An Algebra for Modeling and Simulation of Continuous Spatial Changes

André Fonseca Amâncio, Tiago Garcia de Senna Carneiro

Department of Computing of Federal University of Ouro Preto (UFOP), Ouro Preto, MG, Brazil.
{afancio, tiagogsc}@gmail.com

Abstract. Continuous change models are commonly based on the Systems Dynamics paradigm. However, this paradigm does not provide support for an explicit and heterogeneous representation of geographic space, nor its topological (neighborhood) structure. Therefore, using it in modeling spatial changes still remains a challenge. In this context, this paper presents an algebra that extends the Systems Dynamics paradigm to the development of spatially explicit models of continuous change. The proposed algebra provides types and operators to represent flows of energy and matter between heterogeneous regions of geographic space. To this end, algebraic sets of operations similar to those in Map Algebras are introduced, allowing the representation of local, focal and zonal flows. Finally, case studies are presented to evaluate the usefulness, expressiveness and computational efficiency of the proposed algebra.

Categories and Subject Descriptors: H.4.0 [**Information Systems Applications**]: General; I.6.0 [**SIMULATION AND MODELING**]: General

Keywords: Algebra, Continuous Spatial Change, Modelling and Software Simulation

1. INTRODUCTION

Continuous spatial changes describe continuous flows of energy or matter between regions of geographic space. Although the System Dynamics paradigm (Forrester [1960], Meadows [2008]) is widely used for modeling continuous changes, it does not provide support for an explicit and heterogeneous representation of geographic space and its topological structure (neighborhood). For this reason, it needs to be extended to construct spatially explicit models [d'Aquino et al. 2002] of continuous spatial changes, as of interactions between society and nature.

In this work, the types and operators present in map algebras [Tomlin 1990; Karssenberg et al. 2001; Cordeiro et al. 2009; Schmitz et al. 2013] are used to extend the Systems Dynamics paradigm. Map algebras generally define three sets of operations, defined as proposed by Tomlin [1990]: (a) Local operations - whose value for a location in the output map is calculated from the values for that location in the input maps; (b) Focal operations - whose value of a location in the output map is calculated from the values for the neighborhood of that location in the input map; and (c) Zonal operations - whose value of a location in the output map is calculated from the values for summarized regions in the input map;

Models based on Map Algebra and System Dynamics have completely different syntax, semantics, and execution flows, creating challenges for combining these paradigms. Models based on map algebras are finite sequences of algebraic expressions that cause discrete changes in space. The operations are performed synchronously and immediately, one after the other, as they appear in the model code. Models based on System Dynamics are formulated in terms of differential equations and use infinitesimally small time-steps for numerical integration procedures [Kelly et al. 2013] that simulate continuous changes over time. Since in many models it is common to find interdependence between the several differential equations, they need to be computed simultaneously to avoid error propagation. Therefore, the equations are invoked asynchronously, that is, when they appear in the model code they are only instantiated and, they are executed only after all of them have been invoked. In addition, the combination of these paradigms needs to deal with spatio-temporal dependence generated by feedback loops [Schmitz et al. 2013]. Feedback loops generate data dependencies that need to be resolved for the coherence of simulations, that is, during simulations intermediate states of the shared variables need to be updated and persisted to avoid error propagation. Finally, the modeling activity requires the modeler to be a specialist in the model application domain and in computer programming to be able to code it in the form of algebraic operations or differential equations. Currently, the following questions remain: *How to promote the expressiveness of modeling tools for continuous and spatially explicit change simulation? How to combine different behavioral, spatial and temporal representations in those tools, in a transparent way for the modeler?*

In this context, this work proposes and evaluates through case studies an algebra for the development of spatially explicit models of continuous changes that take place in the geographic space. This algebra extends the Systems Dynamics paradigm proposed by Forrester [1960] with flows transporting energy between heterogeneous stocks which are used to represent distinct regions in the geographic space. In the original paradigm, stocks may represent only homogeneous amount of energy. Therefore, originally there is no need to express how changes caused by flows should be allocated in the stocks (space). Moreover, the original paradigm overall structure and semantics was kept, changes happen simultaneously in a continuous time base and flow rules are interpreted as differential equations. In despite of the space geometry (shape/location) is discret, the state of each location is represented by a continuous variable meaning the amount of energy on it. To this end, we introduce sets of algebraic operations similar to those in Map Algebras, allowing the representation of local, focal and zonal flows. We evaluate in case studies the usefulness, expressiveness and computational efficiency of the proposed algebra.

This paper is an extended version of [Amâncio and Carneiro 2017], presented in XVIII Brazilian Symposium on GeoInformatics (GEOINFO 2017). We extended the algebra for modeling open systems and present two new case studies in which the system exchanges energy with the environment. It is structured as follows: Section 2 presents the related works. In section 3, the algebra that extends the Systems Dynamics paradigm is described as a generic instrument for modeling continuous spatial changes. Section 4 explains how the algebra works. In Section 5, we describe case studies of simplified models to evaluate the usefulness, expressiveness and computational efficiency of an implementation of this algebra in the TerraME tool [Carneiro et al. 2013]. Finally, the discussion of the benefits of using algebra concludes this work.

2. RELATED WORK

Most extensions of the Systems Dynamics paradigm only replicate systems-based models in discrete and regular partitions of space to deal with spatial changes, interconnecting stocks present in these models through spatial neighborhoods. All changes occur instantly and simultaneously (snapshot). Stocks in a locality are linked to stocks of same name in neighboring localities, simulating processes of spatial diffusion or mobility. Generally, the lateral flows are controlled by only one rate fixed by the modeler, with no way to represent heterogeneous lateral flows. The neighborhoods are of stationary topology, typically Moore or von Neumann. This type of approach was called Spatial System Dynamics (SSD) [Ahmad and Simonovic 2004]. Some authors consider it a simplistic extension of Systems Dynamics, it has a slow execution and is only appropriate for feedback loops between two models [Swinerd and McNaught 2012; Sahin and Mohamed 2014]. Therefore, the limitation of these approaches in dealing with heterogeneous, non-stationary and anisotropic spaces, under different spatial and temporal scales, has motivated several innovations [Elsawah et al. 2017]. The Spatial

Modeling Environment (SME) [Maxwell and Costanza 1997] platform was pioneer in this sense, by representing vertical flows between diverse representations of space and allocating different models in different regions. To represent more complex interactions the literature presents approaches based on Individual-Based Modeling [Vincenot et al. 2011], Hybrid Simulations Involving Agent-based [Swinerd and McNaught 2012] and Discrete Event Simulation [Morgan et al. 2017].

On the other hand, several papers propose generalizations and extensions of Map Algebra to represent spatial processes. Camara et al. [2005] present a generalized Map Algebra that uses spatial topological and directional predicates. Frank [2005] discusses how Map Algebra can be formalized for programming, extending it to deal with spatio-temporal data. Cordeiro et al. [2009] extend the Geoalgebra concept (Takeyama [1997] and Couclelis [1997]) by proposing a map algebra that simplifies its use on environmental and dynamic models. Schmitz et al. [2013] combine the concepts of Map Algebra and Model Algebra for the coupling of model components. Camara et al. [2014] introduce the concept of Fields for representations of continuous spatio-temporal variables, demonstrating its use in the construction of a novel Map Algebra. Silva and Carneiro (2016) developed an algebra for models based on spatially explicit agents. However, the authors of this work have not found in the literature algebras that extend the System Dynamics paradigm to operate directly on maps, or that extend Map Algebra to represent continuous flows of energy or matter.

