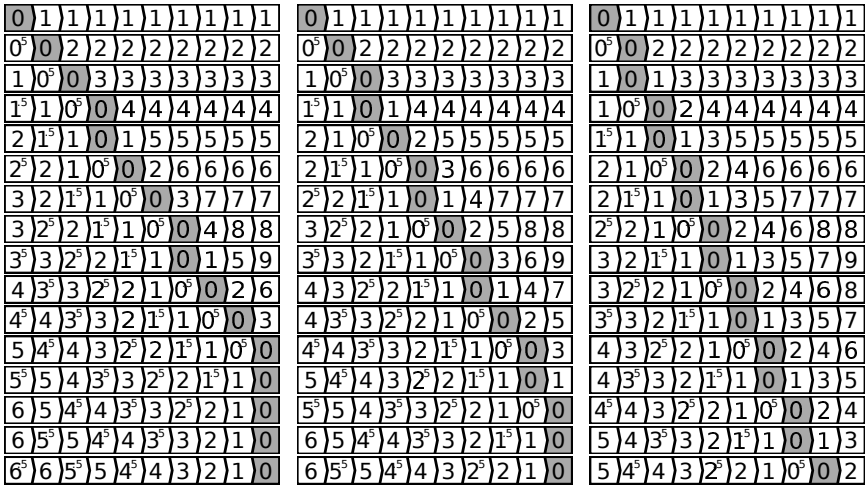
In the cases where the distance field of the moving particles is computed, one will typically want this distance field to be K-Lipschitz continuous in order to have a finitely representable gradient. As explained in Sect. 3.3.2, it is thus required that any site gaining a particle has sufficiently low distance value. To respect the gradient bound K, we therefore need the site value to be less than $K-1$. This way, if it is set to zero in the next configuration, we will be sure that its neighboring sites have values less or equal to K. So we consider the following edge openness function:

$$B[\mathrm{dp}]_t(x,y) = \mathrm{dp}_t(y) - \mathrm{dp}_t(x) \leq K-1. \tag{3.14}$$

As first example, let us consider a single particle, with an increasing movement on an strictly increasing field. If we want to compute a finite state distance field for this particle, we thus need to consider the following system:

$$\left\{ \begin{array}{lcl} \mathrm{dp} & = & D[p] \\ \mathrm{p} & = & M'[p_0, B[\mathrm{dp}] \wedge \mathrm{Dir}[f,\leq]] \end{array} \right.$$

Figure 3.17 shows the evolution obtained with different bound. One can see that the particle trajectory is $\frac{K-1}{K}$-continuous as described in Sect. 3.3.2. However, it is possible to respect the bound without being $\frac{K-1}{K}$-continuous. This is the case in the evolution shown in Fig 3.18. The reason for that is the proximity of another particle that, in fact, bounds the distance values. Indeed, between two particles at distance $2K-1$, all the sites have a distance value lesser than K. So, if one of the particle moves in the direction of the other particle, it can safely run at speed one as many times as it wants.



(a) With $K = 4$ (b) With $K = 3$ (c) With $K = 2$

Figure 3.17: Movement of a single bounded particle is $\frac{K-1}{K}$-continuous

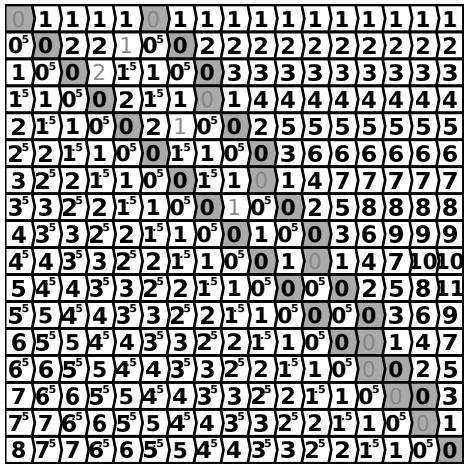


Figure 3.18: Evolution with two particle at distance 5 and $K = 3$

Chapter 4

Density Uniformisation

Contents

Early version of this chapter has been published in [33].

In the previous chapter, we have introduced two basic components: distance fields and Boolean fields representing sets of moving particles. We now consider a problem involving movements of the particles relatively to each other. The problem tackled here is, in fact, a unidimensional version of the repulsion and uniformization problem presented in the introduction chapter. It allows to see how components can be combined to form a coherent dynamic system.

4.1 Problem Statement

4.1.1 Informal Statement

As a general definition, a density uniformization algorithm distributes a given set of points in a given space so that the local density of points is equal every-

where in the space. In our particular case, we consider particles are placed in an unidimensional cellular space, and we want them to move toward a uniform density configuration. Each site hosts at most one particle, in order to work a the finer possible level. Finally, we want the cellular automata rule to be self-stabilizing. That is to say that if no input parameter changes then the rule converges, but if the size of the space or the number of particles change in a smooth way, then the rule adapts its trajectory in a smooth way.

A precise definition of uniform density with discrete particle is hard to give in general, but the unidimensional case is easier to deal with. Indeed, uniform density placements simply correspond to regular placements where the distance between consecutive particles is the same everywhere. Let us consider an idealized continuous space $[0, n]$, and derive precise definition of uniform density placement.

4.1.2 Particle placement and density

The density of particles in a bounded unidimensional space is the number of particle divided by the length of space. Local density at a point x of the space is normally defined as the limit of the density as a considered segment $[x-h; x+h]$ of the space is shrunk to be just the point itself, i.e. $h \to 0$. In our case, considering the limit does not make sense, since we have a finite number of particles. So it is necessary to indicate at which scale h the local density is calculated.

Definition 4.1.1. *The* density *of particles in a segment $[a, b]$ is the number of particles placed on the segment divided by the length $b - a$ of the segment.*

Definition 4.1.2. *The* local density *at point x for scale l is the density of particles in the segment $[x - \frac{l}{2}, x + \frac{l}{2}]$.*

Definition 4.1.3. *A placement has* uniform density *at scale l if the local density for this scale is the same almost everywhere where it is defined.*

For p particles placed in a space $[0, n]$, the density is obviously uniform at the scale $l = n$, since the local density is only defined at point $\frac{n}{2}$ with value $\frac{p}{n}$. An intuitive idea can thus be to require uniform density at the smallest possible scale. Since p particles split any space $[0, n]$ into $p + 1$ particle-free parts, the scale of any placement ρ is lower bounded by the length of the biggest particle-free parts. This is because those parts have null density, while other parts necessarily have non-null density. The only way to minimize the length of those particle-free parts is to equalize their size, thus cutting the space $[0, n]$ into $p+1$ parts of equal length $\frac{n}{p+1}$:

Proposition 4.1.4. *For a set of particle $\{1, \dots, p\}$ in a space $[0, n]$, the smallest possible uniform scale is $\frac{n}{p+1}$, with placement $\rho_i = \frac{n}{p+1} i$ and density $\frac{p+1}{n}$.*

However the local density of this placement is higher than the expected density $\frac{p}{n}$. If we want to have the same density at all uniform scale, a possible placement is $\rho_i = \frac{n}{p}(i - \frac{1}{2})$ for $i \in \{1, \dots, p\}$. This placement is uniform up to the scale $\frac{p}{n}$ and corresponds to cutting the space into p parts of equal size, and to place a particle at the center of each part.

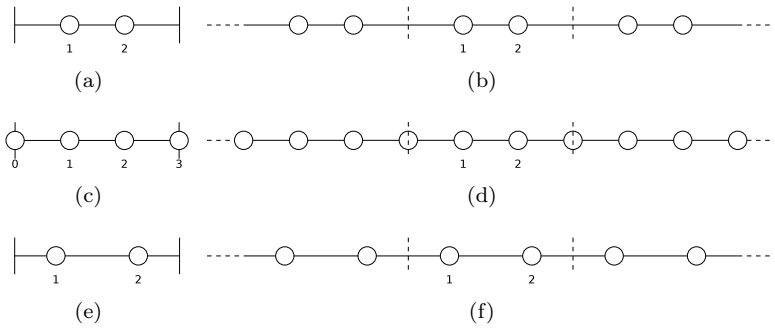


Figure 4.1: Bounded and unbounded space version of the placements

Proposition 4.1.5. *For a set of particle* $\{1, \ldots, p\}$ *in a space* $[0, n]$*, the smallest possible scale with density* $\frac{p}{n}$ *is* $\frac{n}{p}$ *with placement* $\rho_i = \frac{n}{p}(i - \frac{1}{2})$.

In fact, those two solutions are as they are because we consider a bounded space, introducing particularities at the boundaries. Let us clarify the relations between those two solutions and the boundaries by extending them to unbounded spaces using either mirror or cyclic boundaries conditions. In this setting (Fig. 4.1), it is possible to see that the placement of Prop. 4.1.4 needs to be completed with boundary particles in order to correspond to an uniform density placement. This explains the higher density. So if we have particles 0 and $p + 1$ on the boundaries, then we need the placement of Prop. 4.1.4, otherwise the placement of Prop. 4.1.5 has to be considered.

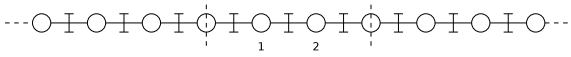
4.2 Solution Description

4.2.1 Walls and Symmetries

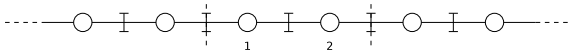
From the previous remarks about uniform placements, it is now possible to derive a global scheme to move each particle to its final position. We introduce a new kind of particle-like objects that we call *walls*. They serve as boundary between the regions of each particle as proposed in the placement of Prop. 4.1.5.

Figure 4.2 shows how walls are placed in uniform density configurations. With them in the space, many symmetries appear. The boundaries of the bounded space is either occupied by particles or by walls, depending on the considered placement. Also, in both situations, walls are exactly at the middle between their neighboring particles, and particles are exactly at the middle between their neighboring walls.

Because of these symmetries, it is clear that the main problem is to be able to move a set of objects to the middles of another set of objects. However, we will need to check the effects of applying these movement laws symmetrically between the particles and the walls, and the effects of the discreteness of the space.



(a) Placement of Prop. 4.1.4: particles on boundaries



(b) Placement of Prop. 4.1.5: walls on boundaries

Figure 4.2: Walls in both uniform density placements

4.2.2 Movement toward the middles

Let us consider the movement of the walls toward the middles between the particles. Since, the middles are the furthest points from the particles, we use a distance field, and ask for the walls to use the gradient of the particles distance field, and to move to local maxima. The distance field $D[p]$ is computed using Eq. (3.8), and the movement is achieved following the results described in Sect. 3.5. We thus obtain the following system:

$$\begin{cases} \mathrm{dp} &= D[\mathrm{p}] \\ \mathrm{dw} &= D[\mathrm{w}] \\ \mathrm{p} &= M'[\mathrm{p}_0, B[\mathrm{dp}] \wedge \mathrm{Dir}[\mathrm{dw}, \leq]] \\ \mathrm{w} &= M'[\mathrm{w}_0, B[\mathrm{dw}] \wedge \mathrm{Dir}[\mathrm{dp}, \leq]] \end{cases} \tag{4.1}$$

The last equation, for example, ask for the walls to move from an initial position w_0, according to the distance field dp of the particle, while taking care of keeping its own distance field dw continuous. The converse is asked to the particles, as dictated by their symmetry. Because we use M' and the non-strict relation $\leq$, we manage the cases where the maxima should be between two sites by occupying both sites (see Figs. 3.4 and 3.16). The last thing to precise is the initial configuration. While p_0 is given freely according to an initial set of particles P, the walls are simply $\mathrm{w}_0 = \neg \mathrm{p}_0$. This way, we ensure that there is a wall between every two consecutive particles, and vice-versa. For the distance fields, we initialize it using one and zeros as in Eq. (3.5).

$$\begin{cases} \mathrm{p}_0(x) &= x \in P \\ \mathrm{w}_0(x) &= x \notin P \end{cases}$$

The symmetries between particles and walls allow to deal with both considered placements in with only one cellular automata. The difference will be naturally implied by the boundaries initial states, without special consideration. Figure 4.3 shows the evolution of the system on a space of length 17, with two particles placed at position 1 and 3 (the first site number is zero). The left column corresponds to the particles and their distance field, and the right column to the walls and their distance field.

4.3 System Evolution and Energy

First of all, Figure 4.3 shows that the system indeed evolves to a fixpoint where the particles and the walls are placed regularly in space. The evolution begins with the distance field of the particles updating its values according to the

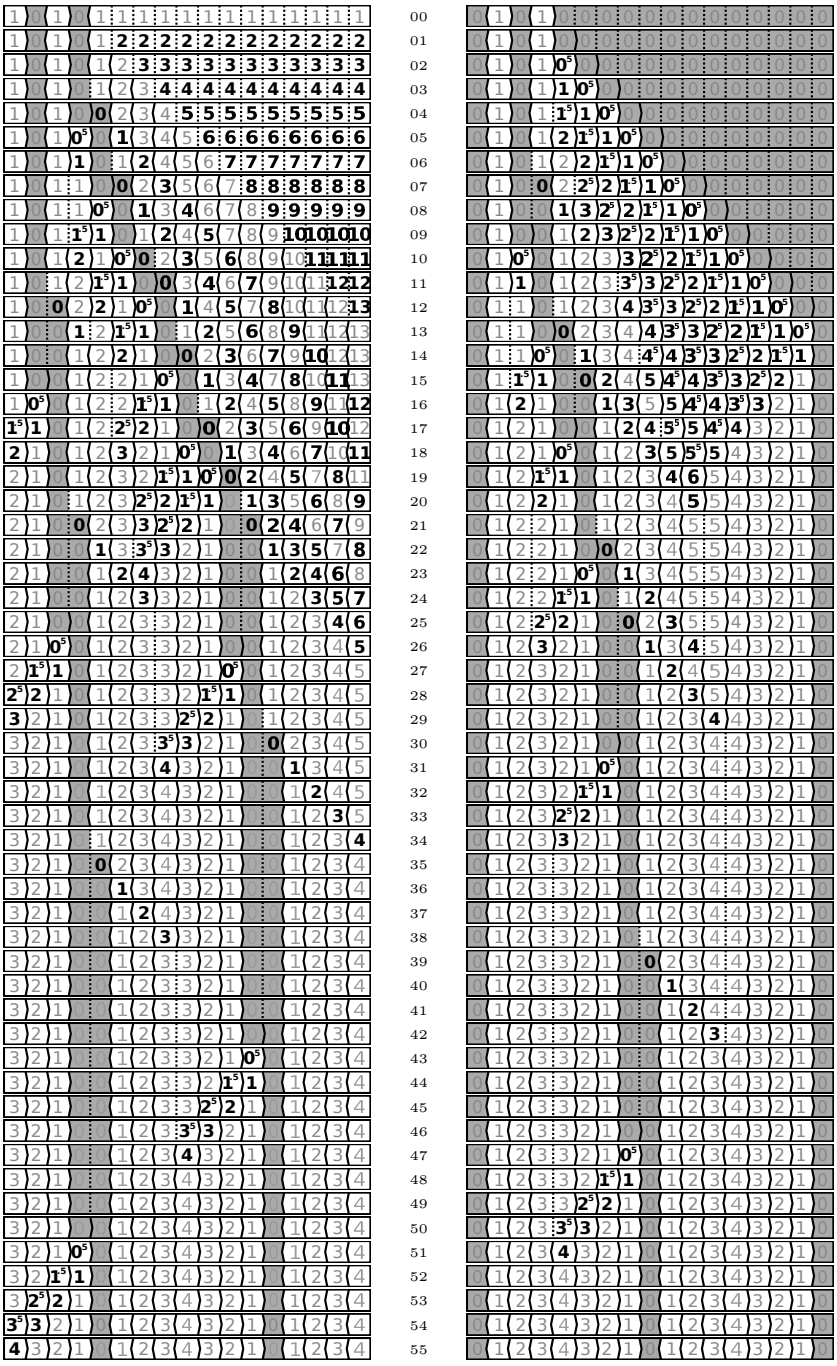


Figure 4.3: Evolution of the system Eq. (4.1)

absence of particle in the right of the space. As the particle distance field is updated, the rightmost wall shrinks of one site per transition. Also, as the wall shrinks, the wall distance field update its values, and to make the long story short, the walls distance field updates affect the particles, that in turn affect their distance field and so on. In fact, the evolution reflects the circular relation between the distance fields, the walls and the particles. This behavior is easily identifiable from time 27 where only a few updates travel in the space, acting as signals.

In Fig. 4.4, the same evolution is drawn, the walls, the particles, and their distance fields configurations being superposed into one figure per time. In the last seven lines of the figures, it is possible to see the signals traveling in the shape of the distance fields.

In order to study the dynamics of the cellular automaton, we identify these signals traveling between particles and walls and causing their displacements in the cellular space. From the interaction between the particles, the walls and the signals, we abstract the behavior of our two-layered cellular automaton into a consistent space-time diagram showing how the cellular automaton evolves as a whole.

4.3.1 Circular relation and perturbation propagation

As said before, the updates of values propagate from the distance field of the particle to the walls, then from the walls to their distance field, then from the walls distance field to the particles, which in turn propagate to their distance field and so on. This can be viewed as a propagation of perturbation from the equilibrium state of each component of the system, the state of each component being an input for another component in a circular way:

Indeed, the Boolean fields have their Boolean states evolving depending on the direction indicated by their input distance field gradients. Conversely, the distance fields have their distance values evolving depending on the particles position indicated by their input Boolean field.

In order to identify precisely the nature of the signals, we study the equilibrium states of both fields, and the way each of them return to an equilibrium state after a single modification of their input. This will lead us to the definition of a precise energy function for each component, composed into a global energy. This energy is expressed in terms of number of changes of the Boolean field.

4.3.2 Equilibrium of the Boolean fields

As already observed in the previous chapter, and as it can be observed on the evolution shown in Fig 4.3, the distance field between any two particles is always first strictly increasing, and then strictly decreasing. When studying the Boolean field of the walls for example, we can thus focus on any slice of the space delimited by null distance value on the distance field of the particle. Supposing that the distance field does not change, Figure 4.6 shows the two possible equilibrium configuration for the Boolean field, depending on the parity

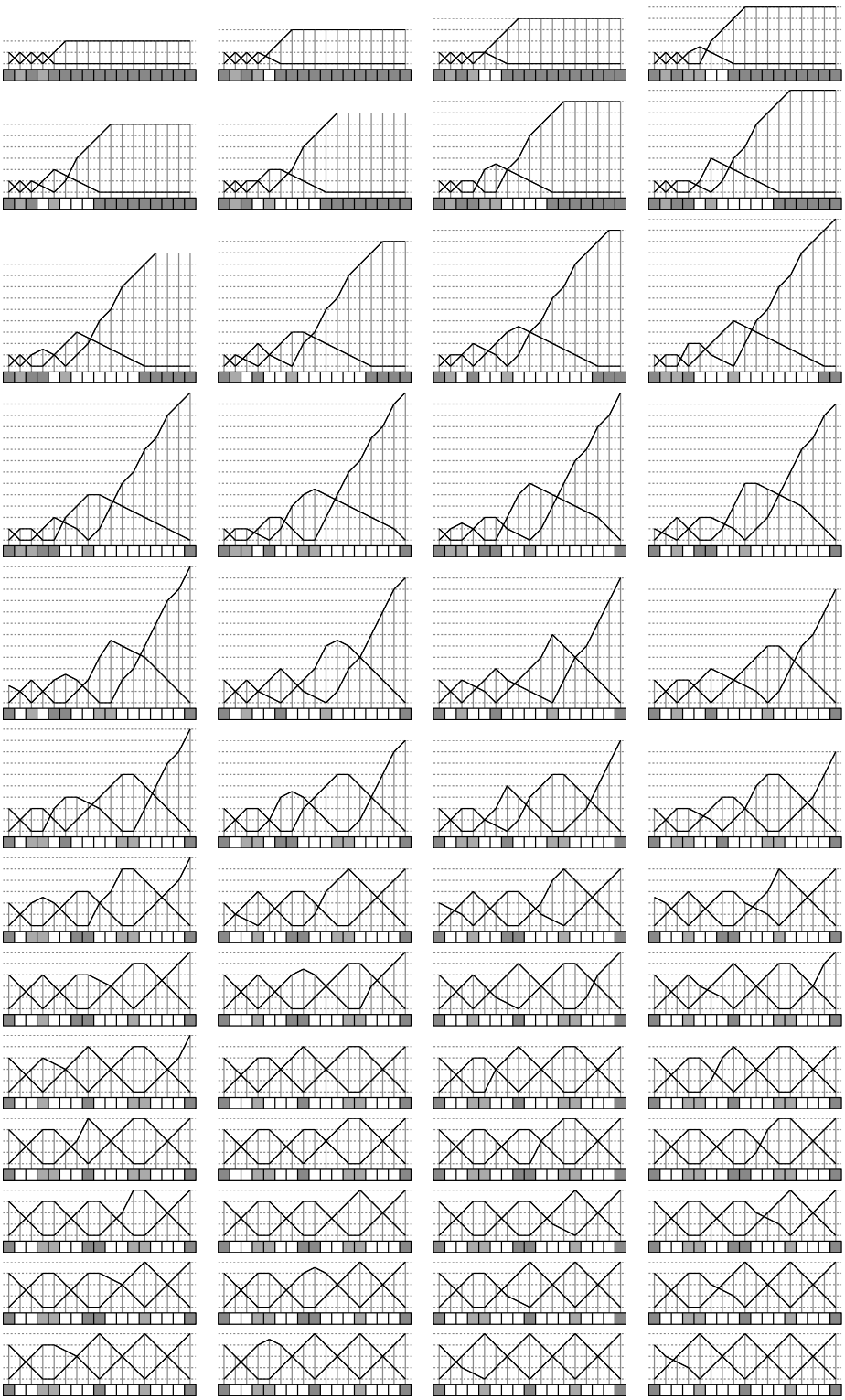


Figure 4.4: Evolution of the system Eq. (4.1)

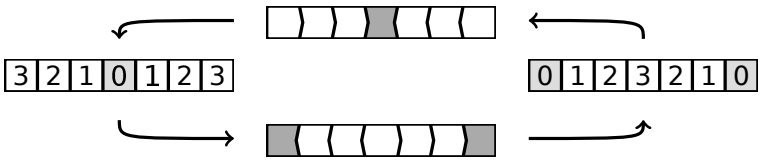


Figure 4.5: Relations between the Boolean and the distance fields

of the size of the slice.

