Topological Computation of Activity Regions

Work in Progress Paper

Martin Potier
U-PEC - LACL
61 avenue du Gal de Gaulle
94010 Créteil Cedex
martin.potier@u-pec.fr

Antoine Spicher
U-PEC - LACL
61 avenue du Gal de Gaulle
94010 Créteil Cedex
antoine.spicher@u-pec.fr

Olivier Michel
U-PEC - LACL
61 avenue du Gal de Gaulle
94010 Créteil Cedex
olivier.michel@u-pec.fr

ABSTRACT

Most of the frameworks and languages available in the field of modeling and simulation of dynamical systems focus on the specification of the state of the system and its transition function. Although we believe that this task has been elegantly solved by the design of the rule-based topological programming language MGS, an interesting challenge remains in the computation of the *activity*, and its topology, exhibited by their discrete event simulation. This additional information can help in optimizing, analyzing and modeling complex systems.

Categories and Subject Descriptors

D.1.1 [**Programming Techniques**]: Applicative (Functional) Programming; F.4.2 [**Mathematical Logic and Formal Languages**]: Grammars and Other Rewriting Systems—*Parallel rewriting systems*

General Terms

Algorithms, Theory, Language

Keywords

Activity tracking, MGS programming language, spatial computing, rule-based modeling and simulation, topology of interactions

1. INTRODUCTION

Complex simulation models consist of a large number of interacting parts. In common modeling tools and languages it is not easy to extract abstractions of the dynamics of the whole system, during, before or after its simulation. The analysis of the many outputs and interactions is long and meticulous. To our knowledge, no established method exists for finding *patterns of interactions* in system structures, during a simulation. Some methods exist for particular domains (multi-agent systems, distributed and parallel simulations, image analysis, etc.), but except for the work proposed by [12] no generic methods have been developed for this purpose. In the simulation context, *activity* is usually used as a phase of the system under study. We do not consider this definition of activity here. Instead, activity is considered as a *locus of interaction* occurring during a simulation step.

We define *active regions* as contiguous active parts that are evolving according to the laws governing the model. These regions are highly dynamical, usually changing at each time step. Moreover, we are interested in giving a topological definition of the active regions and to handle them intentionally. We believe that this definition of activity and activity regions [10] can be used as a central guiding concept to construct generic structures for the analysis and specification of systems. The specification structures, driven by a measure of activity of the simulation, can be used to recognize and enhance the dynamics of sub-components in time, space, and states.

MGS is an experimental programming language dedicated to the modeling and the simulation of a special kind of discrete dynamical systems: *dynamical systems with a dynamical structure*, or $(\textsc{ds})^2$ [3]. Dynamical systems with a dynamical structure arise when the state space is not fixed *a priori* but is jointly computed with the current state during the simulation. In this case the evolution function is often given through local rules that drive the interaction between some system components. MGS offers a new kind of data structure, topological collections, to describe the state of a dynamical system, and a new kind of control structure, transformations, to express local and discrete evolution laws. These two notions permit an easy specification of $(\textsc{ds})^2$.

We focus in this paper on the computation of activity and activity regions for MGS. Section 2 introduces the required technical formalism to understand the next sections. Section 3 investigates the topological computation of activity and activity regions in MGS and preliminary results of their use are given in Section 4. Conclusions are drawn in the last section.

2. A PRESENTATION OF MGS

MGS is a domain specific language dedicated to the modeling and simulation of $(\textsc{ds})^2$. It makes use of topology and rewriting techniques to simulate all kind of systems. It is a rule-based declarative, functional and dynamically typed language.

2.1 Topology of Interactions

A $(\textsc{ds})^2$ is a dynamical system whose state space (*i.e.*, whose structure) evolves jointly with the system itself. Think for instance of a growing multi-cellular organism where cells (which are the constitutive elements of the system) divide, migrate and die: the spatial organization of the system changes with the individual cells behavior which depends on the cells locations in this organization.

