Pergamon

*Mathl. Comput. Modelling* Vol. 23, No. 4, pp. 105–113, 1996

0895-7177/96 $15.00 + 0.00

S0895-7177(96)00006-4

# Computation of Shortest Path in Cellular Automata

A. I. ADAMATZKY
Biophysics Department, St. Petersburg State University
Universitetskaya emb. 7/9, 199034 St. Petersburg, Russia
ada@ada.usr.pu.ru

*(Received May 1994; accepted June 1994)*

**Abstract**—In this paper, we show how to find the shortest path between given nodes of a mesh with weighted edges. We use a cellular automaton which exhibits autowave patterns, where a wave of auto-excitation originates in source node, spreads around the mesh, and modifies states of the cells to make a stationary pattern isomorphed to the shortest path from source node to destination one. For different formulations of the shortest path problem, and various cellular automata (which solve it), we proved the bounds of complexity. For the sake of clarity, we present several examples of cellular automata computation of the shortest path. They are illustrated in detail.

**Keywords**—Shortest path, Cellular automata, Parallel algorithms, Complexity.

## 1. INTRODUCTION

A problem of searching for the shortest path—given a graph with weighted edges, we wish to find a path from one vertex to another through the minimal number of edges with minimal sum of their weights—has had a continued remarkable life from the time of the pioneering works of Dijkstra (see, e.g., [1]). In general, we assume a given graph to be oriented (there are two different edges from vertex $x$ to $y$ and from $y$ to $x$), and therefore it is useful to define 'source' from which we start our path and 'destination' to which we run and where we stop. The shortest path problem has three well-known alterations. They are a single source shortest path ($S^3P$), all pairs shortest path (APSP), and single source single destination shortest path ($S^3DSP$) problems. In the first of them we wish to compute the shortest paths from a given vertex of graph to all others; there are single source and numerous destinations. The second problem asks us to find shortest paths between all the pairs of the vertices. A substance of the third problem is obvious from its formulation.

The $S^3P$ algorithm in Dijkstra's version runs in $O(n)$ time when every edge of given graph has nonnegative weight. There exist effective $S^3P$ algorithms for the sparse graphs which work in $O(\min\{n^{1+(1/k)} + m, m \log n\})$ [2], $O(m(\log\log D))$ [3] and $O(n \log n + m)$ [4] upper times, where input graph has $n$ vertices, $m$ edges and $D$ is a maximal length of shortest path between two vertices; $k$ is a nonnegative integer. Using Floyd's algorithm [5] it is conceivable to solve APSP problem in $O(n)$ upper time; however Fredman [6], and Fredman and Tarjan [4] proved $O(n^3(\log\log n)^{1/3})$ and $O(n^2 \log n + mn)$ time complexity. The most efficient APSP algorithm has been designed by Moffat and Takaoka [7]; it runs in $O(n^2 \log n)$ upper time in the average case. Orda and Rom [8] designed an algorithm with $O(n^2)$ time complexity to find shortest path in mesh with constant weights.

One of the straightforward methods to compute a shortest path in parallel is a solution of the matrix equations (with elements in so-called path algebra) in systolic arrays [10–12]. Thus the computation of APSP on $O(n)$ processors needs in $O(\log n)$ time [13]. In [9], Frederickson proved a message complexity when the shortest path on a planar network was found by a distributed algorithm. It was shown that a shortest path tree can be found in a distributed network of $n$ vertices using $O(n^2)$ messages, but in a planar distributed network using $O(n^{5/3})$ messages. Venkatasubramanian, Krithivasan and Pangan [14] parallelized Dijkstra's serial algorithm for $S^3P$ in the so-called modified cellular graph automaton (the same as network of the two kinds of finite state automata). They proved that a modified cellular graph automaton of $O(n)$ processing elements finds $S^3P$ for the class of weighted graphs of bounded degree and $n$ area in $O(n)$ time.

In this paper, we will consider a mesh of $n$ vertices arranged on discrete lattice (multidimensional in general). Any vertex is connected with $k$'s closest neighbors by links of nonnegative weights; $k$ is much less than $n$. We wish to solve $S^3P$ and APSP problems in parallel, and to do so, we design cellular automaton (CA) with a variable radius of neighborhood as a valuable model of a massively parallel computing device.

The cellular automata are the most material, perceptible and practical models of such living processes (which moreover form a base for biocomputing) as neural networks, cellular and animal populations, molecular liquids and membranes, and excitable and reaction-diffusion media [15–17]. A main feature of the excitable media is that signals can be propagated undamped over a long distance, and furthermore, speed of wave propagation can be variable (in our case a wave runs from node $x$ to node $y$ in the time proportional to the weight of edge connecting $x$ with $y$). When we wish to find a shortest path between the two given vertices $x$ and $y$ of a mesh (detect, or label, edges of the path), we map the given graph onto the cellular array of cellular automaton (cells $x$ and $y$ will be corresponded to vertices $x$ and $y$), excite cell $x$, and a wave of excitation propagates in all directions around the lattice and modifies the states of cells. Computation is assumed finished when the wave reaches destination $y$.

