GCA-w ALGORITHMS FOR TRAFFIC SIMULATION*

Rolf Hoffmann

Technische Universität Darmstadt, FG Rechnerarchitektur
Hochschulstr. 10, 64289 Darmstadt, Germany
hoffmann@ra.informatik.tu-darmstadt.de

(Received March 31, 2011)

The GCA-w model (Global Cellular Automata with write access) is an extension of the GCA (Global Cellular Automata) model, which is based on the cellular automata model (CA). Whereas the CA model uses static links to local neighbors, the GCA model uses dynamic links to potentially global neighbors. The GCA-w model is a further extension that allows modifying the neighbors' states. Thereby, neighbors can dynamically be activated or deactivated. Algorithms can be described more concisely and may execute more efficiently because redundant computations can be avoided. Modeling traffic flow is a good example showing the usefulness of the GCA-w model. The Nagel–Schreckenberg algorithm for traffic simulation is first described as CA and GCA, and then transformed into the GCA-w model. This algorithm is "exclusive-write", meaning that no write conflicts have to be resolved. Furthermore, this algorithm is extended, allowing to deactivate and to activate cars stuck in a traffic jam in order to save computation time and energy.

DOI:10.5506/APhysPolBSupp.4.183
PACS numbers: 87.17.Aa, 92.60.hk, 87.18.Hf

1. Introduction

The GCA-w parallel computing model (GCA with write access) introduced in [2, 3, 4] is an extension of the GCA (Global Cellular Automata) model [5, 6], which in turn is an extension of the CA model. A cell of a GCA can dynamically establish links to any of its global neighbors, whereas a cell of a CA uses only the fixed links to its local neighbors. The CA and GCA model do not allow modifying the state of a neighbor. Therefore, no write conflict can occur, simplifying implementations in hardware or in software. However, for applications, where the amount of active cells in the whole

* Presented at the 2nd Summer Solstice International Conference on Discrete Models of Complex Systems, Nancy, France, June 16–18, 2010.

field is low or is varying over time, or the locations of the active cells are changing, the GCA-w model is a better choice. The GCA-w model allows writing information to its neighbors. This feature is very important because information can actively be transferred to a destination, and the activity of the destination can be switched on or off. Therefore, the GCA-w model is very useful for the description of problems with moving particles or moving agents, or problems with dynamic activities.

Several GCA-w applications were already described in [3, 4] (one-to-all communication, synchronization, moving agents, different random walks of particles, pointer inversion, sorting with pointers, Pascal's Triangle). The purpose of this contributions is not to show all possible features and applications of the model, but rather to show another practical application that can easily be implemented because no write conflicts occur.

It will be shown that the CA based traffic simulation rule (Nagel –Schreckenberg [1]) can easier be modeled and more efficiently be executed using the GCA-w model. This problem is perfectly suited to the GCA-w model because *(i)* the cars decide on their own upon their next location, *(ii)* cars are not moving to the same location (car collisions are excluded by the rules, no conflicts will occur), *(iii)* the number of active cells (cars) is usually much lower than the number of passive cells (spaces), and *(iv)* waiting cars can be deactivated and activated again.

1.1. Informal description of the GCA-w model

Fig. 1 shows the general idea of the GCA-w model: Each cell C is dynamically connected via links h_i to other cells, in the example to A, B, D. Cell C

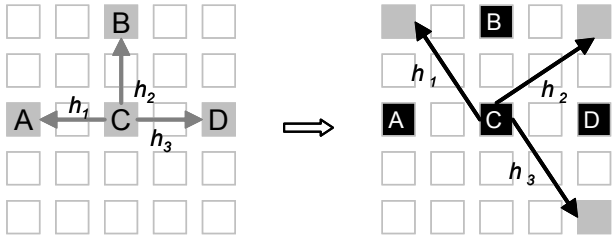

Fig. 1. A 2-D GCA-w example with three links per cell: Each cell is dynamically connected to a global (or locally restricted) set of neighbors (gray), only the activity of the center cell C is shown. The state of C including the links h_i, and the states of its neighbors can be changed (gray to black) by a local rule. Thereby, information can be transferred from C to the current neighbors A, B, D, and the activity of the neighbors may be changed. Write conflicts may occur, *e.g.* if C itself and other cells try to update C at the same time.

updates its own state and the states of its neighbors A, B, D. Thereby the links h_i are also updated. In the following, only one link per cell is assumed, which seems to be sufficient to model most applications. In addition, in the following, the cells will be arranged in a 1-D fashion, although n-D fields can easily be handled by using an n-D indexing scheme.

A potential difficulty is that write conflicts may appear. The worst case scenario is that all cells want to write onto the same cell. In order to reduce the implementation effort to resolve the conflicts, the GCA-w rules should be designed in such a way that either no conflict will occur, or that the amount of conflicts is low or restricted, or that the conflicts can be resolved within a local neighborhood.