The dynamical property of its state space makes difficult the specification of the global evolution function of a $(\textsc{ds})^2$. However by comparing two successive global states, as one could do with two successive frames of a movie, one can identify the part of the system that has evolved and the part where nothing has changed. Doing so for every pair of successive states naturally decomposes the system into elementary atomic parts. Thus a $(\textsc{ds})^2$ may be considered as a population of *subsystems* (the atomic parts) which interact with each other, and where the local interactions give rise to the structure of the whole system. For physical systems, subsystems are spatially localized and only subsystems that are neighbors in space can interact directly. So the interactions between parts are structured by the spatial relationships of the parts. For abstract systems, in many cases the transition function of each subsystem only depends on the state variables of a small set of parts (and not on the state variables of the whole system) and the interactions are structured by the functional relationships between the subsystems.

The idea is then to describe the global dynamics by summing up the local evolutions triggered by local interactions. And if two subsystems s and s' ever interact, it is not because they are known to interact but because they are neighbors. Considering that if a subsystem s interacts with a subset $S = \{s_1, \dots, s_n\}$ of parts it also interacts with any subset S' included in S, it is tempting to provide the set of subsystems with a topology based on this closure property: the *topology of interactions*. It restricts the possible local evolution functions of a subsystem s: the current state of s only depends on the previous state of s and of its neighbors. Furthermore, the evolution function not only specifies the evolution of local states but also the coupled evolution of the topology itself.

2.2 Rule-Based Simulation

The MGS programming language has been developed to support the idea of interaction-based modeling of $(\textsc{ds})^2$. It provides *topological collections*, an original data structure for representing the state of a system based on the topology of interactions, and *transformations*, a rule-based definition of functions on collections for specifying the interaction laws of the system.

Topological Collections

A topological collection is a data structure allowing the representation of a population of subsystems.

The underlying topology of interaction is specified using an *abstract cellular complex* (ACC) which is a mathematical concept from *combinatorial algebraic topology* [9]. An ACC is a combinatorial structure made of objects of various dimensions called *topological cells*, each cell being an abstract representation of a simple part of the domain. We call *k-cell* a topological cell of dimension k: 0-cells are vertexes, 1-cells are edges, 2-cells are faces, 3-cells are volumes, etc. Topological cells are glued together and we say that a k-cell σ_1 is *incident* to a $(k-1)$-cell σ_2, denoted $\sigma_1 \preceq \sigma_2$, if σ_2 is in the boundary of σ_1. Formally the incidence relationship is a ranked locally finite partial order on the set of cells where

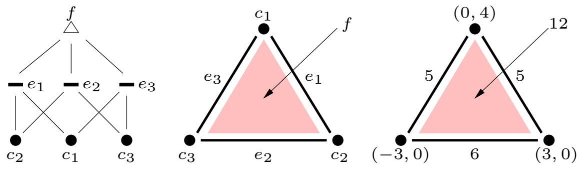

Figure 1: On the left, the Hasse diagram of the incidence relationship of the ACC given in the middle: it is composed of three 0-cells (c_1, c_2, c_3), of three 1-cells (e_1, e_2, e_3) and of a single 2-cells (f). The *closure* of cell e_1 is composed of e_1, c_1 and c_2. The *star* of cell e_1 is the set $\{e_1, f\}$. On the right, a topological collection associates data with the cells: positions with vertexes, lengths with edges and area with f.

the rank coincides with cells dimension. We call *closure* (resp. *star*) of a cell σ the set $\mathrm{Cl}\,\sigma = \{\sigma' \,|\, \sigma' \preceq \sigma\}$ (resp. $\mathrm{St}\,\sigma = \{\sigma' \,|\, \sigma' \succeq \sigma\}$). The incidence relation can be used in numerous ways to specify the neighborhood relationships in a topological collection. In this paper, we consider that two cells σ_1 and σ_2 are neighbors if they are not incident to each other and if they are incident to a common cell either of lower dimension or of higher dimension exclusively:

$$(\mathrm{Cl}\,\sigma_1 \cap \mathrm{St}\,\sigma_2) \cup (\mathrm{St}\,\sigma_1 \cap \mathrm{Cl}\,\sigma_2) = \emptyset$$
$$(\mathrm{Cl}\,\sigma_1 \cap \mathrm{Cl}\,\sigma_2) \neq \emptyset \;\oplus\; (\mathrm{St}\,\sigma_1 \cap \mathrm{St}\,\sigma_2) \neq \emptyset$$

The set of all neighbor cells of some cell σ is given by the *link* denoted $\mathrm{Lk}\,\sigma = \mathrm{St}(\mathrm{Cl}\,\sigma) \triangle \mathrm{Cl}(\mathrm{St}\,\sigma)$, where $\triangle$ denotes the symmetric difference operation.