Figure 4.6: Equilibrium states of the Boolean fields

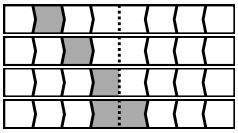


Figure 4.7: From arbitrary to equilibrium Boolean evolution

Let us now consider a non-equilibrium state and its evolution back to equilibrium as shown in Fig 4.7. We can see that among all the open edges, only three of them actually have an effect. Each of the two right oriented edges have two effects: one site loosing a particle and another gaining a particle. The doubly oriented edge has a single effect.

To summarize this in terms of energy initially present at each edge and causing the evolution, we can say that any right oriented edge having a particle on its left has an energy of 2, since it will eventually leads to two Boolean state modifications. The same thing hold for doubly oriented edges that have energy 1 as long as one of their site has no particle.

From this statement, it may seem that the energy of an edge depends on arbitrary distant site states. However, it is not the case since the required information can be deduced using both gradients values on the edge. Indeed, the direction to a particle is indicated by its associated distance field. So everything can be expressed in terms of sign of gradients of the particles and the walls layer.

Figure 4.8 show the same evolution as Fig. 4.7 with the distance fields associated to each layer. It it possible to see that the edges having an energy correspond to those whose gradients are not of opposite signs. When they are both positive, the energy has to be 2. When one is null but the other is positive, the energy is 1, and two null gradients have no energy. We obtain the following formula. In fact, this equation takes into account the direction of the movements, since it gives -2 for two negative gradients. Thus, it corresponds to

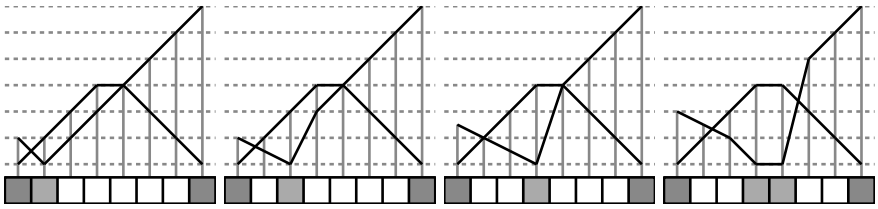


Figure 4.8: From arbitrary to equilibrium Boolean evolution

a momentum, whose absolute value is the energy of the edge.

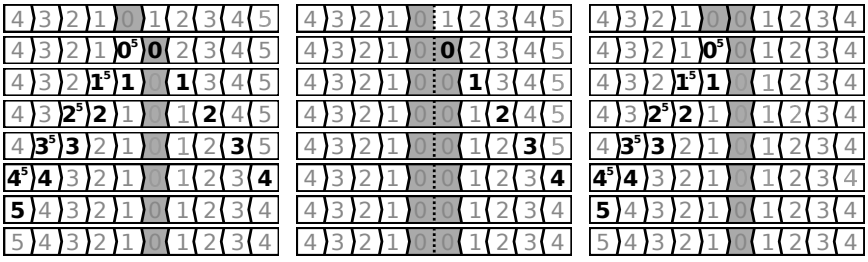
$$Q_t^{\mathrm{pw}}(x, x+1) = \mathrm{sgn}(\Delta \mathrm{dp}_t(x, x+1)) + \mathrm{sgn}(\Delta \mathrm{dw}_t(x, x+1)). \tag{4.2}$$

$$\mathrm{sgn}(x) = \begin{cases} -1 & \text{if } x < 0 \\ 0 & \text{if } x = 0 \\ 1 & \text{if } x > 0 \end{cases}$$

One thing has to be noted here. Eq. (4.2) is totally symmetric on the distance fields, so there is no need to consider it twice. It corresponds directly to the sum of the momentum of both particle and wall Boolean fields.

4.3.3 Equilibrium of the distance fields

Let us now consider the effects of a modification of the Boolean fields on the equilibrium of the distance fields, and analyze how the the momentum is transformed. In fact, Figure 4.8 already shows an evolution starting with an equilibrium distance field which is modified as the particle moves. The equilibrium of the distance field is characterized by local difference of 1 or -1 or zero, with the 0 of the distance field corresponding to the particles. Let us see what happen when a particle moves.



(a) one unit move (b) semi-move: addition (c) semi-move: removal

Figure 4.9: Effect of movements on the distance field

Looking at the configurations at the modification time, it is possible to see that extending a particle to the right leads to a difference of 2 on the corresponding edge. For a shrinking of one particle on the left, we obtain two differences of -0.5. By conservation of the momentum, they both correspond to a momentum of 1. This can, indeed, be retrieved by comparing these differences with the equilibrium differences 1 and -1:

$$Q_t^{\mathrm{dp}}(x, x+1) = \Delta\mathrm{dp}_t(x, x+1) - \mathrm{sgn}(\Delta\mathrm{dp}_t(x, x+1)). \tag{4.3}$$

When a particle moves many times in the same direction, the signals are added on top of each other, since the gradient becomes bigger and bigger. Then, looking at the evolution after the modification, it can be observed that theses differences are just translated to obtain the next configuration. Indeed, applying the min operator locally on a monotonic function just translates it in the increasing direction. In the distance field local transition function (Eq. (3.8)) applies min but also add 1 to the distance values. Therefore translation vectors are respectively $(1, 1)$ for increasing parts, and $(-1, 1)$ for decreasing parts. Figure 3.12 illustrates this fact.

Just to make things clear, let us also consider the transfer of the energy from the distance fields to Boolean fields. We know that as the distance values are updated, the local maxima are displaced. This maxima is at the middle between two particles. If a particle moves of 6 sites, the maxima is shifted of 3 units. In fact, the momentum is conserved since moving of 3 units amounts to remove 3 times and add 3 times. With a odd number, the numbers of removals and additions differ of one.

4.3.4 Global Equilibrium and Energy

Now that we have defined the momentum for all fields, we can consider the equilibrium of the whole system. First, let us consider the combination of all fields at a given edge. We obtain the following:

$$\begin{aligned}
Q_t(x, x+1) &= Q_t^{\mathrm{pw}}(x, x+1) + Q_t^{\mathrm{dp}}(x, x+1) + Q_t^{\mathrm{dw}}(x, x+1) \\
&= \mathrm{sgn}(\Delta\mathrm{dp}_t(x, x+1)) + \mathrm{sgn}(\Delta\mathrm{dw}_t(x, x+1)) \\
&\quad +\Delta\mathrm{dp}_t(x, x+1) - \mathrm{sgn}(\Delta\mathrm{dp}_t(x, x+1)) \\
&\quad +\Delta\mathrm{dw}_t(x, x+1) - \mathrm{sgn}(\Delta\mathrm{dw}_t(x, x+1)) \\
&= \Delta\mathrm{dp}_t(x, x+1) + \Delta\mathrm{dw}_t(x, x+1)
\end{aligned}$$

Because it is a sum of differences, this equation keeps the same form when summed over any sub-part $[y, z]$ of the space, or even the whole space:

$$\begin{aligned}
Q_t(y, z) &= \sum_{x \in [y,z]} Q_t(x, x+1) \\
&= \sum_{x \in [y,z]} \Delta\mathrm{dp}_t(x, x+1) + \Delta\mathrm{dw}_t(x, x+1) \\
&= \sum_{x \in [y,z]} \Delta\mathrm{dp}_t(x, x+1) + \sum_{x \in [y,z]} \Delta\mathrm{dw}_t(x, x+1) \\
&= \Delta\mathrm{dp}_t(y, z) + \Delta\mathrm{dw}_t(y, z)
\end{aligned}$$

These equations describe precisely the global aspect of the final configuration shown in the evolution Fig. 4.4. Indeed, if the momentum is null at every edges,

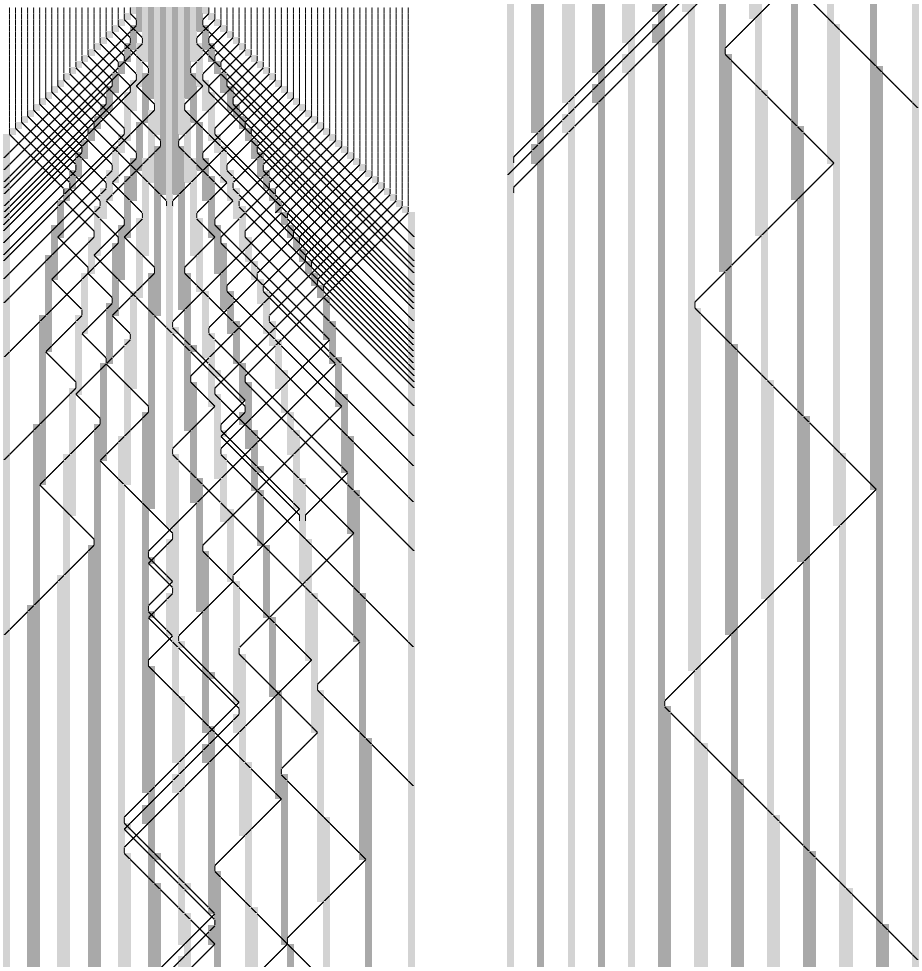


Figure 4.10: Space-time diagram of the uniformization

then each distance field is the opposite of the other one, with some constant:

$$\begin{aligned} Q_t(y,z) &= 0 \\ \Delta\mathrm{dp}_t(y,z) + \Delta\mathrm{dw}_t(y,z) &= 0 \\ \Delta\mathrm{dp}_t(y,z) &= -\Delta\mathrm{dw}_t(y,z) \end{aligned}$$

But this equation does not fully describe the equilibrium state, since it is required for each field to be stable individually. From the definition of each field energy, it is possible to deduce the movements of each local energy. Indeed, the energy of the Boolean fields do not move, while the distance fields energy move to the local maxima. From these considerations, the evolution of all the fields can be summarize into a unique space-time diagram showing the particles, the walls and the energy traveling in the cellular space (Fig 4.10).

This figure exhibits a whole evolution from an initial configuration with a compact set of particles placed slightly to the left of the center of the space. The

final configuration (bottom of the second column) shows a regularly placed set of particles. The figure also shows how the energy decreases. Indeed, the signals that reach the border of the space simply disappear. There are also cases where a positive and a negative signal of same value meet the same source at the same time and no movement of that source is created. When there is no more signal in the system, the fixpoint is reached and the placement is regular.

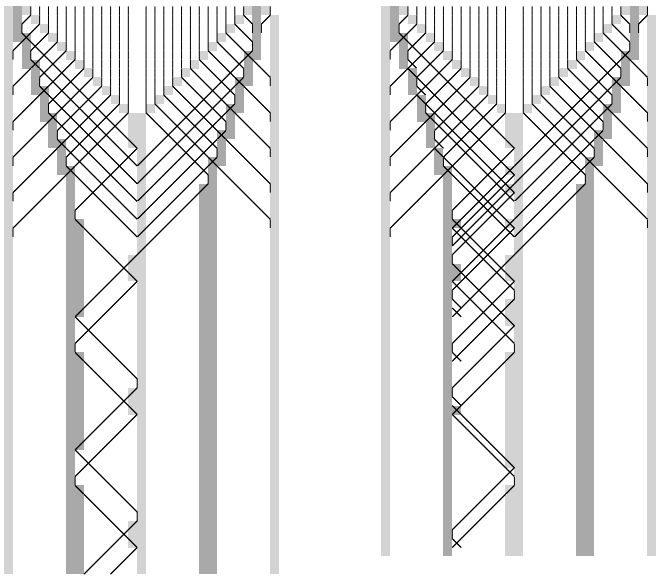
However, it is possible that some signals never leave the system and cause cycles by their interaction. An example is shown in Fig. 4.11(a). Such cycles are created by small signals of opposite signs placed in a region of space such that the displacement caused by one is canceled by the other. However, signals of opposite signs just vanish if they meet. But this vanishing can not happen because both signals have the same speed.

One can notice that relaxing the synchronicity of the system will change the signals movements relatively to each other into Brownian ones and the signals will eventually meet. In the same vein, another solution is to slow down one of the signals probabilistically, the positive one for instance. The slow down is achieved preventing a 1-site particle or wall to become 2-sites when moving to the right on reception of a positive signal. Formally, when such a condition is verified on an edge, its openness is probabilistic:

$$\mathrm{Slow}[f]_t(x,y) = x < y \wedge f_t(x) = f_t(y) \wedge \mathrm{rand}(p),$$

where rand(p) is true with probability p. Figure 4.11(b) shows the results.

$$\begin{cases} \mathrm{dp} &= D[\mathrm{p}] \\ \mathrm{dw} &= D[\mathrm{w}] \\ \mathrm{p} &= M'[\mathrm{p}_0, B[\mathrm{dp}] \wedge \mathrm{Dir}[\mathrm{dw}, \leq] \wedge \mathrm{Slow}[\mathrm{dw}]] \\ \mathrm{w} &= M'[\mathrm{w}_0, B[\mathrm{dw}] \wedge \mathrm{Dir}[\mathrm{dp}, \leq] \wedge \mathrm{Slow}[\mathrm{dp}]] \end{cases}$$



(a) (b)

Figure 4.11: (a) Space-time diagram of cycle with two particles. (b) With the probabilistic rule with $p = 0.8$, the signal is sometimes retained.

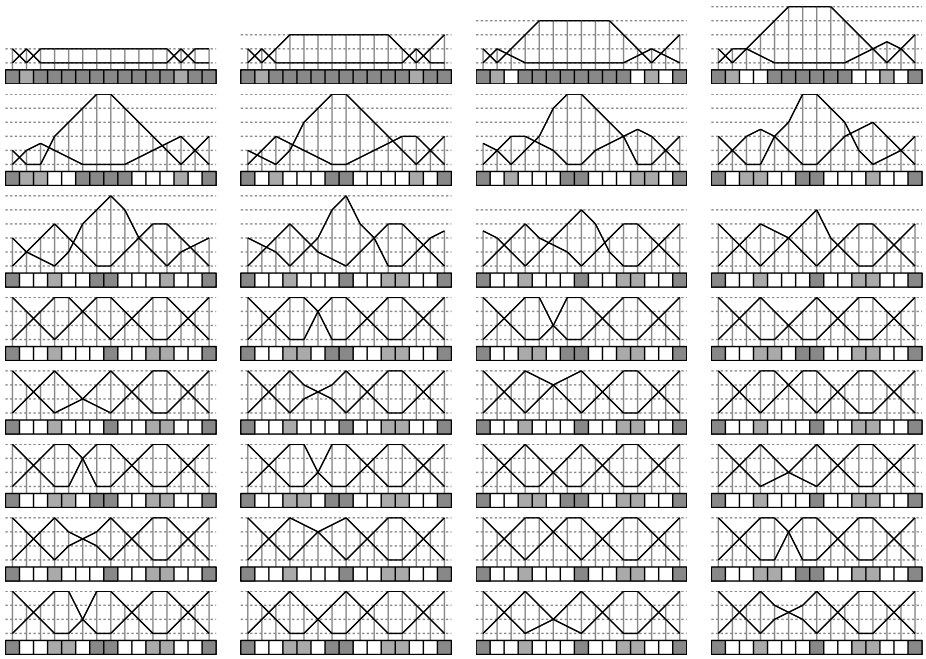


Figure 4.12: Two opposite signals running in cycle between a wall and a particle

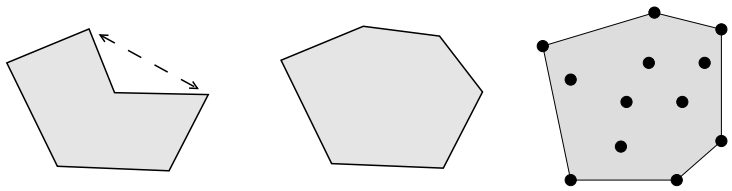
Chapter 5

Convex Hulls

Contents

Earlier version of this chapter has been published in [35].

Until now, we mainly considered unidimensional cellular space. This chapter, and the following tackle multi-dimensional cellular spaces, using the usual bi-dimensional spaces presented in Chapt. 2 as example (Fig. 2.1). In this chapter, we present and solve the convex hull problem. By doing so, we show that a promixity graph is explicitly computed during the construction of the convex hull of arbitrary distant particles. The full mathematical study of this proximity graph is tackled in the next chapter.



(a) Concave set (b) Convex set (c) Convex hull

Figure 5.1: Euclidean convexity: gray: considered set, black points: particles

5.1 Problem Statement

5.1.1 Informal Statement

As a general definition, a convex hull algorithm determines for a given set of particles placed in a space, the set of points of the space that belongs to the convex hull of the particles. A classical notion of convexity exists for Euclidean spaces. However, other kinds of convexity can be defined for Euclidean spaces, and also for other kind of spaces. In order to precisely define the problem, we then have to clarify which convexities are adapted to the cellular spaces we use.

5.1.2 Classical and Abstract Convexity

The most known and studied convexity is defined over Euclidean spaces. In this context, a subset is *convex* if it contains all the segments joining any two of its points. For example, Fig. 5.1(a) shows a non convex set, commonly called concave set, along with a segments that are not contained it the set. In Fig. 5.1(b), it is not possible to find any missing segment. The set is therefore a convex set. The convex hull of a set of particles is the smallest convex set containing the particles. Figure 5.1(c) gives an example of convex hull.

It is possible to characterize Euclidean convex sets and convex hull in many different ways. For example, one can use the fact that the intersection of two convex sets is also convex, and the fact that a straight line (or a flat plane in 3D and so on) cuts the Euclidean space in two convex half-space. Therefore, any convex set is the intersection of many half-spaces, and the convex hull of a set of particles can be defined as the intersection of all half-spaces containing the particles. Sub-parts of theses properties, and of many others, have been selected in different cases to derive different generalizations.

