ResearchGate

See discussions, stats, and author profiles for this publication at: https://www.researchgate.net/publication/325369178

Computational Modeling of Proteins based on Cellular Automata: A Method of HP Folding Approximation

Article *in* The Protein Journal · May 2018

DOI: 10.1007/s10930-018-9771-0

CITATIONS
0

READS
77

3 authors, including:

Azzam Sleit
University of Jordan
129 PUBLICATIONS **469** CITATIONS

SEE PROFILE

Some of the authors of this publication are also working on these related projects:

Improving Friends Matching in Social Networks Using Graph Coloring View project

Data Visualization View project

All content following this page was uploaded by Azzam Sleit on 29 May 2018.

The user has requested enhancement of the downloaded file.

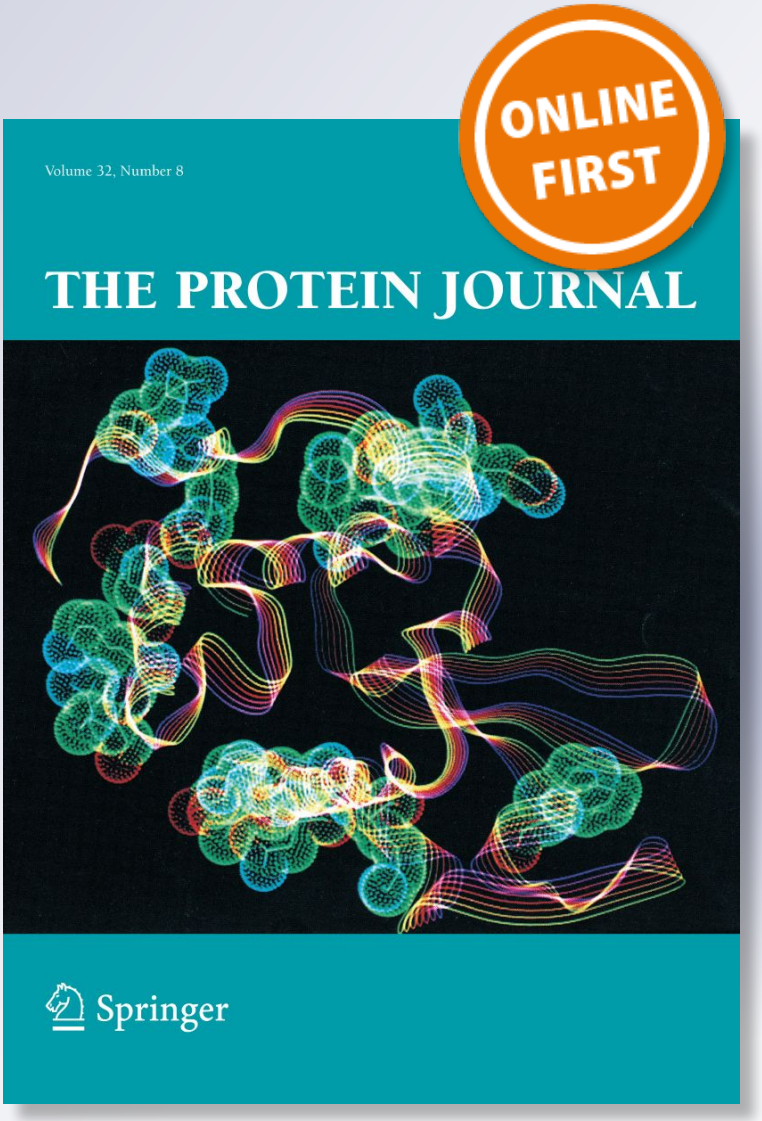
Computational Modeling of Proteins based on Cellular Automata: A Method of HP Folding Approximation

Alia Madain, Abdel Latif Abu Dalhoum & Azzam Sleit

The Protein Journal

ISSN 1572-3887

Protein J
DOI 10.1007/s10930-018-9771-0



Springer

Your article is protected by copyright and all rights are held exclusively by Springer Science+Business Media, LLC, part of Springer Nature. This e-offprint is for personal use only and shall not be self-archived in electronic repositories. If you wish to self-archive your article, please use the accepted manuscript version for posting on your own website. You may further deposit the accepted manuscript version in any repository, provided it is only made publicly available 12 months after official publication or later and provided acknowledgement is given to the original source of publication and a link is inserted to the published article on Springer's website. The link must be accompanied by the following text: "The final publication is available at link.springer.com".

Author's personal copy

The Protein Journal
https://doi.org/10.1007/s10930-018-9771-0

Computational Modeling of Proteins based on Cellular Automata: A Method of HP Folding Approximation

Alia Madain[1] · **Abdel Latif Abu Dalhoum**[1] · **Azzam Sleit**[1]

© Springer Science+Business Media, LLC, part of Springer Nature 2018

Abstract
The design of a protein folding approximation algorithm is not straightforward even when a simplified model is used. The folding problem is a combinatorial problem, where approximation and heuristic algorithms are usually used to find near optimal folds of proteins primary structures. Approximation algorithms provide guarantees on the distance to the optimal solution. The folding approximation approach proposed here depends on two-dimensional cellular automata to fold proteins presented in a well-studied simplified model called the hydrophobic–hydrophilic model. Cellular automata are discrete computational models that rely on local rules to produce some overall global behavior. One-third and one-fourth approximation algorithms choose a subset of the hydrophobic amino acids to form H–H contacts. Those algorithms start with finding a point to fold the protein sequence into two sides where one side ignores H's at even positions and the other side ignores H's at odd positions. In addition, blocks or groups of amino acids fold the same way according to a predefined normal form. We intend to improve approximation algorithms by considering all hydrophobic amino acids and folding based on the local neighborhood instead of using normal forms. The CA does not assume a fixed folding point. The proposed approach guarantees one half approximation minus the H–H endpoints. This lower bound guaranteed applies to short sequences only. This is proved as the core and the folds of the protein will have two identical sides for all short sequences.

Keywords Hydrophobic–hydrophilic model · Folding approximation · Protein folding · Cellular automata · Computational modeling of proteins

1 Introduction

Protein folding approximation algorithms in the hydrophobic–hydrophilic model (HP model) are those algorithms that guarantee a minimum ratio of the number of H–H contacts achieved to the number of optimal H–H contacts. The one-fourth and one-third approximation algorithms are similar in multiple ways. Both algorithms depend mainly on whether the hydrophobic amino acid is placed in an odd or in an even position. Because of the properties of the two-dimensional grid, it is possible to achieve H–H contacts only when a hydrophobic amino acid is placed in an even position, adjacent to another hydrophobic amino acid, which is placed in an odd position.

In both algorithms, a point is assumed to be the predetermined folding point, which splits the string into two. One of the sides of the string considers hydrophobic amino acids placed in odd positions whereas the other side considers hydrophobic amino acids placed in even positions. This arrangement helps in calculating the approximation ratio. The actual folding is done by using a predefined template, where similar blocks are always folded the same way.

This work does not recognize the exact positions of amino acids and does not choose a folding point using a global view. Simply put, each amino acid reads the amino acids in its neighborhood. The neighborhood is limited to the immediate surroundings of the amino acid. This method has the advantage of being flexible, because no folding template is required, and each amino acid could decide its direction of movement based on available information.

The folding approximation approach proposed depends on the concept of cellular automata (CA). CAs are discrete computational models that are studied in computability theory and complex systems. CAs were first proposed as models of

✉ Alia Madain
alia@madaeen.com

[1] Department of Computer Science, King Abdulla II School for Information Technology, The University of Jordan, Amman 11942, Jordan

Published online: 25 May 2018

Author's personal copy

self-reproducing organisms by von Neumann in the forties [1]. CAs are discrete in terms of time and space. A CA contains a set of cells distributed in any number of dimensions. One-dimensional and two-dimensional CAs could be thought of as infinite arrays of cells and infinite grids of cells, respectively.

Cells could be in one state from a set of predefined states. The set of states is finite and could be of any type with any meaning such as integers, binary numbers, or even words and strings. CA goes through iterations which are also called generations. The CA begins with an initial state for each cell, and each cell moves from one state to the other or stays in the same state, depending on a transitional function.

