Chapter 6
Routing by Cellular Automata Agents in the Triangular Lattice

Rolf Hoffmann and Dominique Désérable

Abstract This chapter describes an efficient novel router in which the messages are transported by Cellular Automata (CA) mini-robots or so called CA agents. CA agents are compliant but inconvenient to describe with the CA paradigm. In order to implement agents more efficiently, the CA-w model (with *write* access) is used. Both CA and CA–w models are compared. The other relevant feature in this chapter is the underlying network embedded into the triangular lattice, with more symmetries, thereby providing agents with more degrees of freedom. The router uses six channels per node that can host up to six agents and provides a minimal routing scheme (XYZ–protocol). Each agent situated on a channel has a computed minimal direction defining the new channel in the adjacent node. In order to increase the throughput an adaptive routing protocol is defined, preferring the direction to an unoccupied channel. A strategy of deadlock avoidance is also investigated, from which the initial setting of the channels can be alternated in space, or the agent's direction can dynamically be randomized.

6.1 Introduction

Problem solving with robots and agents has become more and more attractive [1–6]. What are the benefits to use agents for a given problem? Generally speaking, agents are intelligent and their capabilities can be tailored to the problem in order to solve

R. Hoffmann (✉)
Technische Universität Darmstadt, FB Informatik, FG Rechnerarchitektur,
Hochschulstraße 10, 64289 Darmstadt, Germany
e-mail: hoffmann@informatik.tu-darmstadt.de

D. Désérable
Institut National des Sciences Appliquées, 20 Avenue des Buttes de Coësmes,
35043 Rennes, France
e-mail: domidese@gmail.com

G.Ch. Sirakoulis and A. Adamatzky (eds.), *Robots and Lattice Automata*,
Emergence, Complexity and Computation 13, DOI 10.1007/978-3-319-10924-4_6

it effectively, and often in an unconventional way. Important properties that can be achieved by agents are

- **Scalable**: the problem can be solved with a variable number of agents, and faster or better with more agents.
- **Tuneable**: depending on the agent's intelligence, the problem can be solved more efficiently (with a higher quality and faster).
- **Flexible**: similar or dynamically changing situations can be solved using the same agents, e.g. when the shape or size of the environment is changing.
- **Fault-tolerant**: the problem can be solved with low degradation even if some "noise" is added, e.g. dynamic obstacles or temporary malfunctions appear, or some agents break down totally.

Owing to their intelligence, agents can be employed to design, model, analyze, simulate, and solve problems in the areas of complex systems, real and artificial worlds, games, distributed algorithms and mathematical questions.

Robots or agents controlled by a finite state machine (FSM) have a long history in computer science [7], sometimes they are simply called "FSMs", often with the property that they can move around on a graph or grid. For example, searching through the whole environment by an FSM was addressed in [8] and graph exploration by FSM controlled robots was treated in [9]. In order to support the simulation of such applications special languages like [10–12] have been developed.

6.1.1 Cellular Automata Agents

What is a Cellular Automata Agent (CA Agent)? Simply speaking, a CA agent is an agent that can be modeled within the CA paradigm. And what are the most important attributes an agent should have in our context?

1. **Self-contained** (an individual, complete in itself). In CA, this property can be realized by one cell, by a part of a cell, or by a group of cells.
2. **Autonomous** (not controlled by others). Agents operate on their own and control their actions and internal states. In CA, this property can be realized by the own state and the CA rule.
3. **Perceptive** (perceives information about the environment). In CA, this property is realized by reading and interpreting the states of the neighborhood.
4. **Reactive** (can react on the perceived environment). In CA, this property is realized by changing the own state by taking into account the perceived information.
5. **Communicative** (can communicate with other agents). This property means that agents can exchange information, either indirectly through the environment (stigmergy, e.g. pheromones), or directly by perceiving other agents and reacting on them in a perceivable way.
6. **Proactive** (acts on its own initiative, not only reacting, using a plan). In CA, the cell's next state should not only depend on its neighbors' states but also on its

own state. The number of inputs and states should not be too small in order to give the agent a certain intelligence to initiate changes and to deal in advance with difficult situations. And the agent's behavior is to a certain extent not foreseeable, it can be influenced by personal secret information or internal events. In CA, this can be accomplished by hidden states that cannot be observed by the neighbors, or by asynchronous internal triggers (e.g. random generator). As it is difficult to define proactivity in a strict way, it is a matter of viewpoint whether simple classical CA rules (like Game of Life, Traffic Rule 184) shall be classified as multi-agent systems or not.

7. **Local** (acts locally). Agents are small compared to the system size and can only act on their neighborhood. Global effects arise from accumulated local actions.
8. **Mobile** (this feature is not required but very useful). Very often agents are moving around in the environment, and then the neighborhood and the place of activity are moving, too. When moving around, an agent may also change its local environment at the same time.

Usually an agent performs *actions*. *Internal* actions change the state of an agent, either a visible or a non-visible state, whereas *external* actions change the state of the environment. The environment is composed of the ground environment (constant or variable) and the other agents. In CA, an agent is not allowed to change the state of a neighboring cell. Therefore, if an agent wants to apply an external action to a neighboring cell, it can only issue a command that must be adequately executed by the neighbor. For example agent A sends a "kill" command to agent B, then agent B has to kill itself. This example shows that the CA modeling and description of changing the environment is indirect and does not appear natural. Other models are helpful to simplify such descriptions, like the "CA–w model" presented hereafter.

6.1.2 CA and CA–w Models

In order to describe moving agents, moving particles or dynamic changing activities, the CA–w model (Cellular Automata with *write* access) was introduced [13]. Simply speaking, this model allows to write information onto a neighbor. This method has the advantage that a neighbor can directly be activated or deactivated, or data can be sent actively to it by the agent. Thus the movement of agents can be described more easily.

The CA–w model is a restricted case of the more general, "*Global*" GCA–w [14–16]. In GCA–w any cell of the whole array can be modified whereas in the CA–w model only the local neighbors can be. Usually the cells of these models are a composition of (data, pointers). The neighbors are accessed via pointers, that can be changed dynamically like the data by an appropriate rule from generation to generation.

In order to avoid confusion between CA and CA–w, in this context the CA model can be attributed as "classical model" and the CA–w model as "implementation model" although both can be used for description and implementation.

What are the capabilities and limitations of CA–w compared to CA? The main difference is that the CA–w model allows to modify the state of a neighbor. Thereby the activity of a neighboring cell can be switched on or off and data can actively be moved to a neighbor, which is very useful for the description and effective simulation of active particles or moving agents. Comparing their computing power, a CA equivalent to a CA–w with neighborhood $N1$ can be found by extending $N1$ to $N2$ ($N1$ extended by write-distance). For example a CA–w with neighborhood distance 1 (read and write) is equivalent to a CA with neighborhood distance 2. Because of this equivalence, both models can be mapped onto each other.

A drawback is the possible occurrence of write conflicts. There are two solutions to handle conflicts:

- Use a conflict-resolving function, for example by applying a reduction operator (max, +, ...) or using a random or deterministic priority scheme.
- Avoid conflicts by algorithmic design, meaning that the parallel application of all rules never cause a conflict.

The second solution is more simple and elegant and many applications with agents can be described in this way. Our routing problem with agents herein is implemented by the CA–w model, although it is also possible to model a more cumbersome system by standard CA.

6.1.2.1 Modeling Agents' Mobility

How can an agent move from A to B? In the CA model, a couple of two rules (*copy*–rule, *delete*–rule) must be performed (Fig. 6.1a): the first rule copies the agent from A to B, the second deletes it on A. Both rules have to compute the same moving condition, this means a redundant computation. Two operating modes allow the CA–w to avoid this redundancy:

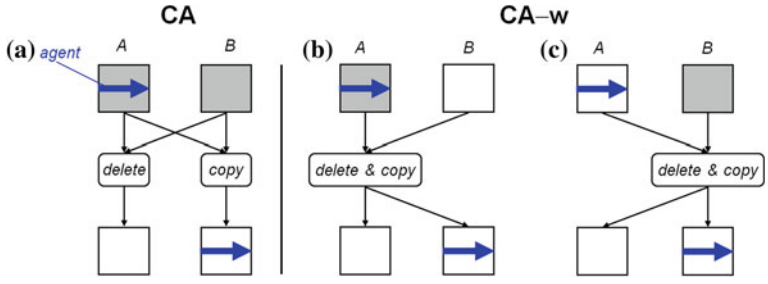

Fig. 6.1 *CA model* **a** cell A deletes the agent and cell B copies it. *CA–w model* **b** cell A deletes and copies the agent or **c** cell B deletes and copies the agent. Active cells executing a rule are shaded

- Cell A (the agent) is responsible for the moving operation (Fig. 6.1b), it computes the moving condition and, if true, applies a rule that deletes itself on A and copies it to B.
- Cell B (the empty front cell) is responsible (Fig. 6.1c), it computes the moving condition and, if true, applies a rule that deletes the agent on A and copies it to B.

The second mode is used for our problem. In this way, concurrent agents wanting to move to the same empty channel can easily be prioritized. The differences between CA–w and classical CA for our routing problem will be illustrated in Sect. 6.3.

6.1.3 Lattice Topology

Choosing the best topology for 2*d* CA agents is not straightforward. We give some insight hereafter to clarify our choice of the network used in this chapter to route agents.

6.1.3.1 Towards an Optimal Tiling

The three regular tessellations of the plane are displayed in Fig. 6.2a where Schläfli symbol $\{p, q\}$ gives an exact definition of the tiling [17]. Their associated dual $\{q, p\}$ tilings are displayed in (b), whence the three possible regular 2*d* lattices in (c), either 3–valent or 4–valent or 6–valent [18].

Two usual tessellations for 2*d* cellular automata are identified in the $\{4, 4\}$ "square" tiling and the $\{6, 3\}$ "hexagonal" tiling. It is observed that the minimal number of neighbors appears in the hexagonal tiling; moreover, the six neighboring cells are adjacent. Thereby, there is no risk of wavering as in the $\{4, 4\}$ case between either a 4–valent von Neumann neighborhood or a 8–valent Moore neighborhood. This matter of symmetry among lattices may have important impacts upon the behavior of their CA. A typical example is well known for lattice-gas automata wherein the 4–valent HPP gas cannot be consistent with the Navier-Stokes equation while the 6–valent FHP ensures consistency [19–21]. Our routing problem herein is embedded into the 6–valent lattice.