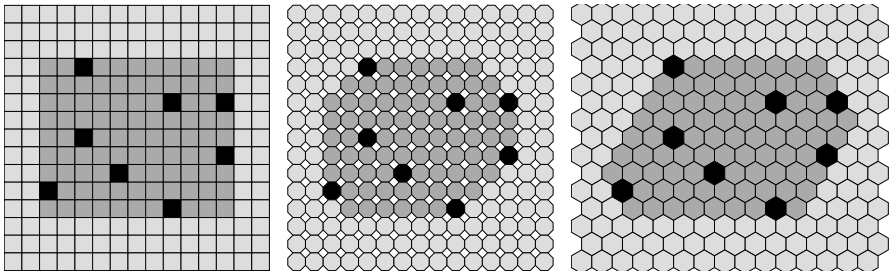
In abstract convexity theory [31, 44], convexity is defined using the relation that the intersection operator puts on the collection of all convex sets. The space itself is then viewed as a simple set. Because of its generality, almost all convexities are specific cases of this one, which is formally defined as:

Definition 5.1.1. *A* convexity $\mathcal{C} \subset \mathbf{2}^S$ *over a set* S *is a collection of subsets of* S*, called* convex subsets *or* convex sets*, that is closed under intersection.*

Definition 5.1.2. *For a given convexity* $\mathcal{C}$ *over a set* S*, the* convex hull $H_{\mathcal{C}}(P)$ *of an arbitrary subset* P *of* S *is the minimal convex set containing* P*.*

(a) 4-square (b) 8-square (c) hexagonal

Figure 5.2: Straight lines in cellular spaces



(a) 4-square with $\theta = 90$ (b) 8-square with $\theta = 45$ (c) hexagonal with $\theta = 60$

Figure 5.3: θ-convex hulls in cellular spaces

5.1.3 Convexity and Cellular Spaces

For the particular cases of cellular spaces, we present here two approaches, the one considered in the state of the art of convex hull computation by cellular automata, and the one that we will consider for our solution.

The first approach is to consider the positions of the sites in Euclidean space. The discreteness of cellular spaces is handle by identifying the angles of the straight lines that fits to the given cellular space as shown in Fig. 5.2. Therefore, a subset is considered to be convex if it is Euclidean convex and it is only composed of straight lines admitted by the cellular space. The 4-square, 8-square, and hexagonal cellular spaces thus respectively allows angles that are multiple of 90°, 45°and 60°. This concept is called θ-convexity, and examples of θ-convex hull are shown in Fig. 5.3.

Definition 5.1.3. *In the Euclidean plane, θ-convexity is the collection of all intersections of half-spaces whose angle is a multiple of the angle θ.*

The second approach consists in forgetting any Euclidean space and only considers cellular spaces and their associated metric. Indeed, Euclidean convexity is a particular case of *metric convexity*. This latter is defined over metric spaces, and replace the segments used to define Euclidean convexity by the more general concept of shortest path. More precisely:

Definition 5.1.4. *A subset P is metric convex if it contains all the shortest paths joining any two of its points, i.e. $\forall x, y \in P, [x, y] \subseteq P$.*

Figure 5.4 shows an example of metric convex hull for each considered cellular space. Comparing with the θ-convex hulls of Fig. 5.3, we can see that they are equivalent for the 4-square and hexagonal cellular spaces. In fact, this equivalence holds for any set of particles. However, they differ for the 8-square cellular space, where metric convex hull is bigger than the θ-convex one.

Conceptually, θ and metric convexities are equivalent on the 8-square cellular space if one associates a length of $\sqrt{2}$ to the diagonal edges. Indeed, the diagonals length is the factor that decides whether the two paths showed in Fig. 5.5 are of same length or not. It is also possible to note that the 8-square θ-convex hull can be obtain as an the intersection of the 4-square metric convex hull and the 8-square metric convex hull. Indeed, the set of half-spaces on the 4-square metric convexity, completed with those of the 8-square metric convexity corresponds to the set of half-spaces on the 8-square θ-convexity. Working with metric convexity thus allows to obtain more generality.

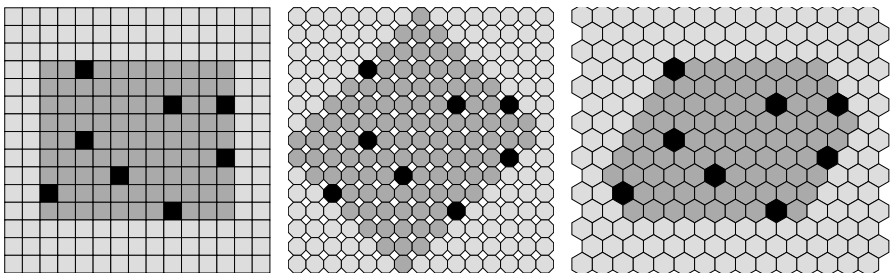
5.1.4 Formal Problem Statement

Given any set of static particles P, the goal is to find a rule R which converges to a convex hull configuration. Formally, there has to be some instant t', depending on P, such that for any $t > t'$, we have $R_t(x) \equiv x \in H_{\mathcal{C}}(P)$. For the reason given in the previous subsection, we consider the convexity $\mathcal{C}$ to be the metric convexity defined in Def. 5.1.4. Figure 5.4 therefore shows examples of initial and final configuration for the different cellular space. The black cells alone correspond to the initial configurations and the black and gray sites together correspond to the final configurations.

5.2 Pre-existing Solutions

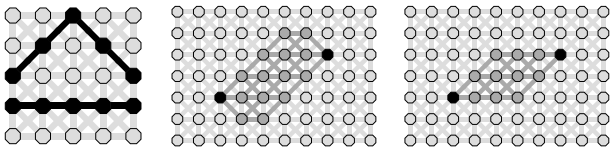
First of all, there is no pre-existing solution that handles the problem at the generality level defined here. Indeed, all of them actually consider the 8-square cellular space with 45-convexity. Also, they both start with a connected region that is then transformed into a convex hull.

The first simple solution is directly related to the well-know majority rule, and works mainly for connected set of particles. The second one needs the



(a) 4-square (b) 8-square (c) hexagonal

Figure 5.4: Metric convex hulls in cellular spaces



(a) Two paths (b) With length 1 (c) With length $\sqrt{2}$

Figure 5.5: Effects of the choice of the diagonals length

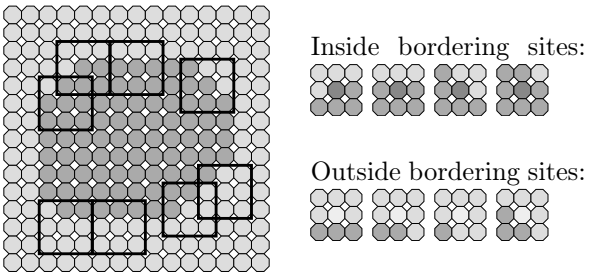


Figure 5.6: Neighborhoods at the border of a 45-convex hull

set of particle to be initially enclosed in a bounded region, which is equivalent to considering that the space itself is bounded. Moreover, this solution has a global stage transition, the first stage shrinking the initial enclosing region, and the second stage re-expanding the region to finally obtain the convex hull. We describe those solutions more precisely in the two following sub-section.

5.2.1 Connected Set of Particles and Majority Rule

In [6], some of the first proposed rules to the convex hull problem are described. These rules can be obtained by examining the local configurations that appear in the convex set configuration, and contrasting them with those that appear in non-convex connected set of particles.

Considering the convex set shown in Fig. 5.6 for example, we can see that sites that are at the outside border of the convex set always have less than 4 neighboring sites in the convex hull. Also, considering any site of the convex hull, we can notice that if its removal leads to a non convex set, then it has at least 4 neighboring sites in the convex set. We can therefore consider the following rule that selects a site if at least 4 neighboring sites are already selected. Figure 5.7 shows an evolution of this rule from a connected set of particle.

$$\text{majo}_{t+1}(x) = x \in P \vee \text{card}\{\, y \in N(x) \mid \text{majo}_t(y) \,\} \geq 4$$

Being just a counter on the neighborhood with a threshold, this rule is just a particular case of the voting rule. The behavior of the voting rule is described in [26], along with its convex hull behavior when the threshold is half the number of neighbors. This is the case here, since we consider the threshold 4 while there are 8 neighbors per sites:

$$\text{majo}_{t+1}(x) = x \in P \vee \text{card}\{\, y \in N(x) \mid \text{majo}_t(y) \,\} \geq \frac{card(N(x))}{2} \qquad (5.1)$$

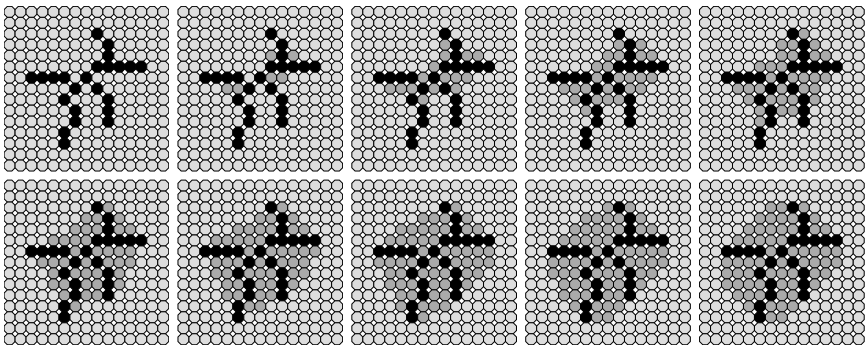


Figure 5.7: Majority rule on a connected set of particle

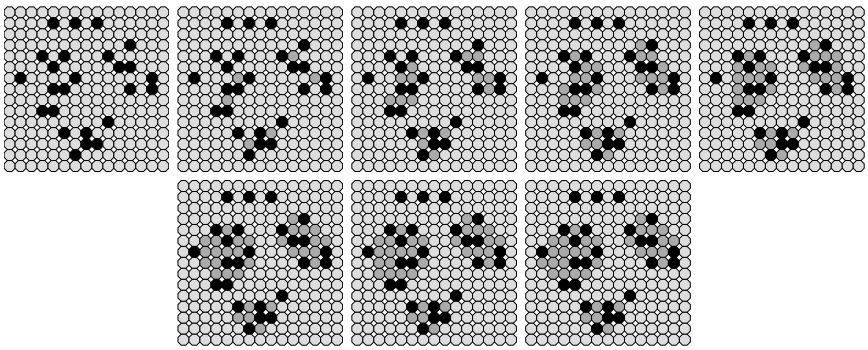


Figure 5.8: Convergence to many convex hulls

In fact, this rule has an interesting, although not necessarily surprising, behavior when used on non-connected set. Indeed, the set of particles does not need to be connected, but simply denser enough, since only quantity matters. In the general case, a set of several partial convex hulls are obtained. The interesting thing is that, as more and more sites are selected by the rule, many partial convex hulls can merge to form bigger convex hulls that can also merge, and so on. For example, Figure 5.8 shows an evolution that converges to many convex hulls. But moving just one particle of the initial configuration of Fig. 5.8 leads to the evolution shown in Fig 5.9. This latter converges to one global convex hull.

5.2.2 Set of Particles Enclosed in a Connected Region

In [46, 14], a rule is proposed that does not require the set of particles to be connected but instead to be enclosed in a connected, compact and bounded marked region. In other words, this rule requires the space to be bounded, considering the space itself as marked region. An example of initial configuration is shown in Fig. 5.10(a).

The proposed rule plays with the same idea of identification of interesting neighborhood at the border, and consists of two globally successive stages. The first one erodes the marked region to obtain a connected region which is included

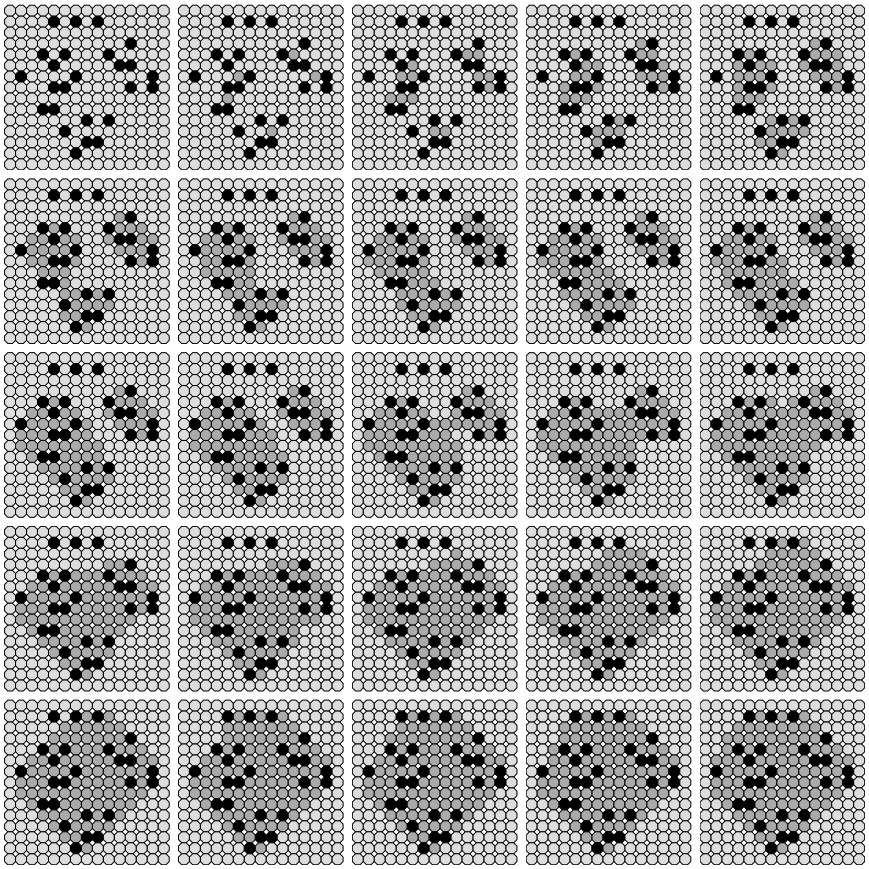
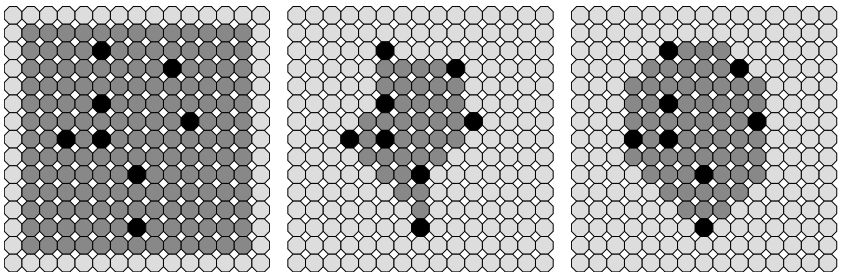
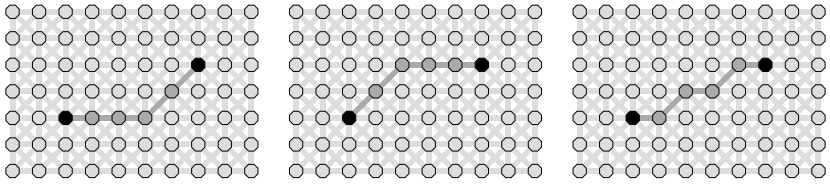


Figure 5.9: Convergence to one global convex hull



(a) Initial configuration (b) After Erosion (c) After Expansion

Figure 5.10: Stages from wrapped seeds to their convex hull: The initial wrapping (a) is shrunk into (b) and is then grown to convex hull (c)



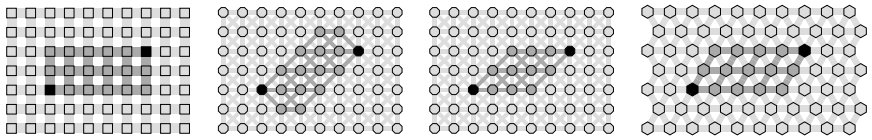
(a) (b) (c)

Figure 5.11: Example of many minimal isometric sets for one given set: with two points, it amounts to consider different shortest paths

in the convex hull. This ensures the minimality condition (see Fig. 5.10(b)). After a detection of the end of this stage, the second stage is launched, which corresponds to a slightly modified version of the solution described in the previous subsection. Thus, the eroded connected region is re-expanded into the 45-convex hull of the set of particles (see Fig. 5.10(c)).

The crucial part of that solution is the erosion stage. In fact, the eroded region corresponds to a minimal isometric set, such that between any two sites of the region, at least one shortest path is contained in the set. The problem is that there can be many minimal isometric sets for a given set of points, and the intersection of all of them can actually be disconnected. For example, a minimal isometric set for two sites is simply made of one shortest path joining them (see Fig. 5.11). For the example shown in Fig. 5.11, the intersection of all the shortest paths consists of the extremities and nothing else.

Therefore, the rule has to be defined in order to prevent different sites to make choices that are not coherent with the others sites when removing a site from the region. Otherwise, the eroded region can be non-connected. This is done in two stages, by indicating at first the wish list removal, and then by choosing a preferred direction to resolve conflicts.



(a) 4-square (b) 8-square (c) 8-square-$\sqrt{2}$ (d) hexagonal

Figure 5.12: Intervals for different grids

5.2.3 Comparison with our Solutions

Our approach tackles the problem at the full generality level defined in the problem statement. For the first solution presented in Sect. 5.2.1, we have a finer rule that follows directly from the convex hull definition. It does not need more states but is able, for example, to handle the cases of diagonally connected set. For the second approach presented in Sect. 5.2.2, we are able to construct the convex hull on infinite spaces without anything else than the particles at the initial state. Our cellular automaton converges locally: any bounded region of the space eventually converges. It is the best that can be done since there are signals lost in the infinity of the space, looking for other points arbitrarily far from its emission source. Also, we do not break any symmetries of the spaces by choosing a preferred direction in the rule. Everything only relies on distances, and is thus completely rotational invariant. Informally, we can summarize by simply saying that our approach is "more natural".

The rule applies directly to many bi-dimensional grids, including the ones listed in Fig. 2.1. It also works for the tridimensional counterpart of those. Last, it can easily be applied to bigger neighborhood, resulting in faster convergence when needed.

5.3 Solution Description

The main anchor of our approach is the fact that constructing the metric convex hull is in fact a matter of adding all missing shortest paths. Between two points x and y, the union of all paths joining them is called the interval. Figure 5.12 shows some example of intervals.

Definition 5.3.1. *An* interval $[x, y]$ *between two points* x *and* y *in a metric space* (M, d) *is the set of points* $\{ z \in M \mid d(x, z) + d(z, y) = d(x, y) \}$. *Points of* $[x, y]$ *are said to be* between x *and* y.

So let us define the operator $I(P) = \bigcup\{[x, y] \mid (x, y) \in P^2\}$ that adds the shortest paths joining a set of points. It is easy to see from Def. 5.1.4 that a set C is metric convex if and only if $I(C) = C$. Also, the convex hull of a set of points P is the limit of the sequence $I(P), I(I(P)), \ldots, I^n(P)$. Indeed the limits $L = I^n(P)$ verifies the convexity condition $I(L) = L$, the minimality condition being ensured by the fact that $I(\bullet)$ only adds points that have to be in the convex set.

In the following, we start off by considering the easiest detectable shortest paths: those that are entirely contained in the neighborhood of a site. This will

lead us to the definition of local metric convexity. We will also observe strong similarity with the majority rule described in Sect. 5.2.1 and explain their relation.

Our next step will be to consider the detection of arbitrarily long shortest paths. So given two arbitrarily distant particles, we will show how to detect the shortest paths joining them. We restrict the study to the hexagonal cellular space cases. In this setting, we will observe the behavior of the obtained rule when many particles are in the space. This will lead us to the definition of the new concept of metric Gabriel graph. From it, we will be able to generalize the results to all the considered cellular spaces and their counterpart of higher dimensionality.

5.3.1 Hull for Local Metric Convexity

5.3.1.1 The local metric convexity rule

In order to easily fit in the locality constraints of CA, we consider first the detection of paths entirely contained in the neighborhood of a site. If we consider a neighborhood of radius r, this amounts to say that we detect all shortest paths of length $2r$ or less. To complete a set P with those paths, each site lying on a shortest path between two marked sites of its neighborhood marks itself as expressed in the following rule:

$$\mathrm{conv}_{t+1}(x) = x \in P \vee \exists \{y_0, y_1\} \subset \mathrm{conv}_t \cap B(x,r) \backslash \{x\}; x \in [y_0, y_1] \qquad (5.2)$$

Let us mention that testing $x \in [y_0, y_1]$ with hop count metric and $r = 1$ is equivalent to testing $d(y_0, y_1) = 2$ since $x \in [y_0, y_1] \Leftrightarrow d(y_0, x) + d(x, y_1) = d(y_0, y_1)$ and $y \in B(x, 1) \backslash \{x\} \Leftrightarrow y \in N(x) \Leftrightarrow d(x, y) = 1$. For the $\{1, \sqrt{2}\}$ metric, the test has to be done explicitly, and the neighborhood radius r is $\sqrt{2}$ since we consider the 8 neighbors of each site.