The rest of the paper is organized as follows. Section 2 gives us the backgrounds for cellular automata and a problem we consider here. In Section 3, we design cellular automata algorithms to detect shortest path on the lattice, prove the complexity bounds for them, and show a few simple but illustrative examples. An informal discussion of the results of this work can be found in Section 4.

## 2. BACKGROUNDS

In the classical meaning, a cellular automaton can be defined as a $d$-dimensional lattice $L$ of $n$ cells, cell states set $Q$, neighborhood function $u$, and cell states transitions function $f$. A neighborhood function assigns a tuple of the closest cells $u(x)$ with every cell $x \in L$, $u : L \to L^k$, whereas a local transitions function $f$ maps a set of neighborhood states into set $Q$ of cell states, $f : Q^k \to Q$. Under these circumstances, any cell $x$ of lattice $L$ calculates its next state $x^{t+1}$ on the state of its neighborhood $u(x)^t$ at previous time step in the following way: $x^{t+1} = f(u(x)^t)$. A configuration (global state) of a cellular automaton is a mapping $c : L \to Q$. Evolution of cellular automaton with initial configuration $c$ is a series of the configurations transformed one from another: $c^0 \to c^1 \to c^2 \to \cdots \to c^t \to c^{t+1} \to \cdots$.

In numerous papers concerning cellular automata, the last ones are defined with many differences with the above classical definition. Maybe it is wrong, maybe not. No one has said where cellular automata finishes and other structures begin, though there is one a silent agreement: cellular automata are the local systems, which means that $k \prec n$. As to us, for the sake of clarity we supplied a common cellular automaton with a so-called pointer and a finite vector; both of them could be expressed through the expanded set of cell states, but it would lead us to unnecessary complications. Thus, in this paper we will use a cellular automaton (CA) $\mathfrak{T}$ consisting of a discrete $d$-dimensional lattice $L$ of $n$ cells; every cell $x \in L$ has states of a finite nonempty

set $Q$, neighborhood $u(x)$, a so-called pointer $p_x$ with states of a finite nonempty set $Y$, and vector $w_x$. A vector $w_x$ contains of the weights of input edges of vertex which was mapped to $x$. The sets $Q$ and $Y$ have the following elements: $Q = \{+, \#, \bullet, 0, 1, 2, \ldots, \nu\}$ and $Y = \{1, 2, \ldots, k, \lambda\}$, $|Q| = \nu + 5$, $|Y| = k + 1$. Every element $w_{xy}$ of vector $w_x$, $y \in u(x)$, can be in one of the states of set $\{\infty, 0, \ldots, \nu\}$. We will write $w_{xy} = \infty$ when vertices (nodes) $x$ and $y$ are not connected with one another by an edge oriented from $y$ to $x$. A neighborhood $u(x)$ of the cell $x$ is a tuple of $k$ cells of $L$ : $u(x) = (y_1, \ldots, y_k)$, where $0 \leq |x - y_j|_{L_\infty} \leq r$, $j = 1, 2, \ldots, k$, $r \in N$, $k = (2r + 1)^d$; $r$ is neighborhood radius and $k$ is neighborhood size (number of neighbors).

Let $G = \langle V, E \rangle$ be an oriented graph of $n$ vertices arranged on the $d$-dimensional discrete lattice, every vertex $v \in V$ of which is connected with no more than $k$ neighboring vertices $v_1, v_2, \ldots, v_k$ by input edges $v_1 v, v_2 v, \ldots, v_k v \in E$ of weights $w(v_1, v), w(v_2, v), \ldots, w(v_k, v) \in \{0, 1, \ldots, \nu, \infty\}$. If for some pair $v'v'' \in V$ we write $w(v', v'') = \infty$ then we keep in mind that there is no edge $v'\,v''$ in set $E$ (vertices $x$ and $y$ are not linked with any edge). To be successfully mapped onto cellular lattice, graph $G$ must have the following principle feature: $k \prec n$, i.e., indegree of any vertex is bounded by constant $k$ which is much less than $n$. Let $p = (v_0, \ldots, v_m)$ be a shortest path from vertex $v_0$ to vertex $v_m$ of graph $G$ and $l(p)$ be a length of $p$. Then we have that $l(p) = \min\{l(p') : p' = \{v_0, \ldots, v_m\}\}$.

## 3. RESULTS

Let us map graph $G$ onto CA $\mathfrak{T}$ in such a way that if vertex $v$ is embedded into cell $x$ then all the vertices $v_1, \ldots, v_k$ connected with $v$ by edges of $E$ will be mapped into cells $y_1, \ldots, y_k$, and form neighborhood $u(x)$. Moreover, the weights of input edges of $v$ are collected in the vector $w_x = (w_{xy_1}, \ldots, w_{xy_k}) = (w_{x1}, \ldots, w_{xk})$, where $w_{xj} = w(v_j, v)$, $v_j$ corresponds to cell $y_j \in u(x)$.

