Modelling and Simulation of Several Interacting Cellular Automata

Marta Pla-Castells

IRTIC – Universidad de Valencia
C/ Catedrático José Beltrán, 2.
46980, Paterna – Valencia (Spain)
email: marta.pla@uv.es

Ignacio García-Fernández

Departament d'Informatica – Universidad de Valencia
Av. de la Universidad s/n
46100, Burjassot – Valencia (Spain)
email: ignacio.garcia@uv.es

Rafael J. Martínez-Durá

IRTIC – Universidad de Valencia
C/ Catedrático José Beltrán, 2.
46980, Paterna – Valencia (Spain)
email: rafael.martinez@uv.es

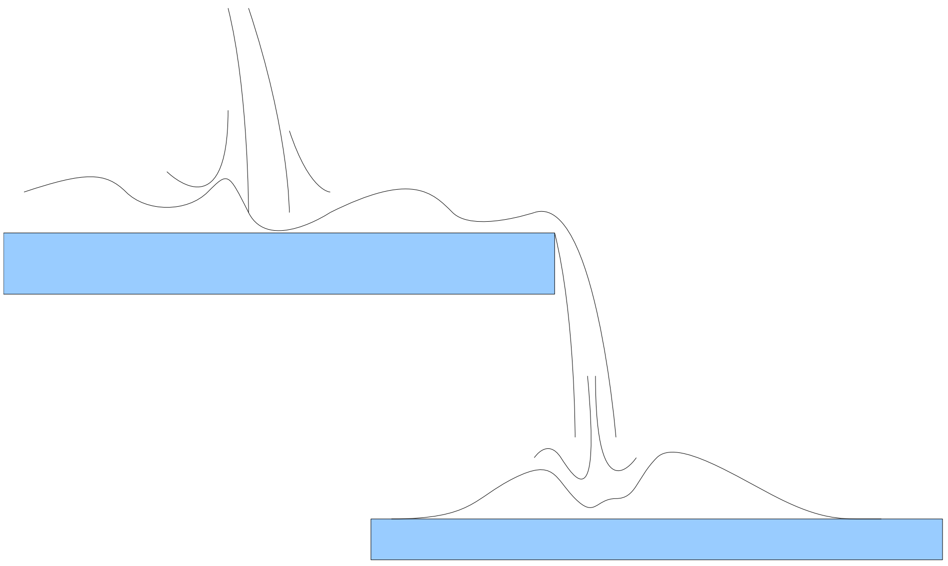
Abstract: Cellular Automata are used for modelling and simulation of many systems. In some applications, the system is formed by a set of subsystems that can be modelled separately, but, in such cases, the existence of interactions between these subsystems requires additional modelling and computer programming. In this paper we propose a modelling methodology for the simulation of a set of Cellular Automata models that interact with each other. The modelling methodology is described, together with an insight on implementation details. Also, it is applied to a particular Cellular Automata model, the Sanpile Model, to illustrate its use and to obtain some example simulations.

Keywords: Cellular Automata models, sandpile model, pedestrian evacuation

Reference to this paper should be made as follows: Pla-Castells, M., García-Fernández, I. and Martínez-Durá, R. (xxxx) 'Modelling and Simulation of Several Interacting Cellular Automata', *International Journal of Computer Aided Engineering and Technology*, Vol. x, No. x, pp.xxx–xxx.

Biographical notes: Marta Pla-Castells studied mathematics at University of Valencia and had her PhD in Computer Science in 2009. She is lecturer at Algebra department in Universty of Valencia and researcher at IRTIC. Her main research interests are Virtual Reality applications for training and interactive simulation.

Figure 1 An example in which two subsystems exchange matter. This situation can be found in many applications of the Cellular Automata methodology.



Ignacio Garcia-Fernandez studied mathematics at University of Valencia. He had his PhD in Computer Science in 2009 and he is professor at Informatics Departament of University of Valencia. His main research interest is physics based interactive simulation and animation.

Rafael J. Martinez studied informatics and had his PhD in Computer Architecture in 1999. Since 2000 he is the head of simulation group LSYM at IRTIC, and has leaded several research projects within the field of simulation.

1 Introduction

Modelling and simulation of spatially distributed physical phenomena is one of the most common applications of Cellular Automata [CD05]. In such cases, the evolution of the automata is usually related to the exchange of some kind of material, information or energy between adjacent cells. The simulation of this kind of systems using a Cellular Automaton requires the representation of the system domain using the Cellular Automaton grid. This can be done by means of the definition of different types of cells in order to simulate boundary conditions. However, there are situations in which the use of a unique Cellular Automaton to represent the whole domain is not straightforward.