The conv rule has roughly the same global behavior than the majority rule presented in Sect. 5.2.1 in the sense that it generally produces a set of partial convex hulls and that the convex hulls can merge to form bigger convex hulls during the evolution. This can be observed in the evolution shown in Fig. 5.13.

However, the conv rule is more precise than majo. Indeed, it detects strictly more neighborhoods than the majority rule. As an example, one can note that the initial configuration of Fig. 5.13 is a fixpoint for the majority rule. More importantly, the conv rule produces a global convex hull for diagonally connected set of particles, where majo can only deal with horizontally and vertically connected set of particles as depicted in Figs. 5.14 and 5.15.

5.3.1.2 Formal summary of conv properties

Let us summarize the main properties of conv in few precise and concise formal statements. First, let us define the convexity which is obtained by using this rule:

Definition 5.3.2. *The* local metric convexity *of radius r is the collection of subsets of the space that are locally metric convex at radius r.*

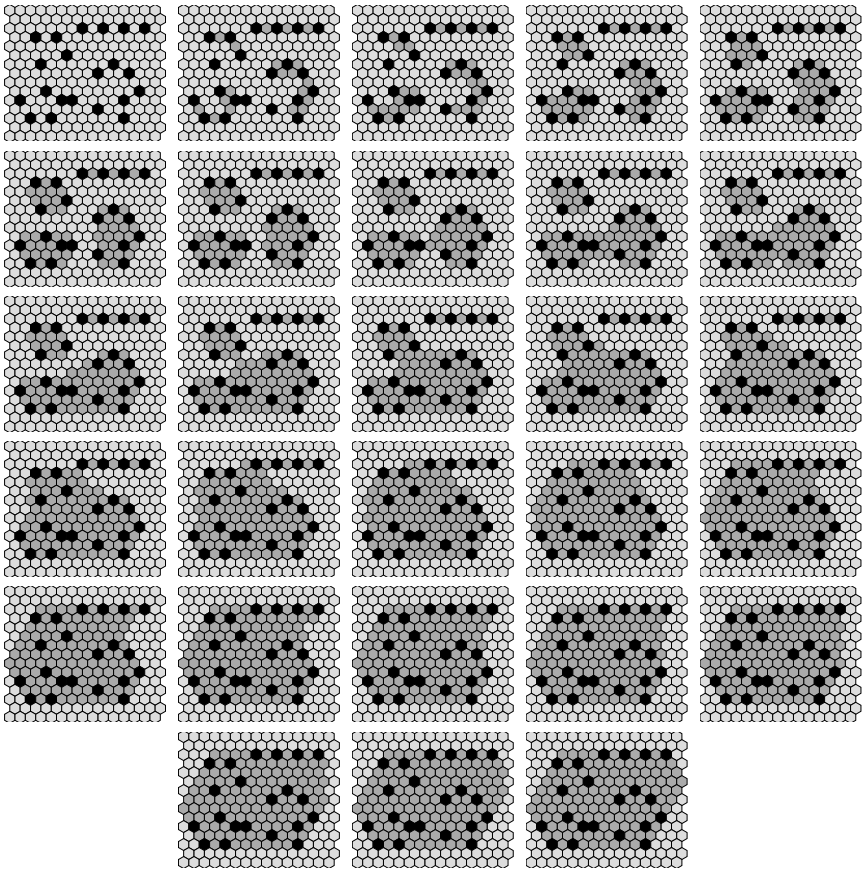


Figure 5.13: Evolution of the conv rule with neighborhood radius 1

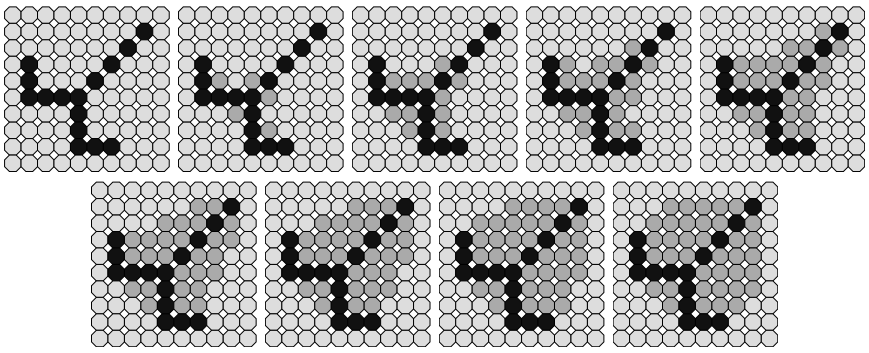


Figure 5.14: Evolution of conv on a diagonally-connected set of particle

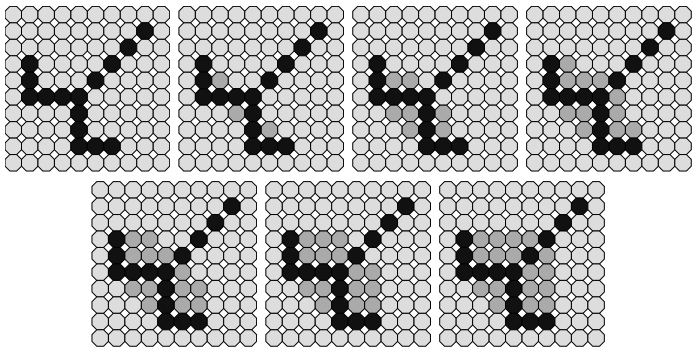


Figure 5.15: Evolution of majo on a diagonally-connected set of particle

Definition 5.3.3. *A subset is* locally metric convex *at radius r if for any two of its points x and y of distance $d(x, y) \leq r$, all shortest paths joining them are in the subset.*

Proposition 5.3.4. *The conv rule defined in Eq.* (5.2) *applied with a neighborhood radius r on an initial configuration corresponding to a set of particle P converges to the local metric convex hull at radius r of P.*

Now, let us identify formally the cases where conv produces the global metric convex hull. Informally, let us define r-connectedness as the property, for a set of particles, to be able to go from a particle to any other particle by jump from particle to particle, with jump size less or equal to r.

Proposition 5.3.5. *For any r-connected set of particles $P \subset S$, the local metric convex hull of radius r of P corresponds its metric convex hull.*

It is interesting to note that in terms of r-connectedness, we can simply say that the majo rule produces the $\{1, \sqrt{2}\}$ metric convex hull for 1-connected set of particles on 8-square cellular spaces.

5.3.1.3 Considering larger neighborhood radius

In all previous figures, we have used either a neighborhood radius of 1 or $\sqrt{2}$. Let us now consider larger neighborhood radius and look at the resulting evolution. An example with $r = 3$ is shown in Fig. 5.16. On the initial configuration, on the top, we can see pairs of particles separated of distances 6, and 5; and on the right, distances 4 and 2. After one transition, we see that some sites are marked at the middles of each pair.

More generally, at each transition, a path of length $2r$ whose extremities are marked will only have its middle marked, thus splitting itself into two shortest path of length r. More generally, any path of length $r < l \leq 2r$ will see its a centered segment of size $2r - l$ detected at its middle, thus splitting itself into two shorter segments. Paths of length $l \leq r$ will simply be entirely marked, since all of its points are able to see the both extremities in their neighborhood.

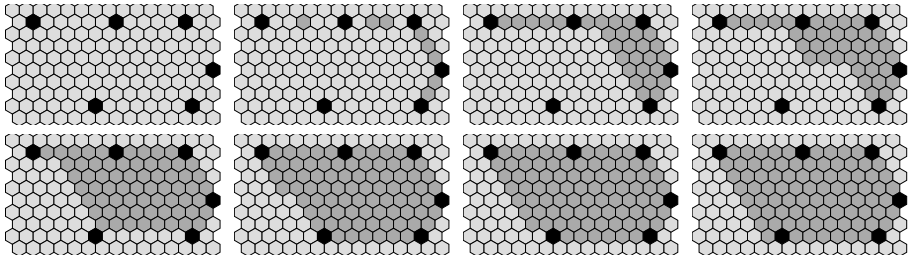


Figure 5.16: Evolution of the conv rule with neighborhood radius 3

5.3.2 Hull for Global Metric Convexity

Since we have reached the local convexity goal, involving only shortest paths of length $2r$ or less, let us now target shortest path of arbitrary length. Given two particles, the goal is then to draw the shortest paths linking them. In this setting, the goal is to select sites of the interval between the particles, as shown in Fig. 5.12.

5.3.2.1 Paths Joining Two Distance Particles

As for the previous case, we know that the middles of the paths are the first to have enough information to be marked. For a site z to know if it is at the middle between two points x and z, it needs to check $z \in [x, y]$ as before, but in the case $d(x, z) = d(y, z)$. Determining the value $d(x, z) = d(y, z)$ is easily achieved using a single distance field, as it has already been done in Chapt. 4. However, determining $z \in [x, y]$ is more tricky, since we need to check a last condition: $d(x, y) = 2d(x, z) = 2d(y, z)$. Our approach has been to use the gradient of the distance field to determine if the sites $\{x, y\}$ are at opposite directions. Before attacking the general case, we start by identifying what we need to detect on the particular case of the hexagonal cellular space.

In order to consider all the local distance field configurations corresponding to the middle of a shortest path on an hexagonal cellular space, we need to take into account all the ways two hexagons can touch each other. From this, we can construct a rule that simply mark a site if its neighborhood is the hand-made list of middle configurations. We do not gives the details since only the results really matter for the moment. Let us just consider one example.

Figure 5.17 show the evolution of a distance fields from two distant particles (see Chap. 3 for the rule and reason why we can use it even in finite state context). We can see at time 4 that there are particular local configurations that can only happen at the middles between two particles. All the other configurations that appear somewhere else in the rest of the evolution have to be discarded for the middle detection rule. Using this hand-made rule, we can obtain the evolution of Fig. 5.18 until time 5 where the middles appears.

The rest of the evolution is obtained easily. Let us consider any site y of the interval, except the middles. It has all required information since it knows its distance value $d(y, z)$ to the nearest particle z, the distance value of any

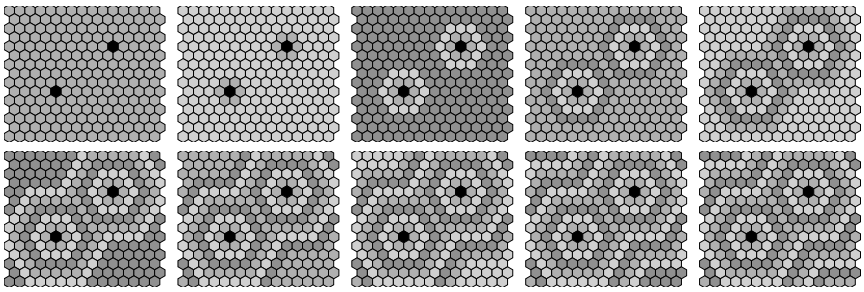


Figure 5.17: Distance fields from two particles

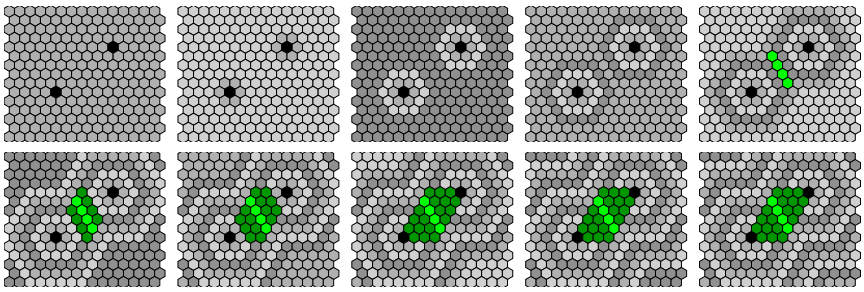


Figure 5.18: Detected middles and paths back to both particles

neighbor $d(x, z)$ to the same nearest particles, and the distance $d(x, y)$. It can thus determine whether it is on a shortest path joining x and z, i.e. $y \in [x, z]$, or not. If it is the case, and x is marked as being in the convex hull, y can also mark itself. In fact, we can see that this propagation simply corresponds to a movement from the middles back to the particles. Therefore, we can express it by opening the good edges for the traffic from the middles back to the particles (see Sect. 3.5).

$$\text{back}[f]_t(x, y) \;=\; d(x, y) + f_t(y) = f_t(x) \tag{5.3}$$

5.3.2.2 Pairwise Construction of the Hull ?

We now have a way to construct the convex hull of two particles using one distance field for the pair. When considering many particles, we cannot build many distance fields as we need the number of states to be constant. The question that naturally arises concerns the way each particle perturbs the distance field of the others. Indeed, we actually have a way to connect two particles and thus, we only need to connect them pairwise and locally while ensuring a global connectedness. If we can have a connected result, the conv rule described in Sect. 5.3.1 in parallel with the other rules will lead to the global convex hull.

Figure 5.19 shows the evolution of the pairwise convex hull described in the previous section (Sect. 5.3.2.1) in the presence of many particles. We can see that we obtain a connected set which looks like a proximity graph. Let us make the link between the considered algorithm (distance field, middle detec-

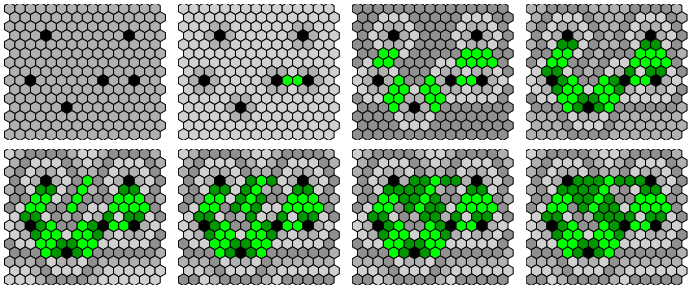


Figure 5.19: Construction of pairwise convex hull

tion, movement back to the particles) and the resulting connected set.

We have said that we detect the middles from the distance fields. When many particles are present, the detection of a middle may be prevented by the destroying of the information due to the presence of another particle nearby. Using the Voronoi diagram vocabulary, we can say that we can detect the middles between two particles only if it belongs to the Voronoi boundary between the Voronoi regions of the considered particles. When another particle is too close, a middle may fall in Voronoi region of that particle, thus preventing any detection.

So it seems that *two particles are connected by this proximity graph if there is a shortest path joining them included in their Voronoi regions.* This very interesting relation with the Voronoi diagram defines actually an existing proximity graph related to Delaunay graph, namely Gabriel graphs. This graph is connected, among other good properties, but is only defined for Euclidean spaces. Since the hand-made list of local configurations that only works for the hexagonal grid is not in the spirit of this work, the next logical step is to further study Gabriel graphs and to generalize them so that they can be drawn on any considered cellular spaces in a connected way, thus allowing our convex hull algorithm to solve the problem with full generality.

To summarize, let us write the system corresponding to the construction of the convex hull. If we denote as cent$[f]$ the field detecting the shortest path middles on the distance field f:

$$\begin{cases} \text{dp} &= D[P] \\ \text{m} &= \text{cent}[\text{dp}] \\ \text{g} &= M'[\text{m}, \text{back}[\text{dp}]] \\ \text{gconv} &= \text{conv}[g] \end{cases} \tag{5.4}$$

Chapter 6

Gabriel Graphs

Contents

Early version of this chapter has been published in [36].

Gabriel graphs are proximity graphs introduced in [19] to study sets of geographical points. They are now used in many domains such as wireless [28] and sensors networks for routing and communication management purpose. They also serve as tools to study proximity of points in order to cluster them, in domains like data mining, data and multivariate analysis [8], and machine learning [37, 12]. In computer graphics [39, 29], they can help to convert a set of points into a 3D surfaces and to obtain information about such surfaces. Their wide spread use is due to their numerous properties. Indeed, they are related to Voronoi diagrams, Delaunay graphs, planar graphs, minimum spanning trees,

nearest neighbor graphs, and represent or contain optimal solutions for some classes of energy-minimizing problem [28].

In the previous chapter, we have seen how the construction of the convex hull of a set of particles in a cellular space leads naturally to consider this proximity graphs. In this chapter, we describe a new generalization of Gabriel graphs that we call *metric Gabriel graphs*. The former carries properties of Gabriel graphs in any metric space, which allows to use them in the considered cellular spaces in particular. We also present the cellular automata that draws the metric Gabriel graphs.

6.1 Mathematical Problem Statement

In this section, we show that the original Gabriel graph does not accommodate the particularities of the cellular space. We highlight the fact that the original definition relies on many uniqueness properties of the Euclidean space, the uniqueness of the segment linking two points for example. Cellular spaces do not have these uniqueness properties, so a generalization of the original definition is needed.

6.1.1 Original Gabriel graphs

Gabriel graphs have originally been designed for Euclidean spaces in terms of angle [19]. Considering a set of points, its purpose is to connect by an edge those considered to be close. Due to properties of Euclidean spaces, many equivalent definitions have been used, one of them being:

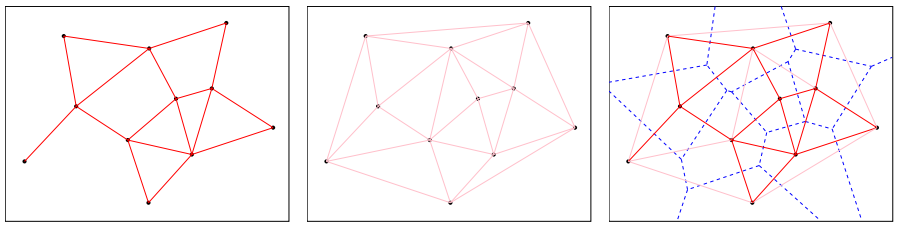
Definition 6.1.1. *The Gabriel graph $G_G = (V_G, E_G)$ of a set of particles P in a Euclidean space of arbitrary dimension $\mathbb{R}^n$ has $V_G = P$ as set of vertices, and an edge $\{x, y\} \in E_G$ between two particles x and y of P if and only if the ball using the segment $[xy]$ as diameter does not contain any other particle.*

This mathematical structure has many good properties and relationships with other well-known graphs. For instance, for a Euclidean space of any dimension, and for any set of particles, we have $G_{NN} \subseteq \text{MST} \subseteq G_G \subseteq G_D$ where G_{NN} is the nearest neighbor graph, MST is any minimum spanning tree and G_D is the Delaunay graph [1, 27].

The relationship between Gabriel and Delaunay graphs can be made more explicit by using Voronoi Diagrams [9, 5]. In fact, the edges of the Delaunay graph connect particles of adjacent Voronoi region, while the Gabriel graph retains only edges that do not pass through the Voronoi region of another particle (see Fig. 6.1).

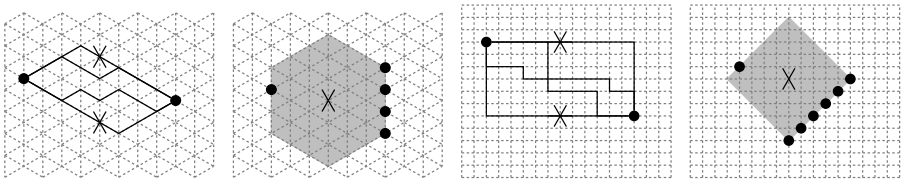
6.1.2 Relationship with general metric spaces

Despite the fact that Gabriel graphs are defined for Euclidean spaces, their definition only involves distances, so that a generalization to arbitrary metric spaces seems natural. However, there are significant differences between Euclidean and general metric spaces, which can break Gabriel graphs properties such as their



(a) Gabriel graph (b) Delaunay graph (c) Both on Voronoi Diagram

Figure 6.1: Gabriel and Delaunay graphs and Voronoi Diagram Relations



(a) Hexagonal cellular space (b) Hexagonal cellular space (c) 4-Square cellular space (d) 4-Square cellular space

Figure 6.2: Non-uniquenesses in hexagonal and square grids: (a,c) lines indicate possible shortest paths, and crosses indicate many possible centers for the balls, (b,d) the isolated point forms a diameter for the ball with any of the other points

connectedness as we will show.

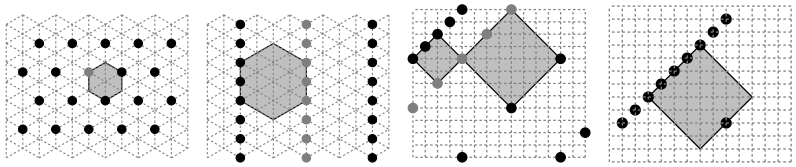
Indeed, the original definition mainly relies on the existence of enough: *ball using the segment* $[xy]$ *as diameter does not contain any other particles.* In an Euclidean space, for a given closed ball $B(c, r)$ and a given point x such that $d(c, x) = r$, there exists a unique point y such that the segment $[xy]$ is a diameter for that ball, i.e. $d(x, y) = 2r$.