The concept of CA is that a set of cells could move from one state to another based on simple local rules, causing an interesting global behavior. The abilities of this model could be summarized by the possible global behaviors in which complex and chaotic behavior could result from these rules that are deterministic, local, and depend only on a certain neighborhood as their input. Hence, the main reason for using CA is its simplicity and abilities.

The design of local rules is not straightforward. Setting this essential limit of not having a global view could require multiple tasks to be done and undone; sometimes, an action needs to be taken based on current local information, it may not need to be taken based on new information. The minimum number of H–H contacts is guaranteed based on the fact that the two sides of the protein core will be identical after folding which makes the number of ones participating in H–H contacts one half except any existing hydrohphobic amino acids at the endpoint. The approach applies for very short strings only, which were proven to have a core of identical sides after folding.

This paper is organized as follows: Sect. 2 summarizes related background; Sect. 3 gives the related work regarding the use of CA in building protein models in addition to the 1/3 and 1/4 approximation algorithms; Sect. 4 presents the proposed folding algorithm; Sects. 5 and 6 discuss the proposed approach properties and the folding results, and finally, Sect. 7 concludes the work done and gives direction to future work.

2 Background

2.1 Hydrophobic–Hydrophilic Model

The HP model for proteins was proposed and further analyzed by Dill [2–4], who made the folding problem simpler for analysis. The model focuses on the hydrophobicity of amino acids, and excludes complicated details.

The model classifies any amino acid as either hydrophobic or polar (hydrophilic). The lowest energy is measured by the number of H–H contacts; the more the contacts, the lower the energy measured would be. An H–H contact could be defined as a contact between two hydrophobic amino acids that are not connected in the proteins primary structure but occur in two adjacent cells in the two-dimensional grid, where adjacent cells are two cells with a common boundary.

Connected neighbors are connected to each other in the original chain of amino acids, whereas topological neighbors are placed in neighboring positions on the two-dimensional grid, but are not connected in the original protein chain. H–H contacts, by definition, are topological neighbors. Folding could be done in a two-dimensional or three-dimensional lattice. The folding process is a self-avoiding walk, where the protein does not cross itself.

The following notations have been put forward to enhance clarity. Assume that the protein primary structure is a binary string, where 1 represents hydrophobic amino acids and 0 represents hydrophilic (polar) amino acids. We would refer to the binary number 1 at odd positions as odd-1's and that at even positions as even-1's, as in a previous study [5].

We also refer to the 1's at the endpoints as endpoint-1's and H–H contacts as 1–1 contacts. Amino acids of the protein's primary structure or bits of binary strings are also referred to as the elements of the string. Finally, zeros at even positions are designated as even-0's and zeros at odd positions are odd-0's. The notation used in this document is summarized in Table 1.

2.2 Upper Bound of 1–1 Contacts

The use of a regular two-dimensional square lattice implies two important features that affect the resulting folded protein: first, each hydrophobic amino acid in the chain could have a maximum of four contacts; some of those places are occupied by connected neighbors. The number of places occupied by connected neighbors depends on the position of the amino acid. In case of internal amino acids, two of the four possible contacts are occupied by connected amino acids from the original chain. However, in amino acids at the end points of the protein binary string, one of the four possible contacts is occupied by a connected amino acid.

Second, for two amino acids to be topological neighbors, they must be located in positions with different parities, because two amino acids at even (odd) placements cannot be topological neighbors. The upper bound could be calculated by simply counting the number of odd-1's and the number of even-1's, and choosing the minimum (M). The maximum number of 1–1 contacts is:

$$Contacts \leq 2 \times M + 2 \quad (1)$$

Each element in M could have two 1–1 contacts, except those at the end points, which result in the formula $2 \times M + 2$. In a previous study [5], the author showed that the upper bound could not be used to achieve an approximation algorithm with a ratio better than 1/2.

Author's personal copy

Table 1 Notation used in describing HP proteins as binary strings

HP model notation	Binary notation
Hydrophobic amino acid (H)	1
Polar amino acid (P)	0
H–H contacts	1–1 contacts
Polar amino acids at odd positions in the 2D lattice	Odd-0's
Polar amino acids at even positions in the 2D lattice	Even-0's
Hydrophobic amino acids at odd positions in the 2D lattice	Odd-1's
Hydrophobic amino acids at even positions in the 2D lattice	Even-1's
Hydrophobic amino acids at the endpoints	Endpoint-1's
A set of amino acids	Bits
Protein primary structure, sequence, chain	Binary string

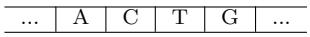


Fig. 1 Mapping DNA to CA

3 Related Work

3.1 Applications of Cellular Automata

Cellular automata have captured the attention of several generations of researchers, leading to an extensive body of work. CAs are being studied from many widely different angles, and the relationships of these structures to existing problems are being constantly discovered. For example, they are used in modeling heterogeneous traffic flow [6, 7], in pattern recognition [8], and in multimedia security applications [9, 10].

A starting point to explore the possibilities of modeling the evolution and the role of DNA sequences in terms of CA has been provided in a previous study [11]. A study of the overall behavior of gene networks in terms of CA has also been conducted [12].

One method to map DNA to one-dimensional CA is to map the sugar-phosphate backbone of DNA to the lattice in CA, and to map the four bases Adenine (A), Cytosine (C), Thymine (T), and Guanine (G) to the possible state of CA cells, where every cell could be in one of the four states in a certain generation, as shown in Fig. 1. This mapping strategy has been used in previous studies as in [13–15].

Variants of cellular automata were used in building models of different processes in the central dogma of molecular biology. Fuzzy CA was used as a pattern classifier to identify the coding region of a DNA sequence [16]. Brownian CA was used to model duplicating DNA, transcription of mRNA from DNA, and splicing to extract exon regions [17]. Probabilistic CA was used to model DNA sequence evolution [18].

Combinations of CA and evolutionary algorithms were used to tackle the protein folding problem. Genetic algorithms combined with CA were used to predict the secondary structure of proteins [19]. Artificial neural networks were used in combination with CA to model the temporal folding process of the protein primary sequence, which yielded the final folded protein conformation [20–22].

In Ref. [23] a 3D theoretical model is proposed. The model is based on 3D CA and heuristic rules of chemistry and thermodynamics. In Ref. [24] HP model folding based on cellular automata and simple rules was proposed, where the rules move the polar amino acids in a way that surrounds the hydrophobic amino acids from both directions, which allows for horizontal 1–1 contacts only. In Ref. [25] CA is used to represent protein translation and the work suggests methods of enhancing protein folding approximation.

Elementary Rule 84 was used for predicting protein subcellular location [26], classifying proteins based on their structure class [27–29], predicting G-protein-coupled receptor functional classes [30], and predicting transmembrane regions in proteins [31].

A flowchart of the classification process, along with a discussion of the potentials and challenges of using cellular automata and its variants as models of proteins has been provided previously [32]. We use a classic cellular automata to fold proteins presented in the 2D HP model. The CA is not combined with any evolutionary algorithm. The proposed approach guarantees a minimum of one half the 1–1 contacts minus the endpoint-1's for all short sequences.

3.2 Protein Folding Approximation

In general, protein folding is complex even in a two-dimensional lattice with the simplest alphabet. The problem with unbounded alphabet size is NP-complete [33] and the problem with two-dimensional and three-dimensional protein folding in the HP model was shown to be NP-complete in previous studies [34, 35].

NP-complete problems are often addressed by using heuristic methods and approximation algorithms. The objective of a heuristic is to produce a solution in a reasonable time frame that is good enough for solving the problem at hand.

Author's personal copy

Approximation algorithms are efficient algorithms that find approximate solutions to NP-hard optimization problems with provable guarantees about the distance of the returned solution to the optimal one.

Many evolutionary algorithms have been proposed to tackle the HP folding problem [36]. For example, genetic algorithms in combination with different mechanisms [37, 38], ant colony optimization [39], differential evolution [40], and thermal cycling [41]. Approximation algorithms, on the other hand, provide a guaranteed ratio of the number of achieved 1–1 contacts to the upper bound.