In Fig. 2 some situations are depicted, showing some possible interactions (reads and writes) between cells. In situation (b) cell i reads j, and modifies its own state and the state of its neighbor j (the basic "idea" of GCA-w). In situation (c) cell i modifies itself and is modified by j and k, a conflict with three write accesses occurs that has to be resolved. Situation (d) is a "exclusive-write situation": Cells i and j exchange data, and cell p modifies cell k and p. No write conflicts occur. In the following the GCA-w model is described in detail.

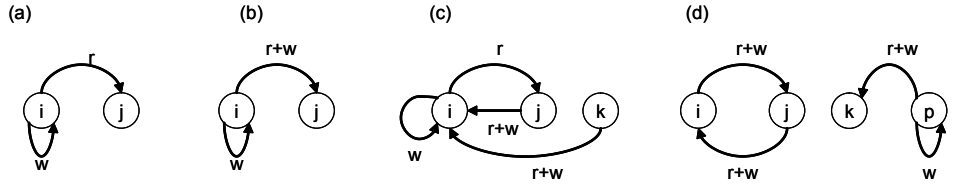

Fig. 2. Some read/write situations: (a) Cell i reads from cell j and modifies its own state; (b) Cell i reads j and modifies its own state and the state of its neighbor j (the basic "idea" of GCA-w); (c) Cell i modifies itself and is modified by cell j and k, but there occurs a conflict of three write accesses which has to be resolved; (d) Cell i modifies j and *vice versa*. Cell p modifies cell k and p. No write conflicts occur (exclusive write situation).

1.2. Formal description of the GCA-w model

Global Cellular Automata with Write Access: GCA-w= $(I, A, B, \delta, h, f, g, e)$

$$\text{Index Set} : I = \{0, 1, \ldots, i, \ldots, N-1\} \tag{1}$$
$$\text{Active States} : A = \{a_1, a_2, \ldots, a_n\} \tag{2}$$
$$\text{Passive States} : B = \{b_1, b_2, \ldots, b_m\} \tag{3}$$

$$\text{Don}'\text{t} - \text{Write} - \text{Symbol respectively Dead} - \text{State} : \delta \tag{4}$$
$$\text{States} : Q = A \cup B \cup \{\delta\} \tag{5}$$
$$\text{Configurations} : Q^N \tag{6}$$
$$\text{A Configuration} : L = (q_0, q_1, \ldots, q_i, \ldots, q_{N-1}) \in Q^N, i \in I \tag{7}$$
$$\text{Neighbor}'\text{s Address (in case of absolute addressing) } h(i,q) : \\ h : I \times Q \to I \tag{8}$$
$$\text{Neighbor}'\text{s Address (in case of relative addressing) } h^{\text{rel}}(i,q) : \\ h^{\text{rel}} : I \times Q \to I^{\text{rel}} = \{0, \pm 1, \pm 2, \ldots, \pm(N-1)\} \tag{9}$$
$$\text{such that } h(i,q) = (i + h^{\text{rel}}(i,q)) \text{mod } N \tag{10}$$
$$\text{Local} - \text{Rule } f(i,q,q^*), q^* = \textit{neighbor}'\textit{s state} : \ f : I \times Q \times Q \to Q \tag{11}$$
$$\text{Write} - \text{Rule } g(i,q,q^*) : \ g : I \times Q \times Q \to Q \tag{12}$$
$$\text{Conflict} - \text{Rule } e(i,f,g^0,g^1,\ldots,g^j,\ldots,g^{N-1}) : \ e : I \times Q^{N+1} \to Q \tag{13}$$

Rule Application (Synchronous Updating) $\forall i \in I$:

$$q_i := \begin{cases} e(i, f(i, q_i, q_{h(i,q_i)}), g^0_{\to i}, \ldots, g^j_{\to i}, \ldots, g^{N-1}_{\to i}) & \text{IF } q_i \in A \quad (14) \\ \delta & \text{IF } q_i = \delta \quad (15) \\ e(i, \delta, g^0_{\to i}, \ldots, g^{j=i}_{\to i} = \delta, \ldots, g^{N-1}_{\to i}) & \text{IF } q_i \in B \quad (16) \end{cases}$$

where $\forall (i,j) \in I \times I$:

$$g^j_{\to i} = \begin{cases} g(j, q_j, q_{h(j,q_j)}) & \text{IF } h(j,q_j) = i \quad \text{AND} \quad q_j \in A \quad (17) \\ \delta & \text{IF } h(j,q_j) \neq i \quad \text{OR} \quad q_j \notin A \quad (18) \end{cases}$$

The cells are arranged as a sequence $\langle a_i \rangle_{i \in I}$ of cells, each cell is labeled by i (1). The constant label can also be accessed by the cell itself in order to use non-uniform rules depending on i. A cell can be in an active state (2), or in a passive/inactive state (3), or in a dead-state δ. These three classes of states are also called *operational states*. These three operational states are distinguished in order to switch on or off the activity of cells and thereby allowing to reduce the computational effort (dead cells are totally excluded from the computation, because they remain dead forever, passive cells do not compute themselves but can be activated by other cells). In addition, passive or dead cells can be used to define a termination condition, *e.g.* if all cells are dead, or if all cells are passive.