In an arbitrary metric space, this uniqueness is lost as shown in Fig. 6.2(b,d). In the same vein, a pair of points is not a diameter for only one ball in an arbitrary metric space. Indeed, the uniqueness of this ball in Euclidean space is implied by the uniqueness of the shortest path linking two points. When many shortest paths exist between two points of a metric space, they can have different middles, which leads to many balls, all having the same radius, but each being centered at a different middle (see Fig. 6.2(a,c)).

Fig. 6.3 gives an example of loss of connectedness due to the fact that there may be no ball having only two particles. We therefore need to modify the definition to take into account these particularities in a meaningful way.

6.1.3 Pre-Existing generalizations

Generalizations of Gabriel graphs have already been proposed in different domains. For example, geodesic Gabriel graphs are a generalization proposed for non-linear manifolds [12] use in classification, generalized Gabriel graphs for



(a) Each point (b) Each line (c) Each diagonal (d) Subcase of (c)

Figure 6.3: Hexagonal and square grid: the Gabriel graph is not connected. Grey points show one of the connected components and gray balls give the reason of the non-connectedness

wireless ad hoc network [28] and elliptic Gabriel graphs for computer graphics [39, 29]. All these generalization consider a change in the shape of region considered for edges selection. The goal is not to conserve the Gabriel graph but rather to modify it to see if better results can be obtained. Moreover, the connectedness is not ensured, which is not a problem for domains like classification or computer graphics where it can even be a feature.

Our goal is to conserve the Gabriel. We look for a generalization which gives back the original Gabriel graph when applied to Euclidean spaces, but can also be used for non-Euclidean spaces while preserving all of its general properties. So we need a new generalization of Gabriel graphs.

6.2 Metric Gabriel Graphs

In order to find a definition that fits our need, and cope with non-uniquenesses, we redefine the Gabriel graph from basic principles. Indeed, it seems that its properties are consequences of three principles: connectedness, minimality, and locality. From them, we deduce a new definition characterizing the original Gabriel graphs, but applicable to any metric space.

6.2.1 Principles of Gabriel Graphs

Let us informally derive the consequences of the three principles that we have identified: connectedness, minimality, and locality.

Minimality and connectedness Let us start with a complete graph connecting all pairs of particles. We want to remove as many edges as possible in order to obtain minimality, while preserving the connectedness. So we consider *minimum spanning trees*.

Locality and connectedness While removing the edges, we don't want to arbitrarily choose an edge when many solutions are equivalent, otherwise there will be a need for a global coherence to achieve connectedness. So we consider now the *union of all* minimum spanning trees.

Locality and minimality For an edge to decide if it is part of a minimum spanning tree or not, it needs to take into account arbitrarily far particles to

ensure minimality of the global result. Instead, we want the edge to decide its status by examining only nearby particles. So we finally consider the union of all *local* minimum spanning trees.

These principles characterize Gabriel graphs in a general way. Especially, the locality principle informally explains the natural adequation with the cellular automata framework. We will now see that their direct applications to metric spaces define a new structure that generalizes Gabriel graphs into *metric Gabriel graphs*.

6.2.2 From Principles to Definition

In order to obtain a definition from the principles, the only missing piece is a precise definition of what are the "nearby particles of an edge", i.e. what does local mean. The original Gabriel graph definition actually considers the particles contained in the smallest ball $B(c, r)$ containing the edge $\{x, y\}$. With this last ingredient, we obtain the following definition:

Definition 6.2.1. *The metric Gabriel graph $G_{MG} = (V_{MG}, E_{MG})$ of a set of particles P has $V_{MG} = P$ as set of vertices and an edge $\{x, y\} \in E_{MG}$ between two particles x and y if and only if there is a ball $B(c, r)$ such that $d(x, y) = 2r$ and $\{x, y\}$ is an edge of a minimum spanning tree of $P \cap B(c, r)$.*

This definition may look complex, and in fact, we show latter that it can be simplified into the equivalent definition given in Prop. 6.2.8. However, it expresses directly the principles and allows convenient formal manipulations by the use of the cut and cycle properties of minimum spanning trees (See Appendix A.2). As for the original definition, the definition relies of the existence of some balls having special properties. Let us call them *metric Gabriel ball*:

Definition 6.2.2. *A ball $B(c, r)$ is* a metric Gabriel ball *for a set of particles P if and only if there is two particles x and y such that $d(x, y) = 2r$ and $\{x, y\}$ is an edge of a minimum spanning tree of $P \cap B(c, r)$.*

6.2.3 Preservation of the Properties

Proposition 6.2.3. *Metric Gabriel graphs are connected graphs.*

Proof. By contradiction, we suppose that G_{MG} is not connected. Let x and y be the two closest particles belonging to different connected components. Since we consider a complete length space, there is a shortest path joining x and y. Let c be the middle of this shortest path. Let us consider a ball $B(c, r)$ such that $d(x, y) = 2r$.

Since x and y belongs to different connected components, we already know that $\{x, y\}$ is not a metric Gabriel edge. This means that it is not an edge for any minimum spanning tree of $P \cap B(c, r)$ neither.

Let us consider the path $v_0, \ldots, v_{l-1}$ connecting $x = v_1$ and $y = v_{l-1}$ in a minimum spanning tree of of $P \cap B(c, r)$. Adding the edge $\{x, y\}$ to this path leads to a cycle. By the cycle property of minimum spanning trees, every edge

$\{v_i, v_{i+1}\}$ of the path is such that $d(v_i, v_{i+1}) < d(x, y)$.

Since x and y are the closest point belonging to different connected component, $d(v_i, v_{i+1}) < d(x, y)$ means that v_i and v_{i+1} belongs to the same connected component. By transitivity, all v_i belongs to the same connected component, even $v_0 = x$ and $v_{l-1} = y$, which is a contradiction. □

Proposition 6.2.4. *Metric Gabriel graphs contains all minimum spanning trees*

Proof. It is the union of all local minimum spanning trees. □

Proposition 6.2.5. *Metric Gabriel graphs are sub-graphs of the Delaunay graph made of the edges that do only pass through the Voronoi regions of its extremity particles.*

Proof. To prove this, we mainly need to consider the middle of a path joining two metric Gabriel neighbors x and y. This middle corresponds to the center of a metric Gabriel ball, so to prove that the path does not pass through any other Voronoi region, we need to prove that the middle is part of the Voronoi Diagram. This means that there is no particle closer to the center than x and y. This corresponds directly to the next proposition. □

Proposition 6.2.6. *If a ball $B(c, r)$ is a metric Gabriel ball for a set of particles P, then the radius r of this ball equals the distance $d(c, P)$. Thus, any metric Gabriel ball is entirely determined by its center.*

Proof. By contradiction, let us assume that the ball $B(c, r)$ is a metric Gabriel ball but $r \neq d(c, P)$. So we have two particles x and y, and a minimum spanning tree as given by Def. 6.2.2. If $r < d(c, P)$, then $P \cap B(c, r) = \emptyset$, which contradicts the existence of two particles. If $r > d(c, P)$, then there is a particle z of P such that $d(c, z) < r$. Let us consider the triangular cycle made of x, y, and z. By the cycle property of minimum spanning trees, the longest edge of the cycle belongs to no minimum spanning trees. But $d(x, z) \leq d(c, x) + d(c, z) < 2r$ and $d(y, z) \leq d(c, y) + d(c, z) < 2r$ while $d(x, y) = 2r$. This contradicts the existence of the minimum spanning trees implied by Def. 6.2.2. □

Proposition 6.2.7. *A ball $B(c, r)$ is* a metric Gabriel ball *for a set of particles P if and only if $P \cap B(c, r)$ can be partitioned into $\{P_0, P_1\}$ with $d(P_0, P_1) = 2r$.*

Proof. Firstly, by the cut property, the existence of a cut $\{P_0, P_1\}$ is equivalent to the existence of a minimum spanning tree containing an edge $(x, y) \in P_0 \times P_1$ such that $d(x, y) = d(P_0, P_1)$. Then, having $d(P_0, P_1) = 2r$ implies that $d(x, y) = 2r$, since $2r$ is the diameter of the ball $B(c, r)$. Those two facts prove the equivalence. □

This latter proposition can be used to give a definition of metric Gabriel graphs directly comparable with the original one.

Proposition 6.2.8. *The metric Gabriel graph $G_{MG} = (V_{MG}, E_{MG})$ of a set of particles P has $V_{MG} = P$ as set of vertices and an edge $\{x, y\} \in E_{MG}$ between two particles x and y of P if and only if there is a ball $B(c, r)$ such that there is a cut $\{P_0, P_1\}$ of $P \cap B(c, r)$ with $(x, y) \in P_0 \times P_1$ and $d(P_0, P_1) = 2r$.*

Compared to the original definition of Gabriel graphs, this definition deals with the non-uniqueness of diameters by replacing the requirement of having two points x and y such that $d(x, y) = 2r$ by the requirement of having two sets of points P_0 and P_1 such that $d(P_0, P_1) = 2r$. The non-uniqueness of balls for a given diameter is managed by requiring only the existence of at least one such ball. This means that whenever diameters and balls are unique, this definition is equivalent with the original one, as it is the case for Euclidean spaces.

Proposition 6.2.9. *For any Euclidean space, and any set of particles P of that space, the Gabriel graph and the metric Gabriel graph are identical.*

6.3 Metric Gabriel Graphs on Cellular Automata

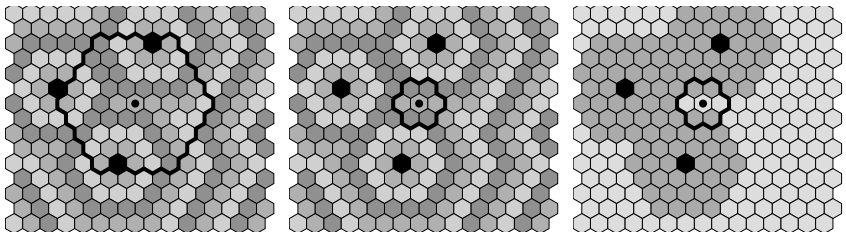
Now that we have completely defined mathematically the structure that we want to construct, we need to study the properties that will allow us to effectively construct it. In the previous chapter, we already considered its construction in fact. It consists of three fields. The first one is the distance field of the particles. The second field detects the middles of the shortests path joining neighboring particles. These middles correspond to the centers of the metric Gabriel balls. The third field draws the shortest paths from the metric Gabriel ball centers to their corresponding particles.

The main field is the one detecting the metric Gabriel ball centers using the values of the distance fields. From the definitions, we know that to determine if a site x is the center of a metric Gabriel ball of P, we need to examine $P \cap B(x, d(x, P))$. Let us consider a set P of three particles, their distance field, a site x, and its associated ball $B(x, d(x, P))$ as depicted in the first column of Fig. 6.4(a).

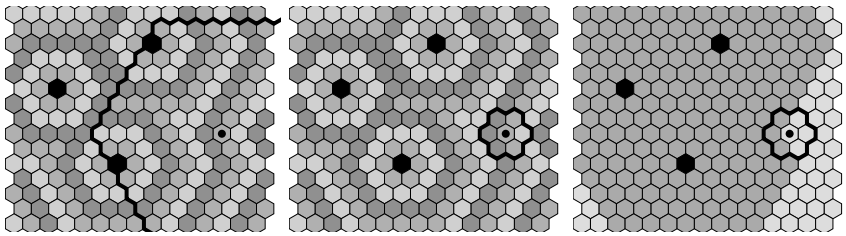
6.3.1 Distance fields and dilations

Examining the ball $B(x, d(x, P))$, we can see that the two particles contained in it have distance 8, which is twice the radius 4 of the ball. So this site is a metric Gabriel ball center. We wish to determine this by using the distance values present in the neighborhood $B(x, 1)$ of the site x. In the second column of the figure, we see that the lowest distance values $d(x, P) - 1$ present in the neighborhood can be partitioned in two parts of distance 2, which is twice the radius of the neighborhood. This means that $B(x, 1)$ is a metric Gabriel ball for the set $B(P, r') = \{ y \mid d(y, P) \leq r' \}$ with $r' = d(x, P) - 1$, as explicitly shown in the third column. Note that the set $B(P, r')$ does not look like a set of points. It is a large connected region obtained by *dilation* of the three particles.

In Fig. 6.4(b), the considered site x is not the center of any metric Gabriel ball. Indeed, the two particles contained in $B(x, d(x, P))$ are at distance 8, which is less than twice the radius 7 of the considered ball. Again, this is reflected in the neighborhood of the site, since the intersection between $B(P, d(x, P) - 1)$ and $B(x, 1)$ consists of three adjacent dark gray cells, and can not be partitioned in two parts of distance 2, as it was in the previous case.



(a) Example of a metric Gabriel ball center



(b) Example of a non-metric Gabriel ball center

Figure 6.4: Relation between distance fields, metric Gabriel balls, and dilations

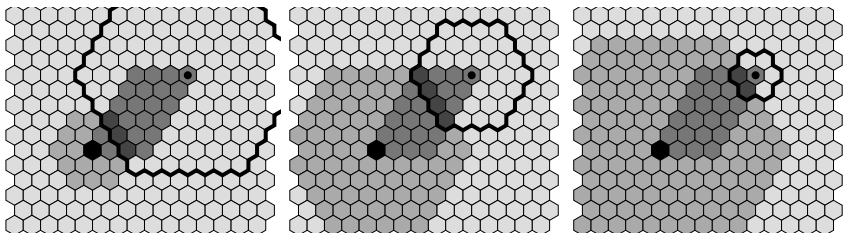


Figure 6.5: Intersections between dilations and neighborhoods are interval slices

Calling the sets $B(P, r')$ the *dilations* of P, we can see that there is a correspondence between the centers of the metric Gabriel ball of P, and the centers of the metric Gabriel ball of its dilations, at least for the hexagonal cellular space. Let us consider the general case.

6.3.2 Dilations and interval slices

In the rest of the chapter, we intensively work with the intersections between the dilations of P and the neighborhoods of a given point x, for different neighborhood radius, i.e. $B(P, r') \cap B(x, r)$. However, this latter has useful properties because of the particular way the radius r' and r are chosen, i.e. $r' = d(x, P) - r$. Indeed, this can be rewritten as $r' + r = d(x, P)$, which corresponds to the triangular equality, which characterizes points of shortest paths between x and P.

This is illustrated in Fig. 6.5 where such intersections are exposed for different neighborhood radius, with the intervals (Def. 5.3.1) depicted for comparison. In fact, the points of the intersection are the points of the shortest paths from x to P that are at distance r from x. We call this an *interval slice*.

Definition 6.3.1. *An* interval slice $[a, r; b]$ *corresponds to the set of points* $\{x \in$

$[a;b] \mid d(a,x) = r\} = \{x \mid d(a,x) = r \wedge d(x,b) = d(a,b) - r\}$.

Proposition 6.3.2. *The intersection between two balls $B(a, r_a)$ and $B(b, r_b)$ such that $r_a + r_b = d(a,b)$ is the interval slice $B(a, r_a) \cap B(b, r_b) = [a, r_a; b]$.*

Proof. To prove the equality between the sets $B(a, r_a) \cap B(b, r_b)$ and $[a, r_a; b]$, we prove the double inclusion. At first, the points of $[a, r_a; b]$ are at distance r_a from a, and at distance $d(a,b) - r_a = r_b$ to b, so they are clearly points of the intersection $B(a, r_a) \cap B(b, r_b)$. Next, any point x of $B(a, r_a) \cap B(b, r_b)$ has distances $d(x,a) \leq r_a$ and $d(x,b) \leq r_b$. But it is impossible that $d(x,a) < r_a$ nor $d(x,b) < r_b$, since it would violate the triangular inequality $d(x,a) + d(x,b) \geq d(a,b)$. Thus, $d(x,a) = r_a$ and $d(x,b) = r_b$, which means that x is a point of the interval $[a, r_a; b]$. □

6.3.3 Correspondence between interval slices

Let us summarize what we know about the correspondence between the original set of particle and its dilations. At first, we wish to study the relation for a site x between being a center of metric Gabriel ball for P[1], and being a center of metric Gabriel ball for a dilation $B(P, r')$. Using Prop. 6.2.7 and Prop. 6.2.6, it amounts to study the partitionability of $B(P,0) \cap B(x, d(x,P))$ and $B(P, r') \cap B(x, r)$ into two parts of distance $2d(x,P)$ and $2r$ respectively, with $r = d(x,P) - r'$. Finally, using the interval slices notation, we express this as the study of the partitionability of $[x, d(x,P); P]$ and $[x, r; P]$ into two parts of distance $2d(x,P)$ and $2r$.

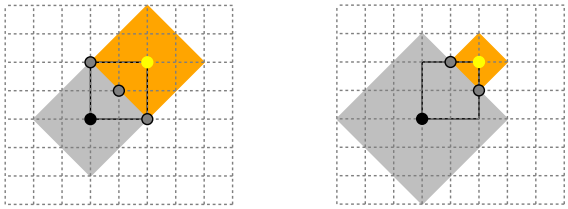
We now continue the reasonning by the following remark. Considering that the slice $[x, d(x,P); P]$ is partitionable into two parts $\{P_0, P_1\}$ of distance $d(P_0, P_1) = 2d(x,P)$ means that P_0 and P_1 are diametrically opposed in the ball $B(x, d(x,P))$. Informally, for any $r \leq d(x,P)$, the slice $[x, r; P]$ also admit diametrically opposed parts $\{Q_0, Q_1\}$, corresponding to the expremity of a subpart of the diameter $\{P_0, P_1\}$. Let us prove this.

Proposition 6.3.3. *Let P be any set of particles, x any point of the space, and $r \in [0, d(x,P)]$. If $[x, d(x,P); P]$ admits a partition $\{P_0, P_1\}$ such that $d(P_0, P_1) = 2d(x,P)$, then $[x, r; P]$ admits a partition $\{Q_0, Q_1\}$ such that $d(Q_0, Q_1) = 2r$.*

Proof. Considering each parts P_0 and P_1 separately, we construct $Q_0 = [x, r, P_0]$ and $Q_1 = [x, r, P_0]$. It is trivial that $Q_0 \cup Q_1 = [x, r; P]$, from $P_0 \cup P_1 = [x, d(x,P); P]$. Also, from $d(P_0, P_1) = 2d(x,P)$, and the triangular inequality $d(P_0, P_1) \leq d(P_0, Q_0) + d(Q_0, Q_1) + d(Q_1, P_1)$, we have that $2r \leq d(Q_0, Q_1)$. But it is not possible that $d(Q_0, Q_1) > 2r$ so $d(Q_0, Q_1) = 2r$. □

We would like to also have that if $[x, d(x,P); P]$ is not partitionable, then neither is $[x, r; P]$, in order to have an equivalence rather than an simple implication. Unfortunately, this is not true in general. Figure 6.6 gives an example in the 4-square cellular space, whith $[x, d(x,P); P]$ being a singleton, obviously not partitionable in two parts. It is shown that with $r = 2$, the points of the interval slice, at distance 2 from near to near, are not partitionable in parts of

[1]The set P can be considered as its own dilation with radius 0, i.e. $B(P, 0)$



(a) $r = 2$ (b) $r = 1$

Figure 6.6: In both case, the interval slice is 2-connected. With $r = 2$, there is partition since $2 < 2r$, but with $r = 1$, $2 = 2r$

distance $2r = 4$. But with $r = 1$, the points of the interval slice are still at distance 2, but this is now twice the radius as required for partitioning.

In fact, the "square holes" of the 4-square cellular space prevents the points of the interval slices to be closer than 2. The structure of the hexagonal and 8-square cellular spaces implies a bound of 1. Therefore, the neighborhood radius has to be strictly greater than 1 and 0.5 respectively. However, this only takes into account the case where $[x, d(x, P); P]$ is a singleton.

In order to study the many particles case, this is enough to work with two particles. Indeed, suppose that for any pair $\{p_0, p_1\}$ of a non-partitionable slice $[x, d(x, P); P]$, we have that the slice $[x, r; \{p_0, p_1\}]$ can neither be partitioned. It means that $[x, r; p_0]$ and $[x, r; p_1]$ can not be placed in different part of any partition of $[x, r; P]$. Therefore, $[x, r; P] = \bigcup_{p \in P} [x, r; p]$, can neither be partitioned.