Consider, as an example, the system of Figure 1: It is composed by two flat surfaces, one atop the other, and a fluid that is spread on the upper one. As soon as the fluid reaches the edge of the upper subdomain, it will start falling onto the lower one, thus exchanging matter from one subsystem to the other.

Situations equivalent to this example can be found in different applications. Problems such as crowd dynamics in evacuation simulation [Zho09, PM08], traffic simulation

[AM06], lava and fire spread [VAN$^+$07, AVSB08] or tumour propagation [RAM$^+$06] may involve the simulation of several interconnected levels and sub-regions. If the simulation domain can be easily decomposed in several subdomains, then a set of independent Cellular Automata models can be used. But, in this case, the interaction between the different subdomains has to be defined and simulated. Although some authors have used the simulation of more than one domain, the issue of connecting the subdomains is solved by modelling the interactions in a custom manner adequate to the particular problem [RAM$^+$06, PM08]. However, a general methodology for this problem has not been addressed.

In this work a modelling formalism is proposed that makes it possible to handle such situations in a very straightforward and general way. The paper is structured as follows. In what remains of this section, some definitions and notations that will be used later in this work are presented. Then, the modelling methodology for several interacting automata is presented. Next, several implementation details are given in the form of data structures and general procedures. Finally, the methodology proposed is used to simulate a particular example, the Sandpile Model, showing several applications.

1.1 Cellular Automata Formalization

In order to present the proposed scheme, first we state the basic notation that is used within this work. A classical two dimensional Cellular Automata based on a regular grid is considered. Every cell of the grid has a set of variables that describe the state of every cell during the simulation.

The evolution of every cell is governed by a local rule that, without loss of generality, will be considered the same for every cell. In order to determine the state change of a cell, the local rule takes the current state of the cell itself and of a group of cells of the grid, which are called its neighbourhood.

A Cellular Automata can be formally described as

$$CA = \langle \mathbf{M}, \mathbf{V}, \mathbf{S}, \varphi \rangle \qquad (1)$$

where

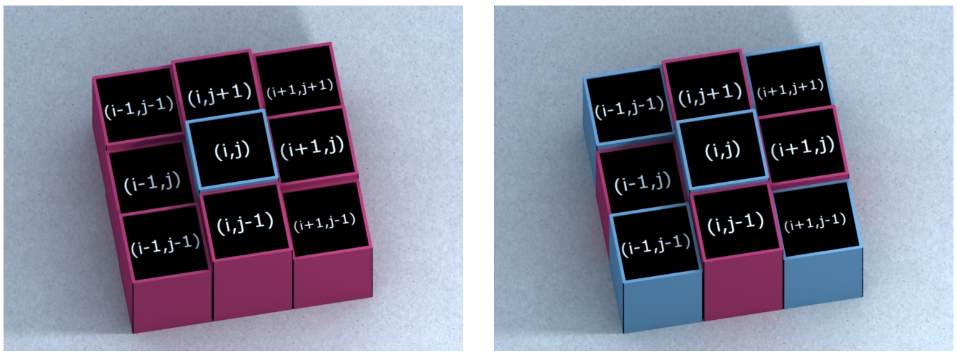
- $\mathbf{M} = \{(i,j) \in [1,N] \times [1,M] \subset \mathbb{N} \times \mathbb{N}\}$ is the set of cells that form the Cellular Automata.
- $\mathbf{V}$ is the neighbourhood of a cell.
- $\mathbf{S}$ is the set of possible states for a cell.
- $\varphi : \mathbf{S}^{|\mathbf{V}|+1} \to \mathbf{S}$ is the local rule or the transition rule of the Automata.

The neighbourhood $\mathbf{V}$ of cell (i,j) will be considered as a subset of its adjacent cells

$$\mathbf{V} \subseteq \mathbf{Adj} := \{(i+k, j+l) : k,l = -1,0,1\} \setminus \{(i,j)\}.$$

This definition of neighbourhood includes the Moore neighbourhood, which corresponds to the complete set $\mathbf{V}_{moore} = \mathbf{Adj}$, and the Von Neumann neighbourhood, $\mathbf{V}_{vn} = \{(i-1,j),(i+1.j),(i,j-1),(i,j+1)\} \subset \mathbf{Adj}$ (see Figure 2). Von Neumann and Moore neighbourhoods are the most common ones in the Cellular Automata literature [CD05].