If a cell is active, it computes all local functions h, f, g, e. If a cell is passive, it does not compute its local functions, but it can be switched into another operational state from outside, *e.g.*, it can be changed into active. If a cell is dead, it will stay dead forever.

The symbol δ has two interpretations: *(i)* it denotes the command "Don't Write", mainly used as output of the function g in order to avoid writing to a neighbor, and further *(ii)* it encodes the dead-state.

The address function h defines the actual neighbor (absolute address, index) in access (read and write), see also Fig. 3. The neighbor's address can also be determined by the use of the relative addressing function h^{rel} (9), (10). Relative addressing is often more adequate and more general to describe spatial relative situations. The local rule f computes the cell's new state in case of no conflict (similar to a CA or GCA rule). The *write-rule* computes a state value that can be written to another cell (including the own cell, a special case). The *conflict-rule* is dedicated to resolve the conflicts. It receives $N + 1$ messages (write-values), the own rule value f and messages $g^j_{\to i}$ from all cells j. Not all messages need to be activating: A non-activating δ-message is interpreted as a Don't Write-command, meaning that a sender j does not want to write to a receiver i. A message is activating if the sender j is active and the receiver i is selected (17). (As a special case, g may produce δ; also f may produce δ.) If the sender is passive or the receiver is not selected, then δ is received.

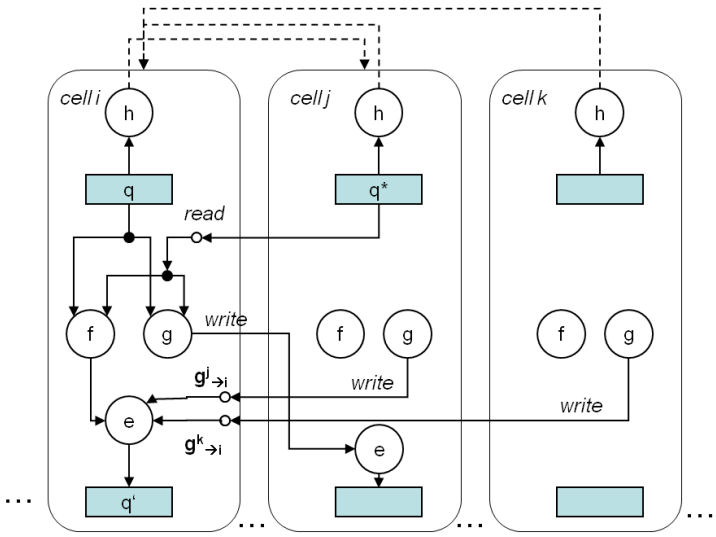

Fig. 3. Cell i selects cell j as neighbor using the address function h. The next own data f and the write data g to be stored in j are computed by cell i. Write conflicts may occur and have to be resolved by the local rule e.

Rule Application. All cells are updated synchronously in parallel. Only cells that are not dead need to be updated. If the cell is active (14) then the own rule value f and the messages $g^j_{\to i}$ from all cells j have taken to be into

account. In case that a δ-message (Don't Write) is received, it is disregarded by the conflict rule e. If the cell is passive (16) then the no computation of h, f, g takes place and the default values $f = \delta$ and $g = \delta$ are assumed, and no neighbor is selected. Although the cell is passive, the conflict-rule has to be awake because there might arrive activating messages.

At a first glance the GCA-w model seems to be too complex because of the conflicts and not very useful. But in many applications the complexity of the conflict resolution can be reduced significantly, *e.g.* if the dynamic neighborhood access patterns are restricted or are known in advance, or if the rules are defined in an *Exclusive-Write* way, meaning that no write conflicts are induced by the "GCA-w" algorithm (defined by the local functions). The following car movement algorithm is such an exclusive-write algorithm.

In the case of modeling cars with conflicts, two phases have to be used: *(i)* send concurrent requests which are resolved by the target cell, *(ii)* the result of the arbitration is checked by the requesters, and only the winner will move the car (delete on own position and write to target position). Another solution would be the introduction of two subphases (for this case the model has to be extended): *(i)* send concurrent requests to the target, evaluate the arbitration and return acknowledge signals, *(ii)* the winner (acknowledge = true) moves the car.