6.1.3.2 Towards an Optimal 2*d* Finite-Sized Network

Once the valence had been settled, the question is to define a finite-sized toroidal network. As a matter of fact, there is a relationship between a compatible arrangement and some associated tessellation of the plane. Tiling the $\{4, 4\}$ tessellation with a finite-sized "prototile" is examined in [22]. In general, the topologies related to plane tessellations belong to the family of multi-loop and circulant networks [23].

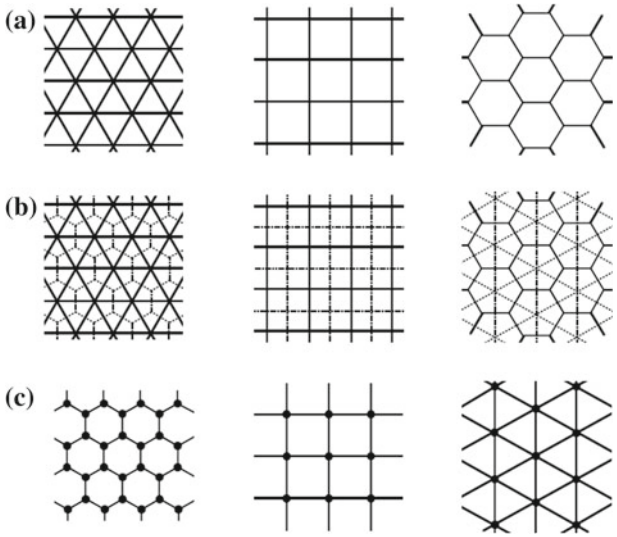

Fig. 6.2 **a** The three regular tessellations of the plane: $\{3, 6\}$ with 3–gons, $\{4, 4\}$ with 4–gons, $\{6, 3\}$ with 6–gons where $\{p, q\}$ is the Schläfli symbol; a triangular cell is surrounded with twelve neighbors, a square cell with eight neighbors, a hexagonal cell with six *adjacent* neighbors. **b** Associated dual tilings: $\{6, 3\}$, $\{4, 4\}$, $\{3, 6\}$ (in *dashed lines*); $\{q, p\}$ is the dual of $\{p, q\}$. **c** The three induced regular $2d$ lattices: *3*–valent, *4*–valent, *6*–valent

The hexagonal (or triple loop) case was investigated in [24] in order to exhibit graphs with minimum diameter. They proved that the maximum order of a triple loop graph with diameter n is $N = 3n^2 + 3n + 1$. The grid representation of the graph is a hexagonal torus with n circular rings of length $6n$ arranged around a central node. Incidentally, this family of "honeycombs" H_n was encountered elsewhere, arising in various projects such as FAIM-1 [25], Mayfly [26], HARTS [27] and more recently with the EJ networks [28].

When tiling the plane with H_n prototiles, it can be observed that the axes joining the center of these prototiles and the symmetry axes of the $\{6, 3\}$ tiling do not coincide. On the contrary, we have defined a new family of hexavalent networks stabilizing the symmetry axes, that provide these networks with the highest symmetry level for a 6–valent finite lattice. A relevant illustration of this discrepancy between prototiles can be found in [29].

6.1.3.3 Arrowhead and Diamond

Our networks belong to a family of hierarchical Cayley graphs [30]. As a consequence, this property facilitate as far as possible any routing or global communication procedure. The graphs of this family are denoted elsewhere as "*arrowhead*" or "*diamond*" in order to avoid confusion with H_n family. The reader is referred

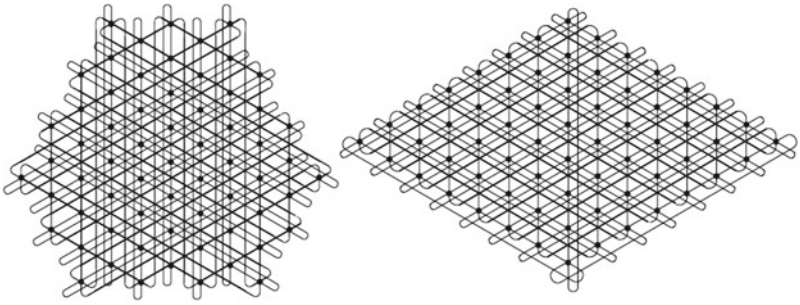

Fig. 6.3 Arrowhead and diamond with $N = 4^n$ vertices for $n = 3$

to [31] for more details about the genesis of these graphs, displayed in Fig. 6.3, and some of their topological properties. A very important one is that as Cayley graphs they are vertex-transitive, that means that any vertex behaves identically. Practically, this property involves a unique version of router code distributed among all nodes. It was also shown that these graphs provide a good framework for routing [32] and other global communications like broadcasting [33] and gossiping [34]. A survey of global communications in usual networks is given in [35] but we focus hereafter on the routing problem.

Arrowhead and diamond are isomorphic and the diamond itself is isomorphic to an orthogonal representation of the "T–grid" like in Fig. 6.5 with $n = 2$. Therefore, it is easy to map "T" into the arrowhead by a simple coordinate transformation. In the sequel, "T" –or "T_n" if the "size" n is relevant– will always denote the orthogonal representation of the diamond. It is interesting to observe that the k–ary 2–cube [36] ($k = 2^n$ herein) can be embedded into by eliminating one direction of link, namely the "diagonal" direction in the orthogonal diamond. For clarity's sake, the "S–grid" "S" –or "S_n" as well– will also denote our 2^n–ary 2–cube in the sequel as a subgrid of T. Note that another family of "augmented" k–ary 2–cubes was investigated elsewhere [37] for any k but which coincide with T_n only when $k = 2^n$.

The tori are well suited for physical ergodic systems with periodic boundary conditions [38]. For a finite space with robots or multiagents, non-periodic boundaries can also be defined with boundary conditions (bounce-back, absorption and so forth). To conclude this topological presentation, let us hope that our S–T family might reconcile von Neumann and hexagonal $2d$ cellular automata and activate exciting challenges in CA and robot worlds.

6.1.4 The Problem: Routing

Let us consider the approach based on agents transporting messages from a source node to a destination node and following a minimal route (or shortest path). The nodes are connected via twelve unidirectional links, namely two in each of the six directions,

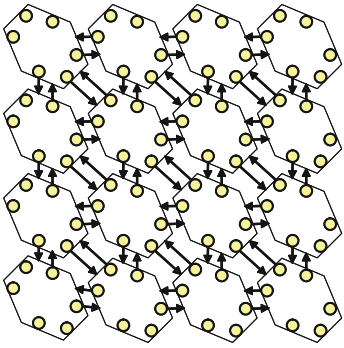

Fig. 6.4 Each node of the network contains six buffers (channels) and is connected to its neighbors by 6 input and 6 output links

that corresponds with a full-duplex or double lane traffic (Fig. 6.4). Each node is provided with six *channels* (sometimes called *buffers* according to the context) and one channel may host at most one agent transporting a message. Each agent moves to the next node, defined by the channel's position it is situated on. When moving to the next node, an agent may hop to another channel, defined by the agent's direction in its minimal path.

A *message transfer* is the transfer of one message from a source to a target *and* each agent shall perform such a message transfer. A set of messages to be transported is called *message set*. A *message set transfer* is the successful transfer of all messages belonging to the set. Initially k agents are situated at their source nodes. Then they move to their target nodes following their minimal path. When an agent reaches its target, it is deleted. Thereby the number of moving agents is reduced until no agent is left. This event defines the end of the whole message set transfer. Note that the agents hinder each other more at the beginning (due to congestion) and less when many of the agents have reached their targets and have been deleted. No new messages are inserted into the system until all messages of the current set have reached their targets. This corresponds to a barrier-synchronization between successive sets of messages. Initially each agent is placed on a certain channel (with direction to the target) in the source node and each agent knows its target. The target node of an agent should not be its source node: message transfers within a node without an agent's movement are not allowed.

The goal is to find an agent's behavior in order to transfer a message set (averaged over many different sets) as fast as possible, that is, within a minimal number of generations. We know from previous works that the agent's behavior can be optimized (e.g. by a genetic algorithm) with respect to the set of given initial configurations, the initial density of agents, and the size of the network. The goal is not to fully optimize the agent's behavior but rather to design a powerful router with six channels that outperforms the ones developed before [39, 40].

6.1.4.1 Related Work

Target searching has been studied in many variations: with moving targets [41] or as single-agent systems [42]. Here we consider only stationary targets, and multiple agents having only a local view. This contribution continues our preceding work on routing with agents on the cyclic triangular grid [39] and on non-cyclic rectangular $2d$ meshes [43]. In a recent work [40], tori S and T were compared; evolved agents, with a maximum of one agent per node, were used in both cases. It turned out that routing in T is performed significantly better than in S.

The novelty herein is that *six* agents per node are now used, with one agent per channel, instead of *one* agent per node therein. Another difference is that in [39] the agent's behavior was controlled by a finite state machine (FSM) evolved by a genetic algorithm, whereas here the behavior is handcrafted. To summarize, the goal is to find a faster router in T, using six agents per node and bidirectional traffic between nodes, at first modeling the system as CA–w and then discussing whether the routing algorithm is deadlock-free or not. Usually deterministic agents with synchronous updating are not deadlock-free. Therefore a small amount of randomness can be added to a deterministic behavior [44] in order to avoid deadlocks.

Communication protocols in hexagonal networks were already studied for H_n or EJ topologies [27, 28]. An adaptive deadlock-free routing protocol was proposed recently [45] using additional virtual channels. In our approach, if the minimal route is blocked, the alternative minimal route is attempted in order to minimize deadlocks. Further possibilities to avoid deadlocks are proposed in Sect. 6.4.3. Note that we do not address the problem of fault tolerance networks on VLSI chips [46, 47]. A general insight on adaptive routing can be found in [48].