Figure 2 The figure shows the Moore neighbourhood (left) and the Von Neumann neighbourhood (right), which are used in many Cellular Automata models. The cells that belong to the neighbourhood are coloured in red.



The set of possible states of a cell, $\mathbf{S}$, is the Cartesian product of the sets that host the state variables of a cell. In the classical definition of Cellular Automata, such spaces are usually formed by a finite set of elements [CD05], although many applications consider more general sets, including real values [Tof84, CD05].

The local rule $\varphi : \mathbf{S}^{|\mathbf{V}|+1} \rightarrow \mathbf{S}$ is a function that obtains the evolution of cell (i, j) using its state and the state of the cells that belong to its neighbourhood. This function, together with the cell's state variables, is what actually defines the dynamic model of the system.

1.2 Open Cellular Automata and interactions

The previous definition of Cellular Automata considers a single spatial domain for the model, that takes the form of the set of cells $\mathbf{M}$. However, there are systems in which the domain can be naturally decomposed into several subdomains.

Let's consider a Cellular Automata model to simulate the movement of a crowd of people in a building. The Cellular Automata grid represents a room and the state of each cell is related to its occupancy. A transition rule can be defined to simulate the behaviour of a crowd during an evacuation [Zho09, SKM12, ZJS12]. In such a model, the cells that represent an staircase or a gate also *throw occupancy* out of the system.

Another example can be found in the Sandpile Models [BTW87]. In the Sandpile Model, cell state represents the amount of sand that is hosted in the cell, and transitions represent material flow in a granular system. A sandpile model is said to be *open* if sand exits the system when it reaches the grid frontier. In this case, the frontier cells have the particular property that can *send material* outside the system. If we consider a system that involves several sandpiles, then the outgoing flow of one sandpile can fall on another sandpile, becoming an input flow. This model will be discussed in more detail later on.

Indeed, we can find examples in industrial processes in which not only there exist relationships between different subsystems but, in addition, these relationships vary with time. In the case of sandpiles this would happen if the relative position of the various subsystems vary with time; e.g a set of conveyor belts to transport bulk material, or an excavator loading sand into a truck.

The notation described will be used in the next sections to propose a modelling methodology for a system with several interacting subdomains. Each subdomain will be modelled with an independent Cellular Automaton, and the different submodels will be interconnected.

2 Cellular automata interaction

The previous examples show that, if we have several subsystems modelled by means of Cellular Automata, we can be interested in simulating an exchange of matter, energy or other type of information between them. Moreover, the relationship between the different subdomains can vary along time during the simulation.

2.1 Interaction modelling

In order to model such situations we propose a modelling methodology that is based in identifying data flow and defining rules for connecting different Cellular Automata cells. The methodology can be decomposed in the following basic steps:

- Identification of the cells of the different automata that act as connection cells.
- Definition of models for the interactions that are to be simulated.
- Definition of a rule, or set of rules, that determine which cells need to be connected to simulate the interactions.
- Monitoring of the existing connections during the simulation, to modify or remove them.

Next, these four steps are explained in more detail.

2.1.1 Identification of emitting cells

In order to formalize the problem, and to allow a systematic definition of the procedure, first we are going to define an special type of cells, which will be called *emitting cells*. An *emitting cell* will be any cell of a Cellular Automaton that, due to the problem definition, is able to throw information out of the system and, potentially, send it to a cell of a different automata.

The first step of the methodology is the identification of such cells in every one of the involved Cellular Automata. This identification will depend on the properties of the system that is being simulated and on the kind of relationship that is to be modelled.

2.1.2 Modelling of the connections

The second step is to define a model for the information exchange that has motivated the definition of the *emitting cells*. The model will depend on the particular problem and on the features that we need to represent with our Cellular Automata model.

However, in many cases the information exchange will be the same that also takes place between adjacent cells during the update of the Cellular Automata. In this case, a general approach can be used that greatly simplifies the simulation phase.

When the interaction between an emitting cell and another Cellular Automaton follows the same model as the one described by the local rule, the easiest way to set the relationship is the modification of the neighbourhood of the emitting cell. By doing this, the simulation of the interaction is featured when the local rule, φ, is applied to the emitting cell during the update of the Cellular Automaton.

