aCell: A Cell Model Using Artificial Chemistry for Exploring Computation in Nature

Chien-Le Goh [1], Yong Kheng Goh [2] and Hong Tat Ewe [3]
[1] *Faculty of Computing and Informatics, Multimedia University, Cyberjaya, Malaysia*
[2, 3] *Lee Kong Chian Faculty of Engineering and Science,*
Universiti Tunku Abdul Rahman (Sungai Long Campus), Selangor, Malaysia
clgoh@mmu.edu.my (Corresponding Author); gohyk@utar.edu.my; eweht@utar.edu.my

***Abstract*—This study proposes aCell, a new artificial cell model for exploring computation in nature using artificial chemistry. The model represents molecules as spheres and simulates chemical reactions through collision-based rewriting rules. Cell components like membranes are modeled abstractly. Experiments demonstrate aCell's capabilities in forming larger molecules from basic building blocks, modeling reversible reactions, implementing logic gates, storing state, and solving optimization problems. The unified framework supports simulating both essential biology and computation using the same artificial chemistry constructs. aCell provides a flexible platform for investigating information processing in natural cells and bio-inspired computing.**

Keywords: Simulation System, Artificial Cell, Computation, Artificial Chemistry, Artificial Life

1. INTRODUCTION

Natural computing has three main branches. They are computing inspired by nature, synthesis of natural phenomena in computers and computing with natural materials [1]. Artificial life ("AL" or "Alife") is an area of studies under the second branch of natural computing. Its main goal is to study life by creating virtual entities which have life-like behaviors. According to one of the pioneers of Alife, C. Langton, artificial life is the study of synthetic systems that exhibit behavioral characteristics of natural living systems. It complements the traditional biological sciences concerned with the analysis of living organisms by attempting to synthesize life-like behaviors within computers and other artificial media [2]. It also complements the field of artificial intelligence by identifying the source of intelligent rational behaviors from the bottom up instead of top down.

Research on artificial life can be conducted at various levels, from cell, tissue, organ, organ system, organism, population and community to ecosystem. Much research work has been done to simulate processes and life-like behaviors of entities such as cells, neurons, genes, immune systems, membranes, plants, bacteria and swarms of animals like ants, termites, bees, spiders, fish and birds [3]. Among the entities, simulation of cells is especially important because cells are the basic building blocks of life. Cells can grow, reproduce, process information and respond to stimuli. An exceptionally large variety of cells exist in varied sizes and shapes. Some move and have changing structures while some are stationary and have stable structures. Organisms on earth exist as a single cell or contain up to trillions of cells [4].

Nowadays, with the knowledge gained from molecular cell biology, it is difficult to deny that powerful information processing capabilities exist in a cell. A cell presents a rich area of exploration for synergies between biology and computation. Research into the information processing capabilities of a cell can provide a better insight into both biology and computation. It has the potential to reveal the similarities and differences between computation designed by humans and computation that occurs in nature. If a good working model of a cell can be created, even if only partially realistic and highly abstracted, the general behaviors of cells can be studied from the perspective of computer science. Algorithms in nature can be learned through the model. With the ever-increasing adoption of artificial intelligence in information systems, discovery of new artificial intelligence algorithms can enhance the

capabilities of information systems. Alife as a form of bio-inspired computing is a major area of investigation.

For simulating cells in the context of Alife, artificial chemistry ("AChem") is one of the approaches used. It is an approach used to model biochemical molecules and simulate the chemical reactions of these molecules at a high abstract level. It attempts to capture fundamental properties of biochemistry without modeling specific actual chemical processes [5]. Chemical molecules are commonly represented with alphabets like A, B, C, ... in AChem for abstraction. Artificial cells are cells simulated with AChem and programming codes.

In this paper, a new artificial cell model based on AChem is proposed. We shall call the model aCell for ease of reference in this paper. The research objective of aCell is to propose an artificial cell model for exploring computation in nature, using artificial chemical reactions with minimal usage of programming codes. aCell aims to be grounded to cell biochemistry to enable relatively easy direct mapping of real-life cell biochemical reactions to aCell reactions. Furthermore, reactions and molecules are to be used as much as possible to simulate the behavior of cells and to sense the status of cells.

aCell is built on top of the foundation laid by the organic builder [6] and membrane computing [7], [8]. Membrane computing and the organic builder both employ AChem. They both simulate molecules and chemical reactions using rewriting rules.