The remainder of this chapter is structured as follows. Section 6.2 deals with the topology of the T–grid and presents the XYZ–protocol computing the minimal route. Section 6.3 shows how the routing can by modeled as a multi-agent system. An analysis of the router efficiency is discussed in Sect. 6.4 and some deadlock situations are pointed out before Summary. This work finalizes a previous one investigating this novel router in the triangular grid with six channels [49].

6.2 Minimal Routing in the Triangular Grid

6.2.1 Topology of S and T

Consider the square blocks in Fig. 6.5 with $N = 2^n \times 2^n$ nodes where n denotes the size of the networks. The nodes are labeled according to the XY–orthogonal coordinate system. In the left block, a node (x, y) labeled "xy" is connected with its four neighbors $(x \pm 1, y)$, $(x, y \pm 1)$ (with addition modulo 2^n) respectively in the W–E, N–S directions, giving the 4–valent torus S_n. In the right block, two additional links $(x - 1, y - 1)$, $(x + 1, y + 1)$ are provided in the diagonal NW–SE direction (Z–coordinate), giving the 6–valent torus T_n. Because their associated

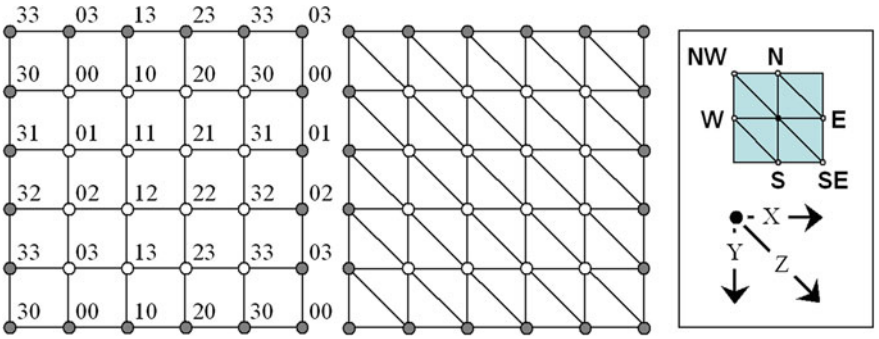

Fig. 6.5 Tori S_2 and T_2 of order $N = 16$, labeled in the XY coordinate system; redundant nodes in *grey* on the boundary. Inset: orientations W–E, N–S, NW–SE according to an XYZ reference frame

graphs are regular their number of links is, respectively, $2N$ for torus S_n and $3N$ for torus T_n. Both networks are scalable in the sense that one network of size n can be built from four blocks of size $n - 1$. The S–grid is just displayed here because it is often interesting to compare the topologies and performances of S_n and T_n, two networks of the same size.

An important parameter for the routing task in the networks is the diameter. The diameter defines the length of the shortest path between the most distant pair of nodes and provides a lower bound for routing or other global communications; such a pair is said to be *antipodal*. The exact value of the diameters in S_n and T_n is given by

$$D_n^S = \sqrt{N}; \quad D_n^T = \frac{2(\sqrt{N} - 1) + \varepsilon_n}{3} \tag{6.1}$$

where $\varepsilon_n = 1$ (resp. 0) depends on the odd (resp. even) parity of n and where the upper symbol identifies the torus type; whence the ratio denoted

$$D_n^{S/T} \approx 1.5 \tag{6.2}$$

between diameters. In this study, only the diameter D_n^T will be considered, denoted simply D_n in the sequel [50].

6.2.2 Minimal Routing Schemes in S and T

The basic, deterministic routing schemes are driven by the Manhattan distance in S [36] and by the so-called "hexagonal" distance in T [28, 32]. They are denoted as "rectangular" and "triangular" herein. Considering a source "A" and a target "B" as

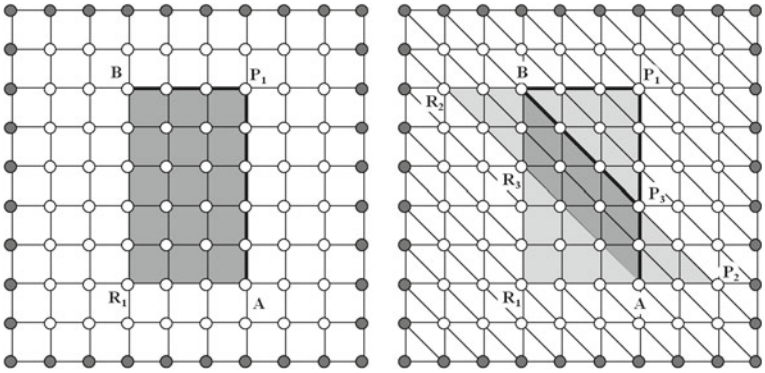

Fig. 6.6 Networks S_3 and T_3 of order $N = 64$. Routing paths from a source "A" to a target "B": rectangular routing in S (*left*), triangular routing in T (*right*). In the rectangular routing, axis systems $X_A Y_A$ and $X_B Y_B$ intersect at P_1, R_1 and yield the rectangle $A P_1 B R_1$ in general. In the triangular routing, axis systems $X_A Y_A Z_A$ and $X_B Y_B Z_B$ intersect at P_i, R_i $(i = 1, 2, 3)$ and yield three parallelograms $A P_i B R_i$ in general; in this case, the parallelogram $A P_3 B R_3$ is "minimal"

shown in Fig. 6.6, we choose to find a shortest path from A to B with *at most one change* of direction.

In the square grid on the left part, the construction yields the rectangle $A P_1 B R_1$. In order to ensure a homogeneous routing scheme, from an usual convention the agent is carried following one direction first, following the other direction afterwards. This orientation will be specified in the following subsection. Under these conditions, a route $A \to B$ and a route $B \to A$ will follow two disjoint paths and each of them is made of two unidirectional subpaths, that is $A \to P_1 \to B$ and $B \to R_1 \to A$ respectively. In a particular case, A and B may share a common axis and the routes $A \to B$ and $B \to A$ need a (full-duplex) two-lane way $A \leftrightarrow B$. Note that in a finite-sized torus, not only the "geometric" rectangle $A P_1 B R_1$ should be considered but rather a "generalized" rectangle, because the unidirectional subpaths may "cross" over the boundaries of the torus.

In the triangular grid on the right part, the construction involves three generalized parallelograms of the form $A P_i B R_i$. Among them, there exists a "minimal" one that defines the shortest path. It is the purpose of the following to detect it and to move CA agents within it.

6.2.3 Computing the Minimal Route in T (XYZ–Protocol)

The following abbreviations are used in the routing algorithm:

$sign(d) = (0, 1, -1)$ IF $(d = 0, d > 0, d < 0)$ for any integer d and

$\bar{d} = d - sign(d) \cdot M/2$, where $M = 2^n$ is the length of any unidirectional cycle.

STEP 0. The offsets between target (x', y') and current (x, y) positions are computed.

$(dx, dy) := (x' - x, y' - y)$.

STEP 1. The deviations are contracted to the interval $[-M/2, +M/2]$.

$dx := \bar{dx}$ IF $|dx| > M/2$; $dy := \bar{dy}$ IF $|dy| > M/2$

If $sign(dx) = sign(dy)$ then the minimal path is already determined and the diagonal is used as one of the subpaths. Note that the path length is given by $max(|dx|, |dy|)$ if the signs are equal, by $|dx| + |dy|$ otherwise.

STEP 2. One of the following operations is performed, only if $dx \cdot dy < 0$. They comprise a test whether the path with or without using the diagonal is shorter.

$$dx := \bar{dx} \text{ IF } |dx| > |dy| \text{ AND } |\bar{dx}| < |dx| + |dy| \quad // \, |\bar{dx}| = \max(|dx|, |dy|)$$

$$dy := \bar{dy} \text{ IF } |dy| \geq |dx| \text{ AND } |\bar{dy}| < |dx| + |dy| \quad // |\bar{dy}| = \max(|dx|, |dy|)$$

STEP 3. This step forces the agents to move in the same direction if source and target lie opposite to each other, namely at distance $M/2$ on the same axis. Thereby collisions on a common node on inverse routes are avoided.

$(dx, dy) := (|dx|, |dy|)$ IF $(dx = -M/2)$ AND $(dy = -M/2)$
$dx := |dx|$ IF $(dx = -M/2)$ AND $(dy = 0)$
$dy := |dy|$ IF $(dy = -M/2)$ AND $(dx = 0)$

Then a minimal route is computed as follows:

(a) If $dx \cdot dy < 0$ then
$[dz' = 0]$ move FIRST $dx' = dx$ steps, THEN move $dy' = dy$ steps

(b) If $dx \cdot dy > 0$ then calculate
(1) $dz' = sign(dx) \cdot \min(|dx|, |dy|)$ // steps on the diagonal
(2) $dx' = dx - dz', \quad dy' = dy - dz'$

$[dy' = 0]$ move FIRST dz' THEN dx', or
$[dx' = 0]$ move FIRST dy' THEN dz' .

This algorithm yields a minimal route and uses a cyclic priority for the six directions, two or one of them which are used in a valid minimal route. For short, the algorithm uses the priority scheme:

$[dx' = 0]$ move FIRST dy' THEN dz',
$[dy' = 0]$ move FIRST dz' THEN dx',
$[dz' = 0]$ move FIRST dx' THEN dy'.

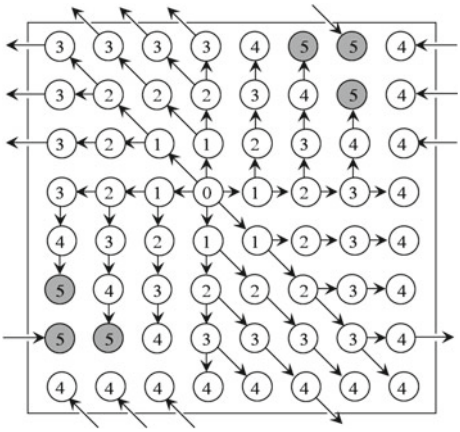

Fig. 6.7 This directed graph is a spanning tree of the torus showing the minimal path according to the XYZ–protocol from a source node "0" to any other node for a 8×8 network ($n = 3$, $N = 64$). The maximal distance (longest path) is the diameter $D_3 = 5$ for this graph (refer back to Eq. (6.1). Six antipodals are highlighted. Note that for n even ($n > 2$) this routing scheme would display twelve antipodals [50]

This priority scheme means: use "FIRST `dirR` THEN `dirL`" where `dirR` and `dirL` define the "right" minimal subpath and the "left" minimal subpath respectively,[1] viewed from the "observer" agent as in Fig. 6.6.