Let $x_s$ and $x_d$ be source and destination cells of CA $\mathfrak{T}$, corresponding with source and destination vertices of given graph $G$, respectively. First, we will discuss the single source, single destination shortest path ($S^3$DSP) problem. This can be continued to $S^3$P and APSP in a similar way. At the beginning of $S^3$DSP computation we assume $x_s^0 = +$. The computation will stop when cell $x$ passes in state $\#$ or every cell of $L$ is in state $\#$ or $\bullet$ (the second constraint will be used to stop computation when there exists no path from $x_s$ to $x_d$). CA $\mathfrak{T}$ resembles an autowave medium; therefore we can say that wave of states $+$ (wave of excitation) runs in all directions around the lattice from cell $x_s$ until it is in $x_d$, or passes all the cells of $L$. When the excitation wave is in cell $x$, the pointer $p_x$ changes its initial state $\lambda$ to some state of $\{1, 2, \ldots, k\}$ in order to point on index of such neighbor of $x$ that have 'sent' state $+$ to cell $x$. The path can be extracted easily from the final configuration of $\mathfrak{T}$ by back-tracking over the pointers from cell $x_d$ toward cell $x_s$.

Let us consider the regular orbit of cell $x$. Let at time $t$ cell $x$ be in state $x^t = \bullet$ (quiescent-like state), and some of its neighbors from $u(x)$ be in state $+$ at time $t$. We keep in mind that cells $x, y_1, \ldots, y_k$ correspond to vertices $v, v_1, \ldots, v_k$, and vector $w_x = (w_{x1}, \ldots, w_{xk})$ presents the weights of edges oriented from $v_1, \ldots, v_k$ to $v$. Cell $x$ finds such neighbor $y$ that $y^t = +$ and $w_{xy} = \min\{w_{xy'} : y' \in u(x) \wedge y'^t = + \wedge w_{xy'} \neq \infty\}$ ($w_{xy}$ corresponds to weight $w(v_y, v)$). After this, cell $x$ passes in state $x^t = w_{xy}$. Starting from state $w_{xy}$, cell $x$ jumps in state 0, decreasing its current state on unit step at every step of time $x^{t+1} = x^t - 1$. It will take place until $x^t = 0$ or there exists such neighbor $y$ that $y^t = +$ and $w_{xy} = \min\{w_{xy'} : y' \in u(x) \wedge y'^t = +\}$ and $w_{xy}^t < x^t$. Any cell of $\mathfrak{T}$ passes from state 0 to state $+$ unconditionally, as well as from state $+$ to state $\#$.

At the beginning of computations, pointers of all cells are in state $\lambda$; however if cell $x$ changes its state to $w_{xy}$, then for some neighbor $y_j$ pointer $p_x$ saves index $j$ of this neighbor $p_x^t = j$. Notice that in the decreasing $w_{xy}$ down to 0, pointer $p_x$ can be modified when condition "$\exists y \in u(x) : y^t = + \wedge w_{xy} = \min\{w_{xy'} : y' \in u(x) \wedge y'^t = +\}$ and $w_{xy}^t < x^t$" is held. The state of pointer is fixed up and becomes constant after cell $x$ departs from state 0. To compute $S^3$P we

must accept the following condition to complete evolution of $\mathfrak{T}$ at time $t$ : $\forall x \in Lx^t \in \{\#, \bullet\}$. Let $x$ be in state $\#$ at time $t$. Then one can extract the shortest path from $x_s$ toward $x_d$ by back-tracking over the pointers. The main procedure will be as follows: label cell $x_d$ and cell $y$ such that $p^t_{x_d} = y$, label $y'$ such that $p^t_y = y'$, and so on until cell $x_s$ is labeled.

Now, it is time to discuss in detail how cells work. Every cell $x$ of $\mathfrak{T}$ changes its state $x^t$ at time $t$ to the next state $x^{t+1}$ depending on the state $u(x)^t$ of neighborhood $u(x)$ by the rule

$$x^{t+1} = \begin{cases} \#, & x^t \in \{\#, +\}, \\ +, & x^t = 0, \\ \bullet, & x^t = \bullet \wedge (\neg \exists y \in u(x) : y^t = +), \\ w_{xy}, & (\exists y \in u(x) : y^t = +) \wedge (((x^t = \bullet) \vee (x^t > 0)) \wedge w_{xy} \\ & = \min\{w_{xy'} : y' \in u(x) \wedge y'^t = +\} \wedge w_{xy} < x^t \wedge w_{xy} \neq \infty), \\ x^t - 1, & (x^t > 0) \wedge (\neg \exists y \in u(x) : y^t = + \wedge w_{xy} = \min\{w_{xy'} : y' \in u(x) \wedge y'^t = +\} \wedge w_{xy} < x^t). \end{cases}$$

In the same time, pointer $p_x$ of cell $x$ passes from state $p^t_x$ to $p^{t+1}_x$ by the rule

$$p^{t+1}_x = \begin{cases} y, & (\exists y \in u(x) : w_{xy} = \min\{w_{xy} : y' \in u(x) \wedge y'^t = +\} \wedge y^t = +) \wedge ((0 < w_{xy} < x^t) \vee (x^t = \bullet)), \\ p^t_x, & \text{otherwise.} \end{cases}$$

At the beginning of computations, we assume $x^0_s := +$, $\forall x \in L\ x \neq x_s : x^0 := \bullet$ and $p^t_x := \lambda$. It must be noticed from the above rules that state $\#$ is absorbed and transitions $+ \to \#$, $\# \to \#$ and $0 \to +$ are unconditional.