The membrane computing model is a model capable of computation by mimicking membranes and cell behavior. However, it operates in an abstract manner forgoing the simulation of physical properties of molecules such as the locations of molecules, the movement of molecules and the triggering of reactions based on collisions. Reactions can occur when the necessary molecules exist within a membrane regardless of the locations of the molecules (Fig. 2).

On the other hand, the organic builder is a model not proposed for computation. It is proposed to simulate molecules and reactions in a simulation space which has molecules moving about freely (Fig. 1). The molecules can react when they collide based on a list of defined reactions. It simulates the physical properties of molecules such as their locations, their movement and molecular bonds. It has been shown to be able to simulate cell membranes, cell reproduction and gene mutation.

aCell combines the major features of the membrane computing model and the organic builder so that these two models can complement each other. aCell uses its own methods to simulate endothermic and exothermic reactions, an explicit flow of molecules into and out of the simulation space, cell membranes, cell DNA, changes to membrane permeability based on the energy levels of cells, aging of cells, cell reproduction and cell mutation based on cell fitness levels. Furthermore, the simulation space of aCell is a 3D space instead of a 2D plane.

The next section describes existing research work related to aCell. It is followed by descriptions of the building blocks of aCell in section 3 and the processes of aCell in section 4. Section 5 presents the experiments conducted to verify the capabilities of aCell. Section 6 discusses the significance of aCell, and section 7 presents the conclusion of this paper.

2. Literature Review

There are broadly three areas of research work related to aCell. They are simulation of chemistry, simulation of cells and computation using artificial cells or artificial chemistry. They are described briefly in the following subsections. Thereafter, the features adopted by aCell are described.

2.1. *Simulation of Chemistry*

In this subsection, five notable approaches to simulating biochemistry are described. These approaches are used to investigate natural chemical reactions.

The Organic Builder [6], [9] is an artificial chemistry system where the atoms are represented as circles moving around randomly in a two-dimensional area. The atoms can react among themselves according to preset reactions rules when they collide with each other. They can also form bonds among themselves to form molecules. Fig. 1 illustrates how a simulation space in the Organic Builder can look like. This example contains molecules a2, b0, c0 and d1. The line between two molecules is the molecular bond between them. Molecules forming a loop by molecular bonds are considered a cell. In the example, molecule d1 is the DNA molecule in a cell. Reactions i) b0 + b0 => b0b0 where two molecules bond together, and ii) b0b0 + c0 => a2c0 where two bonded molecules are

transformed to two other bonded molecules are examples of reactions supported.

The Organic Builder has been used to discover the reaction rules needed for the processes involved in cell reproduction. DigiHive [10] has further enhanced this line of work by adding more features closer to the real world.

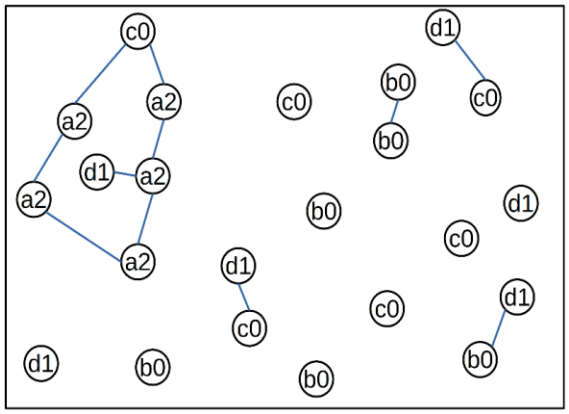

Fig. 1: An example of the Organic Builder.

An attempt to bring chemically realistic model to AChem is the synthon approach [11]. The purpose is to use a model of atoms and molecules to study the physical properties of complex chemical reactions. A synthon is defined as $S(A) = \langle W, A, R, E \rangle$ where W is the set of vertices representing virtual atoms, A is the set of vertices representing atoms, R is the vertices representing the electrons explicitly modeled and E is the set of edges associating vertices from sets W to A and from sets A to W. The virtual atoms are used to represent the part of a molecule that is not relevant to the reactions under investigation. A reaction is defined as $S(A) \overset{T}{\Rightarrow} S'(A)$. A tool based on the synthon model has been used successfully to visualize polymerization and de-polymerization reactions.