The minimal routes (FIRST `dirR` THEN `dirL`) are depicted as directed graph in Fig. 6.7 for a 8×8 network. The number in the nodes represents their distance from the source node "0".

6.2.4 Deterministic Routing

From the above protocol "FIRST `dirR` THEN `dirL`" the agent follows always the "right" minimal subpath. This means that the agent changes its moving direction accordingly. The minimal path can be computed only once at the beginning and stored in the agent's state. During the run, the agent updates the remaining path to its target, decrementing its `dirR` counter until zero, then decrementing its `dirL` counter if any, until completion. It is also possible to recompute the minimal path at each new position. This was done in the simulation and that yields the same result.

The problem with deterministic routing is that it is not deadlock-free (see deadlock discussion in Sect. 6.4.3). Another problem with this protocol is that it may not be optimal with respect to throughput, especially in case of congestion. But it should be noted that congestion usually is not very high, because there are *six* channels available in each node. Formally, the deterministic routing is secure for an agent alone.

[1] An equivalent symmetric protocol would be "FIRST `dirL` THEN `dirR`".

6.2.5 Adaptive Routing

The objective for adaptive routing is (i) to increase throughput, and (ii) to avoid or reduce the probability of deadlocks. This protocol was manually designed and it is a simple algorithm defining the new direction of an agent. During the run, if the temporary computed direction (e.g., `dirR`) points to an occupied channel, then the other channel (e.g., `dirL`) is selected no matter this channel is free or not. A minimal adaptive routing may be roughly denoted as "EITHER `dirR` OR `dirL`". The path from source to target remains minimal on the condition that it remains inside the boundaries defined by the minimal parallelogram.

In order to increase the throughput when the system is congested or to avoid deadlocks securely, the agent's behavior could be more elaborated. The agent could obey to an internal control automaton (finite state machine as in [39]) and this automaton can be optimized by genetic algorithms [51].

It would also be useful to allow the agent to *deviate* from the minimal route in case of congestion. Referring back to Fig. 6.6, going from source A to target B, the agent could route *out* of its minimal parallelogram and move within an extended area like the trapezium AP_1BR_3 or even the rectangle AP_1BR_1 although the minimal route is of course prioritized. As a consequence, *three* possible moving directions, instead of *two*, remain adaptively possible. The three other directions backwards are not allowed.

6.3 Modeling the Multi-Agent System

This section is the core of this chapter. The dynamics of moving agents is described, the impact of the *copy–delete* rules in the CA–w and CA models is emphasized and some programming issues are revealed.

6.3.1 Dynamics of the Multi-Agent System

The node structure, the channel state and how agents and arbiters cooperate are presented herein, the priority rule derived from the above adaptive protocol is analyzed and the conflict-free transition moving the agents follows.

6.3.1.1 Node Structure

The whole system consists of $2^n \times 2^n$ nodes arranged as in the T-grid of Fig. 6.5. Each node labeled by its (x, y) coordinates contains the 6–fold set

$$\mathcal{C} = \{C_0, C_1, C_2, C_3, C_4, C_5\} = \{E, SE, S, W, NW, N\} \tag{6.3}$$

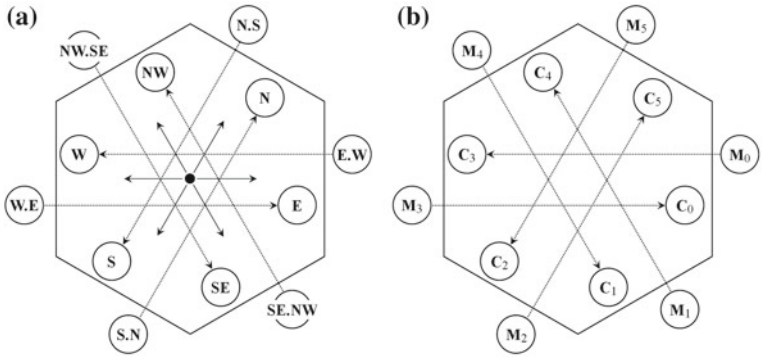

Fig. 6.8 Inner channels of $\mathcal{C}$ oriented (**a**) and labeled clockwise (**b**). Outer channels of $\mathcal{M}$ located in adjacent nodes. The direct neighbor of channel C_i in the adjacent node is denoted by M_i. The cardinal notation "$W.E$" stands for the E–channel of the W–neighbor; the same relative neighborhood is valid for any pair (node, channel) by symmetry: $M_{j\equiv i+3 \pmod 6}$ denotes the i–channel of the adjacent j–neighbor

of channels C_i oriented[2] and labeled clockwise (Fig. 6.8). Index i is called *position* or *lane number* in this context. The position of a channel defines also an implicit direction that defines the next *adjacent* node that an agent visits next on its travel. The direct neighbor of channel C_i in the adjacent node is denoted by M_i where

$$\mathcal{M} = \{M_0, M_1, M_2, M_3, M_4, M_5\} \tag{6.4}$$

and $M_{j\equiv i+3 \pmod 6}$ denotes the i–channel of the adjacent j–neighbor by symmetry. In the cardinal notation, e.g. for $i = 0$, "$W.E$" stands for the E–channel of the W–neighbor.

6.3.1.2 Channel State

Each agent has a direction which is updated when it moves. In other words, the current direction of the agent defines the channel in the next node where the agent requests to move to.

The i–channel's *state* at time t is defined by

$$c_i(t) = (p, (x', y')) \tag{6.5}$$

where (x', y') stands for the agent's target coordinates and $p \in \mathcal{P}$ stands for the agent's direction (a pointer to the next channel) in the set

$$\mathcal{P} = \{-1, 0, 1, 2, 3, 4, 5\} \equiv (\texttt{Empty}, \texttt{toE}, \texttt{toNW}, \texttt{toS}, \texttt{toW}, \texttt{toSE}, \texttt{toN}) \tag{6.6}$$

[2] Except a homeomorphism.

including an empty channel encoded by $\omega = -1$. In a graphical representation, the directions can be symbolized by $(\rightarrow\nwarrow\downarrow\leftarrow\searrow\uparrow)$ according to the inset in Fig. 6.5.

6.3.1.3 Agents and Arbiters

According to the above *adaptive* routing scheme, an agent can move to a *3–fold* subset of channels at most. Recall that the three other directions backwards are not allowed. For example, coming from channel $W.E$ of W–neighbor at $(x-1, y)$, going to channel E or N or SE of current node (x, y) as shown in Fig. 6.9a. In the same way, agents located in outer channels $NW.SE$ and $S.N$ are possible competitors for a part of this subset $\{E, N, SE\}$: channel subset $\{SE, E, S\}$ can be requested by un agent in $NW.SE$ while channel subset $\{N, NW, E\}$ by un agent in $S.N$. The *intersection* of those three requested subsets is the channel E. From this observation, E can be chosen as *arbiter* of three possible concurrent agents. In other words, a priority rule can be locally defined for *this* channel. Arbiter E is C_0 in Fig. 6.9b and this concurrent scheme is invariant by rotation.

This interaction between requesting agents and arbiter channels is formalized hereafter. Let the 3–uple of channels

$$\mathcal{C}_i = (C_{i+1}, C_i, C_{i-1}) \tag{6.7}$$

and let us denote by

$$\mathcal{M}_i = (R_i, S_i, L_i) \tag{6.8}$$

the ordered 3–uple opposite to C_i and where

$$R_i = M_{i+4}, \quad S_i = M_{i+3}, \quad L_i = M_{i+2} \tag{6.9}$$

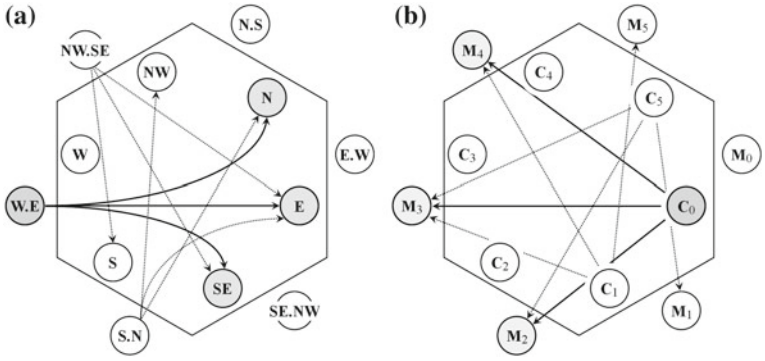

Fig. 6.9 **a** An agent located in the E–channel of the western node $W.E$ can move to one channel in the "opposite" subset $\{E, N, SE\}$. Two agents in channels $NW.SE$ and $S.N$ are possible competitors for a part of this subset. **b** As a consequence, channel C_0 is the arbiter of three possible concurrent agents in the requesting channels M_2, M_3, M_4

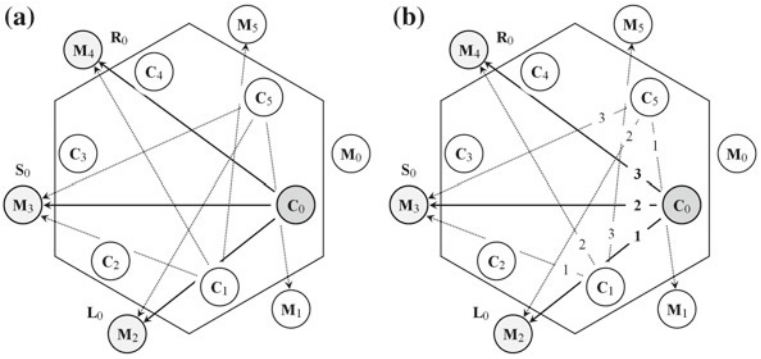

Fig. 6.10 **a** Requesting channels R_0, S_0, L_0 with respect to C_0. The "`right`", "`straight`", "`left`" respective directions are viewed from the "observer" C_0. The same concurrent scheme is valid all around from rotational invariance. **b** Priority rule: a priority order is assigned clockwise by any channel $C_i \in \mathcal{C}$ to its own requesting ordered set $\mathcal{M}_i = (R_i, S_i, L_i)$: `1` to `left`, `2` to `straight`, `3` to `right`

are the `right`, `straight` and `left` outer channels for the three possible incoming agents viewed from the "observer" channel C_i in Fig. 6.10a ($i = 0$ assumed herein). This 3–uple is of special interest because, viewed from channel C_i, the agents may only move from R_i or S_i or L_i to C_i on their route.