Let us consider a small example. We wish to solve an $S^3DSP$ problem in a two-dimensional grid, edges of which have weights $\infty$ or 0. To compute the path we designed a two-dimensional CA, any cell of which has cruciform neighborhood $u(x_{ij}) = (x_{i-1j}, x_{i+1j}, x_{ij-1}, x_{ij+1})$ and 9 states. Figure 1 shows us the given graph. We wish to find the shortest path from the left upper vertex to the right lower vertex. In this concrete case, we have $Q = \{\bullet, +, \#\}$, $Y = \{\mathrm{N,W,S,E}, \lambda\}$, $w_x = (w_{x\mathrm{N}}, w_{x\mathrm{W}}, w_{x\mathrm{E}}, w_{x\mathrm{S}})$. Symbols N, W, E and S are the indices for the northern, western, eastern and southern neighbors of a cell. In Figure 2, the dynamic of CA evolution is shown. In this figure we expose the current state of a cell when it is in one of the states $\bullet$ or $\#$, and the state of the pointer when the cell has been in state $\#$ at least two times. The back-traced (a) and extracted (b) paths can be found in Figure 3.

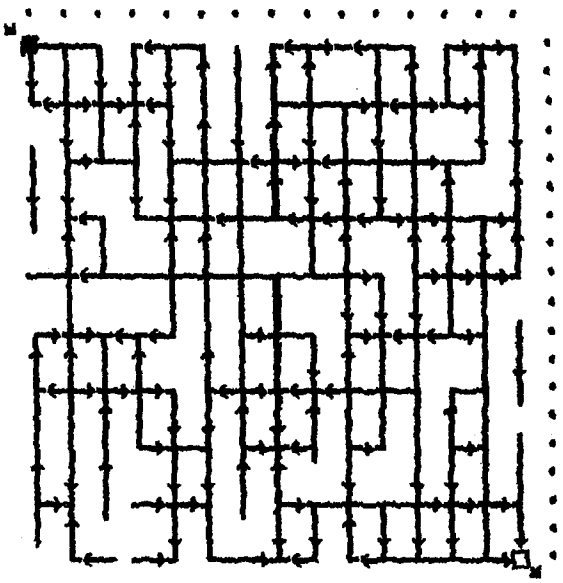


Figure 1. An oriented planar graph $G$: arrows indicate orientation of edges; an intersection of two or more straight segments corresponds to a vertex of $G$; black and empty boxes are source vertex and destination vertex, respectively.

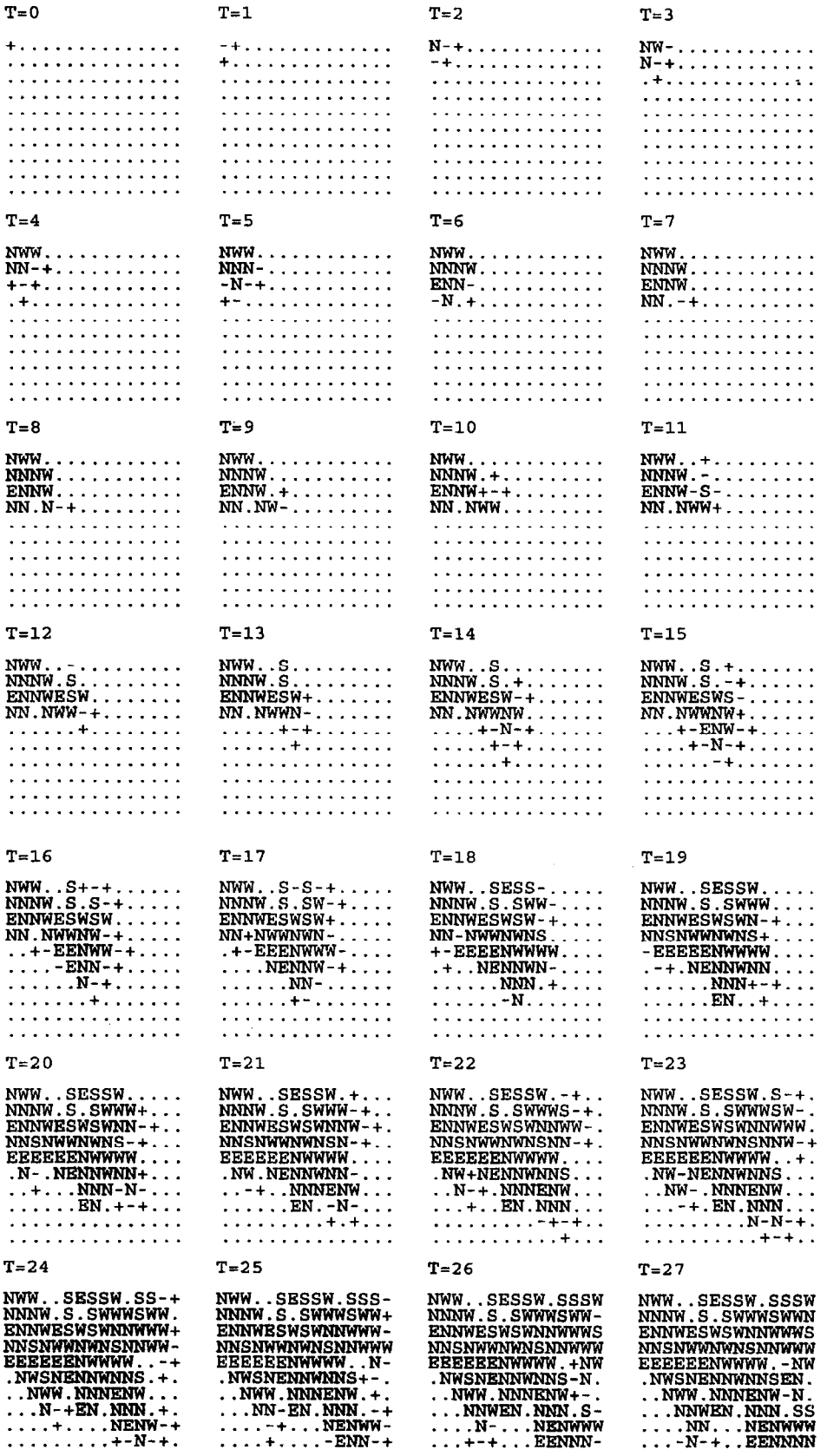
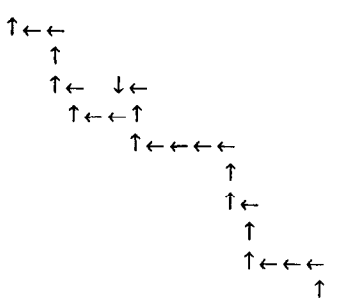
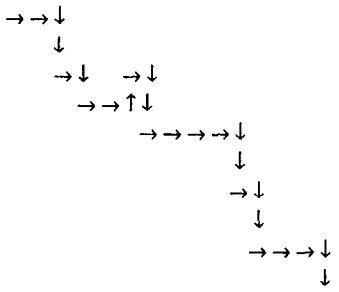