The first approximation algorithm by Hart and Istrail [42] guaranteed the achievement of 1/4 of the 1–1 contacts in the two-dimensional grid. The algorithm does not assume any constraints on the input; hence, any string could be folded and a minimum of 1/4 of the upper bound of 1–1 contacts of that specific protein sequence could be achieved. The work in [43] gives three generalizations of the approximation algorithms based on the folded protein structure, namely, the U-fold, C-fold, and S-fold and specifies the class of Hart and Istrail algorithm as a U-fold.

A one-third approximation algorithm was proposed in [5]. The input of the algorithm must satisfy two constraints, i.e., the length of the protein sequence must be even and the number of odd-1's should be equal to the number of even-1's.

Another interesting approximation algorithm works over any alphabet that may be finite or infinite [44]. The algorithm achieves a 1/4 approximation when the alphabet is binary, and the grid is two-dimensional, which is similar to the approximation ratio of Hart and Istrail algorithm.

The aim here is to suggest an approximation approach to the HP folding problem. The algorithm is based on the main concept of CA that is the dependency on simple local rules to produce some global behavior. The CA approach achieves one half the 1–1 contacts for short strings provided that no 1–1 contacts are endpoint-1's. Initially, all bits in the sequence are assigned to the core of the folded protein. Each cell will run the rules of the CA simultaneously until bits assigned to cells in one side of the core are identical to those assigned to the other side of the core. The proof shows that all short strings end up with two identical sides of the protein core.

4 Cellular Automata Folding Approximation

4.1 Formal Definition

In general, there are three main steps for running a CA: first, the initial configuration G_0 should be prepared, after which the rules must be simultaneously run for each cell; finally, when the criteria for stopping the process are met or no more rules are applicable, stop running the rules. The folding CA could be defined as follows:

(G, Q, V, δ, G_0, F)

1. G is a two-dimensional grid,
2. Q is the finite and non-empty set of possible states,
3. V = (9, ((− 1,− 1), (0,− 1), (1,− 1), (− 1,0), (0,0), (1,0), (− 1,1), (0,1), (1,1))) is the neighborhood,
4. δ:Q × Q^8 ⇒ Q is the transition function that computes the next state of each automaton in the grid, depending on its current state and the states of its eight neighbors.
5. G_0 is the initial configuration,
6. F is the state where no more rules apply, that is when the elements of the two sides of the core are identical and all surrounding cells are stable.

The lattice is two-dimensional, and the shape of the cells is regular and homogeneous. The neighborhood considered is the Moore neighborhood, and the boundaries are empty cells. The neighborhood could be described based on the directions as V = (9, (NW, N, NE, W, C, E, SW, S, SE)), where C is the central cell.

4.2 Possible States (*Q*)

Each state consists of 5-tuples (n, a_1, a_2, a_3, m) where n is a bit representing the amino acid type and a_1, a_2, a_3 are arrows and m is the marker. The use of arrows helps in avoiding the numbering of amino acids. The use of thousands of integers as the alphabet of the CA would make a large number of states possible, especially when combined with markers. The alphabet $\sum = \{\rightarrow, \leftarrow, \uparrow, \downarrow, 1, 0, *, \diamond, \Leftarrow\}$. The elements of $\sum$ could be interpreted as follows:

- $\rightarrow, \leftarrow, \uparrow, \downarrow$: The interpretation depends on the arrow position describing the state of the cell. The arrows at a_1 and a_2 points to connected neighbors from the original protein sequence. If n is assigned a bit then a_1 and a_2 are also assigned since the string is treated as a loop and all assigned cells have two arrows. a_3 indicates the direction of movement. Only a_3 is allowed if it is assigned to an empty cell.
- 1: is a hydrophobic amino acid
- 0: is a polar amino acid
- *: is the current cell, which controls the flow at a certain generation
- $\diamond$: is a blank space
- $\Leftarrow$: Shifts the loop

To make the CA rules clearer, the following notations would be used where the alphabet is split into amino acids (0, 1), arrows in four directions and the marker alphabet:

Author's personal copy

$\sum_n = \{0, 1\}$
$\sum_a = \{\rightarrow, \leftarrow, \uparrow, \downarrow\}$
$\sum_m = \{*\}$ is the marker
$\sum_{ne} = \sum_n \cup \{\diamond\}$
$\sum_{ae} = \sum_a \cup \{\diamond\}$
$\sum_{me} = \sum_m \cup \{\diamond\}$
$\diamond = (\diamond, \diamond, \diamond, \diamond, \diamond)$
$- = (n, a_1, a_2, a_3, m) \ \forall_n \in \sum_{ne} \ \& \ \forall_{a_1,a_2,a_3} \in \sum_{ae} \ \& \ \forall_m \in \sum_{me}$

$\sum_{ne}$, $\sum_{ae}$, and $\sum_{me}$ are comprised of portions of the alphabet in combination with the blank space. Five symbols define the state of the cell. The symbols cover the possible markers. For simplicity, the symbol $\diamond$ is also used to indicate an empty cell instead of $(\diamond, \diamond, \diamond, \diamond, \diamond)$. The dash $(-)$ is also used in the rules when the cell could be in any state, whether it is empty or has any combination of alphabets.

4.3 Initial Configuration (G_0)

The initial configuration of the CA comprises empty cells and assigned cells. The elements of the protein sequence are assigned to cells at the core along with proper markers, at the initial configuration; all cells outside the core are empty cells. The folding takes place over a two-dimensional lattice. Two rows of cells in the lattice are considered to be the core of the protein, in order to fold the protein relative to the core. We would refer to the rows as the main row and the corresponding row.

In order to prepare the initial configuration G_0, assume that all cells are empty cells, and choose any two rows to be the core of the protein. Place the starting bit at the first cell in the first row, and assign the rest of the bits to lattice cells in order, where:

- Every element in the sequence is assigned to one cell.
- Elements that are connected in the input sequence are assigned to adjacent cells.
- If the middle of the sequence is reached, the assignment of bits to cells continues to the second row.

The size of the CA depends on the size of the protein. Before folding the CA has two rows and a number of columns equal to one half the length of the protein sequence. After folding the number of columns and the number of rows does not exceed one half the length of the protein sequence.

There are two constraints on the input protein. First, the sequence must be of an even length for it to fold to a loop on a 2D grid. Secondly, the number of even-1's should be equal to the number of odd-1's. Both constraints do not affect the upper bound because the upper bound depends on M. The conversion of any sequence to an even length could be achieved by adding a zero to the end of the sequence. The equality between the number of even-1's and odd-1's could be achieved by converting a randomly chosen set of additional 1's to 0's [5].

4.4 Transition Function

After preparing the initial configuration, each cell would decide its next state simultaneously. The cell state stays the same by default unless one of the folding rules changes it. The final output of the CA is the folded protein, where the rows representing the core have identical elements.

The neighborhood of each cell is limited and the CA depends on local rules, where each cell applies the rules based on its current state and neighborhood. Because there is no global view of the protein sequence, the folding takes place at each current cell whenever the neighborhood indicates a need to fold. If the folding does not cause the current neighborhood to become more stable, the CA unfolds and shifts the sequence to test another folding point.

On moving from the column of the current folding point (left corner) to the column that ends the protein sequence (right corner), the current state $(*)$ marker moves and checks if the protein core has identical bits assigned at the current column. If both are identical, the current cell checks the two cells to its right (east and south east of the current cell); if those two cells are in the same state, then the current marker moves one step to the right. If those two cells are not the same and the current cell is of the same type as the element in its east neighbor, the current cell starts folding. In this case, if the folding did not work, the current marker moves to the right cell in order to test the possibility of folding again.

At each step, if the folding is necessary, then the CA will end up in one of the following: folding the 0's or folding the 1's or changing the folding point. At first, the cell labeled or assigned a zero starts to move away from the core; if the folding is possible and actually results in two elements of the same type at the core, there is no need to fold at the side assigned a one. If folding is not possible because there are no more cells to move, or no empty cells available to move to or the folding does not result in a cell labeled one, the algorithm unfolds the 0's and folds the 1's instead. The cell labeled one starts to move away from the core. Changing the folding point is necessary when it is not possible to achieve two identical sides in the core based on the current folding point, even after trying to fold 0's and trying to fold 1's.