Conversely, $\mathcal{C}_i$ is the requested channel subset for S_i, as well as $\mathcal{C}_{i-1}$ for L_i and $\mathcal{C}_{i+1}$ for R_i. Now

$$\mathcal{C}_{i-1} \cap \mathcal{C}_i \cap \mathcal{C}_{i+1} = \{C_i\} \tag{6.10}$$

from (6.7). This simple but important property allows to define a *local* priority rule for channel C_i and invariant by rotation.

6.3.1.4 Priority Rule

Each channel $C_i \in \mathcal{C}$ computes the three exclusive conditions selecting the incoming agent that will be hosted next, with a priority assigned clockwise (Fig. 6.10b):

1. Agent wants to move from L_i to C_i, priority 1: $\texttt{LtoC} = (l = i)$
2. Agent wants to move from S_i to C_i, priority 2: $\texttt{StoC} = (s = i) \wedge \neg\texttt{LtoC}$
3. Agent wants to move from R_i to C_i, priority 3: $\texttt{RtoC} = (r = i) \wedge \neg\texttt{StoC}$.

In other words, this rule selects a winner among the three possible concurrent agents requesting channel C_i and the selection is assigned clockwise: first to `left`, second to `straight`, third to `right`, orientation viewed from the observer *channel*. It should be pointed out that this priority scheme is consistent with the protocol "FIRST `dirR` THEN `dirL`" defined in Sect. 6.2.3 but now viewed from the observer *agent*.

6.3.1.5 Moving the Agents

The above priority scheme ensures a *conflict-free* dynamics of moving agents[3] in the whole network. Thus, from the three previous conditions in channel C_i, five cases are distinguished:

- case $\kappa_1 : (p \neq \omega)$ // channel not empty, agent stays at rest,
- case $\kappa_2 : (p = \omega) \wedge \texttt{LtoC}$ // channel empty, agent to be copied from L_i,
- case $\kappa_3 : (p = \omega) \wedge \texttt{StoC}$ // channel empty, agent to be copied from S_i,
- case $\kappa_4 : (p = \omega) \wedge \texttt{RtoC}$ // channel empty, agent to be copied from R_i,
- case $\kappa_5 : (p = \omega) \wedge \neg\texttt{LtoC} \wedge \neg\texttt{StoC} \wedge \neg\texttt{RtoC}$ // channel remains empty.

The new target coordinates $(x', y')^*$ in the channel's state are either invariant if the agent stays at rest (case κ_1) or are copied from L_i or S_i or R_i exclusively, depending of the selected incoming agent hosted and to be received by the channel. Thus the target coordinates $(x', y')^*$ are updated as

$$\begin{aligned}
(x', y')^* &= (x', y') \quad \text{IF } \kappa_1 \\
(x', y')^* &= (x', y')_{L_i} \quad \text{IF } \kappa_2 \\
(x', y')^* &= (x', y')_{S_i} \quad \text{IF } \kappa_3 \\
(x', y')^* &= (x', y')_{R_i} \quad \text{IF } \kappa_4
\end{aligned} \tag{6.11}$$

according to the current channel's state and to the result of the selection.

Since the agent's target coordinates are stuck within its state, the agent must clearly carry them with it when moving. The new pointer to the next node

$$p^* = \varphi_{xy}\,((x', y')^*) \quad (p^* \in \mathcal{P}) \tag{6.12}$$

is then updated by φ_{xy} which yields the new direction from the target coordinates of the selected agent (φ_{xy} is the local updating function in the current node (x, y)). For case κ_5, the direction is irrelevant and the channel remains empty. Finally, the i–channel's *state* at time $t + 1$ becomes

$$c_i(t + 1) = (p^*, (x', y')^*) \tag{6.13}$$

and the new state is updated synchronously. It is assumed that the agents are initially placed on a channel which is part of the minimal route, and the initial direction is one of the minimal directions.

[3] Except special deadlock or livelock situations pointed out in Sect. 6.4.3.

6.3.2 *The CA–w and CA* Copy–Delete *Rules*

The synchronous transition (6.13) to the next timestep is governed by the *copy–delete* operating mode of either the CA or the CA–w model in Fig. 6.1. The impact of their rules is examined hereafter, that will highlight the simplicity induced by the *write*–access in the CA–w model.

6.3.2.1 CA–w Rule

The CA–w model is especially useful if there are no write conflicts by algorithmic design. This is here the case, because an agent is copied by *its* receiving channel, after applying the abovementioned priority scheme and according to the mode displayed in Fig. 6.1c. Thus only *this* receiving channel is enabled to delete the agent on the sending channel at the same time. A further advantage is that only the short-range *copy*–neighborhood is sufficient to move an agent, the wide-range *delete*–neighborhood (necessary for CA modeling and described hereafter) is not needed.

Therefore the 3–fold *copy*–neighborhood $\mathcal{M}_i$ that needs to be checked by C_i in order to receive the hosted agent is given by (6.8), this agent in $\mathcal{M}_i$ is released by C_i when firing the transition (6.13) and following the CA–w *delete–copy* operating mode in Fig. 6.1c.

6.3.2.2 CA Rule

The CA rule differs from the CA–w rule in the fact that the sending channel has to delete *itself* the agent when moving. This means that a separate *delete*–rule is necessary.

In order to release its own agent, the sending channel must be aware of the whole situation in its wide-range neighborhood defined as follows. Knowing that an agent in channel $M_{i+3} \in \mathcal{M}_i$ and wanting to move (Fig. 6.9a) will be selected by its arbiter which belongs to $\mathcal{C}_i$ (Fig. 6.9b), this requesting agent has other possible competing agents lying in $\mathcal{M}_{i+1}$ for C_{i+1} or $\mathcal{M}_i$ for C_i or $\mathcal{M}_{i-1}$ for C_{i-1}. Therefore, the full set of competitors is the union

$$\widehat{\mathcal{M}_i} = \mathcal{M}_{i+1} \cup \mathcal{M}_i \cup \mathcal{M}_{i-1} \tag{6.14}$$

but

$$S_{i-1} = L_i, \quad R_{i-1} = L_{i+1} = S_i, \quad S_{i+1} = R_i$$

from (6.9) whence

$$\widehat{\mathcal{M}_i} = (M_{i+1}, M_{i+2}, M_{i+3}, M_{i+4}, M_{i+5}) \tag{6.15}$$

or, if we exclude the own channel,

$$\hat{\mathcal{M}}_i = (M_{i+1}, M_{i+2}, M_{i+4}, M_{i+5}). \tag{6.16}$$

In addition, the sending channel must also be aware of the *move-to* conditions of C_i, whence the extended *delete*–neighborhood

$$\hat{\mathcal{M}}_i \cup C_i = (LL_i, L_i, R_i, RR_i) \cup (C_{i+1}, C_i, C_{i-1}) \tag{6.17}$$

with seven channels altogether and where "LL_i" and "RR_i" denote the wide-range left and right outer channels in Fig. 6.11b. As a matter of fact and referring back also to Fig. 6.10, it can be observed that only the "isolated" channels are excluded for this wide-range neighborhood.

Note that the cardinality of the neighborhood of a receiving channel is *three* whereas it is *seven* for a sending channel, without counting the own channel. Thus the whole neighborhood in the CA model is the union of the *copy*–and–*delete*–neighborhood with three and seven channels respectively, namely *ten* channels altogether when firing the transition (6.13) and following the CA *delete–copy* operating mode in Fig. 6.1a.

The discrepancy between both CA and CA–w rules described hereabove highlights henceforth the simplicity of the CA–w model. It should be noted that the channels of a node can be seen as the partitions of a cell as in "partitioned CA" [52]. Therefore a similar modeling can be done by partitioned CA. Another way of modeling such a system would be to use a hexavalent FHP–like lattice-gas [20]; but here the purpose is to avoid the two-stage timestep in order to save time, with only one clock cycle instead of two.

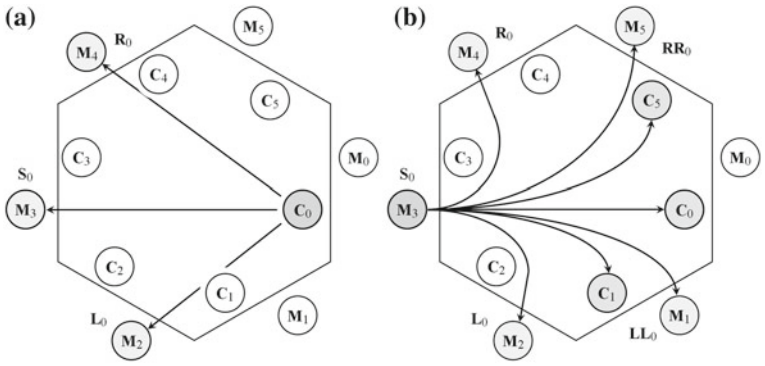

Fig. 6.11 Wide–range neighborhoods: **a** The 3–neighborhood $\mathcal{M}_0$ of receiver C_0. **b** The 7–neighborhood $\hat{\mathcal{M}}_0$ of sender S_0 extended to LL_0 and RR_0. Channels C_0 and S_0 coincide either as receiver or as sender

6.3.3 Programming Issues

Some more details are revealed for the reader interested in writing a simulation program. The following pseudo-codes show the algorithms for implementation of the CA and CA–w models. Missing items and notes "(*)" are explained afterwards.