2.1.3 Configuration of the system of Cellular Automata

Once the *emitting cells* have been identified, and the relationships have been modelled, it is necessary to define which cells need to be connected to each other. This will be done by means of a set of rules and criteria that decide whether an emitting cell has to be connected to a certain cell of another Cellular Automaton.

As it happens with the emitting cells, the identification of these criteria will depend on the properties of the system that is being modelled. However, they will often be related to the position of the cells, due to the kind of systems that are usually modelled by means of cellular automata.

2.1.4 Update of the connections

It has been illustrated previously that the interaction between two or more Cellular Automata can change during the simulation as a result of, for example, the displacement of the different subsystems.

Thus, the connection rules will be used before the beginning of the simulation, to configure the simulation scenario, but it can be necessary to check them also after every simulation step to decide whether any relationship needs to be added, modified or removed.

After presenting the main steps of the proposed methodology, next we overview the way they can be applied to perform a simulation that involves several interacting Cellular Automata.

2.2 Interaction simulation

Cellular Automata are a computational approach to modelling systems and, often, they are used to simulate the modelled systems by means of a computer program. The previous modelling scheme can be applied very easily to an existing Cellular Automata computer model following the next steps:

1. Build a list of Cellular Automata that contains all the automata that are to be simulated.
2. In the data structure used to implement a Cellular Automaton, build a list that holds all the *emitting cells* of the automaton.
3. Either
   (a) implement the functions that decide if an emitting cell needs to be connected to any cell, or
   (b) set up manually the connections of emitting cells.
4. If the local rule of the automaton is not enough to simulate the connections, then implement the necessary models.
5. Run the simulation.
   (a) Update every automaton. Call the connection models, when necessary.
   (b) After every step run the procedures that update the connections.

Next section presents in more detail the data structures that allow the implementation of the cell connections and the simulation of the relationships.

3 Cell neighbourhood implementation

In this section, a set of data types and procedures are presented. They are used to store the Cellular Automata information and to define and implement the neighbourhood of a cell in such a way that foreign cells (cells from another Cellular Automaton) can be used during a simulation. For sake of simplicity they are presented for a one-dimensional Cellular Automaton, in which cells are arranged in a linear array. Later in this section the generalization from this case to the two-dimensional case is discussed.

The data structures and the procedures that are presented next have been written using the C programming language conventions, especially in what regards pointers, which are expressed with asterisks after the type name, although they can, obviously, be translated into any programming language.

3.1 Overall Description of the Implementation

Usually, a Cellular Automaton is implemented by means of a sequential data structure that stores the cells' data. When implementing the local rule, the neighbourhood is defined by means of explicit reference to the indexes of the elements of the array.

In opposition to this approach, and in order to allow the communication between different Cellular Automata, we propose the use of a data structure to represent cells that uses pointers to refer to the cells that belong to its neighbourhood. With this cell data type, the Cellular Automata implementation will become a multiply linked data structure, where the different links define the neighbourhood relationship.

Using this implementation, and once it has been decided that a cell must be connected to another one, the connection will be as simple as a pointer assignment.

3.2 Dynamic Neighbours in One Dimension

Let's consider a one-dimensional Cellular Automaton with N cells. In this case, the grid is an array of cells $\mathbf{M} = \{(i);\ i = 1, \ldots, N\}$ and the neighbourhood of cell (i) is formed by its adjacent cells $\mathbf{V}(i) = \{(i-1), (i+1)\}$.

3.2.1 Cell Data Structure

In order to store the information of a cell, a basic data structure, that will be called `CellState` is needed. This structure will not be defined here, as it is composed by the parameters and state variables of a cell, and it depends on the particular application of the Cellular Automaton. In order to store the complete grid of a Cellular Automaton, an array of `CellState` structures is used. This array could be defined as

```
CellState CA_Grid0[N_CELLS];
```

for a Cellular Automata of size `N_CELLS`.

Computing the evolution of cell (i), which is stored in the data structure `CA_Grid0[i]`, involves the state of its neighbour cells. The simplest way of accessing such cells' information is by explicit indirection of the array elements `CA_Grid0[i-1]` and `CA_Grid0[i+1]`. But, this assumes that the neighbourhood relationship can only happen between adjacent cells of the same Cellular Automaton, and it imposes a limitation in the situation described in the introductory section of this paper.