The reason why the CA always starts with zeros is to prioritize the retention of the hydrophobic amino acids at the core of the protein and move hydrophilic or polar amino acids away from the core so that they surround the hydrophobic amino acids. In other words, assuming that the main cell has a zero bit and its corresponding is assigned a one, cells of the main row fold until a one is reached; if all the bits assigned to the main row after the current cell are zeros,

Author's personal copy

the CA continues to fold zeros from the other row or the corresponding row until a one is reached or the corresponding cell of the current cell is reached. The folding process may not lead to identical bits, either because there are no bits left or because the parity does not allow two identical bits to exist at the core.

If folding zeros does not work, the algorithm applies the same rules for folding 1's. A successful folding of 1's makes the cells at the main and corresponding row hold the same bit, which in this case, is a zero. Folding 1's might also lead to different bits at the current main cell and its corresponding cell. Hence, the algorithm unfolds and shifts the protein loop to change the folding point.

In the following subsections, the rules are stated formally for the horizontal movement of bits in Sect. 4.4.1, the rules of keeping the chain connected in Sects. 4.4.2 and 4.4.3, and the rules for changing the folding point in Sect. 4.4.4. The rules are simple and does not require heavy computation. They depend on the neighborhood defined in order to determine the cell next state. The rules are deterministic, unambiguous and keep the chain connected.

4.4.1 Horizontal Propagation Rules

In order to specify the direction of movement in any generation of the folding CA, arrows are used and moved through cells. In this subsection, we would discuss the propagation of left and right arrows.

If a non-empty cell has a neighbor to its left with a "left arrow" state, then it would move to the left and substitute the state of its left neighbor. The "left arrow" would move to the right in order to shift the next non-empty cell as well. In this manner, the whole row of non-empty cells moves one step to the left. Folding and changing the folding point requires this process. The rules of the left arrow propagation are as follows:

Rule 1. $\delta(-,-,-,-,(\diamond,\diamond,\diamond,\leftarrow,\diamond),(n,a_1,a_2,a_3,m),-,-,-,-) = (n,\leftarrow,\rightarrow,a_3,m) \quad \forall_n \in \Sigma_n \quad \& \quad \forall_{a_1,a_2} \in \Sigma_a \quad \& \quad \forall_{a_3} \in \Sigma_{ae} \quad \& \quad \forall_m \in \Sigma_{me}$

Rule 2. $\delta(-,-,-,(\diamond,\diamond,\diamond,\leftarrow,\diamond),(n,a_1,a_2,a_3,m),-,-,-,-) = (\diamond,\diamond,\diamond,\leftarrow,\diamond) \quad \forall_n \in \Sigma_n \quad \& \quad \forall_{a_1,a_2} \in \Sigma_a \quad \& \quad \forall_{a_3} \in \Sigma_{ae} \quad \& \quad \forall_m \in \Sigma_{me}$

Rules 1 and 2 are applicable at the same time. Their effects are observed whenever a cell with a left arrow state has at its right a non-empty cell with an amino acid. Figure 2 shows an example of left arrow propagation that moves the zero one step to the left.

Rule 3 shows how the right arrow moves from one cell to the other. The rule states that if a cell reads its right neighbors state and finds the right arrow, in the next generation, it would have the right arrow indicating its state as well.

Rule 3. $\delta(-,-,-,(n,a_1,a_2,\rightarrow,m_1),(n',a_3,a_4,\diamond,m_2),-,-,-,-) = (n',a_3,a_4,\rightarrow,m_2) \quad \forall_{n,n'} \in \Sigma_n \quad \& \quad \forall_{a_1,a_2,a_3,a_4} \in \Sigma_a \quad \& \quad \forall_{m_1,m_2} \in \Sigma_{me}$

The rule restricts the move to the cell that has amino acids as part of its state. If any type of cells is allowed, the arrow would move to infinity, because the grid is theoretically infinite. Figure 3 shows an example of applying Rule 3 to a corner cell.

As Rule 3 does not include any empty cells, there should be a rule that deals with identifying the last element of the row. This is dealt with by Rule 4, which allows the right arrow to be assigned to the empty cell.

Rule 4. $\delta(-,-,-,(n,a_1,a_2,\rightarrow,m),\diamond,\diamond,-,-,-) = (\diamond,\diamond,\diamond,\rightarrow,\diamond) \quad \forall_n \in \Sigma_n \quad \& \quad \forall_{a_1,a_2} \in \Sigma_a \quad \& \quad \forall_m \in \Sigma_{me}$

The propagation of the right arrow to the right stops when it is assigned to the cell after the last non-empty cell. This is when the cells actually move one step to the right and the right arrow is replaced by Rules 5 and 6. Figure 4 shows the assignment of the right arrow to the empty cell following the last non-empty cell, which stops its propagation to the right.

Rules 5 and 6 move non-empty cells to the right one by one, and propagate the right arrow to the left. This is done by swapping the cell containing the right arrow with cells containing amino acids as part of their state. If the cell with the right arrow state has an empty cell to its right, it turns to an empty cell, because of Rule 7.

Rule 5. $\delta(-,-,-,(n,a_1,a_2,a_3,m),(\diamond,\diamond,\diamond,\rightarrow,\diamond),-,-,-,-) = (n,a_1,a_2,\diamond,m) \quad \forall_n \in \Sigma_n \quad \& \quad \forall_{a_1,a_2} \in \Sigma_a \quad \& \quad \forall_m \in \Sigma_{me} \quad \& \quad \forall_{a_3} \in \Sigma_{ae}$

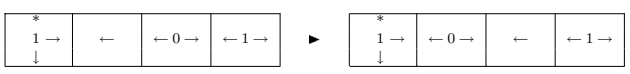


Fig. 2 An example of left arrow propagation.

Fig. 3 An example of right arrow propagation

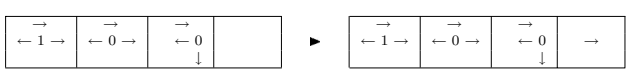


Fig. 4 Stopping the propagation of the right arrow

Author's personal copy

Fig. 5 Shifting non-empty cells to the right

Fig. 6 Deleting the right arrow from the first cell

Rule 6. $\delta(-,-,-,-,(n,a_1,a_2,a_3,m),(\diamond,\diamond,\diamond,\rightarrow,\diamond),-,-,-) = (\diamond,\diamond,\diamond,\rightarrow,\diamond) \quad \forall_n \in \Sigma_n \quad \& \quad \forall_{a_1,a_2} \in \Sigma_a \quad \& \quad \forall_m \in \Sigma_{me} \quad \& \quad \forall_{a_3} \in \Sigma_{ae}$

Rule 7. $\delta(-,-,-,\diamond,(\diamond,\diamond,\diamond,\rightarrow,\diamond),-,-,-,-) = (\diamond,\diamond,\diamond,\diamond,\diamond)$

Rules 5 and 6 are applied at the same time. Figure 5 shows how both rules are applied to a corner cell. This is possible because the rules do not restrict the non-empty cell to be an inner cells.

Finally, the cell with a right arrow state is replaced with an empty cell when it is not possible to shift more non-empty cells, as shown in Fig. 6.

4.4.2 Neighbors of Endpoints and Inner Cells

If an inner cell moves from one place to the other, it might end up at one of the four corners of non-empty cells in the two-dimensional grid. Rules 8–11 change the arrows pointing to connected neighbors when the cell is an endpoint; each rule works on one corner.

Rule 8. $\delta(\diamond,\diamond,\diamond,\diamond,(n,a_1,a_2,a_3,m),-,-,-,-) = (n,\rightarrow,\downarrow,a_3,m) \quad \forall_n \in \Sigma_n \quad \& \quad \forall_{a_1,a_2} \in \Sigma_a \quad \& \quad \forall_{a_3} \in \Sigma_{ae} \quad \& \quad \forall_m \in \Sigma_{me} \quad \& \quad (a_1,a_2) \neq (\rightarrow,\downarrow)$

Rule 9. $\delta(\diamond,\diamond,\diamond,-,(n,a_1,a_2,a_3,m),\diamond,-,-,-) = (n,\leftarrow,\downarrow,a_3,m) \quad \forall_n \in \Sigma_n \quad \& \quad \forall_{a_1,a_2} \in \Sigma_a \quad \& \quad \forall_{a_3} \in \Sigma_{ae} \quad \& \quad \forall_m \in \Sigma_{me} \quad \& \quad (a_1,a_2) \neq (\leftarrow,\downarrow)$

Rules 8 and 9 change the arrows pointing to neighbors when the cell is at the upper left corner and the upper right corner, respectively. Rules 10 and 11 change the arrows at the lower left corner and the lower right corner, respectively.

