Self-stabilizing Distributed Algorithms by Gellular Automata

Taiga Hongu, Masami Hagiya

Self-stabilizing Distributed Algorithms by Gellular Automata

Taiga Hongu(✉) and Masami Hagiya

The University of Tokyo, Tokyo, Japan
hongu314@g.ecc.u-tokyo.ac.jp, hagiya@is.s.u-tokyo.ac.jp

Abstract. Gellular automata are cellular automata with the properties of asynchrony, Boolean totality, and non-camouflage. In distributed computing, it is essential to determine whether problems can be solved by self-stable gellular automata. From any initial configuration, self-stable gellular automata converge to desired configurations, as self-stability implies the ability to recover from temporary malfunctions in transitions or states. In this paper, we show that three typical problems in distributed computing, namely, solving a maze, distance-2 coloring, and spanning tree construction, can be solved with self-stable gellular automata.

Keywords: Gellular automata · Solving a maze · Distance-2 coloring · Spanning tree construction · Self-Stability

1 Introduction

Many studies have been conducted to implement cellular automata using physical or chemical materials, such as [6,7,13]. These include recent efforts to implement cellular automata by reaction-diffusion systems in porous gels [4]. One motivation for implementing cellular automata using gels is to develop smart materials that can autonomously respond to external environments.

The term *gellular automata (GA)* was coined in [3], where the diffusion of DNA molecules is controlled by opening and closing holes between cells. Gellular automata were later formalized as cellular automata with the features of asynchrony, Boolean totality, and non-camouflage in [10,11], where two types of DNA molecules were assumed, one for states of cells and the other for signals transmitting states.

In the research along the latter direction, the computational universality of gellular automata was shown [10], and the computational power of gellular automata as distributed systems was investigated in [9]. Self-stability is a crucial factor in distributed computing. According to [1], self-stability is the ability of a system to converge to states with desired conditions from any initial state. If gellular automata are self-stable, they recover desired conditions even if temporary malfunctions occur in transitions or states. Smart materials are expected to have this property.

H. Zenil (Ed.): AUTOMATA 2020, LNCS 12286, pp. 1–13, 2020.
https://doi.org/10.1007/978-3-030-61588-8_7

In our previous study, we developed self-stable gellular automata that solved a maze [12] using a distributed algorithm similar to Lee's algorithm [5] under the restrictions that the number of states is finite and state transitions are asynchronous. However, this system takes time to detect undesired situations, such as loops.

In this paper, we reconsider the transition rules and target configurations of the gellular automata and present new transition rules that can solve a maze in a relatively short time. Moreover, we examine two other typical problems in distributed computing: distance-2 coloring and spanning tree construction. Like solving a maze, we confirm that these problems can be solved with self-stable gellular automata and explain how to design suitable systems for this purpose.

There are a number of studies on cellular automata solving maze problems such as [8], but we could not find self-stable ones except ours. Self-stable cellular automata for k-coloring are proposed by a very recent study [2], but typical distributed problems such as mazes and spanning trees are not dealt with. We conjecture that their definition of stability is derived from ours, but detailed comparison is left for future work.

The gellular automaton for solving a maze and others are demonstrated by the simulator available at https://cell-sim.firebaseapp.com/. Select "New Maze" in "Simulation Target."

2 Solving a Maze

2.1 Definitions

In this paper, a two-dimensional square lattice and von Neumann neighborhood are assumed. Each cell in the square lattice has a state from the following set.

$$\{W, B, S, T_0, T^*, T^\dagger, R\} \cup \{P'_i, P''_i, P^*_i, P^\dagger_i \mid i = 0, 1, 2, .., n-1\}$$

The states T_0, T^*, and $T^\dagger$ are denoted T. The states P'_i, P''_i, P^*_i, and $P^\dagger_i$ are collectively denoted P_i. If i is arbitrary, P_i is simply denoted P. The parameter n is the number of states in P_i and is equal to 5 in this section.