Figure 2. An evolution of cellular automaton $\mathfrak{T}$ in the computation of $S^3DSP$. Symbol $-$ means a state #. A state of pointer for cell $x$ is shown in the picture only if $x$ was in state # at least two times.



(a) Extraction of the inversed path by back-tracking over pointers.



(b) The shortest path.

Figure 3. The results of computation of $S^3DSP$.

LEMMA 1. *Let $G$ be a two-dimensional integer grid of $n$ nodes, and some edges of it be cut off, and $\mathfrak{T}$ be a cellular automaton of $n$ cells, each of which has four neighbors and nine states. Then $\mathfrak{T}$ can find a single source single destination shortest path ($S^3DSP$) on $G$ in $O(n)$ upper time, and single source (all destinations) shortest path ($S^3P$) in $O(n)$ upper time and can extract path in $O(n^2)$ upper time.*

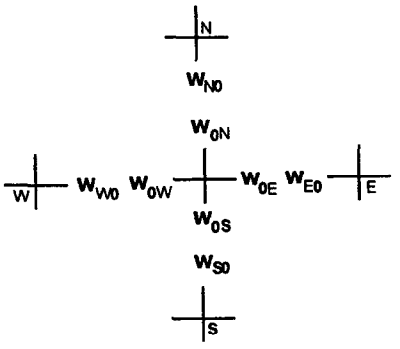
PROOF. Any edge of $G$ can be or not be (i.e., it may have 0 or $\infty$ weight). Therefore, the cell states set will be $Q = \{+, \bullet, \#\}$. Every vertex of a rectangular grid is connected with no more than 4 neighboring vertices (northern, western, eastern and southern), so $Y = \{\mathrm{N,W,S,E}, \lambda\}$. The state of cell $x$ can be presented by a tuple $(x^t, p_x^t)$ of proper state of cell and state of pointer, respectively. If $x^t = \bullet$, then $p_x^t = \lambda$ obligatory; however $p_x^t \in \{\mathrm{W,S,N,E}\}$, when $x^t \in \{+, \#\}$. Thus, any cell $x$ of $\mathfrak{T}$ has nine states. The longest path in $G$ consists of $n-1$ nodes; therefore, it can be computed in $O(n)$ time. A simple application of the above reasons to all the vertices of $G$ gives us $O(n^2)$ time for APSP.