The PCRaster approach extends map algebra for the development of spatio-temporal environmental models [Burrough 1998; Wesselung et al. 1996]. However, it does not explicitly represent the flow operator from System Dynamic Theory in its algebra. It is assigned to the modeler the responsibility to implement a set of operations (map = map +/- change()) to simulate outflows from one storage or/and inflows to another. Those operations are interpreted as difference equations computed only once at each simulation time step. No numerical integration methods are applied. In contrast, we propose an extension of System Dynamic Theory to the development of geospatial models, with an explicit representation of flow operations, reducing the modeler responsibility of properly implement, simulate and compute flows of energy.

Regarding usability, Frank [2005] and Silva and Carneiro [2017] describe algebras as facilitators for model specification, since modelers did not need to become experts in different languages and modeling tools to describe models. The use of a given algebra allows the description of model components focused on the model objectives and not on its implementation [Schmitz et al. 2013]. Cardelli [1997] reinforces the idea that simplifications in models reduce the effort to understand it by future applications and prevent possible mistakes made by users. Frank [2005] says that the descriptions of algebra processing steps can be formalized and optimized. Finally, Schmitz et al. [2013] present evidence that the automation of the interaction routines between space regions, through algebra native operators guarantees the integrity and improves the readability of the models.

3. TYPES OF ALGEBRA OPERATORS

The algebra proposed in this work is a generic tool for modeling continuous spatial changes, it can be implemented in several tools and languages according to the ideas presented in Silva and Carneiro (2016). Here, algebra components are specified from abstractions of their operators [Frank 1999].

The algebra operators act over spatial types that represent stocks of energy or matter (attributes) localized in the geographical space. Space topology (neighborhood and proximity relations) is also represented allowing diffusive flows. Operators are subdivided into creation operators, responsible for creating and relating types, and flow operators, responsible for defining how changes occur (behavioral rules) in relation to time and space. Finally, the execution of the operator coordinates the simultaneous and interleaved execution of changes during simulation.

Table I. Spatial types.

(a) Basic types	**(b) Collections**
Cell: (name, attributes, neighbors) —name : String —attributes: [Attribute] —neighbors: SpatialNeighborhood	CellularSpace: (cells, dimension) —cells: [Cell] —dimension: (width: Number, length: Number)
SpatialNeighborhood : (type, d, self, cells) —type: String —d: (width: Number, length: Number) —self: Boolean —cells: [Cell]	Trajectory: (cs, selectFunction, sortFunction, cells) —cs: CellularSpace —selectFunction: Boolean Function (Cell) —sortFunction: Boolean Function(Cell, Cell) —cells:[Cells]

3.1 Spatial types

There are four spatial types present in the algebra: *Cells, CellularSpaces, Trajectories* and *Neighborhoods*. There are two basic spatial types (Table I (a)): cell and neighborhood. A *Cell* represents the stocks of a space location and contains a list of attributes and a list of neighboring cells. *Neighborhoods* represent the space connectivity and can represent areas of influence, adjacency or proximity relations. Moore and von Neumann neighborhoods are often used for spatially explicit modelling.

CellularSpaces and *Trajectories* are collections (Table I (b)), that is, they represent sets of entities of the same type, in this case cells. *CellularSpaces* represent regions in the geographic space. All cells in a *CellularSpace* are composed by the same set of attributes, which can assume distinct values during simulation. *Trajectories* are collections that select and order cells from a *CellularSpace*, allowing the modeler to filter the cells on which operators must focus and to establish the order in which those operators must traverse the *CellularSpace* performing changes.

3.2 Creation Operators

Creation operators are intended to ensure that all basic types belong to, at least, one collection. In this way, after creating basic types, the modeler needs to relate them to a collection in order to use than as operands in other operators. Table II presents the definition of creation operators.

The *CellularSpace* creation operator uses a Cell instance that provides the archetype for cloning the other cells it aggregates; the dimension of the *CellularSpace* determines the number of Cells that it will contain. Cells can be neighbors of each other determining the *CellularSpace* topology. A *SpatialNeighborhood* for a *CellularSpace* depends on the type and dimension of the neighborhood and on whether cells are self-contained in their own neighborhood relations. Modelers may also provide a filter function to exclude undesired neighborhood relations and a weight function to determine the strength of each relation.

3.3 The Flow operator

In this algebra, the Flow operators (FLOW - Table III) use only collections (*CellularSpace* and *Trajectory*) as operands and represents continuous transference of energy or matter between regions of space. The differential equation supplied as the first operator parameter determines the amount of energy transferred between regions.

Flow operations are classified as local, focal and zonal and their semantics depend on the parameters reported at the moment they are invoked, as described in Table IV and illustrated in Figure 1 .

Table II. Creation operators

createCell: Cell Function (name, attributes) —name : String —attributes: [Attribute]	createCellularSpace: CellularSpace Function(cell, dimension) —cell: Cell —dimension: (width: Number, length: Number)
createTrajectory: Trajectory Function(cs, select, sort) —cs: Cellular Space —select: Boolean Function(Cell) —sort: Boolean Function(Cell, Cell)	createSpatialNeighborhood: SpatialNeighborhood Function(cs, type, dimension, self, filter, weight) —cs: CellularSpace —type: String —dimension: (width: Number, length: Number) —Self: Boolean —filter: Boolean Function(Cell) —weight: Number Function(Cell, Cell)

Table III. Flow operator

FLOW (f(), a, b, step, Collection1, "Attribute", "Neight1", Collection2, "Attribute", "Neight2")
—f(): Differential equation that describes, as a function of one or two parameters, the rate of change (point derivative) of energy f (t, y) at time t, where t is the simulation current instant time, and y is the past value of the rate of change f ().
—a: Number - Beginning of the integration interval.
—b: Number - End of integration interval.
—step: Number - An infinitesimal time interval used in numerical integration.
—Collection1: Cellular Space or Trajectory - A collection of cells that will be used to calculate and subtract flow output.
—Attribute1: String - Name of the attribute of the cells contained in the collections over which the flow will operate.
—Neight1: Neighborhood - Neighborhood name defined on the source collection of the energy flow. Optional.
—Collection2: Cellular Space or Trajectory - Target collection of energy flow.
—Attribute2: String - Name of the attribute of the cells contained in the collections of the energy flow.
—Neight2: Neighborhood - Neighborhood name defined over the recipient collection of the energy flow. Optional.