In order to overcome this limitation we propose to use pointers in order to refer to the neighbours of a cell. This new data structure will make it possible to modify very easily the neighbourhood of a cell to make any other cell to become its neighbour, no matter its position in the grid and, even, whether it belongs to the same Cellular Automaton or not. Using this idea, the basic cell data structure is as follows:

```
struct Cell {
  CellState StateInfo;
  Cell * Prev;
  Cell * Next;
};
```

Next, each attribute of the `Cell` structure is described briefly:

`StateInfo` An structure of type `CellState`, that contains the basic information for a cell, including its state variables.

`Prev` A pointer to a `Cell` structure, that stores the data of the neighbour that is to the left of the cell, usually cell $(i-1)$.

`Next` A pointer to a `Cell` structure, that stores the data of the neighbour that is to the right of the cell, usually cell $(i+1)$.

Again, the Cellular Automata grid can be defined by means of an array, but now using the new data type, as

```
Cell CA_Grid[N_CELLS];
```

The `Cell` structure is general, in the sense that it allows the implementation of standard Cellular Automata, just by pointing `Prev` and `Next` to their default values, the structures of cells $i-1$ and $i+1$ respectively.

3.2.2 Basic Methods

Next, three basic methods are defined to show the use of the `Cell` data structure to define dynamic neighbours. First, procedure `Reset_Neighbours()` makes it possible to use the `Cell` structure in the traditional way, setting its neighbours as its adjacent cells.

```
Reset_Neighbours(int i)
{
  CA_Grid[i].Prev = &CA_Grid[i-1];
  CA_Grid[i].Next = &CA_Grid[i+1];
}
```

If, on the contrary, we are interested in linking cell (i) to another arbitrary cell, no matter whether it belongs to the same automata or not, the following procedures can be used.

```
Set_Neighbour_Prev(int i,Cell *_c)
{
  CA_Grid[i].Prev = _c;
}
```

```
Set_Neighbour_Next(int i,Cell *_c)
{
  CA_Grid[i].Next = _c;
}
```

3.3 Cell Evolution

At this point it is important to note that the computation of the update rule is not affected at all by this definition of the data structures. The only difference is the use of the value of the structures pointed by `Prev` and `Next` instead of using an explicit indirection of the array.

Moreover, this fact has an interesting collateral effect. Using this data structure and the related methods it is possible to make the neighbourhood relationship a non symmetric relationship; the reason is that cell a can take cell b as a neighbour while cell b keeps its original neighbourhood, thus allowing one-way information flow.

As an example, consider the system described by Figure 1. In this case, a cell of the upper Cellular Automaton will have as a neighbour one of the cells of the lower Cellular Automaton. However, in the lower Cellular Automaton we still want that cell to keep its original neighbours to obtain the evolution of the lower subsystem.

3.4 Dynamic Neighbours in Two Dimensions

The data structures presented for the one-dimensional Cellular Automata can be easily extended to be used in the two-dimensional case. Basically, the `Cell` structure needs to be extended so that it has references to the eight possible cells of its neighbourhood.

This can be done either by defining eight pointers using appropriate names, or by using an array of eight pointers as it is shown in the next piece of code

```
struct Cell2D {
  CellState StateInfo;
  Cell2D * Neighbours[8];
};
```

where `Cell2D * Neighbours[8]` represents an array of eight pointers to `Cell2D` structures. The `Reset_Neighbours()` method also needs to be extended accordingly.

The methods that were defined for neighbour assignment, `Set_Neighbour_Prev()` and `Set_Neighbour_Next()`, are now unified in one single function, which receives as a parameter the index of the neighbour to assign.

```
Set_Neighbour(int i,int n,Cell *_c)
{
  CA_Grid[i].Neighbours[n] = _c;
}
```

Thus, the step from one to two dimensions is rather straightforward and, as in the one-dimensional case, it does not affect the calculus of the evolution of the Cellular Automata. Based on this implementation of the Cellular Automata, arbitrary neighbourhood configurations can be defined in order to build a simulation scenario.

4 Interaction of two sandpiles

In order to illustrate the proposed methodology, the complete procedure is described for a particular Cellular Automata model; the Sandpile Model [BTW87]. Briefly, a sandpile model is a Cellular Automaton on a regular square grid in which the value of the state variable represents the height of a sand pile over the grid.