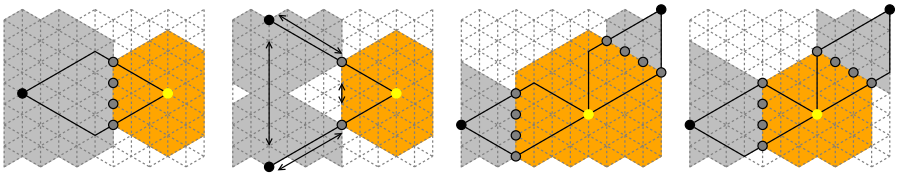
There is no particular problem with the two-particles cases. Figure 6.7 gives an example with the hexagonal cellular space. When considering two particles, we can see that each time we reduce the neighborhood radius r of a certain amount, the distance between interval slices $Q_0 = [x, r; p_0]$ and $Q_1 = [x, r; p_1]$ decrease at least by the same amount. So if $d(p_0, p_1) < 2d(x, P)$, then $d(Q_0, Q_1) < 2r$.

Proposition 6.3.4. *Let P be any set of particles, x any site of the **hexagonal or 8-square cellular space** such that $d(x, P) \geq r$ with $r = 1$. $[x, d(x, P); P]$ admits a partition $\{P_0, P_1\}$ such that $d(P_0, P_1) = 2d(x, P)$ if and only if $[x, r; P]$ admits a partition $\{Q_0, Q_1\}$ such that $d(Q_0, Q_1) = 2r$.*

Proposition 6.3.5. *Let P be any set of particles, x any site of the **4-square cellular space** such that $d(x, P) \geq r$ with $r = 2$. $[x, d(x, P); P]$ admits a partition $\{P_0, P_1\}$ such that $d(P_0, P_1) = 2d(x, P)$ if and only if $[x, r; P]$ admits a partition $\{Q_0, Q_1\}$ such that $d(Q_0, Q_1) = 2r$.*

6.3.4 Odd distances and edge-centered metric Gabriel balls

Until now, we have only considered integral radius. This means that we have only taken into account the pairs of particles of even distance $2r$. When the distance is odd, the center is, in fact, the middle of an edge. To be able to consider the edges middles, we complete the space with segments of unit length



(a) One particle (b) Two particles (c) $r = 4$ (d) $r = 3$

Figure 6.7: (a) A particle produces a non-partitionable slice (b) Two non separable particles produce a non-partitionable slice (c,d) Slightly more complex examples with 2 particles

between any pair of neighboring sites. Therefore, the distance of the middle of an an edge to its extremities is $\frac{1}{2}$.

Considering the edges does not introduce any particular problem, since most of the results applies to arbitrary metric space. Only the neighborhood radius needs to be adjusted for this special kind of metric Gabriel ball center. Since the neighborhood radius has to be strictly greater than 1 and 0.5 for hexagonal/8-square cellular space and 4-square cellular space respectively, this gives the same radius 1.5 for all these spaces.

Proposition 6.3.6. *Let P be any set of particles, x any edge* ***middle of the hexagonal, 4-square or 8-square cellular space*** *such that $d(x, P) \geq r$ with $r = 1.5$. $[x, d(x, P); P]$ admits a partition $\{P_0, P_1\}$ such that $d(P_0, P_1) = 2d(x, P)$ if and only if $[x, r; P]$ admits a partition $\{Q_0, Q_1\}$ such that $d(Q_0, Q_1) = 2r$.*

6.3.5 The metric Gabriel ball centers field

We now have all required information to detect the centers of metric Gabriel balls of a set of particles P from its distance field $D[P]$. We detect both the sites being center and the sites being an extremity of an edge center:

$$\mathrm{cent}_t(x) = \begin{cases} \bot \text{ if } t = 0 \\ \top \text{ if } \overline{\mathrm{cent}}_t(x, x) \\ \top \text{ if } \exists y \in N(x), \overline{\mathrm{cent}}_t(x, y) \\ \bot \text{ otherwise;} \end{cases}$$

Here, $\overline{\mathrm{cent}}_t(x, y)$ is defined as the function which determines whether the point xy at the middle between x and y is a center. If $x = y$, then we consider a site, and if $d(x, y) = 1$ then we consider an edge. According to our results, in both cases, we simply need to check whether $Q = [xy, r_{xy}; P]$ admits a partition or not. But if P is close enough, we don't need to dilate it. More precisely, close enough means here $r_{xy} = \overline{D}[P]_t(x, y)$. Otherwise, r_{xy} takes the appropriate value depending on the cellular space and on whether xy is a site or an edge, as stated in Props. 6.3.4, 6.3.5 and 6.3.6.

Since checking that xy is a metric Gabriel ball center amounts to check that Q can be partitioned into $\{Q_0, Q_1\}$ such $d(Q_0, Q_1) = 2r_{xy}$, we introduce the

closeness relation $C_{2r_{xy}}(y,z) \equiv d(y,z) < 2r_{xy}$. Indeed, anytime two sites y and z are at distance $d(y,z) < 2r_{xy}$, they cannot belong to different part of any considered partition. Therefore, $\{Q_0, Q_1\}$ exist only if there is at least two classes of separable sites in Q, i.e. $|Q/C^+_{2r_{xy}}| \geq 2$, where $C^+_{2r_{xy}}$ is the transitive closure of $C_{2r_{xy}}$. We obtain:

$$\begin{aligned} \overline{\text{cent}}_t(x,y) &= |\overline{Q}_t(x,y)/C^+_{2r_{xy}}| \geq 2 \\ \overline{Q}_t(x,y) &= \{\, z \in B(xy, r_{xy}) \mid D[P]_{t-1}(z) + r_{xy} = \overline{D}[P]_{t-1}(x,y) \,\} \end{aligned}$$

The last missing piece is to complete the definition of the distance field and balls to take into account the edges middles:

$$\begin{aligned} \overline{D}[P]_t(x,y) &= \min\{D[P]_t(x), D[P]_t(y)\} + \frac{d(x,y)}{2} \\ B(xy,r) &= B(x, r - \frac{d(x,y)}{2}) \cup B(y, r - \frac{d(x,y)}{2}) \end{aligned}$$

6.3.5.1 The resulting cellular automaton

To summarize, the complete cellular automata that draws the metric Gabriel graph of a set of particles P is the following, where back is defined at page 72.

$$\left\{ \begin{array}{rcl} \text{dp} &=& D[P] \\ \text{m} &=& \text{cent}[\text{dp}] \\ \text{g} &=& M'[\text{m}, \text{back}[\text{dp}]] \end{array} \right. \tag{6.1}$$

The evolution of this cellular is shown in Figs. 6.8, 6.9 and 6.10 for hexagonal, 4-connected square and 8-connected square grids. The cellular automaton uses 7 states, which correspond to 3 states for the distances modulo 3, multiplied by 2 states for the g field, plus 1 special state for the particles that always have $\text{dp}_t(x) = 0$ and $g_t(x) = \top$. Its neighborhood radius is 2 for the three considered grids, which can be calculated from Props. 6.3.4, 6.3.5 and 6.3.6.

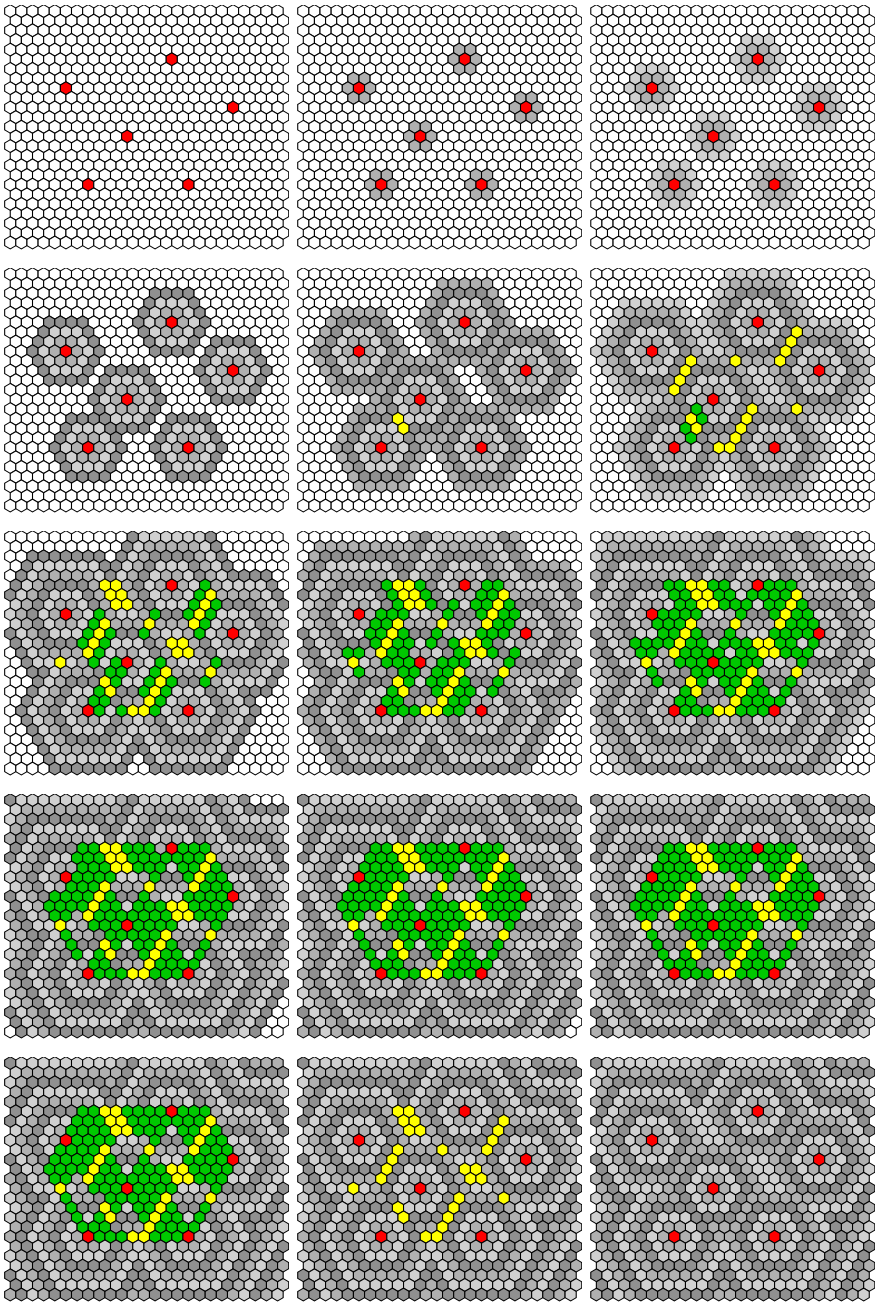


Figure 6.8: Evolution for the hexagonal cellular space. The two last snapshots show the final configuration with, firstly back hidden and then back and cent hidden. Red: particles, gray: D, yellow: cent, green: back

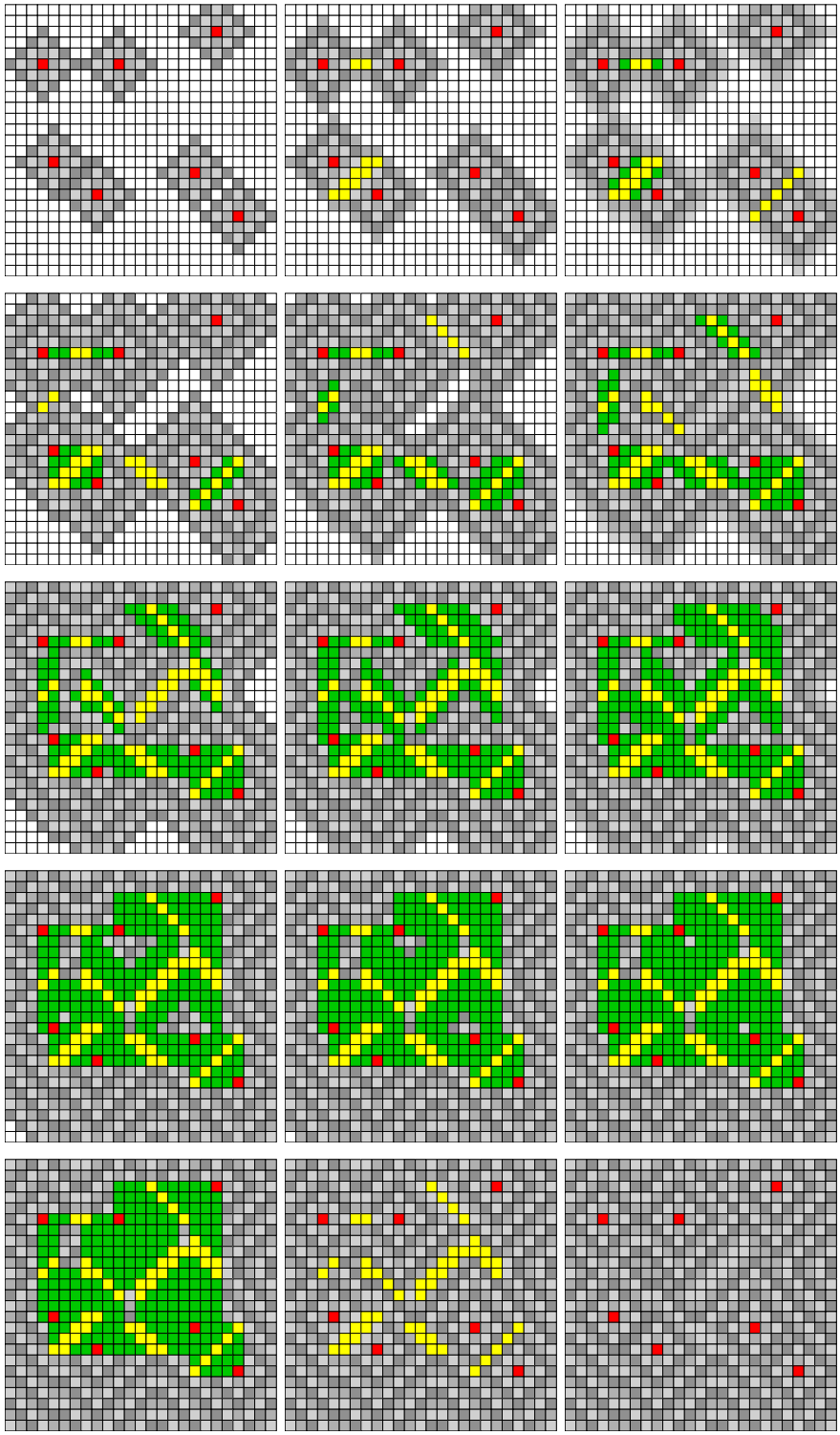


Figure 6.9: Evolution for the 4-square cellular space

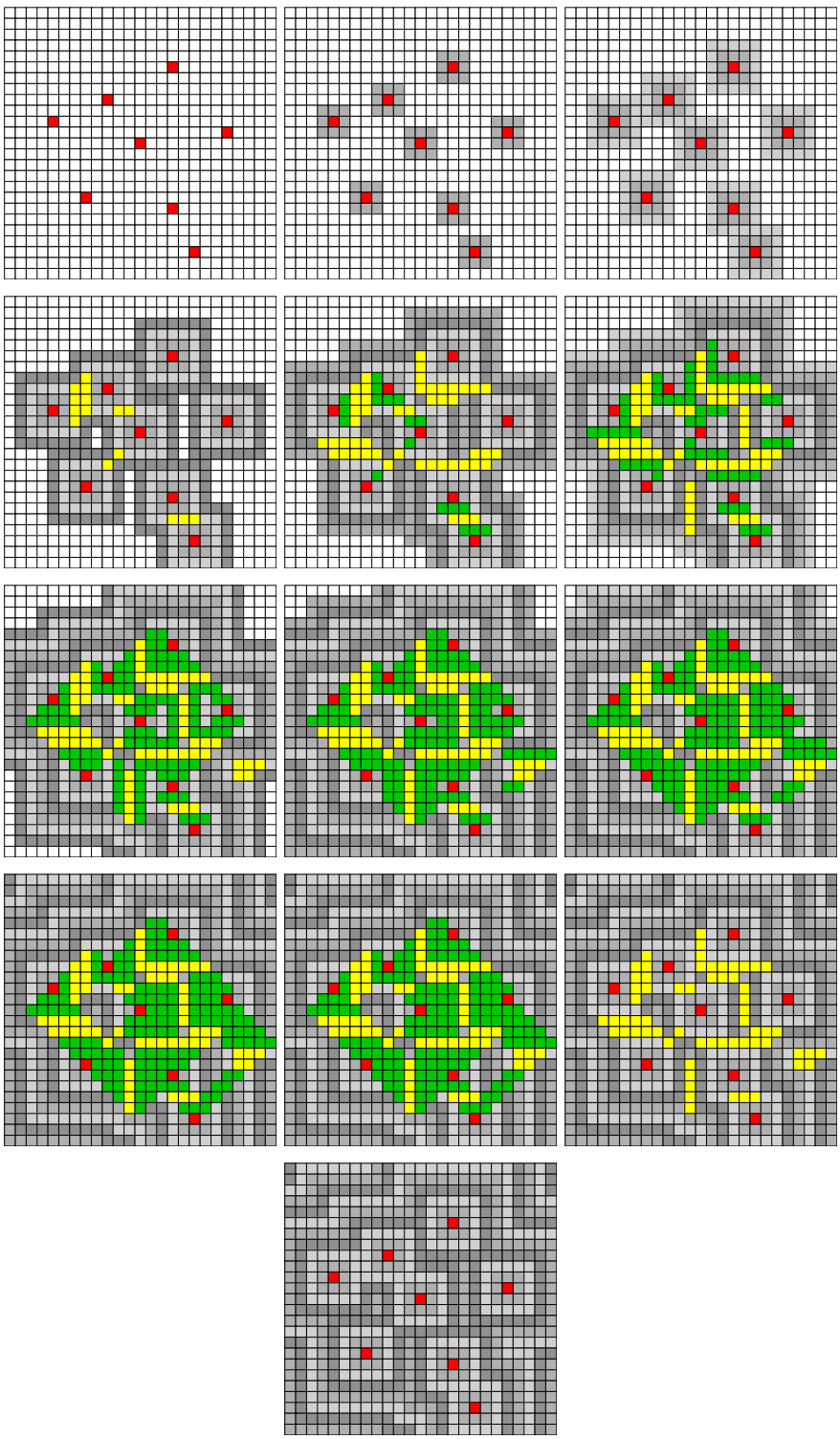


Figure 6.10: Evolution for the 8-square cellular space

Chapter 7

Conclusion and Perspectives

Contents

Through the three later chapters, we have shown how to use distance information to solve different problems: uniformizing of the distribution of a set of particles, constructing of their convex hull and constructing of the metric Gabriel graphs. The corresponding cellular automata rules have been provided, their formulation allowing them to be used with no changes on many cellular spaces, including 3D cellular spaces.

We now comment on the different remaining tasks that can be done for each of these problems and their building blocks, and with their interesting relation with other concepts. We also propose different directions in which the work and the framework can be extended.

7.1 Conclusion on the obtained results

7.1.1 About distance fields

The distance field computation is at the center of our approach, along with the property linking Lipschitz continuity with finite state, as described in Chapter 3.

The problems that we have considered using distance fields can also be translated into problems using wave propagation. In this case, the distance information is coded in time, which makes them usable only in synchronous systems. As an example, for a site to know that it is at the same distance from two particles, both particles need to emit a wave at the same time, and if the site receives both waves at the same time, it knows that the same distance has been traveled from both particles. Using distance fields, the distance information is encoded directly in the state which makes them more appropriate to use in asynchronous systems. This makes a link with other works in reaction-diffusion computers [4] where a solution for the construction of the Voronoi diagram is given using wave propagation.

Another interesting aspect is the link between our manipulation of the distance field as a set of dilation, as done in Chapter 6, and a mathematical tool known as Morse theory. The relation between Morse theory and Voronoi diagrams and Delaunay graphs has already been identified in Euclidean spaces. A comparison between our analysis and an appropriate discrete version of the Morse Theory needs to be done.

To comment on the relation between Lipschitz continuity and finite-state, this property is a general tool that can be used for other fields or in other contexts. Let us illustrate this with the example of a synchronization algorithm. The synchronization of an asynchronous system can be made by adding a time counter t to all elements of the system, and requiring an element to update itself to $t+1$ only if all of its neighboring elements have the value t or $t+1$ in their time counter. This construction builds a Lipschitz continuous field since the difference of counter values between two neighboring elements is at most 1. This implies that the counter can be represented with only 3 states. This is, of course, just one example and many other example can be found.