The states of the subsystems are represented as labels on topological cells: a *topological collection* C is a partial function that associates values from an arbitrary set V with cells of some ACC (see Figure 1). Thus the notation $C(\sigma)$ refers to the value/state of cell σ in collection C. We call *support* of C and write $|C|$ for the set of cells for which C is defined. Set V is left arbitrary to allow the association of any kind of information with the topological cells: for instance geometric properties ($V = \{-1, 0, 1\}$ for representing orientation or $V = \mathbb{R}^n$ for Euclidean positions) or arbitrary state of a subsystem (a mass, a concentration of chemicals, or a force acting on certain cells, etc.)

The collection C can be written as a formal sum $\sum_{\sigma \in |C|} v_\sigma \cdot \sigma$ where $v_\sigma \overset{\mathrm{df}}{=} C(\sigma)$. With this notation, the underlying ACC is left implicit but can usually be recovered from the context. By convention, when we write a collection C as a sum

$$C = v_1 \cdot \sigma_1 + \cdots + v_p \cdot \sigma_p$$

we insist that all c_i are distinct[1]. Notice that this addition is associative and commutative: the order of operations used to build a topological collection is irrelevant. Using this notation, a *subcollection* S of a collection C is defined as a sub-formal sum $C = S + S'$; subcollection S' is then called the complementary of S in C and one writes $S' = C - S$. Set operations and incidence operations (Cl, St and Lk) are naturally extended on subcollections.

[1]This notation is borrowed from algebraic topology where set V is taken with a commutative group structure which gives to topological chains an abelian group structure [9]. We relax this assumption for topological collection.

Transformations

As previously mentioned, an interaction is a subset of subsystems evolving together due to the satisfaction of some local conditions: the subsystems are neighbor in the topology of interaction and their states match. We advocate that a rule-based programming style fits well the description of interaction laws and that rewriting techniques is the right computation model for modeling and simulating (DS)[2].

Transformations generalize this process to topological collections with the concept of *topological rewriting*. A transformation T is a function specified by a set $\{r_1, \dots, r_n\}$ of rewriting rules $r_i = p_i \Rightarrow e_i$ where each p_i is a pattern and e_i is an expression. An application of such a rule selects a subcollection S matching with p_i that is then substituted by the subcollection resulting from the evaluation of expression e_i. The application of a transformation T on a collection C can be represented by the following diagram:

$$\begin{tikzcd}
C \arrow[d, "T"] & = & S_1 \arrow[d, "r_{i_1}"] & + & \dots & + & S_n \arrow[d, "r_{i_n}"] & + & R \arrow[d] \\
T(C) & = & S'_1 & + & \dots & + & S'_n & + & R
\end{tikzcd}$$

where each S_k is a subcollection of C matching with pattern p_{i_k} and S'_k results from the evaluation of e_{i_k}. Subcollection R consists of unchanged subsystems. The formal semantics is given in [13].

Since it may exist different ways to decompose collection C w.r.t. transformation T, only one of the possible outcomes (randomly chosen) is returned by the transformation. A *rule application strategy* is then used to control the pattern matching process.

3. ACTIVITY TRACKING IN MGS

We are now interested in investigating the concept of *activity* in MGS. Activity [11] is a measure of event occurrences or state changes in a simulation. This measure can be used for different purposes like optimizing simulation or giving a better understanding of dynamics. In the context of MGS, activity will allow us to (1) develop a more efficient pattern matching algorithm, and to (2) identify higher level structures in a model and to track them in simulations. The former will be illustrated in the next section, the latter will be discussed in the conclusion.

In this section, we present a generic way to track an active region throughout the simulation.