COROLLARY 1. *Let $\mathfrak{T}$ be a two-dimensional CA of $n$ cells, each of which has six neighbors and five states. Then, in the conditions of Lemma 1, a $S^3DSP$ problem can be solved in $\mathfrak{T}$ in $O(n)$ time.*

LEMMA 2. *Let $G$ be a two-dimensional rectangular grid of $n$ nodes, and the edges of $G$ have weights of the set $\{0, \ldots, \nu, \infty\}$. Let $\mathfrak{T}$ be a two-dimensional cellular automaton of $n$ cells, each of which has four neighbors and $6+5n$ states. Then $S^3DSP$ problem can be solved in $\mathfrak{T}$ in $O(\nu n)$ upper time.*

PROOF. We designed such a CA (which exhibits autowave patterns) that a wave of excitation starting with source cell (node) spans edges and nodes of the shortest path in the shorter time, and reaches a destination cell (node) earlier than other waves (waves running on the paths longer than shortest); i.e., the nodes belonging to the shortest path passes to state # in the first place. To implement it we assumed that a delay of the switching cell $x$ from state $\bullet$ to state $+$ is proportional to weight $w_{xy}$ when $y$ 'transmits' state $+$ to cell $x$. Therefore, cell $x$ passes at first to state $w_{xy}$, decreases $w_{xy}$ on unit ($w_{xy} = w_{xy} - 1$) every time-step until it is equal 0 or there exists such a neighbor $y''$ that $y''^t = +$ and $w_{xy''} = \min\{w_{xy'} : y' \in u(x) \wedge y'^t = +\}$ and $w_{xy''}^t < x^t$. Cell $x$ modifies the state of the pointer on $w_{xy''}$ in the last case. Let us consider the following instance. Let cell $x$ have neighborhood $u(x) = (y_\mathrm{N}, y_\mathrm{W}, y_\mathrm{S}, y_\mathrm{E})$ in the state $u(x)^t = (\bullet, +, +, +)$, and $x^t = +$. Moreover, we have $w_x = (w_{x\mathrm{N}}, w_{x\mathrm{E}}, w_{x\mathrm{S}}, w_{x\mathrm{W}}) = (4, 9, 3, 1)$. Cell $x$ finds such a neighbor $y$ in state $+$ that $w_{xy}$ is minimal over elements of $w_x$ . In our case, the cell chooses $y_\mathrm{S}$ and $x^{t+1} = 3$, $x^{t+2} = 2$, $x^{t+3} = 1$, $x^{t+4} = 0$, $x^{t+5} = +$, $x^{t+6} = \#$, $p_x^{t+1} = \mathrm{S}$, $p_x^{t+2} = \mathrm{S}, \ldots$. By the definition, weights of edges in graph $G$ are bounded by integer $v$; therefore, if the longest path consists of $n$ vertices and the delay of state transition is $\nu$ for any cell, then $O(\nu n)$ is upper time for computation of $S^3P$. The number of cell states can be proved by analogy with Lemma 1.

Figure 4. An oriented planar graph $G$ every vertex of which has indegree 4. The edges are weighted with elements of set $\{0, 1, 2, \infty\}$, where $\infty$ means that there is no edge. The format of data is as follows:




where $w_{ab}$ is a weight of edge oriented from vertex $a$ to vertex $b$, and 0 designates a central node (cell). The left upper vertex of G is a source vertex but right lower is a destination one.

```
+12+0∞+∞∞+∞0+∞0+∞1+∞∞+∞∞+21+
∞  2  1  ∞  ∞  2  1  0  ∞  1
∞  2  2  2  ∞  ∞  1  1  1  ∞
+∞∞+20+2∞+∞2+∞1+0∞+10+∞∞+∞∞+
2  ∞  1  0  1  0  0  0  1  ∞
1  1  1  1  ∞  2  1  ∞  0  ∞
+∞1+∞∞+∞1+22+∞2+00+∞∞+∞1+1∞+
2  1  0  ∞  2  0  0  ∞  ∞  ∞
∞  ∞  ∞  2  ∞  1  1  ∞  1  ∞
+∞2+1∞+2∞+02+∞∞+21+22+∞2+∞∞+
0  ∞  0  ∞  ∞  ∞  2  1  ∞  ∞
∞  ∞  ∞  1  1  2  0  ∞  ∞  1
+∞0+2∞+20+∞0+00+22+02+12+11+
∞  2  ∞  ∞  ∞  1  0  ∞  ∞  0
1  ∞  ∞  ∞  0  ∞  1  ∞  ∞  2
+∞0+00+22+∞2+∞0+∞∞+1∞+∞∞+1∞+
∞  ∞  ∞  ∞  2  ∞  1  ∞  2  ∞
1  ∞  0  0  2  ∞  1  0  ∞  ∞
+∞∞+∞2+∞0+1∞+0∞+∞∞+∞∞+∞2+∞∞+
∞  ∞  ∞  ∞  0  ∞  0  0  ∞  ∞
∞  ∞  ∞  0  2  ∞  ∞  ∞  1  ∞
+10+∞∞+∞1+∞∞+∞0+∞0+∞0+∞∞+∞∞+
2  0  ∞  ∞  0  2  0  2  ∞  ∞
∞  ∞  2  ∞  ∞  ∞  0  2  2  2
+21+11+∞∞+10+1∞+1∞+00+21+00+
∞  ∞  ∞  0  0  ∞  0  ∞  ∞  ∞
∞  2  ∞  ∞  ∞  ∞  0  ∞  2  ∞
+2∞+∞∞+0∞+1∞+∞0+22+21+20+00+
```

A pictorial example can be found in Figures 4–6.