7.1.2 About density uniformisation

In Chapter 4, we have presented the density uniformization for the unidimensional cellular space. The generalization to multidimensional cellular spaces is still an active ongoing work, because of its central importance in the Blob Computing [24, 23]. The main difficulty is to obtain a perfect result of equal density as it is the case with unidimensional cellular spaces. The solution we have presented is based on a perfectly symmetric system, where two computations of middles are intertwined. In bi-dimensional spaces, this symmetry is lost since the particle are points, while the border between the regions of particles is made of lines. In fact, the bi-dimensional system needs to manipulate Voronoi diagrams, and having a uniform density is described by the property of this Voronoi diagram to be centroidal: having the particles at the center of mass of their Voronoi region. This is a whole subject that is out of the scope of this conclusion chapter. We therefore redirect the reader to the publications on the Centroidal Voronoi Tesselation for information about the non-cellular automata case.

7.1.3 About convex hulls and Gabriel graphs construction

In Chapter 5 and 6, the convex hull and Gabriel graph contruction have only be described for the case of static particles. This is in contrast with the dynamic results obtained for the density uniformization. However, we think that a transformation into a dynamic version should not be too difficult to obtain. Let us sketch the construction here.

In the system building the Gabriel graph, there is a clear order in the dependencies. The graph edges are build from the Gabriel centers. The centers are detected from the distance field, which is itself generated by the particles. Also, in the fields themselves, any distance value always depends on a lower value, and any edge site depends on a further edge sites. Using enough states, it is possible to allow each data to remember why it appeared, so that the disappearing of the cause leads to the disappearing of the effects. A general remark is that any structure built on top of the distance field can instantiate an directed acyclic graph of dependency by using the order relation between the distance values. For the complete convex hull solution, the *conv* rule is added to the Gabriel graph system. This rule is not build on top of the distance field. Therefore, a different order needs to be used.

7.2 Other applications of the framework

7.2.1 Voronoi Diagram Construction

Apart from the three examples tackled here, more structures can be computed in this framework and thus enjoy the same genericity and portability. We have already started to port previous work on the construction of dynamical Voronoi diagrams to this framework. In the same vein as the convex hull problem, previous work on this problem mainly relies on the manual identification of the neighborhood to detect [4]. Using our framework, this identification can be made formally and leads to a precise formulation of the rule. To illustrate, figure 7.1 shows the desired evolution for a static set of particles. The black sites correspond to particles, and gray ones to the part of the cellular spaces belonging to the Voronoi Diagram, i.e. being equidistant to at least two particles.

7.2.2 Firing Squad Synchronisation Problem

The problem studied in this document correspond to geometrical problems, and the Voronoi Diagram construction too. However, other kinds of problems, not directly expressed in terms of geometry, can actually be study in our framework. This is the case for the Firing Squad Synchronisation Problem. In this problem, all the sites are considered as quiescent, except one which start the process. The goal is to have all sites to switch to a fixed state at the same time, regarless of the size of the space, and only with a finite number of state. The difficulty arises because no site knows the size of space nor its distance to the initiating site. This difficulty is of geometrical nature and this synchronisation problem is actually solved using geometrical construction since all know solutions involve recursive division of space in two parts, generally of the same size. These divisions can thus be expressed in our framework, possibly providing a more precise analysis

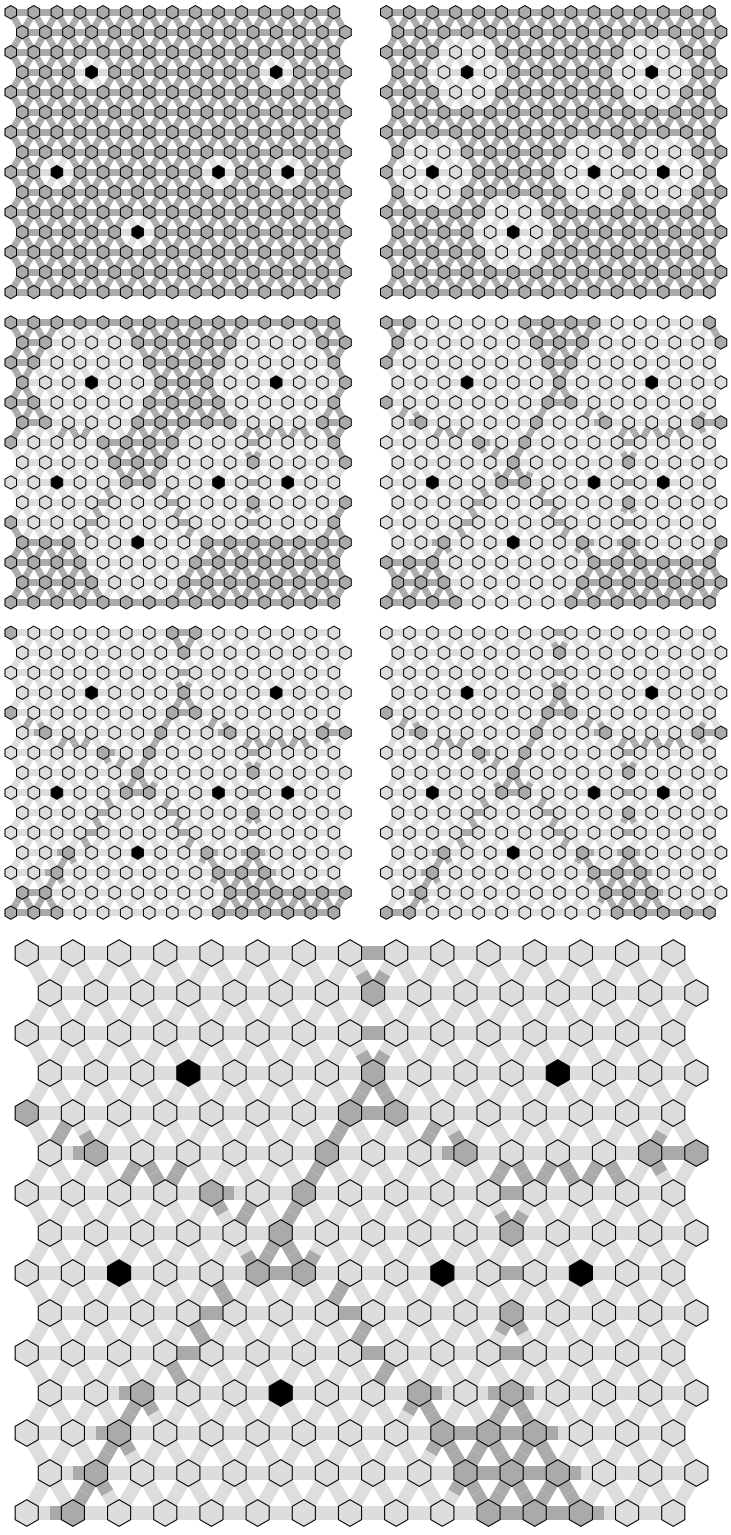


Figure 7.1: Voronoi Diagram at the edge level precision

than the existing ones. In particular, it seems that the recursive nature of the solution correspond to considering an infinite number of distance fields that can be reduced to a finite number of states. But all this is at a very early stage of thinking. This thinking has started in ACRI 2010 conference during a discussion between L. Maignan, J.-B. Yunes and F. Gruau about a new handmade solution for the 4-square cellular space by H. Umeo [47].

7.2.3 Extension of the framework itself

Another aspect that needs further study is about synchronicity of the system. Indeed, while our relation framework assumes full synchronous updates, most part of the presented fields do not rely very strongly on this property. This is because almost all informations are explicitly represented in the states of the sites. For example, the density uniformisation solution seems to be easily adaptable to asynchronous systems, only the movements needs to be corrected to not disappear/reappear at the same but rather appearing before disappearing, which is a more reliable mechanism. In fact, as shown in the corresponding chapter, asynchronous updates can even prevent the system to loop with two opposite signals that never meet, thus ensuring the eventual convergence for a constant size of the space.

The spaces on which the algorithms actually work without any adaptation is actually not entirely identified. Of course, it may be different for each problem. It can be interesting to further study other kind of cellular spaces such as Cayley graphs and other edge-transitive or vertex-transitive graphs. An identification of the regular spaces on which all the algorithms, maybe including a Voronoi diagram construction, may be a good addition to the framework. But of course, a further goal would be to not depend on an regularity anymore...

Appendix A

Mathematical Preliminaries

A.1 Metric Spaces

(From wikipedia) Let (M, d) be a metric space. We define a new metric d_I on M known as the induced intrinsic metric, as follows: $d_I(x, y)$ is the infimum of the lengths of all paths from x to y. If $d_I(x, y) = d(x, y)$ for all points x and y in M, we say that (M, d) is a length space or a path metric space and the metric d is intrinsic.

Definition A.1.1. *A metric $d : S^2 \to \mathbb{R}$ over a set S is a function such that:*

$$\forall (x, y) \in S^2; x = y \Leftrightarrow d(x, y) = 0 \tag{A.1}$$

$$\forall (x, y, z) \in S^3; d(x, y) \leq d(x, z) + d(z, y) \tag{A.2}$$

Definition A.1.2. *A metric space is a pair (S, d) where d is a metric over S.*

The metric is often used with set of points, in which case the former properties do not hold. Then it is defined as $d(X, Y) = \min\{d(x, y) \mid (x, y) \in X \times Y\}$. The notions defined in metric spaces that are used in this paper are: *closed balls*, *shortests paths*, and *intervals*. We also define what we call *dilations*.

A *closed ball* $B(c, r)$ is a generalisation of the concept of disc and sphere and is defined by its center c and its radius r as being the set of points $\{x \in M \mid d(c, x) \leq r\}$. Extending this definition by allowing the center to be a set of points, we define the *dilation* $B(P, r)$ of a set of points $P \subset M$ by the same formula $\{x \in M \mid d(P, x) \leq r\}$ or equivalently $\bigcup\{B(c, r) \mid c \in P\}$.

The *interval* $[x, y]$ between two points x and y as being the set of points $\{z \in M \mid d(x, z) + d(z, y) = d(x, y)\}$, i.e. the points being in a shortest path between x and y. One can note that for two points $\{x, y\} \subset M$, and $0 \leq r \leq d(x, y)$, the set $B(x, r) \cap B(y, d(x, y) - r)$ is the same as $\{z \in [x, y] \mid d(x, z) = r\}$.

A.2 Minimum Spanning Trees

We give here definitions for the specific cases of a set of points taken from a metric space, and the considered graph is the complete graph where the cost of

an edge is the distance between the points composing the edge.

Definition A.2.1. *The* minimum spanning tree *of a connected graph* (V, E_0) *having a cost associated to each of its edge is the connected subgraph* (V, E_1) *which is a tree such that the sum of its edges costs is minimal. Here, the cost is the length of the edge.*

Proposition A.2.2 (Cut Property)**.** *Let* $G(V, E)$ *be a connected graph, and* $\{v_0, v_1\} \in E$ *an edge of this this graph. The edge* $\{v_0, v_1\}$ *is part of a minimum spanning tree of* G *if and only if there is a partition* $\{V_0, V_1\}$ *of the set of vertices* V*, such* $(v_0, v_1) \in V_0 \times V_1$ *and* $d(V_0, V_1) = d(v_0, v_1)$*.*

Proposition A.2.3 (Cycle Property)**.** *Let* $G(V, E)$ *be a connected graph, and* $\{v_0, v_1\} \in E$ *an edge of this this graph. If the edge* $\{v_0, v_1\}$ *is the longest edges of a cycle* $v_0, v_1, \ldots, v_{l-1}, v_l = v_0$ *of a the graph* G*, then it is part of none of minimum spanning trees of* G*.*

Appendix B

Source Code

B.1 distance-field.lisp

```
(in-package #:symfield-lite)

(defun dist-init (src x) (if (funcall src x) 0 1))

(defun dist-iter (src dist x ti)
  (cond ((funcall src x ti) 0)
        ((funcall src x (1- ti)) 1/2)
        (t (apply #'min (mapcar (lambda (y) (+ (funcall dist y (1- ti))
                                               (distance x y)))
                                (neighbors x))))))

(defun dist-edge (dist x y ti)
  (let ((d (/ (distance x y) 2)))
    (min (+ (funcall dist x ti) d)
         (+ (funcall dist y ti) d))))
```

B.2 particle-movement.lisp

```
(in-package #:symfield-lite)

(defun move-iter (in open move x ti)
  (let ((nx (neighbors x)))
    (or (funcall in x ti)
        (some (lambda (y) (and (funcall move y (1- ti))
                               (funcall open y x (1- ti)))) nx)
        (and (funcall move x (1- ti))
             (or (not (some (lambda (y) (funcall open x y (1- ti))) nx))
                 (some (lambda (y) (and (funcall open x y (1- ti))
                                        (funcall open y x (1- ti)))) nx))))))

(defun dir-open (f cmp x y ti)
  (funcall cmp (funcall f x ti) (funcall f y ti)))

(defvar *K* 1000)

(defun bounded-open (d x y ti)
  (<= (- (funcall d y ti) (funcall d x ti)) (1- *K*)))

(defun back-open (d x y ti)
  (= (+ (funcall d y ti) (distance x y)) (funcall d x ti)))
```

B.3 density-uniformization.lisp

```
(in-package #:symfield-lite)

(defun move-far-iter (f dp p x ti)
  (move-iter (field () nil)
             (lambda (x y ti) (and (dir-open f #'<= x y ti)
                                   (bounded-open dp x y ti)))
             p x ti))

(defstruct density
  part ;; Particles
  wall ;; Voronoi regions boundaries
  dist-p  ;; Distances to the particles
  dist-w) ;; Distances to the Voronoi regions boundaries

(defun density-init (part0 x)
  (let ((wall0 (fnot part0)))
    (make-density :part (funcall part0 x)
                  :wall (funcall wall0 x)
                  :dist-p (dist-init part0 x)
                  :dist-w (dist-init wall0 x))))

(defun density-iter (density x ti)
  (struct-field-iter 'density density x ti
                     ((dist-p (dist-iter #'part #'dist-p x ti))
                      (dist-w (dist-iter #'wall #'dist-w x ti))
                      (part (move-far-iter #'dist-w #'dist-p #'part x ti))
                      (wall (move-far-iter #'dist-p #'dist-w #'wall x ti)))))
```

B.4 convex-hull.lisp

```
(in-package #:symfield-lite)

(defun local-convex-iter (in local-convex x ti)
  (or (funcall in x ti) (funcall local-convex x (1- ti))
      (some-pair (lambda (a b)
                   (and (= (distance a b) (+ (distance x a)
                                             (distance x b)))
                        (funcall local-convex a (1- ti))
                        (funcall local-convex b (1- ti))))
                 (neighbors x))))

(defstruct convex-hull
  part  ;; Generators
  dist  ;; Distances to the generator
  cent  ;; Metric gabriel ball centers
  back  ;; Shortest path from centers to generators
  conv) ;; Convex hull of back

(defun convex-hull-init (part0 x)
  (make-convex-hull :part (funcall part0 x)
                    :dist (dist-init part0 x)
                    :cent nil
                    :back nil
                    :conv (funcall part0 x)))

(defun convex-hull-iter (convex-hull x ti)
  (struct-field-iter 'convex-hull convex-hull x ti
                     ((part (part x (1- ti)))
                      (dist (dist-iter #'part #'dist x ti))
                      (cent (gabriel-center #'dist x ti))
                      (back (move-iter #'cent
                                       (lambda (x y ti)
                                         (back-open #'dist x y ti))
                                       #'back x ti))
                      (conv (local-convex-iter #'back #'conv x ti)))))
```

B.5 gabriel-graph.lisp

```
(in-package #:symfield-lite)

(defgeneric gabriel-radius (s x y))
(defmethod gabriel-radius ((s simplicial-space) x y)
  (assert (= (dim s) 2)) (if (eq x y) 1 3/2))
(defmethod gabriel-radius ((s l1-hypercubic-space) x y)
  (assert (= (dim s) 2)) (if (eq x y) 2 3/2))
(defmethod gabriel-radius ((s linf-hypercubic-space) x y)
  (assert (= (dim s) 2)) (if (eq x y) 1 3/2))

(defun gabriel-ball (dist cx cy r ti)
  (let ((dilation-r (- (dist-edge dist cx cy ti) r)))
    (<= 2 (count-connected-components
           (remove-if-not (lambda (x) (<= (funcall dist x ti) dilation-r))
                          (neighbors*-edge cx cy :r r))
           (lambda (x y) (< (distance x y) (* 2 r)))))))

(defun gabriel-center-edge (dist cx cy ti)
  (let ((nr (min (dist-edge dist cx cy ti)
                 (gabriel-radius *space* cx cy))))
       (gabriel-ball dist cx cy nr ti)))

(defun gabriel-center (dist x ti)
  (or (gabriel-center-edge dist x x ti)
      (some (lambda (y) (gabriel-center-edge dist x y (1- ti)))
            (neighbors x))))

(defstruct gabriel-graph
  part  ;; Generators
  dist  ;; Distances to the generator
  cent  ;; Metric gabriel ball centers
  back) ;; Shortest path from centers to generators

(defun gabriel-graph-init (part0 x)
  (make-gabriel-graph :part (funcall part0 x)
                      :dist (dist-init part0 x)
                      :cent nil
                      :back nil))

(defun gabriel-graph-iter (gabriel-graph x ti)
  (struct-field-iter 'gabriel-graph gabriel-graph x ti
                     ((part (part x (1- ti)))
                      (dist (dist-iter #'part #'dist x ti))
                      (back (move-iter #'cent (lambda (x y ti)
                                                (back-open #'dist x y ti))
                                       #'back x ti))
                      (cent (gabriel-center #'dist x ti)))))
```

B.6 spaces.lisp

```
(in-package #:symfield-lite)

;;*****************************************
;;* Metric and Group and Norm (lite version) *
;;*****************************************

(defclass normed-space () ((size :initarg :index-size) spheres balls radius))
(defgeneric dim (s))
(defgeneric zero (s))
(defgeneric add (s p0 p1))
(defgeneric sub (s p0 p1))
(defgeneric norm (s p))
(defgeneric sphere (s r))
(defgeneric ball (s r))
(defgeneric index-size (s))
(defgeneric to-index (s p))
(defgeneric of-index (s i))

;; All normed spaces of this lite version implements zero, add and sub
;; identically. To simplify things, we incorporate 'size slot that
```

```
;; corresponds to the size of the array to be used by fields, so that
;; the dimension of the space is the number of coordinate in the array
;; size.

(defmethod index-size ((s normed-space)) (slot-value s 'size))
(defmethod dim ((s normed-space)) (length (slot-value s 'size)))
(defmethod zero ((s normed-space)) (make-array (list (dim s))
                                               :initial-element 0))
(defmethod add ((s normed-space) p0 p1) (map 'vector #'+ p0 p1))
(defmethod sub ((s normed-space) p0 p1) (map 'vector #'- p0 p1))

;; In order to have fast access to the neighbors of a point, the
;; methods sphere and ball are "memoized".  The first call for a given
;; radius fill the cache and the following calls only read the cached
;; results.  The method 'unit-sphere returns the sphere of radius 1,
;; 'extend-radius fills the field 'spheres and 'balls when necessary.

(defgeneric extend-radius (s r))

(defmethod sphere ((s normed-space) r)
  (let ((r (floor r)))
    (extend-radius s r) (elt (slot-value s 'spheres) r)))
(defmethod ball ((s normed-space) r)
  (let ((r (floor r)))
    (extend-radius s r) (elt (slot-value s 'balls) r)))

;; The "extend-radius protocol" starts at the instance initialization,
;; where the fields 'spheres and 'balls are initialized.  The edges
;; are specific for each space, the radius extension process is
;; identical for all of space.

(defgeneric unit-sphere (s))

(defmethod initialize-instance :after ((s normed-space) &key)
  (let ((edges (unit-sphere s)))
    (setf (slot-value s 'spheres)
          (make-array '(2) :fill-pointer 2 :adjustable t :initial-contents
                      (list (vector (zero s))
                            (apply #'vector edges))))
    (setf (slot-value s 'balls)
          (make-array '(2) :fill-pointer 2 :adjustable t :initial-contents
                      (list (vector (zero s))
                            (apply #'vector (cons (zero s) edges)))))
    (setf (slot-value s 'radius) 1)))

(defun vector= (x y) (and (= (length x) (length y)) (every #'= x y)))

(defmethod extend-radius ((s normed-space) new-r)
  (let ((old-r (slot-value s 'radius)))
    (when (> new-r old-r)
      (let* ((edges (elt (slot-value s 'spheres) 1))
             (old-sphere (elt (slot-value s 'spheres) old-r))
             (new-sphere
              (apply #'vector (remove-duplicates
                               (loop for o across old-sphere append
                                     (loop for e across edges append
                                           (let ((n (add s o e)))
                                             (when (> (norm s n) old-r)
                                               (list n)))))
                               :test #'vector=)))
             (old-ball (elt (slot-value s 'balls) old-r))
             (new-ball (concatenate 'vector old-ball new-sphere)))
        (vector-push-extend new-sphere (slot-value s 'spheres))
        (vector-push-extend new-ball (slot-value s 'balls))
        (setf (slot-value s 'radius) (1+ (slot-value s 'radius)))
        (extend-radius s new-r)))))

;;*************
;;* Hypercube *
;;*************

(defclass hypercubic-space (normed-space) ())
(defmethod to-index ((s hypercubic-space) p) p)
(defmethod of-index ((s hypercubic-space) i) i)
```

```
;; L1-norm ;;

(defclass l1-hypercubic-space (hypercubic-space) ())

(defmethod unit-sphere ((s l1-hypercubic-space))
  (labels ((recurse (d v r)
             (if (zerop d) (when (zerop r) (list (apply #'vector v)))
                 (loop for i from (- r) to r append
                       (recurse (1- d) (cons i v) (- r (abs i)))))))
    (recurse (dim s) nil 1)))

(defmethod norm ((s l1-hypercubic-space) p)
  (reduce #'+ p :initial-value 0 :key #'abs))

;; Linf-norm ;;

(defclass linf-hypercubic-space (hypercubic-space) ())

(defmethod unit-sphere ((s linf-hypercubic-space))
  (labels ((recurse (d v n)
             (if (zerop d) (when (= n 1) (list (apply #'vector v)))
                 (loop for i from -1 to 1 append
                       (recurse (1- d) (cons i v) (max (abs i) n))))))
    (recurse (dim s) '() 0)))

(defmethod norm ((s linf-hypercubic-space) p)
  (reduce #'max p :initial-value 0 :key #'abs))

;;********************
;;* Simplicial grids *
;;********************

(defclass simplicial-space (normed-space) ())

(defun simplicial-vertices (dim)
  (loop for d from 0 to dim collect
       (apply #'vector (loop for i from 1 to dim collect
                             (cond ((< i d) 1/2) ((= i d) 1) (t 0))))))

(defmethod unit-sphere ((s simplicial-space))
  (let ((vertices (simplicial-vertices (dim s))))
    (loop for v0 in vertices append
         (loop for v1 in vertices when (not (eq v0 v1)) collect
              (sub s v1 v0)))))

(defmethod norm ((s simplicial-space) p)
  (multiple-value-bind (cp rp) (of-index s (to-index s p))
    (loop for i from 0 to (1- (dim s)) maximize
         (+ (abs (elt cp i)) (loop for j from (1+ i) to (1- (dim s)) sum
                                   (abs (* 1/2 (- (elt cp j) (elt rp j)))))))))

(defmethod to-index ((s simplicial-space) p) (map 'vector #'floor p))
(defmethod of-index ((s simplicial-space) i)
  (let ((p-res (list (elt i (1- (dim s))))) (p-ref '(0)))
    (loop for j from (- (dim s) 2) downto 0 do
         (let ((ref (+ (car p-ref) (* 1/2 (- (car p-res) (car p-ref))))))
           (setf p-res (cons (+ (elt i j) (mod ref 1)) p-res))
           (setf p-ref (cons ref p-ref))))
    (values (apply #'vector p-res)
            (apply #'vector p-ref))))
```