Context

For the sake of simplicity, we restrict ourselves to patterns involving only at most two interacting elements. For example, pattern `x, y, z` (*i.e.*, matching three elements where `y` is a neighbor of `x` and `z` is a neighbor of `y`) is out of the scope of the following study. However we conclude the section with a paragraph about the generalization.

We illustrate our idea with a simplistic but paradigmatic example, a forest fire spread. This 3-state cellular automaton is easily encoded in an MGS transformation as follows:

```
trans fire_spread = {
  'Forest as x / member('Fire, neighbors x)
        => 'Fire;
  'Fire => 'Ashes;
}
```

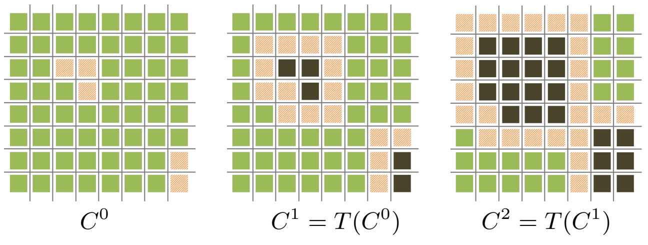

Figure 2: Evolution relation, first three steps of a forest fire simulation. Green, orange (hatched) and black represent Forest, Fire and Ashes respectively.

States are represented by three symbols: `'Forest`, `'Fire` and `'Ashes`. The first rule specifies how forest catches fire when it is neighbor of some fire and the second rule specifies that fire leaves ashes after burning. Figure 2 shows three consecutive evolution steps of this automaton on a square grid topological collection.

In the example of Figure 2, activity is located in a small piece of space – only cells in fire and forest in their neighborhoods evolve – and progresses from neighbor to neighbor. However the usual MGS pattern matching process needs to iterate over the whole collection at each application of transformation `fire_spread`. Given the small number of cells in interaction, the matching process is here quite inefficient. The following study of activity will allows us to target specifically evolving cells during pattern matching. Moreover activity enlightens an important emergent structure of fire spread models, the fire front.

Activity and Interactions

The interpretation of activity in MGS corresponds to the number of interactions occurring at each time step of a simulation. Since interactions happen whenever rules of a transformation match subcollections, activity splits naturally a collection into two parts: the *active* subcollection as the subcollection where interactions *can* occur and the *quiescent* subcollection where they cannot.

Let us define the active and quiescent subcollections in a formal and general way, using the construct of topological collections. Let C be a collection and T be a transformation. We define the *matching function* $\mathbb{M}_T$ of transformation T as the function which maps a collection C onto the set of all subcollections of C matched by a pattern of T. In other words, the matching function $\mathbb{M}_T$ represents a call to the pattern matching process. The *active subcollection* A of C is then defined as

$$A = \bigcup_{S \in \mathbb{M}_T(C)} S$$

that is the merge[2] of all matched subcollections of C. Thus, the *quiescent subcollection* Q corresponds to the complementary of A in C:

$$Q = C - A$$

Figure 3 shows the decomposition into active and quiescent subcollections for the fire spread example.

[2]The union operator is used instead of the addition since some subcollections of $\mathbb{M}_T$ may have overlapping support.

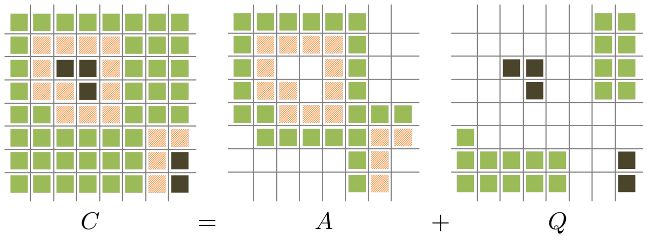

Figure 3: Decomposition of a topological collection into a set of active cells and quiescent cells. Green, orange (hatched) and black represent Forest, Fire and Ashes respectively.

Activity Dynamics

In order to track activity during the simulation of a system, we define a way to compute the evolution of the active region (grow, shrink, carve, etc.) based on the topological properties of the active subcollection.