The state W denotes a *wall* of a maze, which does not make any transitions. The state B denotes a *blank*, which may make a transition to P or R. The states S and T are the *starting point* and the *terminal point*, respectively, and they do not make any transitions. The state R indicates that it is reachable from the terminal point, and a path consisting of P stretches on cells in R.

The superscripts $'$, $''$, * and † are used for detecting junctions by rules (9–21), explained below, and the subscripts i are used for directing paths.

Definition 1 (transition rule). *A transition rule consists of three components: the current state of a cell that makes a transition, a condition to be satisfied by the neighboring cells, and the next state that the cell will take.*

Definition 2 (asynchrony, Boolean totality, non-camouflage). *Cellular automata are asynchronous if cells make transitions asynchronously, that is, each cell may either make a transition by following a transition rule or do nothing at each step. Cellular automata are Boolean totalistic if the conditions of transition rules depend only on neighboring cells being in a particular state, not on the direction or number of cells. Cellular automata are non-camouflage if no conditions of transition rules contain the current state of the cell that makes a transition.*

A transition rule of asynchronous Boolean-totalistic non-camouflage cellular automata is defined as follows.

$$s_1\ (t_1 \wedge \cdots \wedge t_m \wedge \neg t_{m+1} \wedge \cdots \wedge \neg t_{m+n})\ \rightarrow\ s_2$$

In this rule, s_1 is the current state, $t_1 \wedge \cdots \wedge t_m \wedge \neg t_{m+1} \wedge \cdots \wedge \neg t_{m+n}$ is the condition, and s_2 is the next state. This means that a cell in state s_1, whose neighborhood contains cells in states $t_1, \ldots, t_n$ and does not contain cells in states $t_{m+1}, \ldots, t_{m+n}$, can make a transition to state s_2. By the non-camouflage property, s_1 does not appear among $t_1, \ldots, t_{m+n}$.

Definition 3 (configuration, run, step). *A configuration is a mapping from cells at lattice points in a square lattice to states, and a run is an infinite sequence of configurations, each of which, except for the first one, is obtained by applying the transition rules to the previous configuration. A transition step is the process of transforming from configuration C_1 to configuration C_2, which is obtained by having each cell in C_1 make a single transition or do nothing. Due to asynchrony and possibility that several rules can be applied to a state, a configuration sometimes has more than one next possible configuration. In this case, one of them is chosen non-deterministically.*

Definition 4 (passage, path, loop, junction). *A passage is a sequence of neighboring cells, each of which is in state B, R, or P. A maze is connected if there is a passage from the starting point S to the terminal point T in the maze.*

A path is a sequence of neighboring cells in states $\ldots, P_0, P_1, \ldots, P_{n-1}, P_0, \ldots$, where the indices are incremented in $(\mathbb{Z}/n\mathbb{Z})$.

If a path has both ends, that is, a head that is not adjacent to P_{i-1} and a tail that is not adjacent to P_{i+1}, we say that the path is maximal. If the head of a maximal path is adjacent to T and its tail is adjacent to S, the maximal path is called a solving path.

If a path has no ends, it is called a loop. In particular, if a loop has no junctions (described later), we call that loop pure.

We say that a path has a junction if a cell in state P_i in the path is adjacent to two or more different cells in P_{i-1} (or S if $i = 0$) or two or more in P_{i+1} (or T). Such a cell in state P_i is called a collision point, which is also called an entrance in the former case and an egress in the latter case.

Definition 5 (solution). *A solution of a maze is a configuration in which there is only one maximal path, and its head (tail) is adjacent to the starting point S*

(the terminal point T, respectively). If the maze is connected, there are solutions, and if it is not connected, there are no solutions (Fig.1).