The update rule of the sandpile model states that whenever the difference in height between two neighbour cells is higher than a threshold, an *avalanche* happens. This means that the upper cell reduces its height a fixed amount z, and the lower cell increases its height the same amount. Thus, in this case, the interaction between tow cells is set in terms of material exchange when an avalanche takes place.

The interaction between two sandpiles will be considered as a particular case of this situation; when a cell in the frontier of one sandpile receives material, then this material is considered to leave the system, and the cell does not increase its height. When this happens, if another sandpile is placed below the cell, then it receives the material that has been thrown by the upper sandpile.

4.1 Modelling of two interacting sandpiles

The proposed modelling methodology is applied to the case of two interacting sandpiles. Next, the different steps that have been defined are described when they are applied to this particular problem.

4.1.1 Emitting cells

The first step of the methodology is to identify the cells that will potentially interact with other automata. According to the description of the problem of two interacting sandpiles, the *emiting cells* in this case are the so-called *frontier cells* of the automata, which are placed at the edges of the grid (see Figure 3). In general, in a n-dimensional grid of sizes $N_1 \times N_2 \times \cdots \times N_n$, a cell is a frontier cell if and only if the $i-th$ coordinate is 1 or N_i.

Note that a frontier cell is an special cell in the sense that it will have several neighbours undefined, as it does not have adjacent cells in some directions. This can be implemented by initializing the corresponding pointers to `NULL` in the data structures defined previously in this paper.

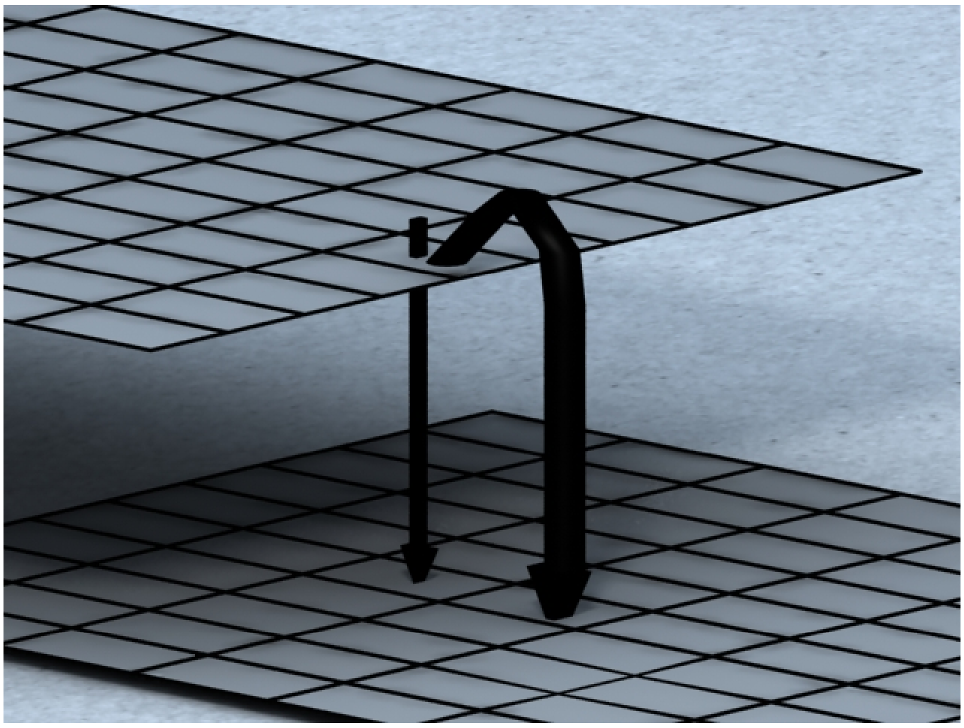
4.1.2 Interaction modelling

The second step in the methodology is to define the necessary interaction models. However, in this case no particular interaction model is necessary, as the same kind of interaction is considered between cells and between the two automata. Thus, by modifying the neighbourhood of the frontier cells of the upper automata the interaction will be automatically simulated when applying the local rule to the modified cells.

4.1.3 Overlapping cells

The third step is the definition of a rule that decides when an *emitting cell* has to be connected to another cell during the simulation. In this case, the decision has to be made according to the relative position of the *emitting cell* in the upper sandpile respect to the lower sandpile.

Figure 3 A frontier cell is an *emitting cell* in the problem of two interacting sandpiles.