Let us consider $(C^0, C^1, \dots)$ the trajectory of collections due to the successive applications of a transformation T, where $C^{i+1} = T(C^i)$ for $i \geq 0$. Using the active-quiescent decomposition on $C^i = A^i + Q^i$, one can notice that the application of transformation T only acts on the active part and leaves Q^i unchanged so that the pattern matching can be restricted to subcollection A^i only:

$$C^{i+1} = T(C^i) = T(A^i|F^i) + Q^i \tag{1}$$

The notation $T(A^i|F^i)$ reflects that the restricted pattern matching process may require some information from Q^i: visiting the neighborhood of some matched element (*e.g.*, in transformation `fire_spread`: `member('Fire, neighbors x)`) does not imply that the visited neighbors are in A^i. These information are denoted F^i which is the subcollection whose support consists of all quiescent cells with a neighbor in A^i: $F^i = \mathrm{Lk}\, A^i$. The decomposition of C^{i+1} proposed in Equation (1) does not coincide with the definition of A^{i+1} and Q^{i+1}. In fact, some cells of $|Q^i|$ may become active and *vice versa*. However the tracking of the active-quiescent frontier during a simulation can be refined by making use of subcollection F^i. Indeed assuming that transformation rules involve at most two neighbor elements, any quiescent cell, at some iteration of the simulation, having no active neighbor cell will observe no change in its environment and then will remain quiescent at the next iteration step. More formally we get:

$$Q^i - \mathrm{Lk}\, A^i \subset Q^{i+1} \qquad A^{i+1} \subset T(A^i|\mathrm{Lk}\, A^i) + \mathrm{Lk}\, A^i \tag{2}$$

The second statement is trivially obtained using complementary and means equivalently that the active part cannot expand further than F^i. Equation (1) is then rewritten

$$C^{i+1} = \Big[T(A^i|\mathrm{Lk}\, A^i) + \mathrm{Lk}\, A^i\Big] + \Big[Q^i - \mathrm{Lk}\, A^i\Big] \tag{3}$$

Optimized Pattern Matching

In the light of Equations (2), the active-quiescent decomposition of C^{i+1} is obviously over-approximated in Equation (3). As an example, on Figure 4, bold lines represent the limit of expansion of the active part at next step but some cells in this subcollection become quiescent. These cells can be identified as the cells of $T(A^i|\mathrm{Lk}\, A^i) + \mathrm{Lk}\, A^i$ which cannot be involved in any interaction at time $i+1$. In other words they are not selected by the pattern matching process:

$$\mathbb{M}_T(C^{i+1}) = \mathbb{M}_T(T(A^i|\mathrm{Lk}\, A^i) + \mathrm{Lk}\, A^i)$$

and the computation of sequence $(A^i)_{i\in\mathbb{N}}$ follows

$$\begin{cases} A^0 &= \bigcup_{S\in\mathbb{M}_T(C^0)} S \\ A^{i+1} &= \bigcup_{S\in\mathbb{M}_T(T(A^i|\mathrm{Lk}\, A^i)+\mathrm{Lk}\, A^i)} S \end{cases}$$

Notice that this recursive definition of A^i does not refer to the quiescent subcollection Q^i anymore, since the neighbor operator Lk exactly targets the interesting cells. As a consequence, activity tracking during the simulation allows us to focus on a reduced part of the collection for pattern matching. The induced optimization is discussed and illustrated in Section 4.

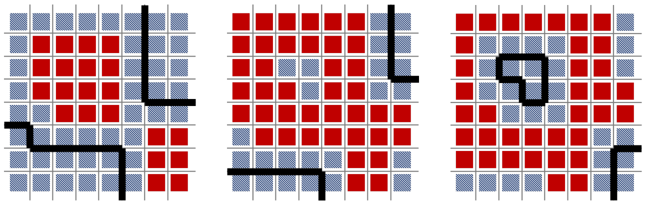

Figure 4: Evolution of the active-quiescent frontier for the three steps of Figure 2. Active cells in red, quiescent cells in light blue (hatched). The bold lines represent the decompositions induced by Equation (3).