The main advantage of the model is that it allows to describe a certain class of algorithms more concise, adequate and less redundant, because of the write access and the activity control. In the CA or GCA model the cells might be switched off, too, but they cannot be switched on again. The GCA-w model can also be used to express indirect communication [12], *e.g.* some cells may share a common communication cell they write information onto to be distributed. Writing to another variable or object is a very common technique, *e.g.* in classical programming languages, therefore this technique is not new as such. The novel idea is to organize the computational task logically as an array of cells with different operational states (active, passive, dead) with write-access onto neighbors using only locally defined rules.

The GCA model can be seen as a submodel of GCA-w. We receive the GCA model from the GCA-w model by:

- No δ symbol is necessary, $Q = A \cup B$.
- Active cells can be passivated.
- No write access to the neighbor is allowed, therefore no write-rule g and no conflict-rule e exists. Access to the neighbor is read-only, write conflicts are not possible like in CA.

- The local-rule f is always used to update the state of the cell ($q := f$), in case that the cell is active. In case that a cell is passive, it remains passive and needs not to be recomputed.

1.3. Related work

The PSA model [9] of computation is a very general and powerful model based on *substitution rules*. It allows also modifying the state of arbitrary target cells (*right side* of the substitution) using a *base* and a *context*. In relation to the GCA-w the base corresponds to the cell under consideration, the context corresponds to the read neighbors and the right side corresponds to the cells which are modified. There is also a relation to the CRCW-PRAM [10,11] model. The PRAM model is based on a physical view with p processors that have global memory access to physical data words whereas the GCA-w is based on logical computing cells tailored to the application. Another difference of the GCA-w model compared to PRAM is the direct support of dynamic links and the rule based approach similar to the CA model.

2. Modeling car movements

We will restrict our model to a single lane with cyclic boundary. N is the length of the lane (number of cells). The maximal speed of a car is $v_{\max}$. Thus, a car can move from its current position i to the position $i + v_{\max}$ at most. The current speed of the car is v. The distance to the car in front is d. A random variable R is available in each cell, changing from generation to generation: $R = 1$ with probability p, and $R = 0$ with probability $(1-p)$. R is used to describe the probabilistic behavior of a driver not to accelerate the car although it is possible, or to slow down the car randomly.

We distinguish Empty (E) cells and Agent (A) cells. In this context, the term *agent* and *car* are used synonymously. We assume that the cars are moving from the left to the right (cyclically).

We will also use the term CA, GCA, or GCA-w algorithm. A CA, GCA, or GCA-w algorithm is given by the set of all cells, its associated rules, and its initial configuration. Such an algorithm based on the local cell's behavior induces a certain global behavior on the whole state of all cells in the field. In order to shorten the algorithms, the state of a synchronous variable in generation $t + 1$ remains the same as in generation t if it is not changed explicitly.

2.1. Nagel–Schreckenberg algorithm and CA rule

The well-known algorithm [1] describes the movement of cars for traffic simulations. The car has a speed v and computes in the steps A1–3 its new speed v'. The new speed is then used in step B to move the car. Note that all cars are moved synchronously in step B.

(A1) **Acceleration.** Increment the current speed v if the maximum speed $v_{\max}$ of the car is not yet reached: $v' = \min(v + 1, v_{\max})$.

(A2) **Slowing down due to the car ahead.** If the gap (distance) d to the preceding car is less than the new speed, reduce it to the size of the gap: $v' = \min(v', d)$.

(A3) **Randomization.** Reduce the new speed by one with probability p, but not below zero: $v' = \max(0, v' - R)$.

(B) **Movement.** Move each car by v' sites (next position is $i + v'$). Set the next speed v (for the generation $t + 1$) to the new speed v' just computed in the current generation t through steps A1–3.

This algorithm can be described in a compact form by the following rule

$v' = \max(0, \min(v + 1, v_{\max}, d) - R))$ *//compute new speed, not* < 0
$AgentAt[i + v'] \leftarrow AgentAt[i]$ *//synchronous move and update*
$v[i + v'] \leftarrow v'[i]$