Rule 10. $\delta(-,-,-,\diamond,(n,a_1,a_2,a_3,m),-,\diamond,\diamond,\diamond) = (n,\rightarrow,\uparrow,a_3,m) \quad \forall_n \in \Sigma_n \quad \& \quad \forall_{a_1,a_2} \in \Sigma_a \quad \& \quad \forall_{a_3} \in \Sigma_{ae} \quad \& \quad \forall_m \in \Sigma_{me} \quad \& \quad (a_1,a_2) \neq (\rightarrow,\uparrow)$

Rule 11. $\delta(-,-,\diamond,-,(n,a_1,a_2,a_3,m),\diamond,\diamond,\diamond,\diamond) = (n,\leftarrow,\uparrow,a_3,m) \quad \forall_n \in \Sigma_n \quad \& \quad \forall_{a_1,a_2} \in \Sigma_a \quad \& \quad \forall_{a_3} \in \Sigma_{ae} \quad \& \quad \forall_m \in \Sigma_{me} \quad \& \quad (a_1,a_2) \neq (\leftarrow,\uparrow)$

Figure 7 shows the application of Rules 8–11; this situation does not occur in the folding CA proposed, the figure is just an illustration of how the rules are applied whenever the cell is at the edge and the arrows pointing to connected neighbors are not updated.

The cell recognizes its position merely through local information, because empty cells cannot be connected neighbors, and cells at the edges are always surrounded by empty cells.

If an endpoint is moved to become an inner cell at the core of the protein, Rule 12 is applied. The rule implies that the arrows directed towards connected neighbors are the left and the right arrows, and therefore, this rule does not apply to inner cells in a fold. It only applies to cells at the core of the protein.

Rule 12. $\delta(-,-,-,(n,a_1,a_2,a_3,m_1),(n',a_4,a_5,a_6,m_2),(n'',a_7,a_8,a_9,m_3),-,-,-) = (n',\leftarrow,\rightarrow,a_6,m_2) \quad \forall_{n,n',n''} \in \Sigma_n \quad \& \quad \forall_{a_3,a_6,a_9} \in \Sigma_{ae} \quad \& \quad \forall_{a_1,a_2,a_4,a_5,a_7,a_8} \in \Sigma_a \quad \& \quad \forall_{m_1,m_2,m_3} \in \Sigma_{me} \quad \& \quad (a_1,a_2) \notin \{(\leftarrow,\uparrow),(\leftarrow,\downarrow)\} \quad \& \quad (a_4,a_5) \neq (\leftarrow,\rightarrow)$

Figure 8 shows how rules are applied simultaneously to change the neighborhood arrows; Rule 12 changes the inner cell arrows and Rule 9 changes the endpoint arrows.

4.4.3 Keeping the Loop

Moving cells should not affect the initial assumption of being a loop. This is because of the fact that cells moving away from each other might cause connected neighbors in the original chain to get separated from each other. Also, the first element and the last element of the protein binary string must follow each other. Rule 13 deals with the upper right

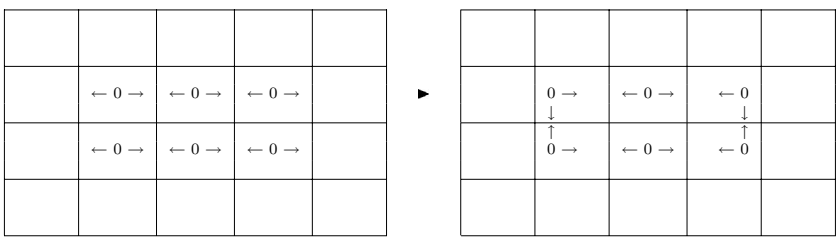


Fig. 7 Changing arrows pointing to neighbors at the endpoints

Author's personal copy

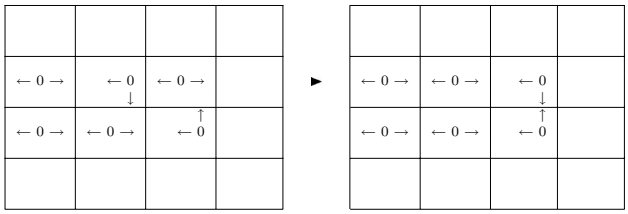


Fig. 8 Neighborhood arrows of inner cells and endpoint

corner of the core of the protein and Rule 14 deals with the lower right corner.

Rule 13. $\delta(\diamond,\diamond,\diamond,-,(\diamond,\diamond,\diamond,\leftarrow,\diamond),\diamond,-,(n,\leftarrow,\uparrow,\diamond,\diamond),\diamond) = (n,\leftarrow,\downarrow,\diamond,\diamond) \quad \forall_n \in \Sigma_n$

Rule 14. $\delta(-,(\diamond,\diamond,\diamond,\leftarrow,\diamond),\diamond,-,(n,\leftarrow,\uparrow,\diamond,\diamond),\diamond,\diamond,\diamond,\diamond) = (\diamond,\diamond,\diamond,\rightarrow,\diamond) \quad \forall_n \in \Sigma_n$

4.4.4 Changing the Folding Point

At any generation, if there is no way for folding to occur in the current arrangement, the CA unfolds all the folded elements and changes the folding point by shifting the protein sequence.

If the protein core cells at the folding point column are assigned different bits, then the CA shifts the sequence and tests another folding point, as shown in Fig. 9. This is because the CA does not fold the endpoints at the beginning of the protein core. Rule 15 adds the shifting flag to the folding point if the cells are assigned different bits. The application of the rule is shown in Fig. 10. The folding point is also changed when there is no folding possible within the current folding point; hence, at any point, the folding might stop so that the CA could change the folding point, according to the rules described in this subsection.

Rule 15. $\delta(\diamond,\diamond,\diamond,\diamond,(n,\rightarrow,\downarrow,\diamond,*),(n',a_1,a_2,\diamond,\diamond),\diamond,(n'',\rightarrow,\uparrow,\diamond,*),-) = (n,\rightarrow,\downarrow,\Leftarrow,*) \quad \forall_{n,n',n''} \in \Sigma_n \quad \& \quad \forall_{a_1,a_2} \in \Sigma_{ae} \quad \& \quad n \neq n''$

The first configuration to the left in Fig. 9 shows the initial configuration of the string (100100) after applying Rule 15; the current element is the upper left element or the first cell in the upper row, which is the folding point. The row containing the current element is the main row. The state of the current cell is $(1,\rightarrow,\downarrow,\Leftarrow,*)$. The state at its corresponding row is $(0,\rightarrow,\uparrow,\diamond,*)$. As the main row has a one and the corresponding row has a zero at the current state, and folding is not possible, the algorithm would shift, as shown in the second CA configuration, in Fig. 9. The situation is the same even after the first shifting process, but this time, the main row has a zero, and its corresponding row has a one; because they do not have the same bit, the algorithm shifts again to the configuration shown on the right in Fig. 9.