Topological Characterization

The previous study can be generalized to transformation involving more than two interacting elements by reconsidering how far the active subcollection can expand at each time step. Let n be the *radius* of an interaction, *i.e.*, the minimum distance (in terms of hops) between interacting elements. The previous restriction was to only consider transformation of radius $n \leq 1$. Let us now consider arbitrary radius $n \in \mathbb{N}$. For example, patterns of the form `f, e / neighborsfold([...],f)` are of radius $n = 2$ since the visited neighbors of `f` can be at distance 2 of `e`.

The expansion F^i_n of A^i for a radius n is given recursively by

$$\begin{cases} F^i_0 &= 0 \\ F^i_{n+1} &= \mathrm{Lk}(A^i + F^i_n) + F^i_n \end{cases}$$

This definition is analogous to the definition of the *wave operator* $W(n)$ in [1] and reveals the topological nature of activity. The wave operator is used for the elaboration of a combinatorial Morse theory; Morse theory is a mathematical tool for studying topology of spaces. Roughly speaking, this theory deals with *Morse functions* – a way to flood the space with some "liquid" – and *critical points* – where the liquid reveals basins, passes and peaks of the topography of the space. We are currently investigating the analogy between Morse theory and activity tracking: Morse functions will correspond to activity propagation and critical points to space-time positions where independent activity zones segregate or collide, pointing out important events in the model.

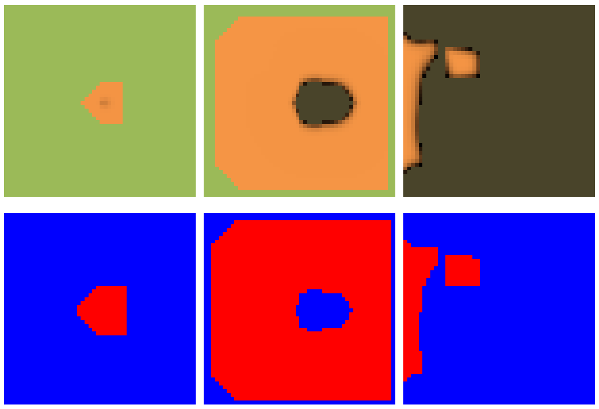

Figure 5: Fire spread (fire is orange, forest is green, ash is brown), Activity (active cells in red, quiescent cells in blue) at iterations 5 (left), 22 (center) and 95 (right) of a simulation with wind blowing from the left and a horizontal symmetrical chain of mountain landscape.

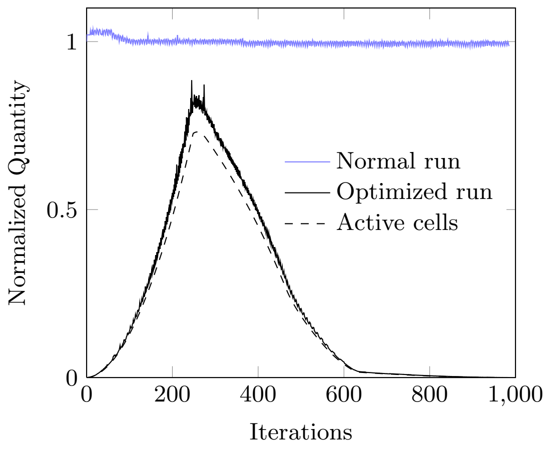

Figure 6: Amount of active cells and computation time for non-optimized and optimized algorithm per iteration on a 500×500 grid. Values have been normalized using the total size of the $(250\,000$ cells) for amounts of cells, and the average computation time of a normal run step for computation times.

4. PRELIMINARY RESULTS

In this section, we consider a more elaborate model of forest fire spreading. Following results of Section 3, information provided by activity are used to speed up simulation by applying rules of transformation in active regions.