In order to set the rule, let's first define this idea more precisely. Let us consider the upper sandpile S_u and the lower sandpile S_l. It is assumed that the vertical position of sandpile S_u is higher than the vertical position of sandpile S_l. We say that a cell $(c^u) \in S_u$ overlaps cell $(c^l) \in S_l$ if the centre of cell (c^u) is inside cell (c^l). Now we can define the rule.

Let's consider a cell c^u in the upper sandpile. If it overlaps a cell c^l of the lower sandpile, then c^u is connected to the lower sandpile. In order to do this, all the missing neighbours of c^u are made to point to the corresponding neighbours in c^l.

4.1.4 Neighbourhood update

In order to set up the neighbourhoods of the upper frontier cells, a procedure will be defined. This procedure can be called once, to configure the scenario at the beginning of the simulation, but also can be called after every simulation step, if the relative position of both sandpiles change along time.

Next, the procedure is described in pseudocode, using the data structures that were defined previously in this paper. The next additional notation will be used:

`Sandpile` A structure that contains the data of a complete sandpile, including its position in the simulation scenario.

`Cell * Is\_Overlaping(Cell *c,Sandpile *s)` A procedure that checks if cell `c` overlaps any cell of the sandpile `s`. If that is the case, returns a pointer to the overlapped cell. It returns `NULL` otherwise.

Fr_U[] An array that contains a list of pointers or references to the frontier of the upper cellular automata.

Sl An structure of type Sandpile with the information regarding the lower sandpile.

Using the previous data structures, the following procedure is applied to very *emitting cell* of the upper automata, prior to the beginning of the simulation and after every simulation step:

Algorithm 1:
```
Input : Sandpile Sl, Cell Fr_U[]
Variables : Cell cl
for c ∈ Fr_U[]
    cl ← Is_Overlapping(c, Sl)
    if cl ≠ NULL
        Assign missing neighbours in cell c using the
        neighbours in cell cl.
else
        Reset neighbours in cell cl.
    fi
end
```

This procedure assumes that the overlapped cell cl is not a frontier cell of the lower sandpile, Sl. The reason is that frontier cells throw away the material they receive and, thus, they are discarded.

5 Results

The example above has been used to simulate two different situations. A simple system composed by a pair of one-dimensional sandpiles, and a more general simulation scenario, which reproduces the load of granular material by a small loader machine.

5.1 A one-dimensional example

In order to illustrate the application of the methodology, first a simple one-dimensional example is presented. Using the methodology proposed, two sandpiles have been set up at different heights. The upper one has one of its edges over the centre of the lower one. During the simulation, a fixed amount of material has been added to the upper system every step. Figure 4 shows a sequence of snapshots of this simulation.

In this case, the system was static; there was no movement of the sandpiles during the simulation. Thus, the Cellular Automaton configuration was defined during the set up stage of the simulation and no update of the cells' neighbourhood was necessary after every simulation step.

5.2 Loader simulation

The modelling methodology presented in this paper has been applied to a more complex simulation application. It has been used for the dynamic model in a virtual reality application that simulates a small loader.

Figure 4 A sequence of the simulation of two sandpiles that exchange material. From bottom to top: the inital state of the simulation (a), the moment when the material reaches the edge of the upper system (b) and the final state (c).

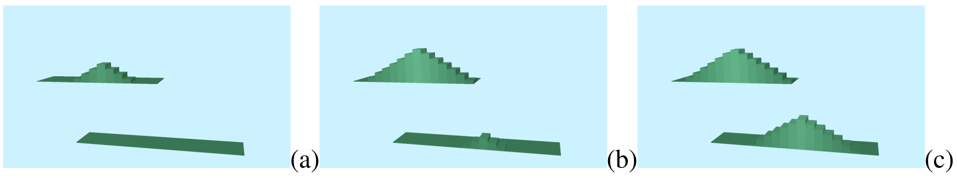
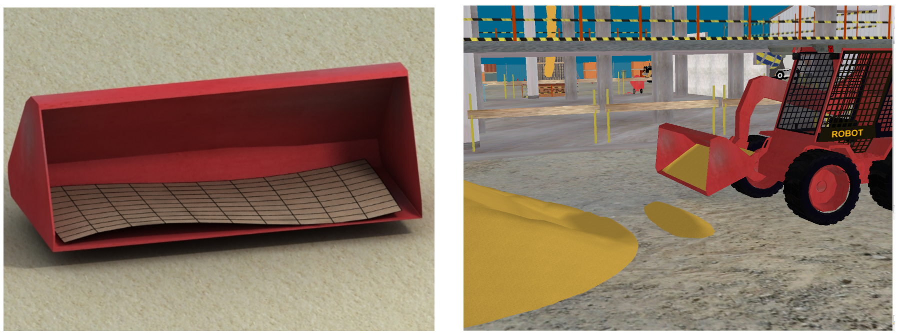


Figure 5 A Cellular Automaton fixed to an excavator bucket (upper figure). When the bucket is over another Cellular Automaton it can load granular material into it.