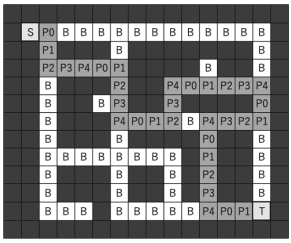

Fig. 1. An example of solutions of a maze (black cells denote W)

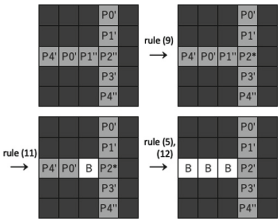

Fig. 2. Reduction of an entrance by rules (7–12)

Definition 6 (fair run). *A run R is fair if a certain configuration C appears in R infinitely often, and any configuration C' that can be obtained from C by a transition step also appears in R infinitely often.*

Throughout this paper, we assume non-deterministic models of computation. In probabilistic models such as Markov processes, probabilities of unfair runs are zero, i.e., runs are fair with probability 1.

Definition 7 (target configuration, self-stability). *Some configurations that are desirable (for a specific purpose) are defined as target configurations. Cellular automata are self-stable if in any fair run from any configuration, a target configuration appears in finite steps, and only target configurations appear after that.*

The above definition of self-stability is generally adopted in the field of distributed computing [1]. Even when a perturbation occurs in a target configuration, a new target configuration eventually appears if cellular automata are self-stable, because we can start a fair run from the resulting non-target configuration.

2.2 Procedure for Solving a Maze

2.2.1 Transition Rules

We introduce 21 transition rules of gellular automata for solving a maze.

(1)	$B\ (T_0)$	$\to\ R$	(12)	$P_i^*\ (\neg P''_{i-1})$	$\to\ P'_i$
(2)	$B\ (R)$	$\to\ R$	(13)	$P_i\ (P'_{i+1} \wedge P''_{i+1})$	$\to\ P_i^\dagger$
(3)	$R\ (S)$	$\to\ P'_0$	(14)	$P_i\ (T \wedge P''_{i+1})$	$\to\ P_i^\dagger$
(4)	$R\ (P_i \wedge \neg P_{i+2})$	$\to\ P'_{i+1}$	(15)	$P''_i\ (P_{i-1}^\dagger)$	$\to\ B$
(5)	$P_i\ (\neg T \wedge \neg R \wedge \neg P_{i+1})$	$\to\ B$	(16)	$P_i^\dagger\ (\neg P''_{i+1})$	$\to\ P'_i$
(6)	$P_i\ (\neg S \wedge \neg P_{i-1})$	$\to\ B$	(17)	$T_0\ (P_i)$	$\to\ T^*$
(7)	$P'_i\ ()$	$\to\ P''_i$	(18)	$T^*\ (P'_i \wedge P''_j)$	$\to\ T^\dagger$
(8)	$P''_i\ ()$	$\to\ P'_i$	(19)	$P''_i\ (T^\dagger)$	$\to\ B$
(9)	$P_i\ (P'_{i-1} \wedge P''_{i-1})$	$\to\ P_i^*$	(20)	$T^\dagger\ (\neg P'')$	$\to\ T_0$
(10)	$P_0\ (S \wedge P''_{n-1})$	$\to\ P_0^*$	(21)	$T^*\ (\neg P)$	$\to\ T_0$
(11)	$P''_i\ (P_{i+1}^*)$	$\to\ B$			

Each rule is actually a schema of rules and represents a number of concrete rules. For example, rule (4) represents $R\ (P'_i \wedge \neg P'_{i+2} \wedge \neg P''_{i+2}) \to P'_{i+1}$ and $R\ (P''_i \wedge \neg P'_{i+2} \wedge \neg P''_{i+2}) \to P'_{i+1}$ for each i, because P'_i and P''_i are collectively denoted by P_i.

If a cell in state B is adjacent to T_0 or R, rules (1–2) change its state to R. In this way, we can detect all reachable cells from the terminal point T. Once a cell in state B adjacent to the starting point S makes a transition to R, rules (3–4) generate a path from S and extend it while making as few loops as possible by preventing the path from joining to an existing path.

Rules (5–6) are intended to reduce dead ends of paths. If the head of a path is not adjacent to T and cannot stretch any more because there are no neighboring cells in state R, it changes back to state B. Similarly, if the tail of a path is not adjacent to S, it changes back to state B.