Note that this rule is not a CA rule because the position of the cell (representing a car) is changed. Nevertheless this rule can be transformed into a CA rule: A cell is dynamically either of type E (Empty) or A (Agent). An agent cell has to be connected to $v_{\max}$ neighbors to the right (the view of the car in order to detect another car in front). An empty cell has to be connected to $v_{\max}$ cells to the left in order to copy a car from the left that wants to move to C. As a cell may be either of type E or A, the neighborhood is the union of the two neighborhoods resulting in a neighborhood of $\pm v_{\max}$. The cell's structure is (Type, v, R). R is a random variable (0 or 1) that is updated in every generation. Temporary variables are: the new speed v', the distance (gap) d to the preceding car, and the distance q from an empty cell back to the following car. The resulting CA rule is the following:

```
if Type = A then
(a) determine d by checking all cells to the right within vmax
(b) compute new speed v'
(c) update: if v' > 0 then Type <- E endif  // delete agent
endif
```

```
if Type = E then
(a) Check all cells to the left within vmax in order to detect an
    agent that wants to move to the own empty cell. Determine the
    distance q to that agent (if there is any)
(b) If an agent is detected at position (i-q) then compute its new
    speed v' using v and R of cell (i-q); then compute its new
    position (i-q+v').

(c) if (v' = q) then
    // agent's next position is the own position, (i-q+v') = i
    // copy agent, use v' of cell at (i-q)
    Type <- A,   v <- (i-q).v' // copy and sync update
    endif
endif
```

Compared to the original algorithm, the CA rule is more complex and not so easy to design and to understand. If the CA algorithm is implemented as a sequential program, the worst case time complexity depends linearly on the neighborhood distance, because an agent has to check the $\min(v_{\max}, d)$ cells to the right, and an empty cell has to check $\min(v_{\max}, q)$ cells to the left. Furthermore, the next speed v' has to be computed by the agent, and also by all the empty cells to the right of the agent within the neighborhood. This redundant computation could be avoided by using a temporary variable holding v', which is computed only once (in phase 1), and then can be accessed (in phase 2) by the empty cells in front of the agent, too.

If the rule were implemented fully parallel in hardware, a lot of hardware resources would be needed because of the wide neighborhood (for real traffic simulations $v_{\max}$ is around 10). Therefore, even in hardware a partial sequential implementation would be more cost effective.

2.2. GCA algorithm

In [7] we have described a GCA algorithm for the same problem. The advantage of this algorithm is that it is less time consuming because the searching to the right and the searching to the left is not necessary, because the relevant positions can directly be computed. The idea is to use a linked list, which is synchronously updated. Each agent is linked to its agent in front, and each empty cell is linked to its agent behind it. In contrast to the description in [7], which is vector based, the algorithm presented here is based on the cell's local view. It is described as a GCA rule to be applied to each cell. In addition, some of the rule conditions are described in more detail.

The cell's state is $(\text{Type}, R, L, v, z)$. R is a random variable. L is the link (pointer). The other variables are only relevant if the cell's type is A. The speed is stored in $v(t)$, and $z(t)$ holds the new speed $v'(t) = v(t+1)$,

already pre-computed in the previous generation $(t-1)$, where t is the current generation.

In the following GCA algorithm, absolute addressing is applied, denoted by ":". *E.g.* $L+L:z$ means $L[i]+z[L[i]]$, where i is the absolute cell's index (address). Fig. 4 shows for an example how the agents and empty cells are connected.

```
if Type = E then
   // E at i is skipped by A, and left cell of agent is E
   if (L+L:z)>i and (L-1):Type = E then
   // L will point to moved leftleft A
   L <- (L-1):L + (L-1):L:z  // (L-1):L points to leftleft A

   // E is skipped by A, and left cell of agent is A
   elseif (L+L:z)>i and (L-1):Type = A then
   // L will point to leftleft A that is behind left A
   L <- (L-1)

   // E is not skipped by A
   elseif (L+L:z)<i then
   // L will point to left A
   L <- L + L:z

   // if A moves to E
   elseif (L+L:z)=i then
   Type <- A // agent is copied
   // L will point to new position of A in front
   L <- L:L + L:L:z  // L:L points to following A
   v <- L:z  // precomputed speed L.z is copied from left A
   // min(v+1, d, vmax), L.L points to A in front
   z <- max(0, min(L:z + 1, L:L + L:L:z - 1, vmax) - R)
   endif
endif

if Type = A then
   // z>0 and left cell E : A moves
   if z>0 and (L-1):Type = E then
   Type <- E
   // L will point to new position of following A
   L <- (L-1):L + (L-1):L:z  // (L-1):L points to following A

   // z>0 and left cell A then A moves
   elseif z>0 and (L-1):Type = E then
   Type <- E
   // L will point to left A that is directly behind
   L <- (L-1)
```

```
   // z=0 : A does not move
   elseif z=0 then
   // L will point to agent in front, moved by z
   L <- L + L:z  // L points to agent in front
   v <- z        // use precomputed speed
   // min(z+1, d, vmax), L+L.z-1 = future dist. to agent in front
   z <- max(0, min(1, L + L:z - 1, vmax) - R)
   endif
endif
```

This rule is quite sophisticated and it was not so easy to design it. The advantage of this rule compared to the CA rule is that the checking of all the neighbors in the $\pm v_{\max}$ neighborhood is not necessary because the entire next cell state can directly be computed by the use of the links. Therefore, this algorithm will run faster if it is sequentially simulated, especially if the car density is low.