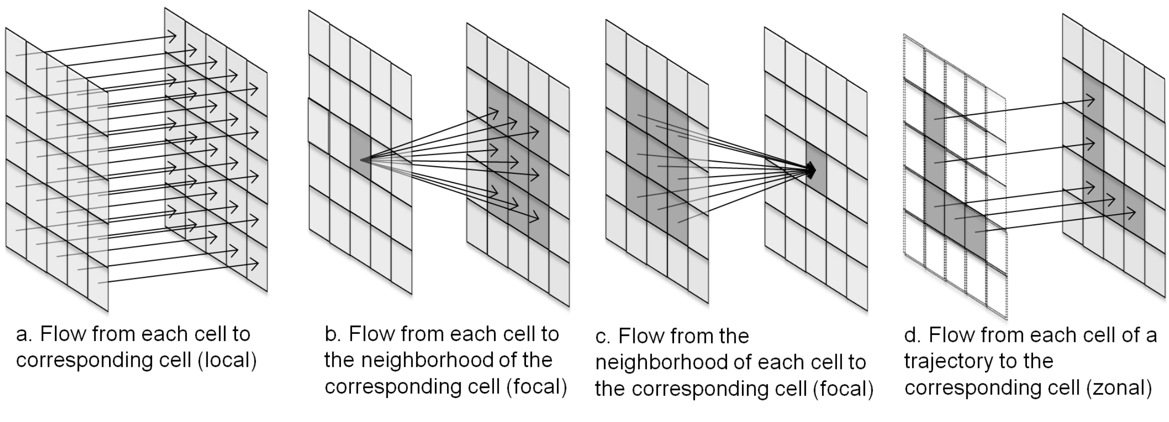

Fig. 1. Flow operator examples between two collections

Flows from the collection A to the collection B are calculated only for cells in intersection A ∩ B. Figure 2 illustrates the possible topological relations between collections and the Flow operator semantics, named by Egenhofer and Herring [1994] as: a. equal, b.contains, c.inside and d.overlap.

Table IV. Behavior rule, syntax and semantics of the flow operator

RULE	SYNTAX	SEMANTICS
Flow local execution; Rule: From Cell To Cel;	flow(Collection, Collection) —Collection —Collection	Each cell in a cellular space transfers part of its attribute stock at a rate defined by f (t, y) to the spatially corresponding cell attribute of another cellular space, Figure 1 (a). Example: precipitation of cloud water to ground.
Flow focal execution; Rule: From Cell To Neigth Of Cell;	flow(Collection, Collection, Neigth) —Collection —Collection —Neight	Each cell in a cellular space transfers part of its attribute stock at a rate defined by f (t, y) to the attributes of cells in the neighborhood of the spatially corresponding cell of another cellular space, Figure 1 (b). Example: Heat dispersion in fire propagation modeling.
Flow focal execution; Rule: From Neight Cell To Cell;	flow(Collection, Neight, Collection) —Collection —Neight —Collection	Each cell from neighborhood of a cell in a cellular space transfers part of its attribute stock at a rate defined by f (t, y) to the cell attribute spatially corresponding to the central cell of the neighborhood of another cellular space, Figure 1 (c). Example: Condensation of water in clouds.
Flow zonal execution; Rule: From Selected Cell To Cell;	flow(Trajectory, Collection) —Trajectory —Collection	Each cell in a trajectory transfers part of its attribute stock at a rate defined by f (t, y) to the spatially corresponding cell attribute of another cellular space, Figure 1 (d). Example: Evaporation of water from a river to clouds.

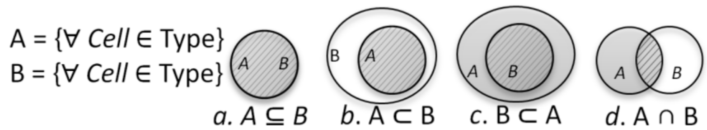

Fig. 2. Venn diagram of the topological relations between two collections

The semantics of some Flow operators are not detailed in the Table IV, such as flows between distinct *Trajectories* (zonal - example: flow of water from the mainland to the ocean at a beach), or flows from a trajectory to its neighborhood (zonal and focal composition - example: heat flowing from fire border), among others. In the Flow operator, it is possible to construct several combinations of collection and neighborhood parameters, both in the source or destination of flows.

Flow operations have special cases in which it represents continuous insertion or elimination of energy from space regions, Figure 6. The differential equation given as the first parameter determines the amount of energy inserted or eliminated from the regions. In these cases, its definition will omit the source collection (energy insertion in the system) or the target collection (elimination of system energy) as described in Table V and illustrates Figure 6.

3.4 Execute operator

The Execute operator described in Table VI starts the simulation execution. The simulation will run until the simulation clock reaches the time received as parameter (finalSimulationTime). All flows have the definition of its integration interval, defined by the **lower** time and upper time limits. The **lowest** time limit for all flows is used as the initial simulation time.

4. SIMULATION EXECUTION AND ITS IMPLEMENTATION IN TERRAME

The proposed algebra simulator was implemented based on the temporal types of the TerraME platform: Timer and Event (Table VII). Timer is a discrete event scheduler that operates according

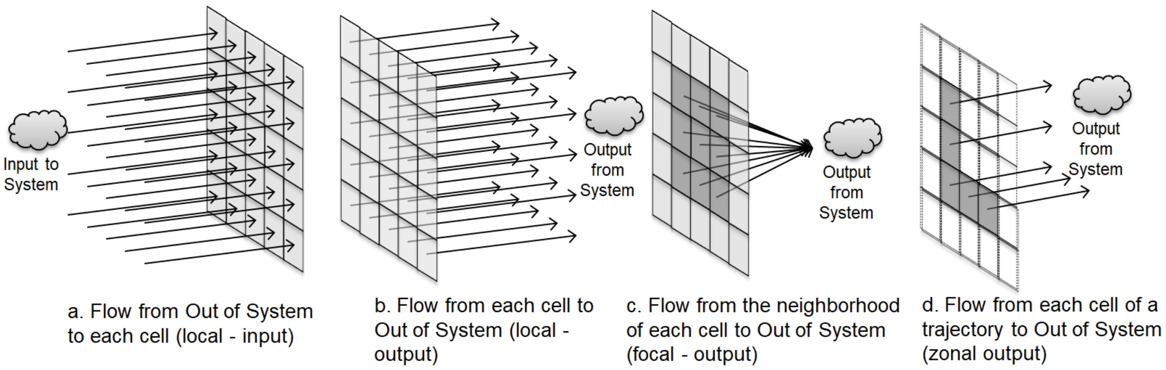

Fig. 3. Flow operator examples between a collection and system input and output

Table V. Behavior rule, syntax and semantics of the flow operator forsystem energy input and output

RULE	SYNTAX	SEMANTICS
Flow local execution; Rule: From Out of System To Cell;	flow(nil, Collection) —nil —Collection	Each cell in a cellular space suffer an energy variation in its attribute stock at a rate defined by f (t, y), Figure 3 (a). Example: population grows in prey predator model.
Flow local execution; Rule: From cell to Out Of System;	flow(Collection, nil) —Collection —nil	Each cell in a cellular space transfers part of its attribute stock at a rate defined by f (t, y) to out of system, Figure 3 (b). Example: Population death in prey predator model.
Flow focal execution; Rule: From Neight Of Cell to Out Of System;	flow(Collection, Neight, nil) —Collection —Neight —nil	Each cell from neighborhood of a cell in a cellular space transfers part of its attribute stock at a rate defined by f (t, y) to out of system, Figure 3 (c). Example: Burning of biomass of front fire border.
Flow zonal execution; Rule: From Selected Cell To Out Of System;	flow(Trajectory, nil) —Trajectory —nil	Each cell in a trajectory transfers part of its attribute stock at a rate defined by f (t, y) to out of system, Figure 3 (d). Example: Absorption of water by human collection from a river.