The idea of circulating the protein loop to change the folding point is straightforward. However, if this change is to be brought about with the use of local rules only, then there are more details to consider. Every cell, be it an inner cell or an endpoint, decides its next state based on what the neighborhood indicates about the relative position of the cell. The transition rules that shift the protein loop in order to change the folding point are defined as follows:

Rule 16. $\delta(\diamond,\diamond,\diamond,\diamond,(n,\rightarrow,\downarrow,\Leftarrow,*),(n',a_1,a_2,\diamond,\diamond),\diamond,-,-) = (n',\rightarrow,\downarrow,\diamond,*) \quad \forall_{n,n'} \in \Sigma_n \quad \& \quad \forall_{a_1,a_2} \in \Sigma_a$

Rule 17. $\delta(\diamond,\diamond,(n,\rightarrow,\downarrow,\Leftarrow,*),\diamond,\diamond,-,\diamond,\diamond,\diamond) = (n,\rightarrow,\uparrow,\diamond,*) \quad \forall_n \in \Sigma_n$

Rule 18. $\delta(\diamond,\diamond,\diamond,(n,\rightarrow,\downarrow,\Leftarrow,*),(n',a_1,a_2,\diamond,\diamond),-,-,-,-) = (\diamond,\diamond,\diamond,\leftarrow,\diamond) \quad \forall_{n,n'} \in \Sigma_n \quad \& \quad \forall_{a_1,a_2} \in \Sigma_a$

Rule 19. $\delta(\diamond,(n,\rightarrow,\downarrow,\Leftarrow,*),(n',a_1,a_2,\diamond,\diamond),\diamond,(n,\rightarrow,\uparrow,\diamond,*),-,\diamond,\diamond,\diamond) = (n,\rightarrow,\uparrow,\diamond,\diamond) \quad \forall_{n,n'} \in \Sigma_n \quad \& \quad \forall_{a_1,a_2} \in \Sigma_a$

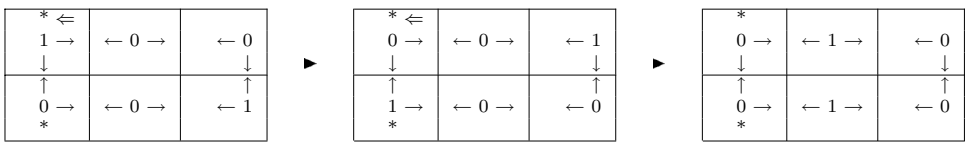


Fig. 9 Changing the Folding Point

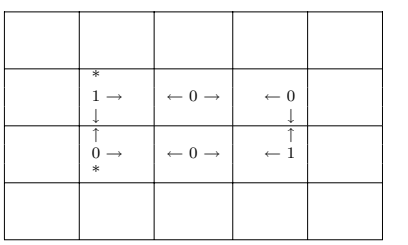


Fig. 10 The shifting flag initiation (generation 1)

Author's personal copy

In Fig. 10, the shifting flag is initiated by Rule 15 because the folding point has a state of $(1, \rightarrow, \downarrow, \diamond, *)$ and its corresponding point has a state of $(0, \rightarrow, \uparrow, \diamond, *)$. The figure shows the initial configuration and first generation of running the folding CA over the string (100100).

Rules 16–19 are applied simultaneously, and in one generation, the CA shifts the folding point (Fig. 11), and all the other cells are arranged accordingly in several following generations. The state of the cell after the folding point is replaced does not allow for folding or changing to yet another point until the current shifting process is complete, because the rule that initiates a change in the folding point (Rule 15) assumes that an empty cell is present at the lower left side of the folding point neighborhood.

Figure 12 shows the movement of the left arrow and the movement of the cells as a loop. The left arrow moves based on Rules 1 and 2. Rules 13 and 14 connect the movement between the two rows containing the loop. These are the CA configurations that are observed directly after the initial configuration and the first generation.

Figure 13 shows the movement of the cells in the second row one by one to the right. The movement is achieved by Rules 5 and 6. Those generations come directly after the ones in Fig. 12. Rule 11 is applied to the neighborhood arrows of the lower right corner.

Figure 14 shows the final output of the process of changing the folding point. As the folding point is still different from its corresponding point, the shifting process would be repeated, as shown in Fig. 9. The last two generations come directly after those shown in Fig. 13, where the lower left corner joins the loop and the right arrow leaves the loop. The right arrow is deleted by Rule 7.

5 Proposed Approach Properties

There are three properties of the one-fourth and one-third approximation algorithms that could be improved. First, the even-1's on one side and odd-1's on the other side help in calculating a minimum number of 1's contributing to 1–1 contacts. This approach ignores all odd-1's at the even side and all even-1's at the odd side. If it was possible to use those 1's and calculate the minimum number of them then the approximation ratio would improve significantly.

Secondly, regarding the use of normal forms in folding, a possible method of improving the approximation ratio is to keep the folding process more dynamic or more aware of the environment. Finally, regarding the use of a predetermined folding point, it was observed that in many strings, a high number of 1–1 contacts could be achieved with different folding points. The proposed approach does not take

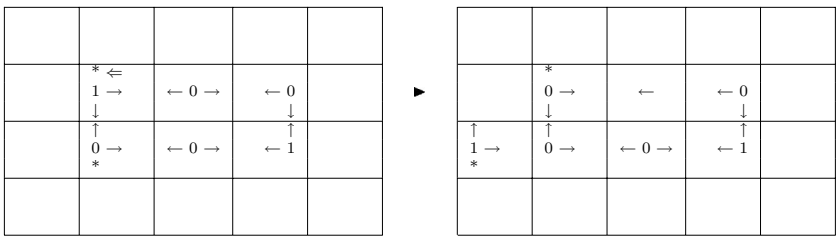


Fig. 11 Changing the folding point (generation 2)

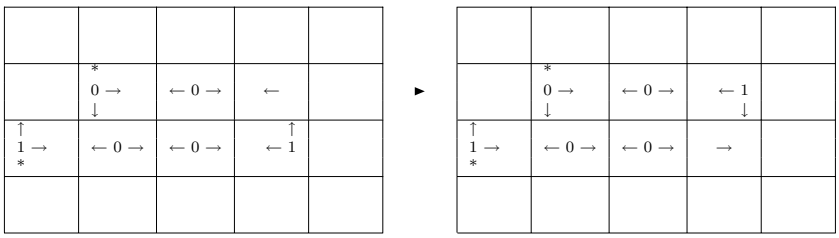


Fig. 12 Maintaining the loop (generations 3 and 4)

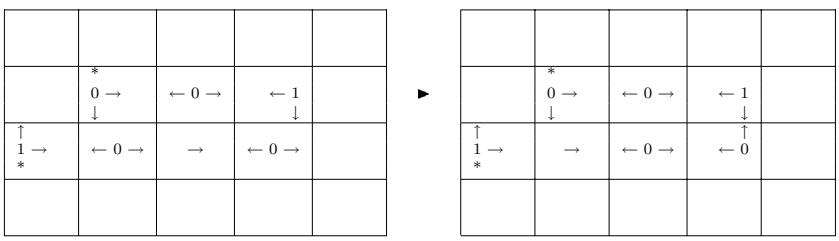


Fig. 13 Moving the second row (generations 5 and 6)

Author's personal copy

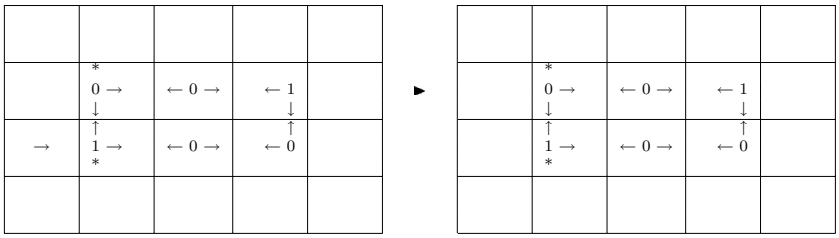


Fig. 14 The final step in changing the folding point (generations 7 and 8)

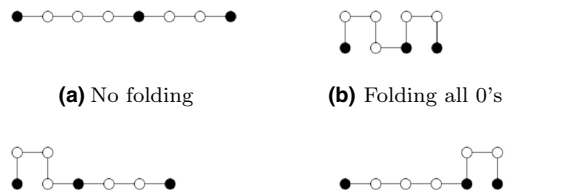


Fig. 15 Folding options of the string (10001001)

the positions of 1's in the string into account and depends on the surrounding environment to make folding decisions.

There are multiple ways to fold the same substring. Choosing one of the folding methods to be a normal form would limit the number of possible matches or 1–1 contacts. In the following graphs, the ones in the protein binary sequence are represented with black vertices and the zeros are represented with white vertices.

Examples of the possible ways to fold the substring (10001001) are given in Fig. 15, which shows two blocks of zeros, one of odd-length and one of even-length, separated by a single 1. Valid possibilities for folding are as follows: fold the two zero blocks (Fig. 15b), fold the odd-length block only (Fig. 15c), fold the even-length block only (Fig. 15d), or fold none of the blocks (Fig. 15a).