6.3.3.1 CA Model

```
FOR EACH cell IN cellfield DO
  // compute own move condition, requires evaluation of copy- and delete neighborhood
1 cell.agent.can_move = (no agent or obstacle in front) AND                    (*)
                        (no other agent with higher priority can move to cell in front)

  // for each possible sending neighbor, requires evaluation of the copy-neighborhood
  // check if an agent wants to move to me and select one with the highest priority
2 neighbor(sending).agent.can_move(to_me) =
                              neighbor(sending).is(agent) AND
                              no other agent with higher priority wants to move to me

  // compute moving rule
3 cell_next<- empty   IF cell.is(empty) AND no agent can move to me // remains empty
           <- agent   IF cell.is(agent) AND receiving cell occupied // blocked
           <- empty   IF cell.is(agent) AND cell.agent.can_move     // delete (*)
           <- neighbor(sending).agent
                IF cell.is(empty) AND neighbor(sending).agent.can_move(to_me) // copy
ENDFOREACH

FOR EACH cell IN cellfield DO
5  cell <- cell_next // synchronous updating
ENDFOREACH
```

Statement 1 evaluates if the agent situated on its cell can move; this evaluation requires an extended neighborhood because of possible conflicts, namely the union of the copy- and delete neighborhood. Statement 2 evaluates if a neighboring agent can move to the current cell; this evaluation requires the copy-neighborhood only. In Statement 3 an agent may move, by deleting it by the sending cell and copying it by the receiving cell. Statement 5 performs a synchronous updating of the whole cell field.

The following simulation using the CA–w model is more simple.

6.3.3.2 CA–w Model

```
FOR EACH cell IN cellfield DO
  // for each possible sending neighbor, requires evaluation of the copy-neighborhood
  // check if an agent wants to move to me and select one with the highest priority
2 neighbor(sending).agent.can_move(to_me) =
                              neighbor(sending).is(agent) AND
                              no other agent with higher priority wants to move to me

  // compute moving rule, now without delete
3 cell_next<- empty   IF cell.is(empty) AND no agent can move to me // remains empty
```

```
            <- agent   IF cell.is(agent) AND receiving cell occupied // blocked
            <- neighbor(sending).agent
                  IF cell.is(empty) AND neighbor(sending).agent.can_move(to_me) // copy

  // extra CA--w operation: write on neighbor (deletion of agent when moving)
4 neighbor(sending).agent <- empty IF neighbor(sending).agent.can_move(to_me) //delete
ENDFOREACH

FOR EACH cell IN cellfield DO
5  cell <- cell_next // synchronous updating
ENDFOREACH
```

Compared to the CA program, the evaluation of the move condition (Statement 1 in CA program (*)) is omitted. Therefore the delete-neighborhood need not to be checked. As a consequence, the `delete(*)` line in Statement 3 of the CA program is also omitted and replaced by the additional Statement 4. Through this statement an agent on a sending cell is deleted by the receiving cell. The advantage of the CA–w program is that it is more concise and less expensive, because the `move` condition in Statement 1 needs not to be computed.

6.4 Router Efficiency and Deadlocks

Two test cases will be used for evaluation, where k is the number of agents, s the number of source nodes and m the number of target nodes:

1. **First Test Case** ($m = 1, k = s$). All agents move to the same common target. We will consider the case $k = N - 1$, meaning that initially an agent is placed on each site (except on the target). In this case the optimal performance of the network would be reached if the target consumes six messages in every timestep ($t = (N - 1)/6$). In addition, the target location is varying, with a maximum of N test configurations in order to check the routing scheme exhaustively. We recall that the T–grid is vertex-transitive, so the induced routing algorithm must yield the same result for all N cases!
2. **Second Test Case** ($k = s = m$). The sources are mutually exclusive (each source is used only once in a message set) as well as targets. Source locations may act as targets for other agents, too. We consider the case $k = N/2$ that was also used in preceding works [39, 40] for comparison. Note that the minimum number of timesteps t to fulfil the task is the longest distance between source and target which is contained in the message set. For a high initial density of agents the probability is high that the longest distance is close to the diameter of the network. Thus the best case would be $t = D_n$.

6.4.1 Efficiency of Deterministic Routing

Using one agent only in the router, it will travel always on a minimal route. More agents are also using a minimal route, but sometimes they have to wait due to traffic congestion.

6.4.1.1 First Test Case

In the first test case scenario, $k = N - 1$ messages move to the same common target from all other nodes. All possible or a large number of initial configurations differing in the target location were tested (Table 6.1). The results are the same for all tested initial configurations. This means that the router works totally symmetric as expected. An optimal router would consume in every generation six agents at the target, leading to an optimum of $t_{opt} = k/6$. It is difficult to reach the optimum, because the agents would need a global or a far view in order to let the agents move simultaneously in a cohort. Here an agent needs an empty receiver channel in front in order to move, thus empty channels are necessary to signal to the agents that they can move.

As the router is completely filled with agents at the beginning (one agent in each node except the target node), there exist some agents which have as travel distance the diameter D_n. Therefore the ratio t/D_n (B/C in Table 6.1) is significantly higher than one, slightly higher than $\sqrt{N}/2$. On the other hand, the ratio $t/(k/6) = B/D$ is quite good and relatively constant, that is $B/D \approx 2$ for large N, which is almost optimal because each agent needs an empty channel in front when moving without deviation on the minimal route. This phenomenon is easy to understand and has a close relationship with Traffic Rule 184 in a 1–dimensional system: a car with a car straight ahead cannot move and must wait for the next timestep.

Table 6.1 First test case: $k = N - 1$ messages travel from all disjoint sources to the same common target

Nodes N	Number of configurations (destinations) checked	(B) Message transfer time [steps]	(C) Diameter	Ratio B/C	(D) N/6	Ratio B/D
$4 = 2 \times 2$	all 4	1	1	1	1	1
$16 = 4 \times 4$	all 16	5	2	2.5	3	1.67
$64 = 8 \times 8$	all 64	23	5	4.6	11	2.09
$256 = 16 \times 16$	all 256	89	10	8.9	43	2.07
$1024 = 32 \times 32$	all 1024	351	21	16.7	171	2.05
$4096 = 64 \times 64$	64	1384	42	32.95	683	2.03

Message transfer time (in *timesteps*) in the T–grid, averaged over the number of checked initial configurations. The time is independent of the position of the target

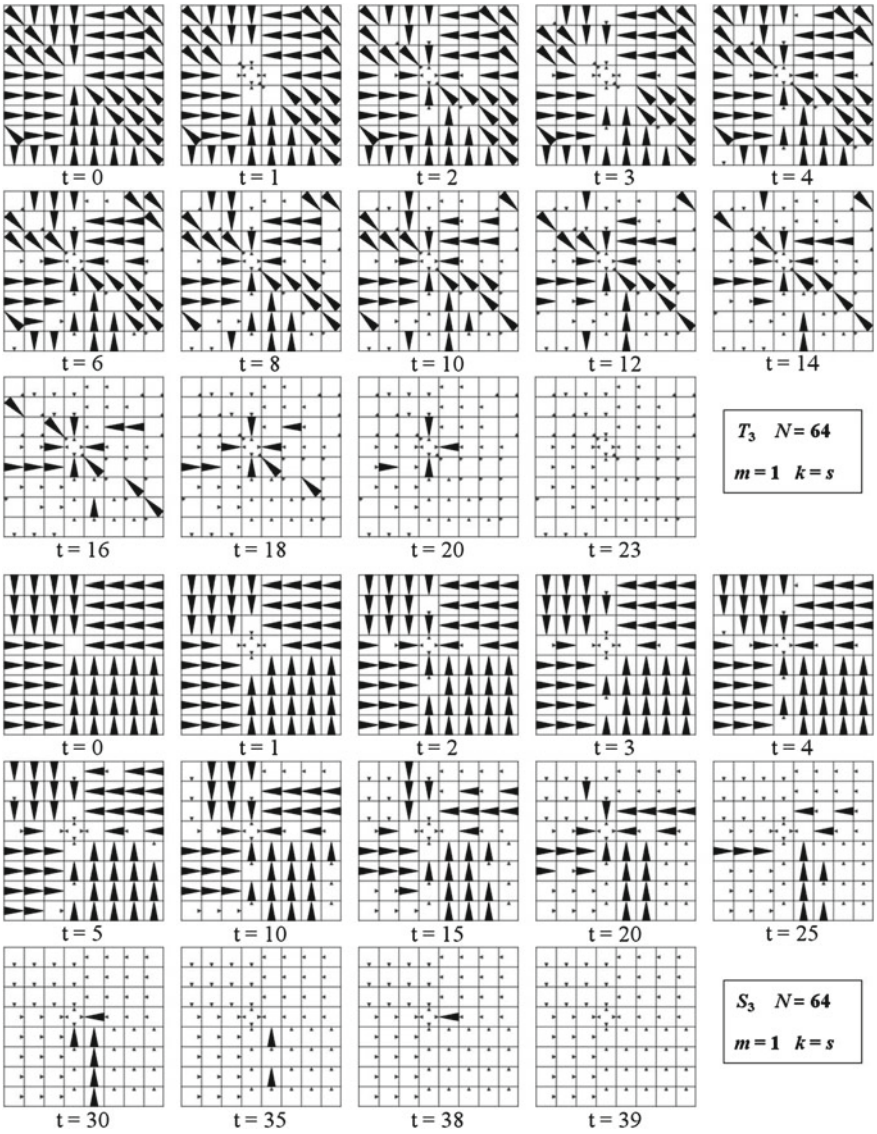

Fig. 6.12 Simulation snapshots for the first scenario in a 8×8 grid T_3, $N - 1$ agents moving to the same target position. Agents are depicted as black triangles, visited channels as small grey triangles: directions are symbolized by $(\rightarrow\nwarrow\downarrow\leftarrow\searrow\uparrow)$. Snapshots on S_3 are also displayed for comparison

A simulation sequence of this case is shown in Fig. 6.12 for the 8×8 grid T_3 and S_3 is also displayed for comparison.