Table VI. Definition of the execution operator

Execute : (finalSimulationTime) —finalSimulaionTime: Number

to Discrete Event Driven Simulation (DEVs) [Wainer 2009]. It maintains a queue of chronologically ordered events and the current time record of the simulation. Events are instants in the simulated time in which the modeler of the TerraME platform performs input and output operations, or computations defined by the modeler. Events are defined by the instant, periodicity, final instant, and action parameters. The instant parameter determines the moment in the simulation in which the event must occur, triggering an action defined by the modeler. The periodicity determines the instant that the event will occur again. The final instant (finalInstant) determines when the event will cease to occur. The action is a function that implements the behavioral rules of the model or commands for TerraME to load, view, and store data. The return value of an action is used as a stop condition, if

Table VII. Definition of temporal types.

Event : (instant, finalInstant, periodicity, action)	Timer : (currentTime, eventQueue)
—instant: Number —finalInstant: Number —periodicity : Number —action: Boolean Function(Event)	—currentTime: Number —eventQueue: [Event]

true the event returns to the Timer queue at the position determined by its periodicity (event.instant = event.instant + event.periodicity), otherwise the event is permanently canceled.

During the simulation, the events are removed from the queue, the simulation current time is updated (currentTime = event.instant), and then the event action is performed. Eventually, the event will be rescheduled if its action returns true.

At the beginning of the simulation, all collections created by the modeler are synchronized through the TerraME's synchronize() function. That is, temporary copies of all cells in each collection are created, recording their immediate state. During the simulation, all readings are performed on the temporary copies of the attributes and the writes are performed directly on the attributes. This strategy ensures that all computations start from the same shared and consistent value, ensuring consistency of the simulations.

The flow operator is implemented according to Algorithm 1 that evolves in two stages: (1) Flow Execution - Behavioral rules (BehavioralRules) are executed, that is, TerraME iterates over all cells of the involved collections, applying the differential equations (flows) defined by the modeler that receive the temporary values as parameters, the results of the equations are written directly on the attributes of cells; (2) Synchronization - Temporary copies of the cells of the collections affected by the flow are updated instantly, causing the changes to be persisted and to be noticed by the next computations. All events present in the algorithm remain re-queued until the end time of the simulation is reached (timer.currentTime == finalSimulationTime).

Algorithm 1 FLOW

```
function FLOW(f(), a, b, step, colle1, attr1, neig1, colle2, attr2, neig2)
    –FLOW EXECUTION
    timer.add( Event(a, b, step, BehavioralRule()))
    –SYNCHRONIZATION
    timer.add( Event(a, b, step, synchronize(colle1, colle2)))
END FLOW
```

5. CASE STUDY

Five case studies are used to evaluate the usefulness, expressiveness, and computational efficiency of the TerraME implementation. Case studies present examples that are only offered for demonstration purposes. We focus on how to express spatial changes with the use of system dynamics extension and may omit some aspects of the problems, so they should not be considered as a reliable modelling solution for the described problem. Case study 1 uses a simplistic "Hello World" model to simulate fire propagation in a forest. Case study 2 simulates the water cycle exemplifying operations frequently used in the representation of continuous spatial changes. The case study 3 evaluates the response time of the simulator implemented in this work. Case study 4 simulates a water spring and the superficial water runoff, exemplifying flows with no energy source. The case study 5 simulates fire propagation conditioned by biomass presence. More detailed descriptions, as well as other examples and codes for reproduction of case studies can be found in [ExtraCases 2017].

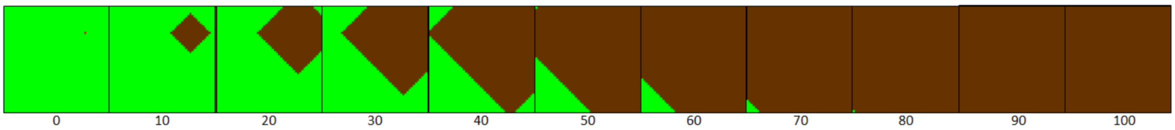

Fig. 4. Heat spread over the cellular space. Green (inert), Brown (Burning).

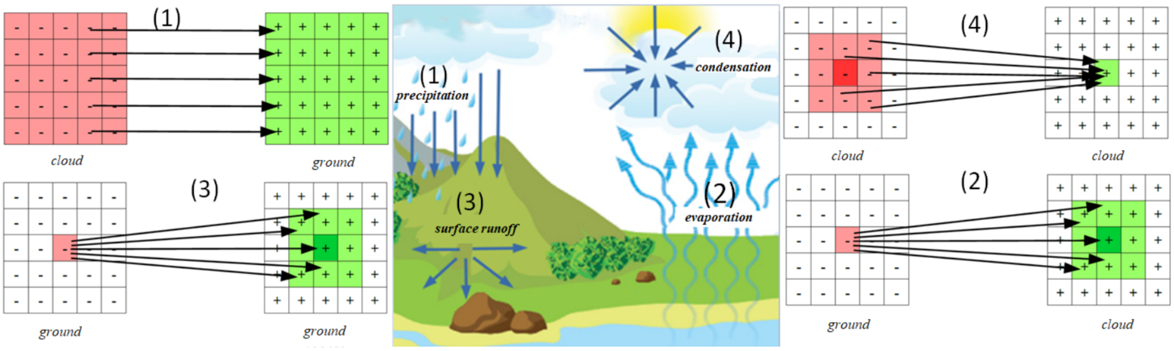

Fig. 5. Illustration of water cycle operations.

5.1 Case Study 1

Fire Spread model (Algorithm 2) is composed of a cellular space (lines 3-4) and a von Neumann neighborhood (line 5) through which fire will propagate. Each cell has the attribute heat that represents the thermal energy stored in it, initially equal to 0 (green). A cell is considered to be burning (brown) if its stock is greater than zero. A random fire starting point is created (line 6), whose value is 1. The flow operator (line 7) simulates heat going from burning cells to their neighbors according to the exponential differential equation defined in line 2. Figure 4 shows a series of images with the simulation result.

Algorithm 2 FireSpred Model

1: $dispersionRate = 0.99$
2: function dispersion $(t, stock)$ return $dispersionRate * stock$
3: createCell$(groundslice, heat = 0)$
4: createCellularSpace$(ground, groundslice, 50)$
5: createSpatialNeighborhood$(groundNeight, ground, vonneumann, nil, true)$
6: ground.RandomCell$(heat = 1)$
7: FLOW$(DISPERSION, 1, 100, 1, ground, heat, nil, ground, heat, groundNeight)$
8: Execute(100)

5.2 Case Study 2

The simplified water cycle model (Algorithm 3 and Figure 5) is composed of four flow operations: (1) Cloud water precipitation to the soil - local flow; (2) Evaporation of soil water to the clouds (with water vapor dispersion) - focal flow; (3) Surface runoff of soil water through neighborhood - focal flow; and (4) Condensation of water in the clouds - focal flow.

In this case study, both cloud and soil are represented by 5x5-sized cellular spaces, which work as water stocks. Algorithm 3 presents the complete model code. All flows defined in lines 16 to 19 have different start and end times. However, they use equal step intervals, 1. These flows have behavior governed by exponential differential equations defined in lines 5 to 8, whose rates are defined in lines 1 to 4.