THEOREM 1. *Let $G$ be an oriented graph of $n$ vertices arranged on the discrete lattice, and every vertex have indegree bounded by integer $k$. Then there exists a cellular automaton $\mathfrak{T}$ of $n$ cells, which has $k$ neighbors and $O(\nu k)$ states; $\mathfrak{T}$ can find $S^3DSP$ on $G$ in $O((\nu + k)n)$ upper time. An $S^3P$ can be extracted from the final configuration of $\mathfrak{T}$ in $O(n^2)$ upper time.*

To obtain the results of the theorem, we add $k$ to $n$ in the time complexity bound in Lemma 2 because cell $x$ finds a neighbor corresponding to the minimal element of $w_x$ in $O(k)$ time.

## 4. CONCLUSION

In this work, we designed cellular automata which are capable of detecting the nodes of a given mesh belonging to the shortest path from one node of the mesh to another one. Here we draw a visible analogy between running patterns, which appear in the evolution/computation, and auto-waves of excitation which are found in the biological media [15–17] and morphogenetical processes. An excitation originates in the source cell, spreads around the cells of the lattice, and the velocity of the conducting of excitation from one cell to another is inversely proportional to weight of edge which links the corresponding nodes in the given mesh. We supply every cell of the CA with a rotating arrow, or pointer, which (after cells have been excited) points to the neighbor which excited this cell. A $S^3DSP$ is simple enough: after excitation comes down, or the CA falls into fixed point of its evolution, we run backward under the leading of the pointers and mark the cells we have passed. This might be done in $O(n^2)$ time in the case of $S^3P$ problem.

In the first part of the paper we discussed ordinary rectangular lattice, some edges of which were cut out. In the $S^3DSP$ case we obtained $O(n)$ time complexity bound in just the time a wave can run from one boundary of a lattice to another. Moreover, under the same conditions an $S^3DSP$ problem can be solved in a CA with a cell neighborhood of size four and nine elements of cell states set, as well as in a CA with cell neighborhood of size six cells and five states of a cell. This fact is in good agreement with the results concerning simulation of one CA in another one.

How complex a CA must be to find the shortest path on the mesh with weighted edges? Each node of it has input degree less or equal to $k$, and every edge has weight from 0 to $\nu$, or $\infty$ when it