We reduce entrances with rules (7–12). First, cells in state P'_i or P''_i switch their states from P'_i to P''_i or P''_i to P'_i. Next, a cell adjacent to both P'_i and P''_i finds itself being an entrance and makes a transition to state P_i^*. Then, one (or more) of the paths joining at the entrance cell disappears gradually, and the entrance changes back to state P'_i.

Figure 2 shows the procedure for reducing entrances. We also reduce egresses with rules (13–16).

Rules (17–21) restrict the number of paths reaching the terminal point T to fewer than one. If the terminal point T_0 is adjacent to cells in state P, it makes a transition to state T^*, and no more cells in R are generated from it. As in the case of junctions, if the terminal point T is adjacent to several cells in P, one (or more) of the paths joining at T disappears gradually.

2.2.2 Self-stability of Solving a Maze

An initial configuration is a configuration that satisfies all of the following conditions:

(I-1) There is just one starting point S and one terminal point T, and these are not adjacent.
(I-2) The number of cells not in state W is finite.

A target configuration is a configuration that does not satisfy the above conditions for an initial configuration or that satisfies all of the following conditions:

(T-1) If the maze is connected, there is only one solving path from S to T^*.
(T-2) There are no maximal paths except solving paths.
(T-3) There are no cells in R adjacent to S, P, or B.
(T-4) If the maze is not connected, there is a cell in T_0 not adjacent to B or P.
(T-5) There are no junctions, that is, there are no cells in P_i adjacent to two or more cells in P_{i-1} (or S if $i = 0$), or two or more in P_{i+1} (or T).
(T-6) There are no cells in P_i^* or $P_i^\dagger$.

We now prove that these gellular automata are self-stable.

Theorem 1 (Self-stability of Solving a Maze). *Gellular automata with the above states, transition rules, and conditions of target configurations are self-stable.*

First, we show that from any initial configuration, a target configuration appears after a finite number of steps. Second, we show that once a target configuration appears, only target configurations appear afterward.

Lemma 1. *Assume that from any initial configuration, a target configuration can be obtained by some transition steps. Then a target configuration appears in any fair run.*

Proof. Assume that no target configuration appears in a fair run. As the cellular space is finite, the number of possible configurations is also finite. Therefore, there exists a configuration C that appears an infinite number of times in the run, and because of fairness, any configurations that can be obtained from C, including a target configuration, also appear in the run. This is a contradiction.

To prove the theorem, we first show that we can obtain a target configuration from any initial configuration by the following operations (i)–(iv) in order.

(i) We spread R by rules (1–2) until they can no longer be applied, then spread P by rules (3–4) until they can no longer be applied.
(ii) By applying rules (5–16), we remove all maximal paths except solving paths. Then there are only solving paths without junctions and pure loops. If there remain cells in P_i^* or $P_i^\dagger$, we get rid of them by applying rules (8,12,16).
(iii) If the maze is connected and there are solving paths, we move to (iv).

If the maze is connected but there are no solving paths, because the terminal point T is not adjacent to P, we change T to T_0 by applying rules (20–21). We then spread R from T_0 only on the passage that will be a solving path without junctions. When R is adjacent to P in a pure loop, we change all

R on the passage to P by applying rules (3–4) and remove the loop and the resulting path using rules (5–16). By repeating this process, a single solving path is obtained.

If there remain cells in P_i^* or $P_i^\dagger$, we get rid of them using rules (8,12,16). There should be no cells in R adjacent to B, S, and P because we spread R only on the passage of a solving path, and there should be no junctions.

If the maze is not connected, we also change T to T_0, as above. We then spread R to all cells reachable from T while removing pure loops by rules (3–4), (7–16). Then there should be no cells in R adjacent to B, S, and P and no junctions. Moreover, T_0 should not be adjacent to B, P.

(iv) If there are two or more solving paths, we keep one and remove the others by applying rules (17–21). Because there are no junctions in the paths, we can remove them by applying rule (5) until just one cell is adjacent to S. Finally, we change T to T^* by rule (17).