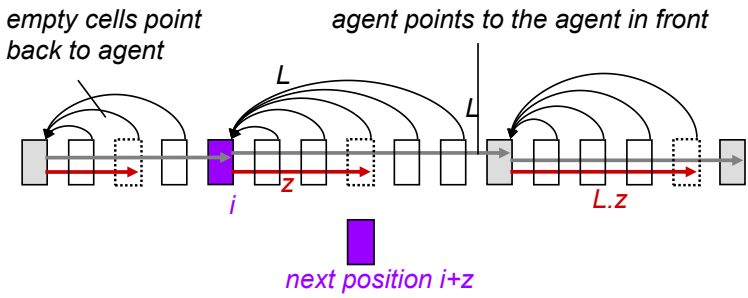

Fig. 4. GCA algorithm. The new speed of the agent was computed in advance and stored in the variable z. The agent will be situated at $(i+z)$ in the next generation.

But there is still a weakness in this algorithm because all cells have to be computed ($O(N)$) (redundant computations of the empty cells). Therefore, a more effective GCA-w algorithm will be presented in the next section.

2.3. GCA-w algorithms

The GCA-w model allows writing to a neighbor. This feature is in particular useful for this problem because the movement of an agent can be described by the cell with type A only. An agent can "beam" itself to the target position and delete itself on the source position. The Nagel–Schreckenberg algorithm is collision-free (no agent can move to the same site), and this feature is preserved when this algorithm is mapped onto the GCA-w model. No write conflicts can appear, and the implementation has not to care about detecting or solving conflicts. Therefore, the hardware or software interpret-

ing the algorithm can be kept simple. The GCA-w model was also used in [8] for the modeling of multi-lane traffic.

GCA-w algorithm with absolute links. The following GCA-w rule can directly be derived from the GCA rule, using only the parts of the rule that describe the moving. The cell's state is $(Type, R, L, v, z)$ as in the GCA rule. Now, the neighbors' state variables x in L, v, z, at position $i + z$, denoted by $z.x$ (meaning access relative to the own position i, $z.x = x[i + z]$), can directly be modified. Thus, the agent beams itself to its next position $i + z$. $(L : z)$ means: access the agent in front via the absolute link L and then read the component z, respectively $L : z = z[L[i]]$.

```
if Type = A then
   if z>0 then  // agent moves
     Type <- E  // delete, L is no longer relevant
     z.Type <- A // agent is beamed to i+z
     // z.L will point to new position of A in front
     z.L <- L + L:z  // L points to A in front
     z.v <- z  // precomputed speed z is written to own position +z
     // min(v+1, d, vmax), L points to A in front
     z.z <- max(0, min(z + 1, (L-i-z) + L:z - 1, vmax) - R)

   elseif z=0 then  // agent does not move
     // L will point to agent in front, moved by z
     L <- L + L:z  // L points to agent in front
     v <- z        // use precomputed speed
     // min(z+1, d, vmax), L+L.z-1 = future dist. to agent in front
     z <- max(0, min(1, (L-i) + L:z - 1, vmax) - R)
   endif
endif
```

This GCA-w algorithm is much shorter and simpler than the GCA algorithm. In addition, it is more efficient because only the agent cells have to be active and no redundant computations are performed. Thus the computing complexity is only $O(z)$, where z is the number of agents. The above algorithm can be simplified (integration of the cases for $(z > 0)$ and $(z = 0)$, and with $z.x = x$ if $z = 0$):

```
if Type = A then
   if z>0 then  Type <- E, z.Type <- A endif
   z.L <- L + L:z
   z.v <- z
   z.z <- max(0, min(z + 1, (L-i-z) + L:z -1, vmax) - R)
endif
```

Note that this algorithm uses relative addressing $z.L = z[i + L[i]]$ and absolute addressing $L : z = z[L[i]]$.

GCA-w algorithm with relative links only. The algorithm above uses absolute addressing for the link L, denoted by $L : z = z[L]$. An equivalent description, using relative addressing for L, $L.z = z[i + L]$, is the following algorithm. This description is "cell based", meaning that the point of view is any cell, and in the description no cell index i is used. Note that $L - 1$ is the actual gap size between two cars, and $L - z + L.z - 1$ is the gap in the following generation, when both cars will have moved. Fig. 5 shows the movements of two cars using the pre-computed new speed z.