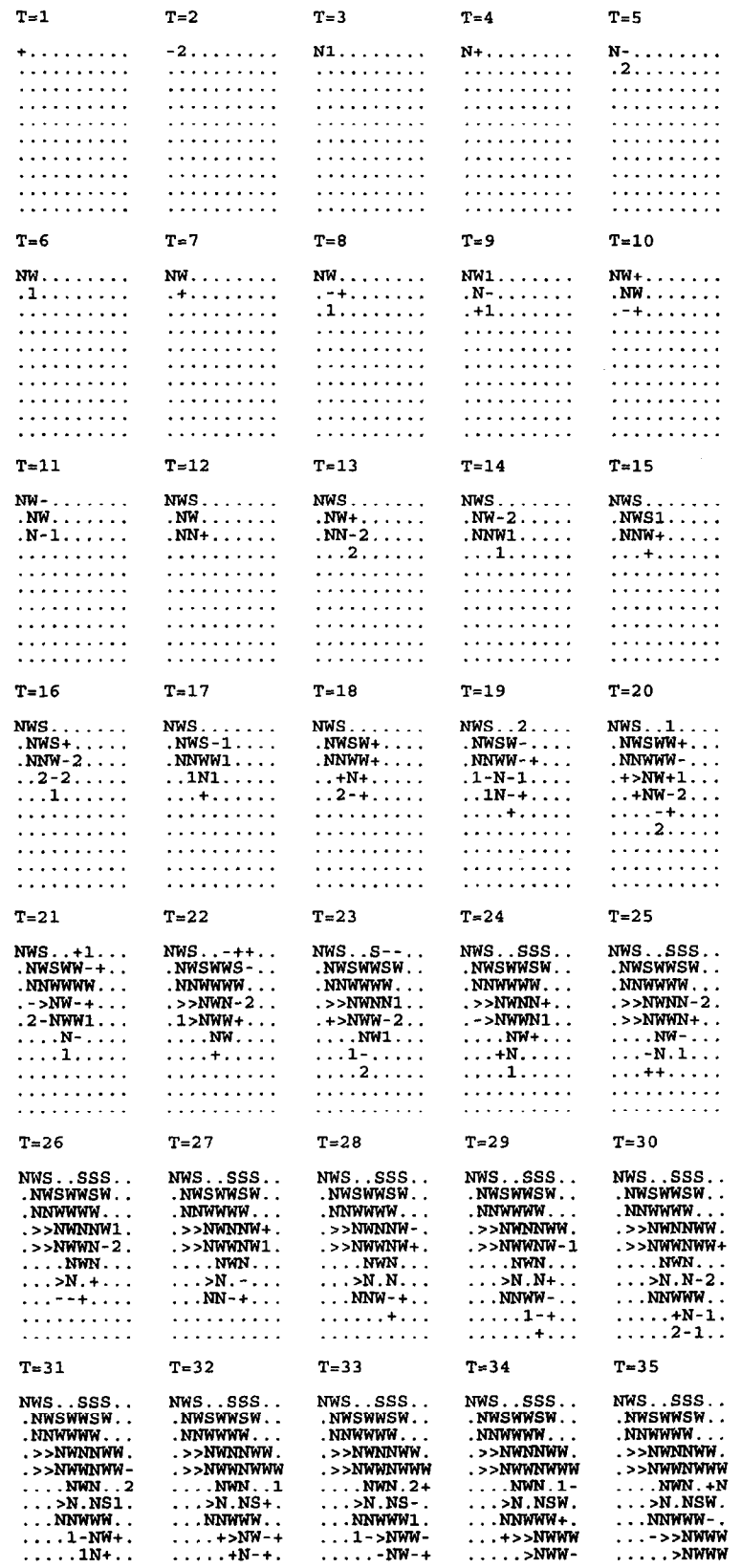



Figure 5. An evolution of cellular automaton $\mathfrak{T}$ which computes $S^3DSP$ on a graph with weighted edges. Symbol $-$ means a state #. A state of the pointer for cell $x$ is shown in picture only if $x$ was in state # at least two times.

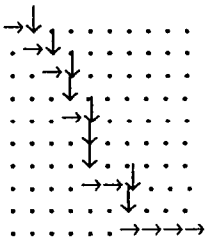


Figure 6. The shortest path.

is absent. Evidently, a size of neighborhood is equal to $k$, and any cell has $O(\nu k)$ states (because of the necessity to simulate delay of transmission of excitation from a cell to its neighbors). This CA will work in $O(n(\nu + k))$ time.

What is the reason that the complexity bounds are so good? It takes place because instead of an arbitrary graph, we used a very restricted form of this—rectangular lattice—to make the structure of a problem most resembling architecture of computing device which solves this problem.

Can we apply our results in a practice? Undoubtedly. The CA algorithms are (in the meantime) biological and, in particular, neural algorithms by nature. For this reason, the implementation of these algorithms in massively parallel processors or neurocomputers is an event of the near future.

## REFERENCES


1. E.N. Dijkstra, A note on two problems in connection with graphs, *Numerische Mathematik* **1**, 269–271 (1959).
2. D.B. Johnson, Efficient algorithms for shortest paths in sparce networks, *J. ACM* **24**, 1–13 (1977).
3. R.G. Karlsson and P.V. Poblete, An $O(m \log \log D)$ algorithm for shortest path, *Discrete Apll. Math.* **6**, 91–93 (1983).
4. M.L. Fredman and R.E. Tarjan, Fibonacci heaps and their uses in improved network optimization algorithms, presented at the $25^{th}$ *Ann. Symp. Found. Comput. Sci.*, Singer Island, FL, October 24–26, 1984, pp. 338–346, Silver Springer, NY, (1984).
5. R.W. Floyd, Algorithm 97: Shortest path, *Comm. ACM* **5**, 345 (1962).
6. M.L. Fredman, New bounds on the complexity of the shortest path problem, *SIAM J. Comput.* **6**, 83–89 (1976).
7. A.M. Moffat and T. Takaoka, An all pairs shortest path algorithm with expected time $O(n \log n)$, *SIAM J. Comput.* **6**, 1023–1031 (1987).
8. A. Orda and R. Rom, Shortest path and minimum delay algorithms in networks with time-dependent edge-length, *J. ACM* **37**, 607–625 (1990).
9. G.N. Frederickson, A distributed shortest path algorithm for a planar network, *Information and Computation* **86**, 140–159 (1990).
10. J.-M. Delosme, A parallel algorithm for the algebraic path problem, In *Parallel and Distributed Algorithms*, (Edited by M. Cosnard *et al.*), Elsevier Science, North-Holland, (1989).
11. H.-W. Lang, Transitive closure on an instruction systolic array, Bericht Nr. 8718, Christian-Albrechts-Universität, Kiel, (1987).
12. C.-H. Huang and C. Lengauer, An incremental mechanical development of systolic solutions to the algebraic path problem, *Acta Informatica* **27**, 97–124 (1989).
13. J. Quinn and N. Deo, Parallel graph algorithm, *ACM Computing Surveys* **16**, 319–348 (1984).
14. S. Venkatasubramanian, K. Krithivasan and C.P. Rangan, Algorithms for weighted graph automaton, *Theoret. Inform. Apll.* **23**, 251–279 (1989).
15. M. Gerhardt, H. Schuster and J.J. Tyson, A cellular automata model of excitable media. II Curvature, dispersion, rotating waves and meandering waves, *Physica D* **46**, 392–415 (1990).
16. M. Gerhardt, H. Schuster and J.J. Tyson, A cellular automata model of excitable media. III Fitting the Belousov-Zhabotinskii rection, *Physica D* **46**, 416–426 (1990).
17. H. Hartman and P. Tamayo, Reversible cellular automata and chemical turbulence, *Physica D* **45**, 293–306 (1990).