B.7 fields.lisp

```
(in-package #:symfield-lite)

;;;;;;;;;;;
;; Space ;;
;;;;;;;;;;;

(defvar *space*)
```

```
(defun set-space (space size)
  (setf *space* (make-instance space :index-size size)))

(defun iter-space (f &key (s *space*))
  (array-iter (lambda (x) (funcall f (of-index s x)))
              (index-size s)))

(defun inbound (p &key (s *space*))
    (every (lambda (x s) (and (>= x 0) (< x s)))
           (to-index s p) (index-size s)))

(defun distance  (p0 p1 &key (s *space*))
  (norm s (sub s p0 p1)))

(defun neighbors  (p &key (r 1) (s *space*))
  (reduce (lambda (r n) (if (inbound n) (cons n r) r))
          (sphere s r) :key (lambda (d) (add s p d))
          :initial-value ()))

(defun neighbors* (p &key (r 1) (s *space*))
  (reduce (lambda (r n) (if (inbound n) (cons n r) r))
          (ball s r) :key (lambda (d) (add s p d))
          :initial-value ()))

(defun distance-edge (x0 y0 x1 y1 &key (s *space*))
  (let ((*space* s))
    (let ((dx0 (/ (distance x0 y0) 2)) (dx1 (/ (distance x1 y1) 2)))
      (+ dx0 dx1 (min (distance x0 x1) (distance x0 y1)
                      (distance y0 x1) (distance y0 y1))))))

(defun neighbors-edge (x y &key (r 1/2) (s *space*))
  (let ((*space* s) (er (- r (/ (distance x y) 2))))
    (union (remove-if (lambda (n) (< (distance y n) er)) (neighbors x :r er))
           (remove-if (lambda (n) (< (distance x n) er)) (neighbors y :r er))
           :test #'vector=)))

(defun neighbors*-edge (x y &key (r 1/2) (s *space*))
  (let ((*space* s) (er (- r (/ (distance x y) 2))))
    (union (neighbors* x :r er) (neighbors* y :r er) :test #'vector=)))

;;;;;;;;;;;;
;; Fields ;;
;;;;;;;;;;;;

(defstruct field space curt memo field-iter time-radius)

(defun mkfield (field-init field-iter &key (space *space*) (time-radius 2))
  (let* ((*space* space)
         (size (index-size space))
         (memo (make-array (cons time-radius size))))
    (array-iter (lambda (x) (setf (apply #'aref memo 0 (coerce x 'list))
                                  (funcall field-init (of-index space x))))
                size)
    (make-field :space space :curt 0 :memo memo
                :field-iter field-iter :time-radius time-radius)))

(define-condition access-to-forgotten-past (error)
  ((field :initarg :field)
   (asked-time :initarg :asked-time)
   (current-time :initarg :current-time)
   (time-radius :initarg :time-radius)))

(defun fieldcall (field x ti)
  (with-slots (curt memo space field-iter time-radius) field
    (cond
      ((<= ti (- curt time-radius))
       (error 'access-to-forgotten-past :field field :asked-time ti
              :current-time curt :time-radius time-radius))
      ((<= ti curt)
       (apply #'aref memo (mod ti time-radius)
              (coerce (to-index space x) 'list)))
      (t (setf curt (1+ curt))
         (let ((*space* space) (offs (mod curt time-radius))
```

```
              (global-rule (lambda (x ti) (fieldcall field x ti))))
           (array-iter (lambda (x)
                         (setf (apply #'aref memo offs (coerce x 'list))
                               (funcall field-iter global-rule
                                        (of-index space x) curt)))
                       (cdr (array-dimensions memo))))
         (fieldcall field x ti)))))

(defun fnot (f) (lambda (x) (not (funcall f x))))
(defmacro field ((&key (x (gensym "x") xp) (ti (gensym "t") tip))
                 &body body)
  `(lambda (,x ,ti)
     ,@(when (not xp) `((declare (ignore ,x))))
     ,@(when (not tip) `((declare (ignore ,ti))))
     ,@body))

(defmacro struct-field-iter (struct sfield x ti system)
  (let* ((names (mapcar #'car system))
         (codes (mapcar #'cdr system))
         (result (gensym)) (nti (gensym))
         (updateds (mapcar (lambda (n) (gensym (symbol-name n))) names))
         (cycles (mapcar (lambda (n) (gensym (symbol-name n))) names)))
    `(let ((,result (make-instance ,struct))
           ,@(mapcar (lambda (x) `(,x nil)) updateds)
           ,@(mapcar (lambda (x) `(,x nil)) cycles))
       (labels ,(mapcar (lambda (name code updated cycle)
                          `(,name (,x ,nti)
                             (cond ((not (= ,nti ,ti))
                                    (slot-value (funcall ,sfield ,x ,nti)
                                                ',name))
                                   (,updated (slot-value ,result ',name))
                                   (,cycle (error "Cycle detected"))
                                   (t (setf ,cycle t)
                                      (setf (slot-value ,result ',name)
                                            (progn ,@code))
                                      (setf ,updated t)
                                      (slot-value ,result ',name)))))
                        names codes updateds cycles)
         ,@(mapcar (lambda (fn) `(,fn ,x ,ti)) names)
         (values ,result)))))
```

B.8 example.lisp

```
(in-package #:symfield-lite)

(defun test-density-uniformization (part0 max-time)
  (labels ((part0 (x) (eq 1 (elt part0 (elt x 0))))
           (dens-init (x) (density-init #'part0 x)))
    (set-space 'l1-hypercubic-space (list (length part0)))
    (let ((dens-field (mkfield #'dens-init #'density-iter)))
      (loop for ti from 0 to max-time do
           (iter-space (lambda (x)
                         (let ((res (fieldcall dens-field x ti)))
                           (format t
                                   (cond ((density-part res) "==")
                                         ((density-wall res) "%%")
                                         (t "  "))))))
           (format t "~%")))))

(defun test-local-convex-hull (part0 max-time)
  (labels ((part0 (x) (eq 1 (elt (elt part0 (elt x 1)) (elt x 0))))
           (conv-iter (conv x ti) (local-convex-iter (field () nil) conv x ti)))
    (let ((h (length part0))
          (w (length (car part0))))
      (set-space 'l1-hypercubic-space (list w h))
      (let ((conv-field (mkfield #'part0 #'conv-iter)))
        (loop for ti from 0 to max-time do
             (loop for y from 0 to (1- h) do
                  (loop for x from 0 to (1- w) do
                       (let ((res (fieldcall conv-field (list x y) ti)))
                         (format t "~a " (if res 1 0))))
```

```
                (format t "~%"))
            (format t "~%"))))))

(defun test-gabriel-graph (part0 max-time)
  (labels ((part0 (x) (eq 1 (elt (elt part0 (elt x 1)) (elt x 0))))
           (graph-init (x) (gabriel-graph-init #'part0 x))
           (graph-iter (graph x ti) (gabriel-graph-iter graph x ti)))
    (let ((h (length part0))
          (w (length (car part0))))
      (set-space 'l1-hypercubic-space (list w h))
      (let ((graph-field (mkfield #'graph-init #'graph-iter)))
        (loop for ti from 0 to max-time do
              (loop for y from 0 to (1- h) do
                    (loop for x from 0 to (1- w) do
                          (let ((res (fieldcall graph-field (list x y) ti)))
                            (format t (cond
                                        ((gabriel-graph-part res) "X ")
                                        ((gabriel-graph-cent res) "C ")
                                        ((gabriel-graph-back res) "B ")
                                        (t (case (mod (gabriel-graph-dist res) 3)
                                             (0 "! ") (1 ". ")
                                             (2 ": ") (t "  "))))))))
                    (format t "~%"))
              (format t "~%"))))))

(test-density-uniformization
 '(0 1 0 1 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0)
 210)

(test-local-convex-hull
 '((0 0 0 0 0 0 0 0)
   (0 1 0 0 0 0 1 0)
   (0 0 0 0 0 0 0 0)
   (0 0 0 0 1 0 0 0)
   (0 0 0 1 0 0 1 0)
   (0 0 0 0 0 0 0 0)
   (0 1 0 0 0 0 0 0)
   (0 0 0 0 0 0 0 0))
 21)

(test-gabriel-graph
 '((0 0 0 0 0 0 0 0 0 0 0 0)
   (0 0 0 0 0 0 0 0 0 0 0 0)
   (0 0 0 0 0 1 0 0 0 0 0 0)
   (0 0 0 0 0 0 0 0 0 0 0 0)
   (0 0 0 0 0 0 0 0 0 0 0 0)
   (0 1 0 0 0 0 0 0 0 1 0 0)
   (0 0 0 0 0 0 0 0 0 0 0 0)
   (0 0 0 0 0 0 0 0 0 0 0 0)
   (0 0 0 0 0 0 0 0 0 0 0 0)
   (0 0 0 0 0 0 0 1 0 0 0 0)
   (0 1 0 0 0 0 0 0 0 0 0 0)
   (0 0 0 0 0 0 0 0 0 0 0 0))
 8)
```

Bibliography

[1] *Handbook of Graph Theory (Discrete Mathematics and Its Applications)*. CRC, 1 edition, December 2003.

[2] F. Gruau A. Dehon, J.-L. Giavitto, editor. *Computing Media and Languages for Space-Oriented Computation 2006*, Dagstuhl international workshop 06361, 2006.

[3] Harold Abelson, Don Allen, Daniel Coore, Chris Hanson, George Homsy, Jr. Thomas F. Knight, Radhika Nagpal, Erik Rauch, Gerald Jay Sussman, and Ron Weiss. Amorphous computing. *Commun. ACM*, 43(5):74–82, 2000.

[4] A. Adamatzky, B. De Lacy Costello, and T. Asai. *Reaction-Diffusion Computers*. Elsevier Science Inc., New York, NY, USA, 2005.

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

[6] Andrew Adamatzky. *Computing in nonlinear media and automata collectives*. IOP Publishing Ltd., Bristol, UK, UK, 2001.

[7] Leonard M. Adleman. Computing with DNA. *Scientific American*, 279(2):54–61, August 1998.

[8] Michaël Aupetit and Thibaud Catz. High-dimensional labeled data analysis with topology representing graphs. *Neurocomputing*, 63:139–169, 2005.

[9] Franz Aurenhammer. Voronoi diagrams - a survey of a fundamental geometric data structure. *ACM Comput. Surv.*, 23(3):345–405, 1991.

[10] Jonathan Bachrach, Jacob Beal, Joshua Horowitz, and Dany Qumsiyeh. Empirical characterization of discretization error in gradient-based algorithms. In *SASO '08: Proceedings of the 2008 Second IEEE International Conference on Self-Adaptive and Self-Organizing Systems*, pages 203–212, Washington, DC, USA, 2008. IEEE Computer Society.

[11] J. Banatre and D. Le Metayer. Programming by multiset transformation. *Commun. ACM*, 36(1):98–111, 1993.

[12] Huajie Chen and Wei Wei. Geodesic gabriel graph based supervised nonlinear manifold learning. In *Intelligent Computing in Signal Processing and Pattern Recognition*, volume 345 of *Lecture Notes in Control and Information Sciences*, pages 882–887. Springer Berlin / Heidelberg, 2006.

[13] Bastien Chopard and Michel Droz. *Cellular Automata Modeling of Physical Systems*. Cambridge University Press, 1998.

[14] A. G. Clarridge and K. Salomaa. An improved cellular automata based algorithm for the 45-convex hull problem. *Journal of Cellular Automata*, 2008.

[15] D. Coore. *Botanical computing: a developmental approach to generating interconnect topologies on an amorphous computer*. PhD thesis, MIT, 1999.

[16] Peter J. Denning. The locality principle. *Commun. ACM*, 48:19–24, July 2005.

[17] A. K. Dewdney. The cellular automata programs that create wireworld, rugworld and other diversions. *Computer Recreations, Scientific American*, pages 146 – 149, January 1990.

[18] U. Frisch, B. Hasslacher, and Y. Pomeau. Lattice-gas automata for the navier-stokes equation. *Phys. Rev. Lett.*, 56(14):1505–1508, Apr 1986.

[19] Ruben K. Gabriel and Robert R. Sokal. A new statistical approach to geographic variation analysis. *Systematic Zoology*, 18(3):259–278, September 1969.

[20] Jean-Louis Giavitto. Topological collections, transformations and their application to the modeling and the simulation of dynamical systems. In *Rewriting Techniques and Applications*, pages 208–233, 2003.

[21] Jean-Louis Giavitto and Olivier Michel. MGS: a programming language for the transformations of collections. Technical Report 61-2001, LaMI, Universite d'Evry, 2001.

[22] B. Gojman, E. Rachlin, and J. E. Savage. Evaluation of design strategies for stochastically assembled nanoarray memories. *J. Emerg. Technol. Comput. Syst.*, 1(2):73–108, 2005.

[23] Frederic Gruau, Christine Eisenbeis, and Luidnel Maignan. The foundation of self-developing blob machines for spatial computing. *Physica D*, 2008.

[24] Frederic Gruau, Yves Lhuillier, Philippe Reitz, and Olivier Temam. Blob computing. In *Computing Frontiers 2004 ACM SIGMicro.*, 2004.

[25] J. Hardy, O. de Pazzis, and Y. Pomeau. Molecular dynamics of a classical lattice gas: Transport properties and time correlation functions. *pra*, 13:1949–1961, May 1976.

[26] Andrew Ilachinski. *Cellular Automata: A Discrete Universe*. World Scientific Publishing Co., Inc., River Edge, NJ, USA, 2001.

[27] Jerzy W. Jaromczyk and Godfried T. Toussaint. Relative neighborhood graphs and their relatives. In *Proc. IEEE*, pages 1502–1517, 1992.

[28] Iyad A. Kanj, Ljubomir Perković, and Ge Xia. Local construction of near-optimal power spanners for wireless ad hoc networks. *IEEE Transactions on Mobile Computing*, 8(4):460–474, 2009.

[29] Changhee Lee, Donguk Kim, Hayong Shin, and Deok-Soo Kim. Efficient computation of elliptic gabriel graph. In *ICCSA (1)*, pages 440–448, 2006.

[30] Thomas Lengauer. VLSI theory. In *Handbook of theoretical computer science (vol. A): algorithms and complexity*, pages 835–866. MIT Press, Cambridge, MA, USA, 1990.

[31] F.W. Levi. On helly's theorem and the axioms of convexity. *J. of Indian Math. Soc.*, pages 65–76, 1951.

[32] Luidnel Maignan. Uniformisation de la répartition spatiale d'un ensemble de points par diagramme de Voronoi: Implémentation sur automate cellulaire, 2007.

[33] Luidnel Maignan and Frederic Gruau. A 1D cellular automata that moves particles until regular spatial placement. 2008.

[34] Luidnel Maignan and Frederic Gruau. Integer gradient for cellular automata: Principle and examples. In *Spatial Computing Workshop at IEEE SASO 2008*, 2008.

[35] Luidnel Maignan and Frédéric Gruau. Convex hulls on cellular automata. In Stefania Bandini, Sara Manzoni, Hiroshi Umeo, and Giuseppe Vizzari, editors, *Cellular Automata*, volume 6350 of *Lecture Notes in Computer Science*, pages 69–78. Springer, 2010.

[36] Luidnel Maignan and Frédéric Gruau. Gabriel graphs in arbitrary metric space and their cellular automaton for many grids. *ACM Trans. Auton. Adapt. Syst.*, 2010, In Press.

[37] Kaustav Mukherjee. Application of the gabriel graph to instance based learning algorithms. Master's thesis, SFU CS School, aug 2004.

[38] R. Nagpal. *Programmable self-assembly: constructing global shape using biologically-inspired local interactions and origami mathematics*. PhD thesis, MIT, 2001.

[39] Joon C. Park, Hayong Shin, and Byoung Kyu Choi. Elliptic gabriel graph for finding neighbors in a point set and its application to normal vector estimation. *Computer-Aided Design*, 38(6):619–626, 2006.

[40] J-P. Patwardhan, C. Dwyer, A. R. Lebeck, and D. J. Sorin. Circuit and system architecture for DNA-guided self-assembly of nanoelectronics. In *Foundations of Nanoscience: Self-Assembled Architectures and Devices (FNANO)*, pages 344–358, 2004.

[41] Gheorghe Paun. *Membrane Computing: An Introduction*. Springer-Verlag New York, Inc., Secaucus, NJ, USA, 2002.

[42] E. Rauch. Discrete, amorphous physical models. *International Journal of Theoretical Physics*, 42(2):329–348, feb 2003.

[43] A. Regev, E. M. Panina, W. Silverman, L. Cardelli, and E. Shapiro. Bioambients: an abstraction for biological compartments. *Theor. Comput. Sci.*, 325(1):141–167, 2004.

[44] Ivan Singer. *Abstract convex analysis*. John Wiley & Sons Inc, 1997.

[45] Tommaso Toffoli and Norman Margolus. *Cellular Automata Machines, A New Environment for Modeling*. The MIT Press, Reading, Massachusetts, 1987.

[46] S. Torbey and S. G. Akl. An exact and optimal local solution to the two-dimensional convex hull of arbitrary points problem. *Journal of Cellular Automata*, 2008.

[47] Hiroshi Umeo and Keisuke Kubo. A seven-state time-optimum square synchronizer. In Stefania Bandini, Sara Manzoni, Hiroshi Umeo, and Giuseppe Vizzari, editors, *ACRI*, volume 6350 of *Lecture Notes in Computer Science*, pages 219–230. Springer, 2010.

[48] E. Winfree. Self-healing tile sets, architecture for dna-guided self-assembly of nanoelectronics. *Nanotechnology: Science and Computation*, 2006.

[49] Daniel Yamins. *A Theory of local-to-global algorithms for one-dimensional spatial multi-agent systems*. PhD thesis, Harvard University Cambridge, Massachusetts, 2007.