Figure 3 shows the operations (i)–(iv).

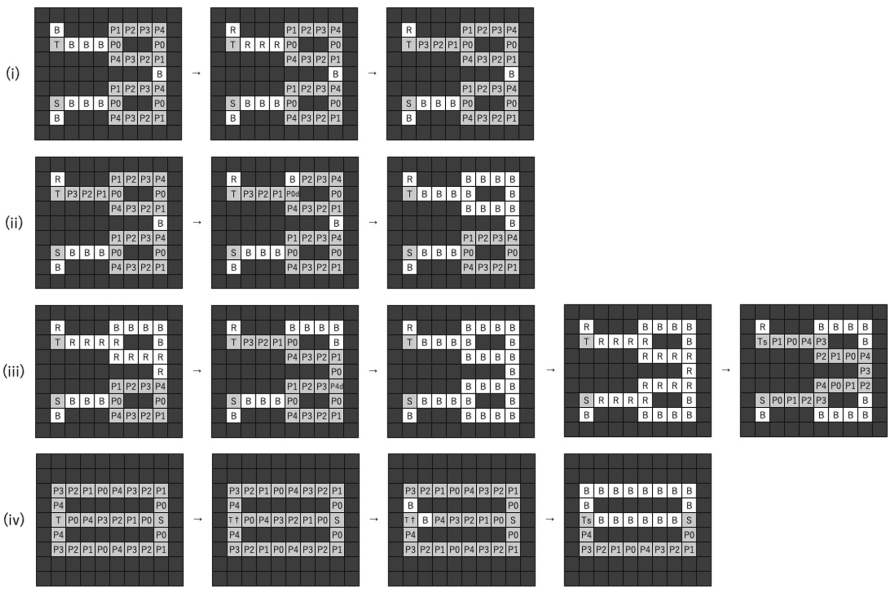

Fig. 3. Procedure of the operation (i–iv)

We now show that only target configurations appear after a target configuration is obtained. Table 1 shows the conditions satisfied after each transition step.

Table 1. Conditions of mazes satisfied after each operation (*init.* denotes initial configurations)

	(I)	(II)	(III)	(IV)	(V)	(VI)	(VII)	(VIII)
init.	x	x	x	x	x	x	x	x
(i)	o	o	x	x	o	x	x	x
(ii)	o	o	o	o	x	x	x	x
(iii)	o	o	o	o	o	o	x	x
(iv)	o	o	o	o	o	o	o	o

(I) There are no cells in R adjacent to S, P.
(II) There are no cells in B adjacent to R.
(III) There are no junctions and no cells in P_i^* or $P_i^\dagger$.
(IV) There are no maximal paths except solving paths.
(V) If the maze is unconnected, there is a cell in T_0 not adjacent to B, P.
(VI) If the maze is connected, there are solving paths.
(VII) If the maze is connected, each of the cells in S or T has just one neighboring cell in P.
(VIII) If the maze is connected, there is a cell in T^*.

After (iv), all of the above conditions (I–VIII) are satisfied. We can see that the conditions of the target configurations are satisfied. (T-1) holds because of (III), (VI), (VII), (VIII). (T-2), (T-3), (T-4), (T-5), and (T-6) hold because of (IV), (II), (V), (III), and (III). (In fact, if and only if all of the conditions (I–VIII) are satisfied, all of the conditions of target configurations (T-1–T-6) are satisfied.) Then only rules (7–8) can be applied to cells under target configurations, and they continue to satisfy (T-1–T-6). Therefore, after the first, only target configurations appear.

3 Distance-2 Coloring

3.1 Definitions

The space of cellular automata for solving the distance-2 coloring problem is the same as that for solving a maze. A state of a cell is either W, which represents a wall, or a pair (c, conf) of a *color state* c and a *conflict state* conf. Cells whose state is a pair (c, conf) is called *colored*.