4.1 An Example of Forest Fire Simulation

Forest fire is a well known and paradigmatic example for the community of discrete event simulation [6]. Here, we implement in MGS a more elaborate, yet efficient, model for forest fire spreading than the basic example used in Section 3. The model is based on the work of [8] and uses the cellular automata (CA) formalism while being both simple and realistic. The forest fire spread model takes into account the main environmental effects: wind (for both speed and direction), type of fuel, and landscape topography. At any time, the state of a cell in the CA is characterized by the ratio of burnt area ranging continuously from 0 (wholesome forest) to 1 (burnt forest - ashes) where $(0, 1)$ represents burning forest. Considering a *fire front* as the *area* separating the forest from the ashes, the main concern of our model is to determine the evolution of the fire front from: the current fire front position, the distribution of rates of fire spread in the forest, the wind speed and direction, as well as the height and shape of the land.

The cyclic 2D square grid used in the CA is represented using a *GBF topological collection* [4]. A GBF is topological collection whose underlying ACC is the Cayley graph of an abelian group presentation. The elements of the group represents the allowed atomic displacements. In MGS, such a collection is specified from a presentation of the group of displacements. The definition of a Moore neighborhood grid, requires four basic displacements `n` (north), `e` (east), `ne` (north-east) and `se` (south-east):

```
gbf Land = < n, ne, e, se ;
             500 n = 0, 500 e = 0,
             ne = e + n, se = e - n >
```

The four additional equations specify that the grid is cyclic of size 500×500 and define relations between diagonal and non-diagonal directions. The reverse directions are automatically considered. Each cell of the GBF is labeled by its state (a record containing the burned out state, the wind direction, the burning rate and the altitude).

Transformation `fire_spread` of Section 3 is extended to take into account dynamics of [8]. Figure 5 shows simulations of the model at different iterations.

4.2 Benchmark

Using the classical pattern matching algorithm, the entire topological collection must be iterated over at every update. For a CA on a square grid of side length n, the process must go through the n^2 cells leading to an update step with a constant time (see Figure 6), while only a fluctuating number of cells have their values changed on the update (which correspond to only a fraction of the grid).

In this section, we use activity information to reduce the cost of pattern matching. The previous example of forest fire spread was rewritten to have the transformation rules only apply to the active region and skip the quiescent part. As with the theoretical view presented in Section 3, the cellular automaton space is split between active cells taking part in a transformation of their neighbors and quiescent cells having no role to play whatsoever in the current iteration. This results in a decent speedup of the whole computation.

Figure 6 presents a plot of the count of active cells for each iteration, and of the computation time by iteration for both the optimized and the regular run of the simulation. Time is normalized by the average computation time of the normal run so that both normal and optimized computation time can be easily compared. The occasional spikes and noise come from the fluctuating work on the test computer.

As can be deduced from Figure 5, at the beginning of the simulation, very few cells are active. Their amount increases dramatically with the spread of fire to only lower once the forest has been consumed and turns into ashes. As a cellular automaton, the computation time is linear with the number of cells: the more cells, the more time with the same computation time by cell. From the figure, we can notice that the computation time of the non-optimized run is roughly the same for every iteration: the pattern matching mechanism must explore all of the 250 000 cells to verify whether a rule applies. Whether a rule matches or not does not change the time required to compute the application of a transformation rule.

We can immediately notice that activity and computation time of the optimized run coincide: it directly depends on the amount of cells to explore, thus the shape of the curve is nearly identical to the variation of the amount of active cells. Cells are never all active at once, that is why, even at its peak, the optimized run always requires less computation time by iteration than the non-optimized one. The cost of maintenance can be read from the difference between the computation time of the optimized run and the active cell count. This optimization is data-dependent, had we changed the initial values, the graph would have been altered too. Therefore, a general or global speed-up does not make sense.

5. RELATED WORK AND CONCLUSION

This paper focuses on the computation of activity and activity regions in the context of the domain-specific language MGS. We have characterized and provided a topological framework to compute these two notions using the link operator of combinatorial topology. Activity regions have been successfully used to reduce the cost of pattern matching in the simulation of an example of fire spread. In this case, activity appears to be a data-dependent optimization technique that can be easily combined with other techniques.

While our approach could appear similar to dependency graph optimizations (as in Gillespie's SSA algorithm by [5]), it is very different since, due to the topological approach followed by MGS, the computation of activity regions described here is *generic*, unlike previously asserted in [11], page 162. Moreover, it is valid for *any kind of topological collections* available in the language. Based on the notion of *spatial interaction*, MGS provides a unified simulation framework encompassing discrete/continuous and deterministic/stochastic formalisms (as in the classical approach of rewriting-based tools in computational biology, by [7, 2] for example, that consider structureless objects - namely only chemical reactions).