Algorithm 3 Water Cicle Model

1: $precipitationRate = 0.2$
2: $evaporationRate = 0.1$
3: $surfacerunoffRate = 0.3$
4: $condensationRate = 0.5$
5: function precipitation $(t, stock)$ return $precipitationRate * stock$
6: function evaporation $(t, stock)$ return $evaporationRatee * stock$
7: function surfacerunoff $(t, stock)$ return $surfacerunoffRate * stock$
8: function condensation $(t, stock)$ return $condensationnRate * stock$
9: createCell$(groundslice, water = randon())$
10: createCellularSpace$(ground, groundslice, 100)$
11: createSpatialNeighborhood$(groundNeight, ground, moore, nil, true)$
12: createCell$(cloudslice, water = randon())$
13: createCellularSpace$(cloud, cloudslice, 100)$
14: createSpatialNeighborhood$(cloudNeight, cloud, vonneumann, nil, true)$
15: createSpatialNeighborhood$(cloudNeight5x5, cloud, nil, 5, true)$
16: FLOW$(precipitation, 2, 7, 1, cloud, water, nil, ground, water, nil)$
17: FLOW$(evaporation, 5, 16, 1, ground, water, nil, cloud, water, cloudNeight)$
18: FLOW$(surfacerunoff, 15, 19, 1, ground, water, nil, ground, water, groundNeight)$
19: FLOW$(condensation, 5, 16, 1, cloud, water, cloudNeight5x5, cloud, water, nil)$
20: Execute(20)

Flow operator 1 at line 16 transfers water from the cloud to soil, simulating rainfall. Flow operator 2 at line 17 simulates evaporation, transporting water from soil to cloud, so that each cell receives a proportion of water proportional to the weights they have in the neighborhood. Flow operator 3 at line 18 simulates the water surface runoff in the soil, transporting water from a cell to its neighbors. Finally, flow operator 4 at line 19 simulates the condensation of water in the cloud, transferring water from neighboring cells to the central cell.

Figure 6 and Figure 7 graphically display the volumes of water stored in the cloud and soil during simulation. Arrows indicate the start points of flows between cellular spaces. In Figure 7, precipitation during instants 2 to 7 causes ground darkening and cloud whitening. In Figure 6, from moment 5, evaporation reduces the slope of curves by combining its effects with the precipitation. After instant 7, continuous evaporation causes cloud darkening and ground whitening in Figure 7. The surface runoff (instants 15 to 19) and condensation (instants 5 to 20) make homogeneous the water stocks in the cells, observed in Figure 7.

5.3 Case study 3

Three abstract models were used to evaluate the computational efficiency of the implementation of algebra developed in this work, all have flows based on exponential differential equations: (1) Local - Containing a local flow; (2) Focal - Containing a focal flow; and (3) Case Study 2 - Containing the 4 flows described in Algorithm 3. Simulations were performed on the Ubuntu 12.04 operating system on an Intel Xeon (R) CPU E5620 2.40GHz x8 32GB memory. The graph in Figure 8 shows the CPU time consumed by the simulation during model execution for cell spaces containing up to two million cells.

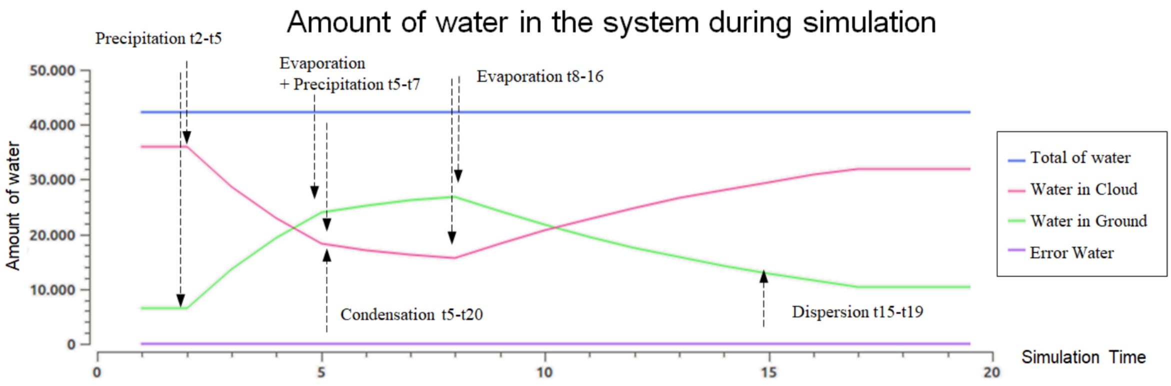

Fig. 6. Graph representing the total water quantity of the model.

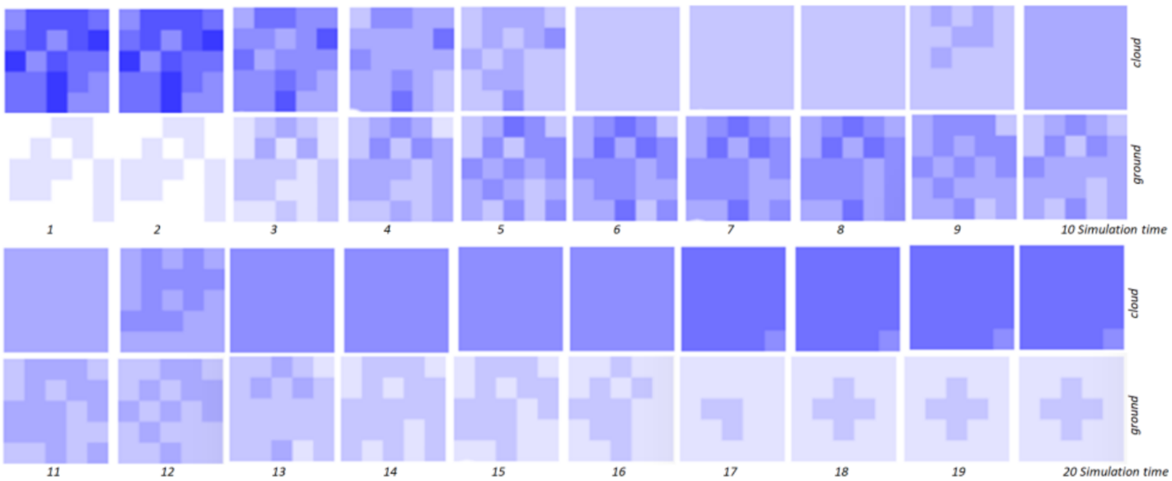

Fig. 7. Representation of the quantity of water contained in each cell of ground and cloud cellular spaces according with simulation time.

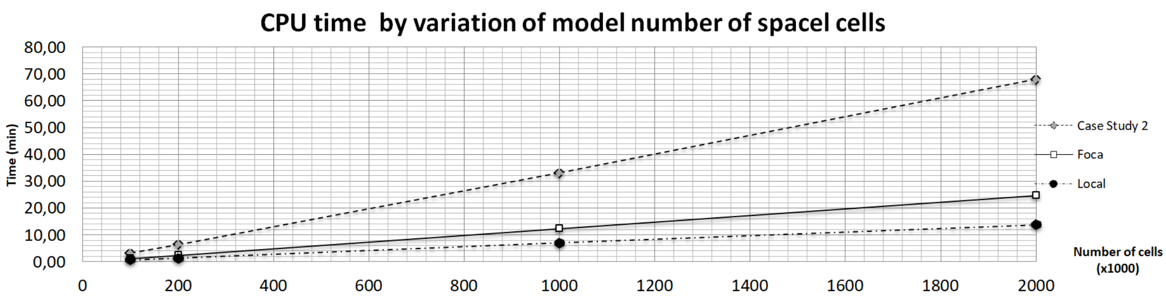

Fig. 8. CPU chart for simulations of large cell spaces.