Figure 16 shows a substring of two zero blocks, separated by a single 1. Both blocks are of even length. It is not possible to fold both blocks at the same time as only one black vertex is available. If an even-length block has more than two zeros it could be further folded as illustrated in Fig. 17b, c.

The folding of 1's so that they are away from the core is shown in Fig. 17e, f; this approach is beneficial when a long substring of 1's have to be folded, and no other substring with the same length is present in the protein sequence. Hence, each element from the long substring of 1's matches with 1's from the same substring. Note that the folded bits in Figs. 15, 16, and 17 are at one side of the main row as it is assumed that the other side will have the corresponding row.

6 Results

For every possible binary string with lengths of 4, 6, 8, and 10, there exists a configuration where every element in the main row is of the same type as the elements in its corresponding row, as we prove in Lemma 2. If there are no 1's

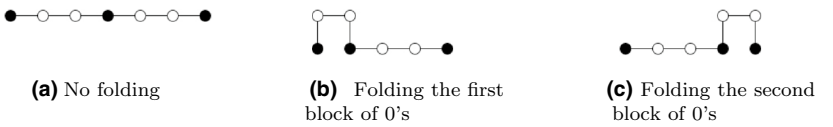


Fig. 16 Folding options of the string (1001001)

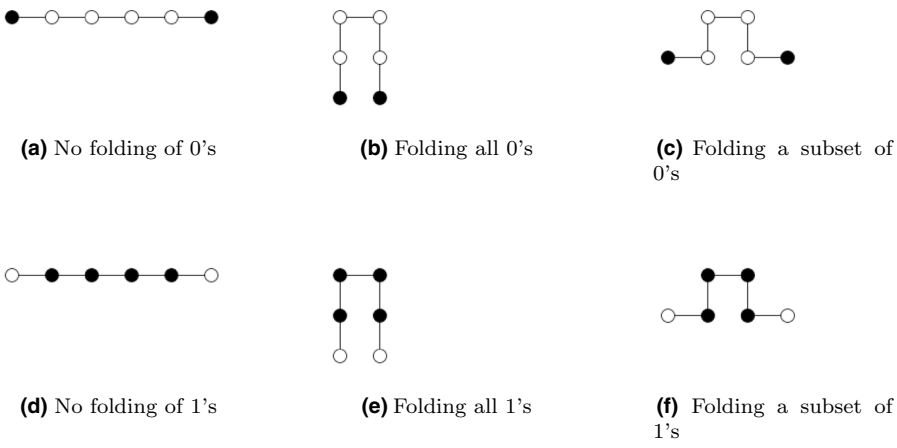


Fig. 17 Folding subsequences with more than two amino acids

Author's personal copy

at the end points and every bit in the main row is the same as the bits in its corresponding row, then every even-1 and every odd-1 would participate in a 1-1 contact, and this is one half the upper bound of $2 \times M$.

Protein sequences are assumed to be of even length, with an even number of 1's in all strings. There are two equal halves of 1's, which include those at even positions and those at odd positions. To calculate the number of zeros in the string, we subtract the number of 1's from the total length of the string. This would result in an even number of zeros, because subtracting even numbers results in an even number. Consequently, an even number of zero blocks of odd-length may be present in the input string. In fact, the constraint is even stricter than that, as shown in Lemma 1.

Lemma 1 *If a language L1 is defined as a set of even-length strings, where each string w has even-1's = odd-1's and even-0's = odd-0's, and the language L2 is defined as a set of even-length strings, where each string w has even-1's = odd-1's, then every string in L2 is contained in L1.*

We could analyze L2 inductively to see that it maintains the property of L1 for each case. Assuming an empty string at first, at each step, the added 1's and zeros must keep the length of the string even. Hence, let us start by adding two consecutive 1's, which occupy one even position and one odd position, and adding any number of even-length 1's occupying as many even positions as the odd positions. A second case is to add two 1's in two scattered positions, in one that is even and one that is odd; the number of zeros that must be padded between those two 1's is even, which implies that even-0's = odd-0's, which is also observed in L1.

In other words, concatenating even-1's and odd-1's would lead to an even-length string; if it is subtracted from the total even length of the protein sequence, it would result in another even-length string of zeros, which in turn could be partitioned into equal even and odd positions.

Lemma 2 *Given a protein string S, the CA folding algorithm finds a folding with at least 1/2 the contacts minus the endpoints for all strings of length 10 or less.*

If a string can be folded at a point where every element and its corresponding element are of the same type, or the two sides of the fold are symmetric, then the number of 1–1 contacts is 1/2 unless the endpoints are 1–1. The number becomes 1/2 minus those endpoints. We can start by analyzing short strings that cannot be folded as a first step. Those are strings of length 4 and length 6.

Based on our first lemma, 1's will occupy (if any) as many even positions as odd and 0's will occupy (if any) as many even positions as odd. There are two even positions and two odd positions in a string of length 4. The possibilities are: All are 1's or all are 0's which results in a symmetric fold. Any combination has at least two consecutive 1's or two consecutive 0's, which can be considered endpoints and results in a symmetric fold.

The possibilities in a string of length 6 are: All are 1's or all are 0's which results in a symmetric fold. Any combination has at least two sets of consecutive bits (1's or 0's), which can fold against each other or become the endpoints and in both cases it results in a symmetric fold. Strings of length 8 and 10 can have all 1's or all 0's which results in a symmetric fold. Also, the strings can have a combination of bits. Both must have at least two sets of consecutive 1's or 0's, which after folding falls back to the case of strings of length 4 and 6 respectively.

The CA approach has exceptions similar to those discussed in the one-fourth algorithm [42]. The 1–1 contacts of a string are zero when the number of 1's in the string is zero, such as (00000000) or if the same string of zeros has two 1's that are connected neighbors, such as (01100000) and (00011000). Those exceptions occur quite infrequently in practice [42].

The criterion for stopping the folding process is that the algorithm has no action to perform and ends with a perfect match of the core sides. The long protein sequence might not emerge into a fold with identical core sides. In order to avoid the infinite loop, the algorithm could be modified by adding two markers to the alphabet $\sum_m$ to start and end the folding process; however, this may result in a poorly folded protein. Another method is to run the CA over all possible folding points and to choose the best folding with the highest number of 1–1 contacts.

7 Conclusion and Future Work

This paper describes a Cellular Automata approach to HP protein folding approximation. The proposed approach achieves an approximation ratio of one half minus the number of the hydrophobic amino acids at the endpoints (if there are any). This is proved in short sequences. HP protein folding approximation using Cellular Automata enhances approximation algorithms in multiple ways. Instead of folding blocks of amino acids based on a normal form or a template, the amino acids were folded dynamically at each step, by considering information from the surrounding environment. The proposed approach does not take the hydrophobic amino acids position into account, and does not assume a predetermined folding point.

The folding approach proposed could be improved by folding blocks of the same type further, in order to make the folded protein more compact, which may introduce additional H–H contacts. It would be beneficial to find new methods for guaranteeing a minimum number of H–H contacts

Author's personal copy

without limiting the approximation approach. Also, there is more room for creativity in extending the two-dimensional lattice model to a three-dimensional one.

Acknowledgements We would like to thank Dr. Khair Eddin Sabri, Dr. Loai Alnemer, and Dr. Rawan Ghnemat for their suggestions and comments that greatly improved the content of this manuscript.