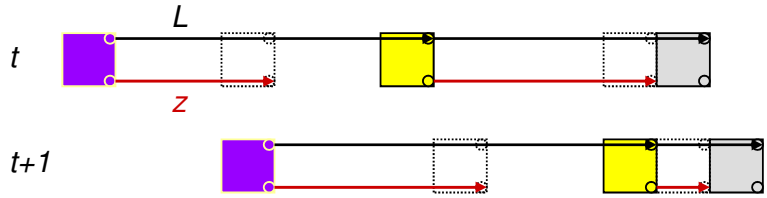

Fig. 5. GCA-w algorithm with relative links. The agent knows already its new position given by z when entering into the current generation.

```
if Type = A then
   // move agent
   if z>0 then  Type <- E, z.Type <- A endif
   // "z." means at destination
   z.v <- z   // new speed = z(t)
   z.L <- L-z + L.z // next relative link
   // compute future position (new speed) one gen. in advance
   z.z <- max(0, min(z + 1, L-z + L.z - 1, vmax) - R)
endif
```

A simulation sequence of this algorithm for 10 generations with $N = 20$ cells, 3 cars and $v_{\max} = 4$ is the following:

```
0   **.*................
1   **.*................
2   **..*...............
3   *.*..*..............
4   *..*..*.............
5   .*...*..*...........
6   ..*....*...*........
7   ...*......*...*.....
8   ....*........*....*.
9   .*...*...........*..
```

The states of the cells in the generations $t = 5, 6, 7$ are

```
                                                    -- 5 generation
1 1 1 0 0 0 1 0 1 0 1 1 0 0 0 1 1 1 1 0   R random
. * . . . * . . * . . . . . . . . . . .   C cars
  1       2     2                         V speed
  1       2     3                         Z future speed
  4       3    13                         L rel.pointer
    5         4      11                   L' new
    1         2       3                   V' new
    1         3       3                   Z' new
  0 1     0   1 0     1                   C' new

                                                    -- 6 generation
0 0 1 0 1 1 1 0 1 0 0 0 1 0 0 0 1 1 1 1   R random
. . * . . . . * . . . * . . . . . . . .   C cars
    1         2       3                   V speed
    1         3       3                   Z future speed
    5         4      11                   L rel.pointer
      7             4       9             L' new
      1             3       3             V' new
      1             3       4             Z' new
    0 1       0     1 0     1             C' new

                                                    -- 7 generation
1 1 1 1 1 1 1 0 1 0 0 0 0 0 1 0 0 1 0 1   R random
. . . * . . . . . * . . . * . . . . . .   C cars
      1             3       3             V speed
      1             3       4             Z future speed
      7             4       9             L rel.pointer
        9                   5         6   L' new
        1                   3         4   V' new
        1                   4         3   Z' new
      0 1           0     1 0       1     C' new
```

GCA-w algorithm with a varying number of active agents. At last an algorithm will be presented that is able to deactivate and activate agents. Agents that are stuck in a traffic jam are deactivated. The advantage is that "sleeping" agents need not to perform any computations and thus energy and computation time can be saved. (Nevertheless sleeping agents have to be aware to be wakened-up, and thus have to perform a minimal listening process.) The rule for deactivation is: If there is an agent directly in front and an agent directly behind, then the blocked agent enters into the passive state Type $= P$. Now the cell's state is $(\text{Type}, R, L, B, v, v')$, where v' is a temporary variable. In Phase 1 every cell computes its next speed v' (the movement offset) and stores it in the temporary variable to be used in

Phase 2 by the cell itself and by the cell's neighbors. B is an additional link, pointing from an agent back to its follower. Fig. 6 shows the movements and the computation of the new links. Fig. 7 shows the deactivation and activation procedure.

```
// PHASE 1
if Type = A then
    // compute new speed v', new position will be i+v'
    v' = max(0, min(v+1, L-1, vmax) - R
endif

// PHASE 2
if Type = A then
    L' = L + L.v' - v'    // compute next forward link
    B' = B + B.v' - v'    // compute next backward link

    v'.L <- L'            // sync. write to position i+v'
    v'.B <- B'
    v'.v <- v'

    if v'>0 then          // if agent will move
       Type <- E, v'.Type <- A
       // activate follower if passive
       if (-1).Type = P then (-1).Type <- A endif
    endif
    if (L'=1) and (B'=-1) then
       Type <- P  // deactivate, if agent will be enclosed by agents
    endif
endif
```

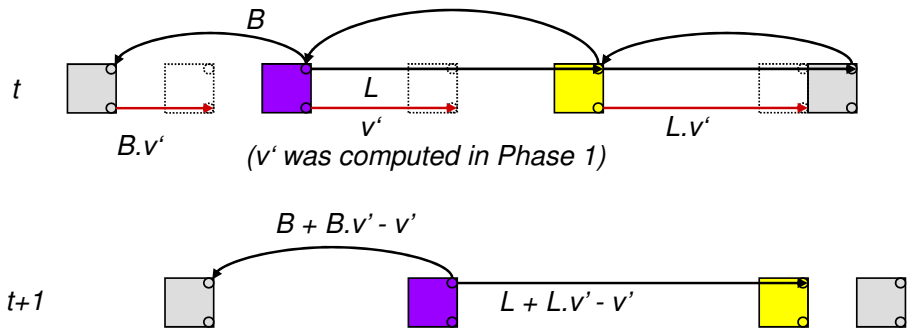

Fig. 6. GCA-w algorithm with a varying number of active agents. Computation of the new links. L points forwards, B points backwards.