5.4 Case study 4

Water spring and runoff model (Algorithm 4) is composed of a cellular space (lines 5-6), a trajectory that represents the water spring location (lines 8-9) and a conditional 3x3 neighborhood (line 7) that selects only cells at the same height or bellow a given cell. The attribute height of a cell varies from 0

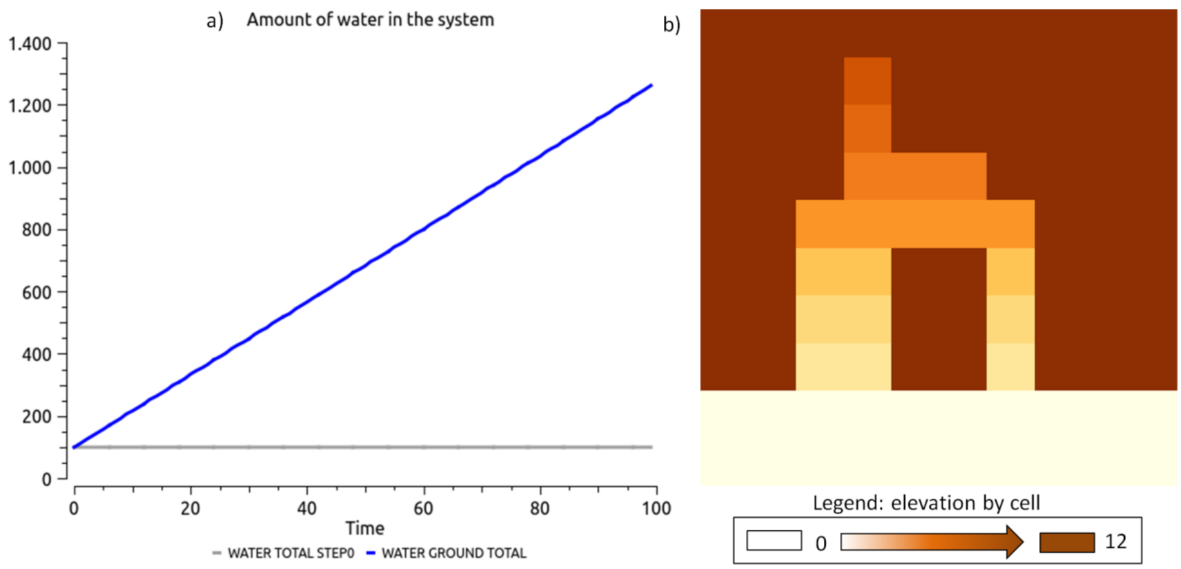

Fig. 9. a) Graph representing the total amount of water in the system; b) Representation of elevation of each cell of ground cellular spaces.

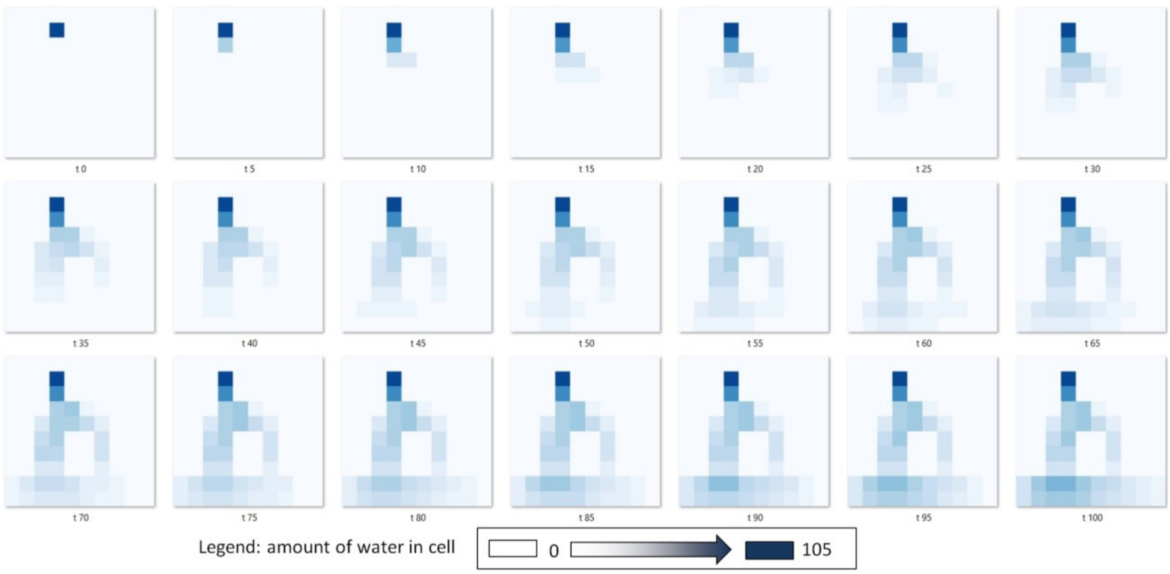

Fig. 10. Representation of the quantity of water contained in each cell of ground cellular spaces according to simulation time.

(white) to 12 (orange) (lines 10-31) (Figure 9-b). The attribute water represents the amount of water stored in the cell location, it varies from 0 (white) to 105 (dark blue). The model is composed of two flows operations: water spring flow - zonal flow from outside into the system (line 32), that simulate the input of water into space (trajectory); and the superficial water runoff - directed focal flow (line 33) that simulate the water dispersion respecting the conditional neighborhood.

This experiment illustrates the use of Flows with no source (or target) collections to simulate the inflow of energy (water spring) from the environment, Figure 9-a, into a system (soil surface) and the use of *Trajectories* as collections operands. It also illustrate how neighborhood filter and weight function (line 7) can be used to drive spatial flows of energy (water runoff from higher to lower cells). Figure 10 shows simulation result as a series of images.