6.4.1.2 Second Test Case

This test case was already used in a previous work [40] and is used for comparison. Therein, the agents were controlled by a finite state machine FSM: optimized, evolved agents were used, choosing a random direction with probability 0.3 % in order to avoid deadlocks, with only one agent per node. Ratio A/B in Table 6.2 shows that even the deterministic router with six channels performs significantly better, with $A/B \approx 2.5$ for $N = 1024$. The main reason is that here a node can host six agents, not only one, and therefore the congestion is considerably lower. The ratio (B/C) is noteworthy and shows that the mean transfer time is close to the diameter. This phenomenon is again easy to understand from Traffic Rule 184 but now in a fluid traffic. A simulation sequence of this case is shown in Fig. 6.13.

6.4.2 Efficiency of Adaptive Routing

An adaptive routing protocol was designed in order to speed up the message set transfer time and to avoid deadlocks during the run, although this could not be proved. When an agent computes a new direction and whenever the channel in that direction is occupied, the agent chooses the other minimal direction if there is a choice at all.

Table 6.2 Second test case: $k = N/2$ messages travel from disjoint sources to disjoint targets

Nodes N	Number of configurations checked for B randomly generated	(A) time steps, FSM controlled agent	(B) time steps, 6 channels non-adaptive	Ratio A/B	(C) Diameter	Ratio B/C	Time steps, 6 channels adaptive
$4 = 2 \times 2$	32	3.756	1	3.76	1	1	1
$16 = 4 \times 4$	256	8.528	2.520	3.38	2	1.260	2.520
$64 = 8 \times 8$	256	14.641	5.852	2.50	5	1.170	5.648
$256 = 16 \times 16$	256	28.848	12.070	2.39	10	1.207	11.574
$1024 = 32 \times 32$	256	58.438	23.367	2.50	21	1.113	22.648
$4096 = 64 \times 64$	256	128.087	45.199	2.83	42	1.076	44.082
$16384 = 128 \times 128$	128	300.330	87.789	3.42	85	1.033	86.668

Message transfer time (in *timesteps*) in the T–grid, averaged over the number of checked initial configurations. Routing with six channels per node performs significantly better (ratio A/B) than FSM controlled agents (one per node)

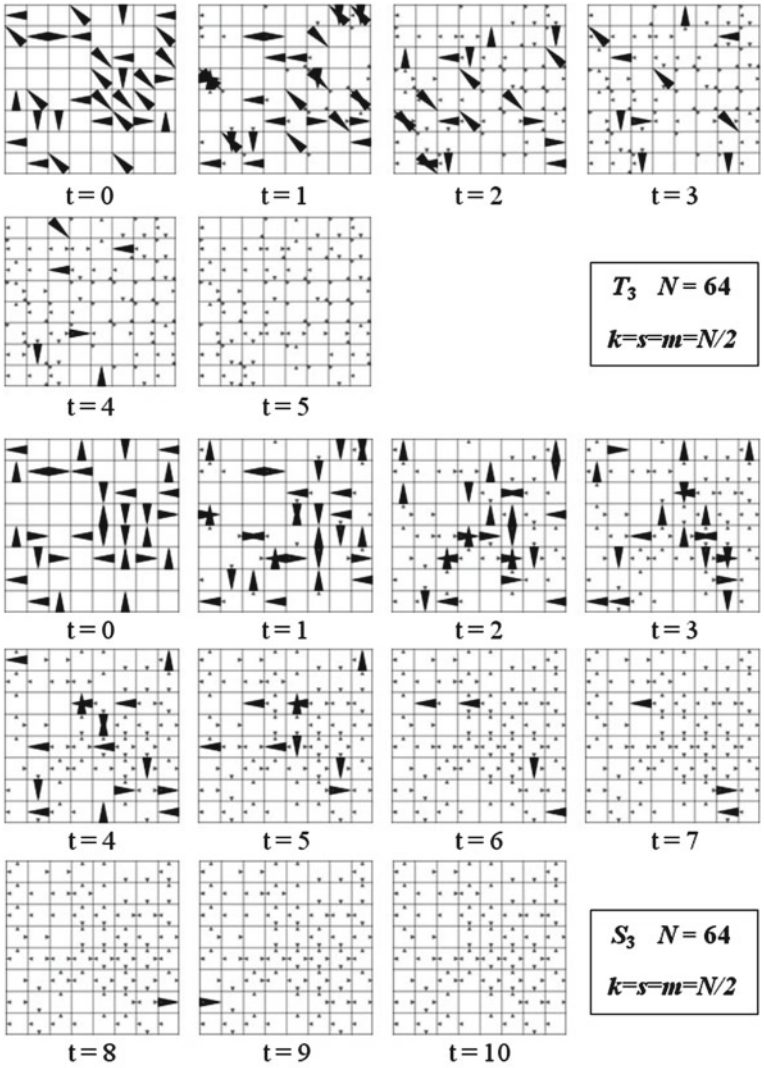

Fig. 6.13 Simulation snapshots for the second scenario in a 8×8 grid T_3, 32 agents moving to their assigned target. Agents are depicted as *black* triangles, visited channels as small *grey* triangles: directions are symbolized by $(\rightarrow\nwarrow\downarrow\leftarrow\searrow\uparrow)$. Snapshots on S_3 are also displayed for comparison

6.4.2.1 First Test Case

For this scenario with a common target the performance of adaptive routing is the same as for the deterministic routing. The reason is that all routes to the target are heavily congested. This means that the adaptive routing can hardly be better, but it is also not worse for the investigated case.

6.4.2.2 Second Test Case

For this scenario with randomly chosen sources and targets, the adaptive routing performs slightly—but only slightly—better. That means that agents' minimal path is seldom rerouted because of the fluid traffic. For example, for $N = 1024$, the message transfer time is reduced by 4.1 %. There seems to be more potential to optimize the behavior of the agents (using an FSM, or using a larger neighborhood) in order to guide them in a way that six agents are almost constantly consumed by the target.

6.4.3 Deadlocks

A trivial deadlock can be produced if all $6N$ channels contain agents (fully packed), thus no moving is possible at all. Another deadlock appears if $M = 2^n$ agents line up in a loop on all the channels belonging to one lane, and all of them have the same lane direction. Then the lane is completely full and the agents are stuck. To escape from such a deadlock would only be possible if the agents can deviate from the shortest path, e.g. by choosing a random direction from time to time. More interesting are the cyclic deadlocks where the agents form a loop and are blocking each other (no receiving channel is free in the loop). Two situations were investigated.

First situation `[Right Loop]` (Fig. 6.14). An empty node Ω is in the center of six surrounding nodes, let us call them A_0, A_1,... A_5 clockwise. Agent at A_0 wants to go to A_2, A_1 to A_3... in short the A_i want to go to A_{i+2} all around. Note that each agent has two alternatives: going first via Ω through the center or going first to a surrounding node (e.g., agent at A_0 can go first to Ω and then to A_2, or first to A_1 and then to A_2). Whether a deadlock appears depends on the initial assignments to the channels. If the initial assignments of all agents are "`use the left channel first`" via surrounding nodes, then the agents block each other cyclically. Otherwise they can move via the center node Ω and no deadlock occurs. Thereby it is assumed that the channels in Ω are empty or become empty after some time and are not part of other deadlocks.

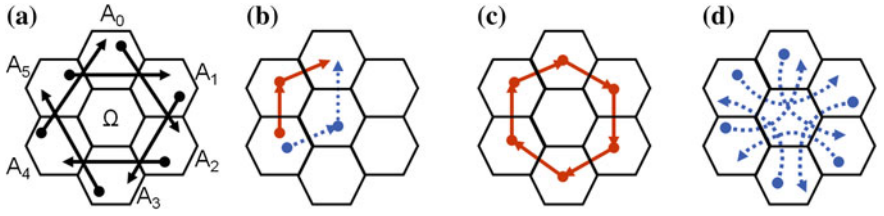

Fig. 6.14 A possible deadlock situation. **a** Agents targets, target = source + 2 mod 6. **b** The alternate paths, two min paths *solid* leftmost first, *dotted* rightmost first. **c** Cyclic deadlock appears if leftmost subpath is taken first. **d** No deadlock if rightmost subpath is taken first

Second situation `[Left Loop]`. This situation is symmetric to the right loop, except that the loop direction is now counterclockwise. A deadlock will appear if the initial directions of all agents are "`use the right channel first`".

If an initial configuration includes a right loop and a left loop, then at least one deadlock will appear if the initial assigned channel is fixed to the left or to the right. There are several ways to dissolve such deadlocks:

1. A spatial inhomogeneity is used, e.g., agents at *odd* nodes use initially the left subroute channel and agents at *even* nodes use the right subroute channel. The partition "odd–even" means $x + y \equiv 1$ or $x + y \equiv 0$ respectively, under addition modulo 2. This kind of partition, among others, was examined and did work for a limited set of experiments. Another similar way would be to randomize the initial subroute/channel assignment. This may be very successful but still there exists a very low probability for a deadlock.
2. The choice between the two minimal subroutes is taken randomly, or the choice may depends on an extended neighborhood.
3. It would be possible to deviate in a deterministic or non-deterministic way from the minimal route: for example an agent could move side-backwards if the whole area in direction of the target is blocked.
4. It would be possible to redistribute the channels during the run, by using a two-stage interaction-advection transition similar to the FHP lattice-gas [20]: move or don't move, then redistribute. In this case, the initially assigned channels and the used channels during the run could be dynamically rearranged.

6.5 Summary

The properties of a family of scalable 6–valent triangular tori were studied herein and for this family a minimal routing protocol was defined. A novel router with six channels per node was modeled as a multi-agent system within the cellular automata paradigm. In order to avoid the redundant computation of the moving condition, the CA–w model was used for implementation, that allows the receiving cell to copy the agent and to delete it on the sending cell. Thereby the description becomes more natural and the simulation faster. Both classical CA and new CA–w models were presented and compared. Each agent has a computed direction defining the new channel in the adjacent next node. The computed direction is a "minimal" direction leading on the shortest path to the target. The novel router is significantly faster (2.5 times for 1024 nodes) than an optimized reference router with one agent per node. In addition, an adaptive routing protocol was defined, which prefers the leftmost channel of a minimal route if the rightmost channel is occupied. Thereby a speed-up of 4.1 % for 1024 nodes was reached.

Deadlocks may appear for special situations when the system is overloaded, or when a group of agents form a loop. In order to avoid some of the deadlocks the initial subpath's direction can be alternated in space, or an adaptive protocol can be used.