The perspectives opened by this work are numerous. We have to internalize in the runtime the computation of the active regions. Indeed, we believe that it is not to the programmer to seek for complex and obfuscating optimizations but to the language (or the execution support) to provide automatic high-level techniques to reduce the execution time.

At the language level, activity regions are not first-class objects: they are computed at each time-step of the simulation, and there is no link between regions computed at two different time steps. *Reifying* the regions to first-class objects and adding operators to handle these new objects will be the key to level up the language to multi-level modeling of complex dynamical systems that exhibits behaviors at multiple levels of descriptions (like in biology the molecular level, the cellular level, the organic level, etc.).

ACKNOWLEDGMENTS

The authors would like to thanks the anonymous reviewers for their comments on a first version of this paper.

The authors are grateful to the community born during the Cargese Interdisciplinary Seminar in 2009 and to the colleagues of the spatial computing initiative. This research is supported in part by the French ANR grant "SynBioTIC" 2010-BLAN-0307-03, U-PEC, IRCAM, Univ. of Évry and ISC-Paris.

6. REFERENCES

[1] U. Axen. *Topological Analysis Using Morse Theory and Auditory Display.* PhD thesis, University of Illinois at Urbana-Champaign, Champaign, IL, USA, 1998.

[2] V. Danos, J. Feret, W. Fontana, R. Harmer, and J. Krivine. Rule-based modelling, symmetries, refinements. *Formal Methods in Systems Biology*, pages 103–122, 2008.

[3] J.-L. Giavitto, C. Godin, O. Michel, and P. Prusinkiewicz. *Modelling and Simulation of biological processes in the context of genomics*, chapter "Computational Models for Integrative and Developmental Biology". Hermes, July 2002.

[4] J.-L. Giavitto, O. Michel, and J.-P. Sansonnet. Group based fields. In I. Takayasu, R. H. J. Halstead, and C. Queinnec, editors, *PSLS'95*, volume 1068 of *LNCS*, pages 209–215, Beaune (France), 2–4 Oct. 1995. Springer Verlag.

[5] M. A. Gibson and J. Bruck. Efficient exact stochastic simulation of chemical systems with many species and many channels. *The journal of physical chemistry A*, 104(9):1876–1889, 2000.

[6] X. Hu, Y. Sun, and L. Ntaimo. DEVS-FIRE: design and application of formal discrete event wildfire spread and suppression models. *SIMULATION*, 88(3):259–279, Oct. 2011.

[7] M. John, C. Lhoussaine, J. Niehren, and C. Versari. Biochemical reaction rules with constraints. *Programming Languages and Systems*, pages 338–357, 2011.

[8] I. Karafyllidis and A. Thanailakis. A model for predicting forest fire spreading using cellular automata. *Ecological Modelling*, 99(1):87 – 97, 1997.

[9] J. Munkres. *Elements of Algebraic Topology.* Addison-Wesley, 1984.

[10] A. Muzy, L. Touraille, H. Vangheluwe, O. Michel, M. Kaba Traoré, and D. R.C. Hill. Activity Regions for the Specification of Discrete Event Systems. In *DEVS'10*, 2010.

[11] A. Muzy, F. Varenne, B. P. Zeigler, J. Caux, P. Coquillard, L. Touraille, D. Prunetti, P. Caillou, O. Michel, and D. R. Hill. Refounding of activity concept? Towards a federative paradigm for modeling and simulation. *SIMULATION*, 89(2):156–177, 2012.

[12] J. J. Shi. Activity-based construction (ABC) modeling and simulation method. *Journal of Construction Engineering and Management*, 1999.

[13] A. Spicher, O. Michel, and J.-L. Giavitto. Declarative mesh subdivision using topological rewriting in mgs. In *Int. Conf. on Graph Transformations (ICGT) 2010*, volume 6372 of *LNCS*, pages 298–313, Sept. 2010.