Algorithm 4 Spring Water and Runoff Model

```
1: inputWaterRate = 0.116
2: runoffRate = 0.22
3: function risingWater (t, stock) return inputWaterRate * stock
4: function runoff (t, stock) return runoffRate * stock
5: Create Cell(groundslice, water = 0)
6: Create Cellular Space(ground, groundslice, 10)
7: Create Spatial Neighborhood(neighHeightGround3x3, ground, nil, 3, true, filter = function(cell, candidate) return cell.height >= candidate.height end, weight = function(cell, candidate) return (cell.height / candidate.height) end)
8: Create Trajectory(trajSpringWater, ground, select = function(cell)return
9: ((cell.x == 3 and cell.y == 1))end)
10: for EachCell (ground, function(cell) do
11:     cell.height = 12
12:     if (((cell.x == 3 and cell.y == 1)))) then
13:         cell.height = 10-cell.y+1 –spring water
14:         cell.water = 100
15:     end if
16:     if (((cell.x == 3 and (cell.y >= 2 and cell.y < 4)))) then
17:         cell.height = 10-cell.y+1
18:     end if
19:     if ((((cell.x >= 3 and cell.x <= 5) and (cell.y >= 3 and cell.y <= 3)))) then
20:         cell.height = 10-cell.y+1
21:     end if
22:     if ((((cell.x >= 3 and cell.x <= 6) and (cell.y >= 4 and cell.y <= 4)))) then
23:         cell.height = 10-cell.y+1
24:     end if
25:     if ((((cell.x >= 2 and cell.x <= 3) and (cell.y >= 4 and cell.y <= 7)) or (cell.x == 6 and (cell.y >= 4 and cell.y <= 7)))) then
26:         cell.height = 10-cell.y+1
27:     end if
28:     if ((cell.y > 7)) then
29:         cell.height = 1
30:     end if
31: end for
32: FLOW(risingWater, 1, 100, 1, nil, nil, nil, trajSpringWater, water, nil)
33: FLOW(runoff, 1, 100, 1, ground, water, nil, ground, water, neighHeightGround3x3)
34: Execute()
```

5.5 Case study 5

Fire Spread by Biomass model (Algorithm 5) consists of a cellular space (lines 5-6), a 3x3 neighborhood (lines 8) and a trajectory that selects only neighbors with biomass greater than 1 (line 7) through from which the fire will spread. Each cell has the heat attribute that represents the thermal energy stored in it, initially equal to 0 (green) and a random biomass value:-1, 1 or 2 (line 5). A cell is considered to be burning (white in Figure 12) if its heat attribute is greater than zero. Two random initial fire points are created (line 9-10), whose heat value = 1 (red). The flow operator (line 11) simulates a focal flow of heat from the cells to their neighbors, according to the exponential differential equation defined in line 3. A second flow operator (line 12) simulates a local flow of biomass burning in the cells,

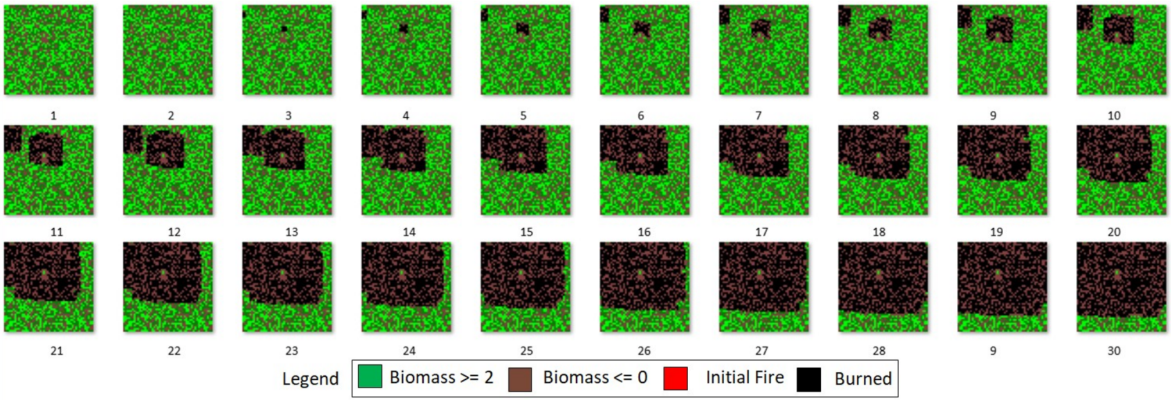

Fig. 11. Representation of fire propagation and biomass burning in the ground cellular space.

according to the exponential differential equation defined in line 4. The trajectory selection function is automatically updated at each simulation step, what makes the trajectory dynamic. Figure 11 and Figure 12 present a series of images with the result of the simulation. In particular, Figure 12 exemplifies the result of the variation heat attribute.

Algorithm 5 FireSpred Biomass Model - Summary

1: $dispersionRate = 0.99$
2: $burnRate = 1$
3: function dispersion $(t, stock)$ return $dispersionRate * stock$
4: function burn $(t, stock)$ return $burnRate * stock$
5: Create Cell$(groundslice, heat = 0, biomass = Random(-1, 1, 2), summaryHeat = 0)$
6: Create Cellular Space$(ground, groundslice, 50)$
7: Create Trajectory$(fireBorder, ground, select = function(cell) returncell.heat > 0$ and $cell.biomass >= 1\ end)$
8: Create Spatial Neighborhood$(neightBurn, ground, nil, 3, false)$
9: gorund.RandomCell$(heat = 1)$
10: gorund.RandomCell$(heat = 1)$
11: $FLOW(funcDisper, 1, 100, 1, fireBorder, "heat", nil, ground, "heat", "neightBurn")$
12: $FLOW(funcBurn, 1, 100, 1, fireBorder, "biomass", nil, nil, nil, nil)$
13: Execute()

6. FINAL CONSIDERATIONS

This paper proposes an algebra for the development of spatially explicit models of continuous changes that evolve in geographic space. This algebra extends the Systems Dynamics paradigm by introducing a set of algebraic operations similar to those in Map Algebra, allowing the representation of local, focal, and zonal flows. Experiments demonstrated how the algebra can be easily used to model and simulate scenarios containing several simultaneous and interleaved energy flows between heterogeneous regions of space. The algebra has good expressiveness and is able to concisely represent models where there are dependencies between variables of several differential equations, that is, feedback loops. Briefly, the algebra contributions can be listed as:

(1) Allowing the definition of rules of behavior in a declarative way;
(2) Providing operators that act on a high level of abstraction, which use collections of cells as operands;

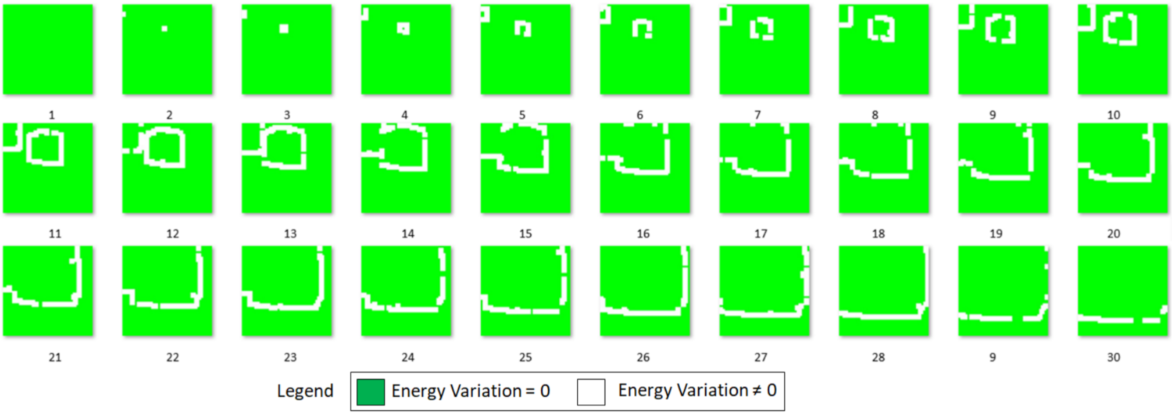

Fig. 12. Representation of the variation of the heat attribute.

(3) Allowing the representation of local, focal and zonal spatial flows;

(4) Shifting the modeler's focus from model implementation to its conception and design, since operators encapsulate implementation difficulties;

(5) Allowing to model and to simulate changes involving spatio-temporal discretizations of different scales (extent and resolution).