In this case, one sandpile is placed on the ground to simulate a heap of granular material and several additional sandpiles are placed over the loader bucket or any other object that needs to accumulate material (see Figure 5). The frontier cells of any moving sandpile are checked after every simulation step to dynamically reconfigure their neighbourhood.

6 Conclusion

This paper has presented a modelling methodology for the simulation of several interacting Cellular Automata. By means of the reconfiguration of the cells' neighbourhood it is possible to define arbitrary connections between the different automata involved.

The methodology has been described formally, and implementation details have also been presented. In addition, it has been developed in more detail for the sandpile model, a particular type of Cellular Automata. Moreover, the applicability of the methodology has been illustrated by means of two applications, which show that it can be used even in complex simulation environments.

The procedures that have been described, based in the modification of cells' neighbourhood, provide with an automated method to intercommunicate two or more automata so that they can share information during a simulation.

Acknowledgement

This work has been supported by Ministerio de Industria y Energia of Spain, proj. FIT-340000-2006-290.

References

[AM06] A. Aponte and J.A. Moreno. Cellular automata and its application to the modeling of vehicular traffic in the city of caracas. In Samira El Yacoubi, Bastien Chopard, and Stefania Bandini, editors, *ACRI*, volume 4173 of *Lecture Notes in Computer Science*, pages 502–511. Springer, 2006.

[AVSB08] A. Alexandridis, D. Vakalis, C.I. Siettos, and G.V. Bafas. A cellular automata model for forest fire spread prediction: The case of the wildfire that swept through spetses island in 1990. *Applied Mathematics and Computation*, 204(1):191–201, 2008.

[BTW87] P. Bak, C. Tang, and K. Wiesenfeld. Self-organized criticality: An explanation of the 1/f noise. *Physical Review Letters*, 59(4):381–384, 1987.

[CD05] B. Chopard and M. Droz. *Cellular Automata Modeling of Physical Systems*. Cambridge University Press, 2005.

[PM08] Nuria Pelechano and Ali Malkawi. Evacuation simulation models: Challenges in modeling high rise building evacuation with cellular automata approaches. *Automation in Construction*, 17(4):377–385, 2008.

[RAM$^+$06] B. Ribba, T. Alarcón, K. Marron, P.K. Maini, and Z. Agur. The use of hybrid cellular automaton models for improving cancer therapy. In *Proceedings of ACRi'06*, pages 444–453, 2006.

[SKM12] Nobuhiko Shinozaki, Shigeyuki Koyama, and Shin Morishita. Evacuation simulation from rooms through a pathway and a stairway by cellular automata based on the public guideline. In *Cellular Automata*, volume 7495 of *Lecture Notes in Computer Science*, pages 743–751. Springer Berlin Heidelberg, 2012.

[Tof84] Tommaso Toffoli. Cellular automata as an alternative to (rather than an approximation of) differential equations in modeling physics. *Physica D: Nonlinear Phenomena*, 10(1-2):117 – 127, 1984.

[VAN$^+$07] Annamaria Vicari, Herault Alexis, Ciro Del Negro, Mauro Coltelli, Maria Marsella, and Cristina Proietti. Modeling of the 2001 lava flow at etna volcano by a cellular automata approach. *Environmental Modelling & Software*, 22(10):1465–1471, 2007.

[Zho09] Shuqiu Zhou. Study on the evacuation simulation based on cellular automata. In *ITCS '09: Proceedings of the 2009 International Conference on Information Technology and Computer Science*, pages 481–484, Washington, DC, USA, 2009. IEEE Computer Society.

[ZJS12] Nuo Zhu, Bin Jia, and Chun-Fu Shao. Pedestrian evacuation with the obstacles based on cellular automata. In *Computational Sciences and Optimization (CSO), 2012 Fifth International Joint Conference on*, pages 448–452, 2012.