A color state c is either c'_i or c''_i $(i = 1, 2, \ldots, n)$, and cells whose color state is c'_i or c''_i are considered to have the same color i. They are sometimes collectively denoted by c_i or c, as in the case of automata for solving a maze. The parameter n is the number of colors used, which is 13 in this section. A conflict state conf is a list of 0 or 1, such as $[0, 1, 0, ..., 1]$, whose length is n. The i-th element is 1 if there are two or more neighboring cells in the color i and is 0 otherwise. A cell in the color i may change its color if $\mathit{conf}[i]$ of any neighboring cell is 1 or it has a neighboring cell in the same color i.

Definition 8 (distance-2 coloring). *A colored cell is called unsafe if there is a neighboring cell in the same color as the cell or a pair of neighboring cells in the same color and safe otherwise. A configuration is called distance-2 colored if there are no unsafe cells.*

The color of any cell in a distance-2 colored configuration is different from those of the cells within a distance of two cells from it.

Figure 4 shows distance-2 coloring with five colors. Cells with different letters have different colors. Panel (a) shows a distance-2 colored configuration, but panel (b) does not because there are colored cells adjacent to two cells in R. (Note that cells W adjacent to more than two cells in the same color are allowed.)

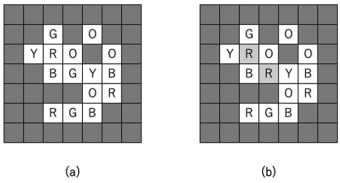

Fig. 4. An example of distance-2 coloring of cells

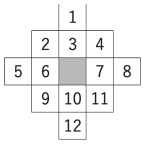

Fig. 5. 12-cells

3.2 Procedure for Distance-2 Coloring

In this section, we introduce the transition rules and the proof of self-stability of the automata for distance-2 coloring.

3.2.1 Transition Rules

The automata for distance-2 coloring have the following seven transition rules. The symbol $*$ expresses an arbitrary color state or conflict state. The symbol $conf|_{[i]=j}$ is a conflict state such that $conf[i] = j$, that is, $[*, \ldots, *, j, *, \ldots, *]$, where the i-th element is replaced by j.

$$
\begin{array}{lll}
(1) & (c'_i, conf)\ () & \rightarrow\ (c''_i, conf) \\
(2) & (c''_i, conf)\ () & \rightarrow\ (c'_i, conf) \\
(3) & (c_i,\ conf|_{[j]=0})\ ((c'_j, *) \wedge (c''_j, *)) & \rightarrow\ (c_i,\ conf|_{[j]=1}) \\
(4) & (c_i,\ conf|_{[j]=1})\ ((\neg c''_j, *)) & \rightarrow\ (c_i,\ conf|_{[j]=0}) \\
(5) & (c'_i, conf)\ ((c''_i, *)) & \rightarrow\ (c'_j, conf) \\
(6) & (c''_i, conf)\ ((c'_i, *)) & \rightarrow\ (c''_j, conf) \\
(7) & (c''_i, conf_1)\ ((*,\ conf_2|_{[i]=1})) & \rightarrow\ (c'_j, conf_1)
\end{array}
$$

These transition rules work as follows.

- (1–2): They switch the color states of cells from c_i' to c_i'' or c_i'' to c_i' to enable cells to recognize whether there are two or more neighboring cells in the same color.
- (3–4): If a cell has two or more neighboring cells in the same color i, they change $conf[i]$ of the cell from 0 to 1. If not, they change it from 1 to 0.
- (5–6): If a cell is adjacent to cells in the same color, they change the color of the cell to an arbitrary one.
- (7): If a cell is in a color state c_i'' and there is a neighboring cell whose $conf[i]$ is 1, they change its color to an arbitrary one.

3.2.2 Self-stability of Distance-2 Coloring

An initial configuration is a configuration in which the number of cells not in state W is finite. A target configuration is a configuration that does not satisfy the above condition for initial configurations or that satisfies all of the following conditions:

(T-1) There are no colored cells whose colors are the same as one of their neighboring cells.
(T-2) There are no colored cells that are adjacent to two or more neighboring cells with the same color.
(T-3) There are no colored cells whose conflict states are not $[0, 0, \ldots, 0]$.