The defined adaptive protocol switches to the other alternate minimal subpath in the case where the channel of the prior subpath is occupied. This adaptive protocol leads also to a higher throughput in the case of congestion. In order to dissolve deadlocks securely, a random or pseudo-random component should be introduced that may also allow the agents to bypass congested routes.

Further work can be aimed towards more intelligent agents in case of congestion through optimizing their behavior by using a genetic algorithm. Moreover, previous comparative works on the performance of agents moving either in the 4–valent S–grid or in the 6–valent T–grid [40, 53], with a speedup on T over S according to their diameter ratio (refer back to Eq. (6.2)), emphasize again our choice of triangular lattice explained in Sect. 6.1.3. The routing protocol could also be simplified by exploring the symmetries of the isotropic triangular grid: it is conjectured that this approach may drastically reduce the cost of the router, at least in a deterministic or adaptive context [54, 55].

References

1. Woolridge, M., Jenning, N.R.: Intelligent agents: theory and practice. Knowl. Eng. Rev. **10**(2), 115–152 (1995)
2. Franklin, S., Graesser, A.: Is it an agent, or just a program?: a taxonomy for autonomous agents, In: Müller, J.P., Woolridge, M., Jennings, N.R. (eds.), Proceedings of ECAI'96 Workshop on Intelligent Agents III, Agent Theories, Architectures, and Languages, pp. 21–35. Springer, New York (1997)
3. Holland, J.H.: Emergence: From Chaos To Order. Perseus Book, Cambridge (1998)
4. Woolridge, M.: An introduction to multiagent systems, 2nd Ed.. Wiley, New York (2002)
5. Pais, D.: Emergent collective behavior in multi-agent systems: an evolutionary perspective. Ph.D. Dissertation, Princeton University, Princeton (2012)
6. Schweitzer, F.: Brownian Agents and Active Particles. Collective Dynamics in the Natural and Social Sciences, Springer Series in Synergetics, Springer, Berlin (2003)
7. Shannon, CL. E.: Presentation of a maze-solving machine. In: 8th Conference of the Josiah Macy Jr. Found., Cybernetics pp. 173–180 (1951)
8. Blum, M., Sakoda, W.: On the capability of finite automata in 2 and 3 dimensional space. In: 18th IEEE Symposium on Foundations of Computer Science, pp. 147–161. (1977)
9. Fraigniaud, P., Ilcinkas, D., Guy, P., Andrzej, P., Peleg, D.: Graph exploration by a finite automaton. In: Fiala, J., et al. (eds.) MFCS 2004, LNCS 3153. pp. 451-462, (2004)
10. Spezzano, G., Talia, D.: The CARPET programming environment for solving scientific problems on parallel computers. Par. Dist. Comp. Practices **1**, 49–61 (1998)
11. Freiwald, U., Weimar, J.R.: JCASim—a Java System for Simulating Cellular Automata, Theory and Practical Issues on Cellular Automata, pp. 47–54. Springer, London (2001)
12. Rosenberg, A.L.: Cellular ANTomata. Adv Complex Syst. **15**(6) (2012)
13. Hoffmann, R.: Rotor-routing algorithms described by CA-w. Acta Phys. Polonica B Proc. Suppl. **5**(1), 53–68 (2012)
14. Hoffmann, R.: The GCA–w massively parallel model. In: Malyshkin, V. (ed.) LNCS 5698, pp. 194–206. (2009)
15. Hoffmann, R.: GCA-w: Global cellular automata with write-access. Acta Phys. Polonica B Proc. Suppl. **3**(2), 347–364 (2010)
16. Hoffmann, R.: GCA-w algorithms for traffic simulation. Acta Phys. Polonica B Proc. Suppl. **4**(2), 183–200 (2011)

17. Grünbaum, B., Shephard, G.C.: Tilings and Patterns. Freeman & Co., New York (1987)
18. Désérable, D.: A Terminology for 2*d* Grids, RR INRIA n° 2346, pp. 1–31 (1994)
19. Hardy, J., Pomeau, Y., de Pazzis, O.: Time evolution of a two-dimensional classical lattice system. Phys. Rev. Lett. **31**, 276–279 (1973)
20. Frisch, U., Hasslacher, B., Pomeau, Y.: Lattice-gas automata for the Navier-Stokes equation. Phys. Rev. Lett. **56**, 1505–1508 (1986)
21. Désérable, D., Dupont, P., Hellou, M., Kamali-Bernard, S.: Cellular automata in complex matter. Complex Syst. **20**(1), 67–91 (2011)
22. Morillo P., Fiol, M.A.: El diámetro de ciertos digrafos circulantes de triple paso. Stochastica **10**(3):233–249 (1986)
23. Bermond, J.C., Comellas, F., Hsu, D.F.: Distributed loop computer networks: a survey. J. Par. Dist. Comp. **24**:2–10 (1995)
24. Morillo, P., Comellas, F., Fiol, M.A.: Metric problems in triple loop graphs and digraphs associated to an hexagonal tessellation of the plane. TR 05–0286, pp. 1–6 (1986)
25. Davis, A.L., Robison, S.V.: The architecture of the FAIM-1 symbolic multiprocessing system. In: Proceedings of 9th International Joint Conferences on Artificial Intelligence pp. 32–38. (1985)
26. Davis, A.: Mayfly—a general-purpose, scalable, parallel processing architecture. J. LISP Symbolic Comput. **5**:7–47 (1992)
27. Chen, M.-S., Shin, K.G., Kandlur, D.D.: Addressing, routing and broadcasting in hexagonal mesh multiprocessors. IEEE Trans. Comp. **39**(1), 10–18 (1990)
28. Albader, B., Bose, B., Flahive, M.: Efficient communication algorithms in hexagonal mesh interconnection networks. J. LaTeX Class Files **6**(1):1–10 (2007)
29. Désérable, D.: Embedding Kadanoff's scaling picture into the triangular lattice. Acta Phys. Polonica B Proc. Suppl. **4**(2):249–265 (2011)
30. Gowrisankaran, C.: Broadcasting on recursively decomposable Cayley graphs. Discrete Appl. Math. **53**:171–182 (1994)
31. Désérable, D.: A family of Cayley graphs on the hexavalent grid. Discrete Appl. Math. **93**(2–3), 169–189 (1999)
32. Désérable, D.: Minimal routing in the triangular grid and in a family of related tori. In: Euro-Par'97 Par. Proceedings of Passau, Germany, LNCS 1300 pp. 218–225. (1997)
33. Désérable, D.: Broadcasting in the arrowhead torus. Comput. Artif. Intell. **16**(6), 545–559 (1997)
34. Heydemann, M.-C., Marlin, N., Pérennes, S.: Complete rotations in Cayley graphs. Eur. J. Comb. **22**(2), 179–196 (2001)
35. Fraigniaud, P., Lazard, E.: Methods and problems of communication in usual networks. Discrete Appl. Math. **53**(1–3), 79–133 (1994)
36. Dally, W.J., Seitz, C.L.: The torus routing chip. Distrib. Comput. **1**, 187–196 (1986)
37. Xiang, Y., Stewart, I.A.: Augmented k-ary n-cubes. Inf. Sci. **181**(1), 239–256 (2011)
38. Metropolis, N., Rosenbluth, A.W., Rosenbluth, M.N., Teller, A.H., Teller, E.: Equations of state calculations by fast computing machines. Chem. Phys. **21**(6):1087–1092 (1953)
39. Ediger, P., Hoffmann, R., Désérable, D.: Routing in the triangular grid with evolved agents. J. Cell. Automata **7**(1), 47–65 (2012)
40. Ediger, P., Hoffmann, R., Désérable, D.: Rectangular vs triangular routing with evolved agents. J. Cell. Automata **8**(1–2), 73–89 (2013)
41. Loh, P.K.K., Prakash, E.C., Performance simulations of moving target search algorithms, Int. J. Comput. Games Technol. **2009**:1–6 (2009) (New York, Hindawi Publ. Corp.)
42. Korf, R.E.: Real-time heuristic search. Artif. Intell. **42**(2–3):189–211 (1990)
43. Ediger, P., Hoffmann, R., Routing based on evolved agents. In: 23rd PARS Workshop on Parallel Systems and Algorithms, ARCS, Hannover. Germany 2010, pp. 45–53 (2010)
44. Ediger, P., Hoffmann, R.: CA models for target searching agents. Automata São José dos Campos. Brazil, ENTCS **252**:41–54 (2009)
45. Shamei, A., Bose, B., Flahive, M.: Adaptive routing in hexagonal torus interconnection networks, IEEE High Performance Extreme Computing Conference (HPEC), (2013)

46. Leighton, F.T., Leiserson, C.E.: Wafer-scale integration of systolic arrays. IEEE Trans. Comput. **C-34**:448–461 (1985)
47. Cole, R.J., Maggs, B.M., Sitarman, R.K.: Reconfiguring arrays with faults—Part I: Worst-case faults. Siam J. Comp. **26**(6):1581-1611 (1997)
48. Duato, J., Yalamanchili, S., Ni, L.: Interconnection Networks. Morgan Kaufmann, San Francisco (2002)
49. Hoffmann, R., Désérable, D.: Efficient minimal routing in the triangular grid with six channels. In: Malyshkin, V. (ed.) LNCS 6873 pp. 152–165, (2011)
50. Désérable, D.: Arrowhead and Diamond Diameters (unpublished)
51. Ediger, P.: Modellierung und Techniken zur Optimierung von Multiagentensystemen in Zellularen Automaten, Dissertation, TU Darmstadt, Darmstadt, Germany (2011)
52. Poupet, V.: Translating partioned cellular automata into classical type cellular automata. Journées Automates Cellulaires, Uzès, France pp. 130–140 (2008)
53. Hoffmann, R., Désérable, D.: All-to-All communication with cellular automata agents in 2d Grids—topologies, streets and performances. J. Supercomp. **6**9(1):70–80 (2014)
54. Désérable, D.: Hexagonal Bravais-Miller routing of shortest path (unpublished)
55. Miller, W.H.: A Treatise on Crystallography. J. & J.J, Deighton, Cambridge (1839)