(6) Providing representation of input and output of energy in the system.

(7) Allowing to model and simulate conditional and dynamic neighborhoods.

The algebra has sufficient expressiveness to represent spatial and continuous diffusive processes of several natures without requiring the modeler to construct algorithms to avoid the propagation of computational errors or inconsistencies that may arise due to the order in which the space or neighborhoods are traveled. Corroborating with Schmitz et al. [2013] we also believe that high-level modeling languages facilitate the modeling process by reducing the programming fundamentals required during model development, reducing errors arising from implementation of feedbacks. The proposed algebra also allows synchronization of simultaneous flows and mechanisms to avoid the propagation of errors due to numerical integrations methods. The results also show that it is possible to use personal computers to simulate flows between millions of cells in a reasonably time. The simulation times grow linearly with the number of cells. Future work includes evaluating the use of this algebra for modeling and simulation of other models found in literature and improving current implementation to simulate large-scale models over high-performance hardware architectures.

REFERENCES

Ahmad, S. and Simonovic, S. P. Spatial system dynamics: new approach for simulation of water resources systems. *Journal of Computing in Civil Engineering* 18 (4): 331–340, 2004.

Amâncio, A. F. and Carneiro, T. G. d. S. An algebra for modeling and simulation of continuous spatial changes. *GeoInfo XVIII - Brazilian Symposium on Geoinformatics* 69 (5): 971–980, 2017.

Burrough, P. A. Dynamic modelling and geocomputation. In *Geocomputation: a primer*. Chichester, UK: John Wiley & Sons, John Wiley & Sons, Chichester, UK, pp. 165–192, 1998.

Camara, G., Egenhofer, M. J., Ferreira, K., Andrade, P., Queiroz, G., Sanchez, A., Jones, J., and Vinhas, L. Fields as a generic data type for big spatial data. In *International Conference on Geographic Information Science*. Springer, In International Conference on Geographic Information Science, Cham, pp. 159–172, 2014.

Camara, G., Palomo, D., de Souza, R. C. M., and de Oliveira, O. R. F. Towards a generalized map algebra: Principles and data types. In *GeoInfo*. In GeoInfo, Brazil, pp. 66–81, 2005.

Cardelli, L. Type systems, handbook of computer science and engineering, chapter 103, 1997.

Carneiro, T. G. d. S., DE Andrade, P. R., Câmara, G., Monteiro, A. M. V., and Pereira, R. R. An extensible toolbox for modeling nature–society interactions. *Environmental Modelling & Software* vol. 46, pp. 104–117, 2013.

Cordeiro, J. P. C., Câmara, G., De Freitas, U. M., and Almeida, F. Yet another map algebra. *Geoinformatica* 13 (2): 183–202, 2009.

Couclelis, H. From cellular automata to urban models: new principles for model development and implementation. *Environment and planning B: Planning and design* 24 (2): 165–174, 1997.

d'Aquino, P., August, P., Balmann, A., Berger, T., Bousquet, F., Brondízio, E., Brown, D. G., Couclelis, H., Deadman, P., Goodchild, M. F., et al. Agent-based models of land-use and land-cover change. In *Proc. of an International Workshop*. Report and Review of an International Workshop. L. R. No.6., Irvine, California, USA, pp. 4–7, 2002.

Egenhofer, M. J. and Herring, J. Categorizing binary topological relations between regions, lines and points in geographic databases, the 9-intersection: Formalism and its use for naturallanguage spatial predicates. *Santa Barbara CA National Center for Geographic Information and Analysis Technical Report* 1 (1): 94–1, 1994.

Elsawah, S., Pierce, S. A., Hamilton, S. H., Van Delden, H., Haase, D., Elmahdi, A., and Jakeman, A. J. An overview of the system dynamics process for integrated modelling of socio-ecological systems: Lessons on good modelling practice from five case studies. *Environmental Modelling & Software* vol. 93, pp. 127–145, 2017.

ExtraCases. Extra case study of an algebra for modeling and simulation of continuous spatial changes. http://bit.ly/2gCmXrg, 2017.

Forrester, J. W.(1961). industrial dynamics, 1960.

Frank, A. U. One step up the abstraction ladder: Combining algebras-from functional pieces to a whole. In *International Conference on Spatial Information Theory*. Springer, Springer, Berlin, Heidelberg., pp. 95–107, 1999.

Frank, A. U. Map algebra extended with functors for temporal data. In *International Conference on Conceptual Modeling*. Springer, Springer, Berlin, Heidelberg., pp. 194–207, 2005.

Karssenberg, D., Burrough, P. A., Sluiter, R., and de Jong, K. The pcraster software and course materials for teaching numerical modelling in the environmental sciences. *Transactions in GIS* 5 (2): 99–110, 2001.

Kelly, R. A., Jakeman, A. J., Barreteau, O., Borsuk, M. E., ElSawah, S., Hamilton, S. H., Henriksen, H. J., Kuikka, S., Maier, H. R., Rizzoli, A. E., et al. Selecting among five common modelling approaches for integrated environmental assessment and management. *Environmental modelling & software* vol. 47, pp. 159–181, 2013.

Maxwell, T. and Costanza, R. An open geographic modeling environment. *Simulation* 68 (3): 175–185, 1997.

Meadows, D. H. *Thinking in systems: A primer.* chelsea green publishing, USA, 2008.

Morgan, J. S., Howick, S., and Belton, V. A toolkit of designs for mixing discrete event simulation and system dynamics. *European Journal of Operational Research* 257 (3): 907–918, 2017.

Sahin, O. and Mohamed, S. Coastal vulnerability to sea-level rise: a spatial–temporal assessment framework. *Natural hazards* 70 (1): 395–414, 2014.

Schmitz, O., Karssenberg, D., De Jong, K., De Kok, J.-L., and De Jong, S. M. Map algebra and model algebra for integrated model building. *Environmental modelling & software* vol. 48, pp. 113–128, 2013.

Silva, W. S. d. F. and Carneiro, T. G. d. S. An algebra for modelling the simultaneity in agents behavior in spatially explicit social-environmental models. *Brazilian Journal of Cartography* 69 (5): 260–272, 2017.

Swinerd, C. and McNaught, K. R. Design classes for hybrid simulations involving agent-based and system dynamics models. *Simulation Modelling Practice and Theory* vol. 25, pp. 118–133, 2012.

Takeyama, M. Building spatial models within gis through geo-algebra. *Transactions in GIS* 2 (3): 245–256, 1997.

Tomlin, C. D. *Geographic information systems and cartographic modeling.* Prentice Hall, Prentice Hall, 1990.

Vincenot, C. E., Giannino, F., Rietkerk, M., Moriya, K., and Mazzoleni, S. Theoretical considerations on the combined use of system dynamics and individual-based modeling in ecology. *Ecological Modelling* 222 (1): 210–218, 2011.

Wainer, G. A. *Discrete-event modeling and simulation: a practitioner's approach.* CRC press, press, 2009.

Wesselung, C. G., KARSSENBERG, D.-J., Burrough, P. A., and DEURSEN, W. Integrating dynamic environmental models in gis: the development of a dynamic modelling language. *Transactions in GIS* 1 (1): 40–48, 1996.