Now we show that these gellular automata are self-stable.

Theorem 2 (Self-Stability of Distance-2 Coloring). *Gellular automata with the above states, transition rules, and conditions of target configurations are self-stable.*

As in the case for solving a maze, we consider the following operations (i–iii).

(i) If there is a cell such that $conf[i] = 1$ for some i and there is at most one neighboring cell in color i, we change its $conf[i]$ from 1 to 0 by applying rule (4).

(ii) If a cell and one of its neighboring cells are in the same color, we change its color according to rules (5–6). Also, if two or more neighboring cells of a cell are in the same color i, we first make both c_i' and c_i'' appear in the neighboring cells by rules (1–2). We then change its $conf[i]$ to 1 by rule (3) and finally change c_i'' by rule (7). In both cases, we choose the color to which the cell changes, except the color of neighboring cells and that of the cells to which they are adjacent. As the number of colors we cannot choose is at most 12 (as in Fig. 5), which is less than the number of colors, 13, we can always choose one, and the number of unsafe cells decreases.

(iii) By repeating (i–ii), we can change all of the unsafe cells to safe ones and their conflict states to $[0, 0, \ldots, 0]$.

Through the above operations (i–iii), a target configuration is obtained. Then only rules (1–2) can be applied to cells in target configurations, which does not change conditions (T-1–T-3). Therefore, after the first, only target configurations appear.

4 Spanning Tree

4.1 Definitions

The space of cellular automata for spanning tree construction is the same as that for solving a maze. A state of a cell is either W, which represents a wall, or a pair (P, C), which should belong to a spanning tree. Here P is called a *tree state* and C is called a *color-conflict state*. A tree state P is one of the following set:

$$\{r_i, t_i[c_j], l_i[c_j] \mid i = 0, 1, \ldots, m-1,\ j = 1, 2, \ldots, n\}$$

A cell whose tree state is r_i, $t_i[c_j]$, or $l_i[c_j]$ is called a *root*, an *inner node*, or a *leaf* of a spanning tree. The index i is called a *wave index*, and c_j is called a *parent color*. If i or j are arbitrary, r_i is denoted r, and $t_i[c_j]$ is denoted by t_i, $t[c_j]$, and t (and similarly for $l_i[c_j]$). A color-conflict state C is a pair of a *color state* and a *conflict state*, similar to the case of distance-2 coloring. The parameter m is the number of r_i, which is 6 in this section.

If a tree state of a cell is $t_i[c_j]$ or $l_i[c_j]$, it points to a neighboring cell whose color is j as its parent. In this manner, we can define a parent-child relation in the cellular space if the parent of each cell is uniquely determined. Figure 6 shows an example of the construction of a spanning tree.

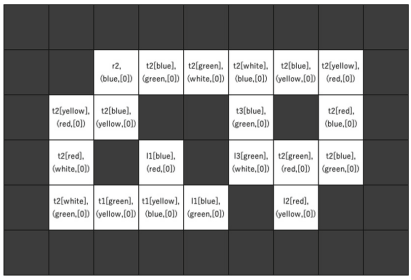

Fig. 6. An example of a spanning tree construction

4.2 Self-stability of Spanning Tree Construction

As in the case of the other two problems, we can construct a spanning tree with self-stable gellular automata. First, we construct the parent-child relation of cells

by distance-2 coloring. If configurations are distance-2 colored, there are no cells with a pair of neighboring cells of the same color, so the parent of each cell is uniquely determined by designating the color of the parent. This enables us to construct a tree in which no cells have two or more parents. Next, we propagate a wave from a root to leaves and from leaves to a root. This enables us to detect a pure loop as cells to which the wave does not propagate. We then add the cells in the loop to the tree.