References

1. Sarkar P (2000) A brief history of cellular automata. ACM Comput Surv 32(1):80–107
2. Dill KA (1985) Theory for the folding and stability of globular proteins. Biochemistry 24(6):1501–1509
3. Lau KF, Dill KA (1989) A lattice statistical mechanics model of the conformational and sequence spaces of proteins. Macromolecules 22(10):3986–3997
4. Dill KA, Bromberg S, Yue K, Chan HS, Ftebig KM, Yee DP, Thomas PD (1995) Principles of protein folding a perspective from simple exact models. Protein Sci 4(4):561–602
5. Newman A (2002) A new algorithm for protein folding in the hp model. In: Proceedings of the Thirteenth Annual ACM-SIAM Symposium on Discrete Algorithms, SODA '02. Society for Industrial and Applied Mathematics, Philadelphia, pp 876–884
6. Li X, Li X, Xiao Y, Jia B (2016) Modeling mechanical restriction differences between car and heavy truck in two-lane cellular automata traffic flow model. Physica A 451(Supp C):49–62
7. Pandey G, Rao KR, Mohan D (2017) Modelling vehicular interactions for heterogeneous traffic flow using cellular automata with position preference. J Mod Transp 25(3):163–177
8. Raghavan R (1993) Cellular automata in pattern recognition. Inf Sci 70(1):145–177
9. Madain A, Abu Dalhoum AL, Hiary H, Ortega A, Alfonseca M (2014) Audio scrambling technique based on cellular automata. Multimed Tools Appl 71(3):1803–1822
10. Abu Dalhoum AL, Madain A, Hiary H (2015) Digital image scrambling based on elementary cellular automata. Multimed Tools Appl 75(24):17019–17034
11. Burks C, Farmer D (1984) Towards modeling dna sequences as automata. Physica D 10(1–2):157–167
12. de Sales JA, Martins ML, Stariolo DA (1997) Cellular automata model for gene networks. Phys Rev E 55:3262–3270
13. Sirakoulis G, Karafyllidis I, Mizas C, Mardiris V, Thanailakis A, Tsalides P (2003) A cellular automaton model for the study of dna sequence evolution. Comput Biol Med 33(5):439–453
14. Mizas C, Sirakoulis G, Mardiris V, Karafyllidis I, Glykos N, Sandaltzopoulos R (2008) Reconstruction of DNA sequences using genetic algorithms and cellular automata: towards mutation prediction? Biosystems 92(1):61–68
15. Mizas C, Sirakoulis GC, Mardiris V, Karafyllidis I, Glykos N, Sandaltzopoulos R (2016) DNA cellular automata. In: Adamatzky A, Martínez GJ (eds) Designing beauty: the art of cellular automata. Springer, Cham, pp 127–128
16. Maji P, Parua S, Das S, Chaudhuri PP (2005) Cellular automata in protein coding region identification. In: Proceedings of 2005 International Conference on Intelligent Sensing and Information Processing, pp 479–484
17. Takata D, Isokawa T, Matsui N, Peper F (2013) Modeling chemical reactions in protein synthesis by a Brownian cellular automaton. In: First International Symposium on Computing and Networking, pp 527–532
18. Elsayed WM, Elmogy M, El-Desouky B (2017) Evolutionary behavior of DNA sequences analysis using non-uniform probabilistic cellular automata model. Cincia e Tcnica Vitivincola 32:137–148
19. Chopra P, Bender A (2006) Evolved cellular automata for protein secondary structure prediction imitate the determinants for folding observed in nature. Silico Biol 7(7):87–93
20. Santos J, Villot P, Dieguez M (2013) Cellular automata for modeling protein folding using the hp model. In: IEEE Congress on Evolutionary Computation (CEC), pp 1586–1593
21. Santos J, Villot P, Diéguez M (2014) Emergent protein folding modeled with evolved neural cellular automata using the 3d HP model. J Comput Biol 21(11):823–845
22. Varela D, Santos J (2017) Protein folding modeling with neural cellular automata using the face-centered cubic model. In: Ferrández Vicente JM, Álvarez-Sánchez JR, de la Paz López F, Toledo Moreo J, Adeli H (eds) Natural and artificial computation for biomedicine and neuroscience: international work-conference on the interplay between natural and artificial computation. Springer, Cham, pp 125–134
23. Madain A, Abu Dalhoum AL, Sleit A (2016) Computational modeling of proteins based on cellular automata. Int J Adv Comput Sci Appl 7(7):491–498
24. Madain A, Abu Dalhoum AL, Sleit A (2016) Protein folding in the two-dimensional hydrophobic polar model based on cellular automata and local rules. Int J Comput Sci Netw Secur 16(9):48–54
25. Madain A, Abu Dalhoum AL, Sleit A (2018) Application of local rules and cellular automata in representing protein translation and enhancing protein folding approximation. Prog Artif Intell. https://doi.org/10.1007/s13748-018-0146-8
26. Xiao X, Shao S, Ding Y, Huang Z, Chou KC (2006) Using cellular automata images and pseudo amino acid composition to predict protein subcellular location. Amino Acids 30(1):49–54
27. Xiao X, Ling W (2007) Using cellular automata images to predict protein structural classes. In: The 1st International Conference on Bioinformatics and Biomedical Engineering, 2007. ICBBE 2007. pp 346–349
28. Xiao X, Wang P, Chou KC (2008) Predicting protein structural classes with pseudo amino acid composition: an approach using geometric moments of cellular automaton image. J Theor Biol 254(3):691–696
29. Kavianpour H, Vasighi M (2017) Structural classification of proteins using texture descriptors extracted from the cellular automata image. Amino Acids 49(2):261–271
30. Xiao X, Wang P, Chou KC (2008) GPCR-CA: a cellular automaton image approach for predicting g-protein-coupled receptor functional classes. J Comput Chem 30(9):1414–1423
31. Diao Y, Ma D, Wen Z, Yin J, Xiang J, Li M (2008) Using pseudo amino acid composition to predict transmembrane regions in protein: cellular automata and lempel-ziv complexity. Amino Acids 34(1):111–117
32. Madain A, Abu Dalhoum AL, Sleit A (2016) Potentials and challenges of building computational models of proteins based on cellular automata. Int J Comput Sci Inf Secur 14(9):1086–1091
33. Paterson M, Przytycka T (1996) On the complexity of string folding. Discret Appl Math 71(1):217–230
34. Crescenzi P, Goldman D, Papadimitriou C, Piccolboni A, Yannakakis M (1998) On the complexity of protein folding (abstract). In: Proceedings of the Second Annual International Conference on Computational Molecular Biology, RECOMB '98. ACM, New York, pp 61–62
35. Berger B, Leighton T (1998) Protein folding in the hydrophobic-hydrophilic (hp) is np-complete. In: Proceedings of the Second Annual International Conference on Computational Molecular Biology, RECOMB '98. ACM, New York, pp 30–39
36. Lopes HS (2008) Evolutionary algorithms for the protein folding problem: a review and current trends. In: Smolinski TG, Milanova

Author's personal copy

MG, Hassanien A-E (eds) Computational intelligence in biomedicine and bioinformatics: current trends and applications. Springer, Berlin, pp 297–315
37. Bokovi B, Brest J (2016) Genetic algorithm with advanced mechanisms applied to the protein structure prediction in a hydrophobic-polar model and cubic lattice. Appl Soft Comput 45(Suppl C):61–70
38. Wang S, Wu L, Huo Y, Wu X, Wang H, Zhang Y (2016) Predict two-dimensional protein folding based on hydrophobic-polar lattice model and chaotic clonal genetic algorithm. In: Yin H, Gao Y, Li B, Zhang D, Yang M, Li Y, Klawonn F, Tallón-Ballesteros AJ (eds) Intelligent data engineering and automated learning—IDEAL 2016: 17th international conference. Springer, Yangzhou, pp 10–17
39. Llanes A, Vélez C, Sánchez AM, Pérez-Sánchez H, Cecilia JM (2016) Parallel ant colony optimization for the HP protein folding problem. In: Ortuño F, Rojas I (eds) Bioinformatics and biomedical engineering: 4th international conference. Springer, Cham, pp 615–626
40. Lopes HS, Bitello R (2007) A differential evolution approach for protein folding using a lattice model. J Comput Sci Technol 22(6):904–908
41. Günther F, Möbius A, Schreiber M (2017) Structure optimisation by thermal cycling for the hydrophobic-polar lattice model of protein folding. Eur Phys J Special Top 226(4):639–649
42. Hart WE, Istrail S (1996) Fast protein folding in the hydrophobichydrophilic model within three-eighths of optimal. J Comput Biol 3(1):53–96
43. Lyngs RB, Pedersen CNS (1999) Protein folding in the 2d hp model. Techreport RS-99-16, BRICS Bioinformatics Research Center, University of Aarhus
44. Mauri G, Pavesi G (2000) Approximation algorithms for string folding problems. In: Proceedings of the International Conference IFIP on Theoretical Computer Science, Exploring New Frontiers of Theoretical Informatics, TCS '00. Springer-Verlag, London, pp 45–58

View publication stats