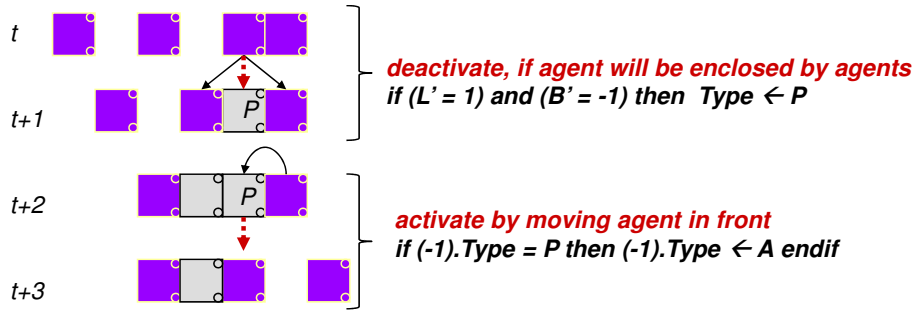

Fig. 7. The agent is deactivated when it stopped and is enclosed by other agents. It is activated again by the agent in front when it starts to move again.

In order to apply only one phase, a synchronously updated state variable $z(t) = v'(t+1)$ can be used to replace the temporary local variable v'. The first phase can be saved by attaching the computation of v' to the end of *PHASE2*: (*PHASE2*, then compute v' and assign it to z). Thereby v' is already available when entering into the next generation (pre-computed and stored already in z). Thus z has a semantically identical meaning as the variables "z" used in the preceding algorithms. On the other hand, the preceding algorithms can be transformed into algorithms with two phases, computing v' in the first phase and use it as z in the second phase.

3. Conclusion

The GCA-w model is well suited to describe problems with moving agents/particles concise and efficiently as it was shown for the traffic simulation. For comparison, the Nagel–Schreckenberg algorithm was first described as a CA rule, and then as a GCA rule. The corresponding GCA-w rule is much shorter, easy to understand, and only the cells of type agent need to be simulated. In addition, a rule was presented, which deactivates cars stuck in a traffic jam, and which activates the car behind the head of the queue, when the head starts to move again. Such an algorithm can also be useful for car-to-car communication systems where a car can be stopped or started automatically depending on the local situation.

I would like to express my thanks to Patrick Ediger (Technische Universität Darmstadt, Computer Architecture Group) for his valuable comments to improve the formal description of the GCA-w model.

REFERENCES

[1] K. Nagel, M. Schreckenberg, *J. Phys. I France* **2**, 2221 (1992).

[2] R. Hoffmann, Fachgebiet Rechnerarchitektur, Technische Universität Darmstadt, Internal Report (1/2009), `http://www.ra.informatik.tu-darmstadt.de/forschung/publikationen`

[3] R. Hoffmann, *Acta Phys. Pol. B Proc. Suppl.* **3**, 347 (2010).

[4] R. Hoffmann, *Lect. Notes Comput. Sci.* **5698**, 194 (2009).

[5] R.Hoffmann, K.-P. Völkmann, S. Waldschmidt, in: Theoretical and Practical Issues on Cellular Automata, Proceedings of the Fourth International Conference on Cellular Automata for Research and Industry, Karlsruhe, 4–6 October 2000, Springer, 2000.

[6] R. Hoffmann, K.-P. Völkmann, S. Waldschmidt, W. Heenes, *Lect. Notes Comput. Sci.* **2127**, 66 (2001).

[7] Chr. Schäck, R. Hoffmann, W. Heenes, Efficient Traffic Simulation Using the GCA Model, talk on APDCM Workshop at IEEE International Parallel and Distributed Processing Symposium IPDPS 2010.

[8] A. Lawniczak, B. Di Stefano, *Acta Phys. Pol. B Proc. Suppl.* **3**, 479 (2010).

[9] S. Achasova, O. Bandman, V. Markova, S. Piskunov, *Parallel Substitution Algorithms, Theory and Applications*, World Scientific, 1994.

[10] J. Keller, Chr. Kessler, J. Träff, *Practical PRAM Programming*, Wiley, 2001.

[11] J. JáJá, *An Introduction to Parallel Algorithms*, Addison-Wesley, 1992.

[12] D. Keil, D. Goldin, in: Proceeding WETICE '03 Proceedings of the 12th International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises, IEEE Computer Society Washington, DC, USA, 2003.