5 Conclusions

In this paper, we showed how to construct gellular automata that solve three problems: solving a maze, distance-2 coloring, and spanning tree construction. As self-stable gellular automata can recover from malfunctions of states and transitions, materials that contain them are able to form structures like blood vessels or neural networks that can repair themselves following external damage or environmental changes.

By adding and changing some states and transition rules, we can also design gellular automata for solving other problems. For instance, gellular automata that solve the Hamiltonian circuit problem can be constructed by modifying those for a maze. The actual construction of these gellular automata remains for future studies. We also plan to improve the automata by decreasing the number of states and transition rules and reducing the number of steps required for them to converge to a target configuration.

Acknowledgements. We thank Akira Yagawa for valuable discussions and implementing the simulator. We also thank the anonymous reviewers for improving the paper. This work was partially supported by Grant-in-Aid for challenging Exploratory Research 17K19961.

References

1. Dolev, S.: Self-Stabilization. MIT Press, Cambridge (2000)
2. Fatés, N., Marcovici, I., Taati, S.: Cellular automata for the self-stabilisation of colourings and tilings. In: Filiot, E., Jungers, R., Potapov, I. (eds.) RP 2019. LNCS, vol. 11674, pp. 121–136. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-30806-3_10
3. Hagiya, M., et al.: On DNA-based gellular automata. In: Ibarra, O.H., Kari, L., Kopecki, S. (eds.) UCNC 2014. LNCS, vol. 8553, pp. 177–189. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-08123-6_15
4. Hosoya, T., Kawamata, I., Nomura, S.I.M., Murata, S.: Pattern formation on discrete gel matrix based on DNA computing. New Gener. Comput. **37**(1), 97–111 (2019). https://doi.org/10.1007/s00354-018-0047-1
5. Lee, C.Y.: An algorithm for path connections and its applications. IRE Trans. Electron. Comput. **EC–10**(3), 346–365 (1961)
6. Peper, F., Lee, J., Adachi, S., Isokawa, T.: Cellular nanocomputers: a focused review. Int. J. Nanotechnol. Mol. Comput. (IJNMC) **1**(1), 33–49 (2009)

7. Scalise, D., Schulman, R.: Emulating cellular automata in chemical reaction-diffusion networks. Nat. Comput. **15**(2), 197–214 (2016). https://doi.org/10.1007/s11047-015-9503-8
8. Tsompanas, M.-A.I., Sirakoulis, G.C., Adamatzky, A.: Cellular automata models simulating slime mould computing. In: Adamatzky, A. (ed.) Advances in Physarum Machines. ECC, vol. 21, pp. 563–594. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-26662-6_27
9. Yamashita, T., Hagiya, M.: Simulating population protocols by gellular automata. In: 57th Annual Conference of the Society of Instrument and Control Engineers of Japan (SICE), pp. 1579–1585. IEEE (2018)
10. Yamashita, T., Isokawa, T., Peper, F., Kawamata, I., Hagiya, M.: Turing-completeness of asynchronous non-camouflage cellular automata. In: Dennunzio, A., Formenti, E., Manzoni, L., Porreca, A.E. (eds.) AUTOMATA 2017. LNCS, vol. 10248, pp. 187–199. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-58631-1_15
11. Yamashita, T., Isokawa, T., Peper, F., Kawamata, I., Hagiya, M.: Turing-completeness of asynchronous non-camouflage cellular automata. Inf. Comput. **274**, 104539 (2020)
12. Yamashita, T., Yagawa, A., Hagiya, M.: Self-stabilizing gellular automata. In: McQuillan, I., Seki, S. (eds.) UCNC 2019. LNCS, vol. 11493, pp. 272–285. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-19311-9_21
13. Yin, P., Sahu, S., Turberfield, A.J., Reif, J.H.: Design of autonomous DNA cellular automata. In: Carbone, A., Pierce, N.A. (eds.) DNA 2005. LNCS, vol. 3892, pp. 399–416. Springer, Heidelberg (2006). https://doi.org/10.1007/11753681_32