BioDrive [12] is an artificial chemistry model which models the reactions of molecules using differential equations. Concentration of each type of molecules in the system is used to determine the reaction rate. Reactions are expressed using differential equations and the effects of a reaction on other reactions are modeled mathematically. The precise locations of molecules are not taken into consideration in this model. Changes in concentrations of molecules against time can be calculated by solving the differential equations.

Another approach to simulate chemistry is an approach which uses squares and tiles [13]. Squares are used to represent the basic elements to configure tiles of different shapes and sizes. The squares and tiles are placed in a well-mixed 'soup' in which they are randomly chosen to collide. The result of a collision is a change to the size and shape of the tiles. This change represents a chemical reaction. This model has been applied to the investigation of the growth and the decomposition of complex molecules in chemistry.

[14] proposed an AChem model based on strings of characters. In the model, a text character is used to represent an atom and a string of characters is used to represent a molecule. A set of recombination rules function as reactions to transform the connected strings from one configuration to another. This model has been applied to model complex biochemical reactions such as oxidation of fatty acids and DNA related biochemical reactions such as replication, transcription and translation. With the introduction of membrane molecules, this model has been extended to include membrane structures and to enable movement of molecules across membranes [15].

Simulation of Cells

Simulation of natural cells at low level is extremely complex research. It is also hampered by the incomplete knowledge we currently have about a cell. Therefore, related research work in this section normally focuses on certain selected cell processes only. The exception is E-CELL which attempts to mathematically simulate a single cell or a group of cells.

E-CELL [16]–[18] is a multi-algorithm, multi-timescale cell simulator. It focuses on simulating biochemical interactions and components within a single cell mathematically without using virtual matters like atoms or molecules. It is efficient for large scale simulations of different timescales. E-CELL has been used to investigate the global behaviors of the liver.

A model that can generate complete organisms possessing metabolism and morphology from a single initial cell was proposed by [19]. It was later extended to use L-systems [20]. It is a hybrid model combining AChem and L-systems. The model simulates a cell growing into an organism of cells in a 2-D grid which contains chemical molecules. Chemical reactions in this model consume or produce energy besides producing new chemical molecules. The action list of the simulated cells contains actions to absorb or release chemical molecules, consume energy, transform a molecule to another via a reaction rule, perish and reproduce new

cells. To grow an organism in an environment, a genetic algorithm is used. The genetic algorithm is applied to evolve two different chromosomes in the organism, one specifying the rules of growth and another specifying cellular actions. Balanced creatures and actions that can develop a suitable metabolism system in an environment are favored in the evolution of the two chromosomes. With this model, artificial creatures that can survive in an environment have been created successfully and they have self-healing abilities.

Although it is not based on AChem but a physical rule of motion, PPS (Primordial Particle System) [21], [22] is able to produce emergent behavior of particles forming cells when a certain combination of parameter values is used. The cells can reproduce, restructure themselves and form a life-like system. aCell does not use the discovery made with PPS because linking it to AChem is currently beyond the scope of aCell. However, it is worth noting as an example that AChem may not be the simplest way to simulate a cell.

2.2. *Computation Using Artificial Cells or Artificial Chemistry*

There are three main research areas using cells and chemistry as inspiration to find new computational approaches. They are membrane computing, cellular automata and evolutionary computation. For computational efficiency, abstraction is normally used to avoid computing the sophisticated internal processes involved.

Membrane computing is an area of computer science that aims to abstract computing ideas and models from the structure and the functioning of living cells. A system based on membrane computing has multi-sets of objects encapsulated in membranes. The objects and the membranes evolve according to some rules and the membranes function as compartments to ensure that specific objects only evolve according to certain rules locally within certain membranes. Fig. 2 shows an example of a simple membrane computing system. There are four membranes labeled 1 to 4. Objects `ab` are encapsulated in membrane 4 and there are two local rules in the membrane. Rule `b->bc` can transform `ab` to `abc`. Thereafter, `b->bc` or `c->aδ` can transform `abc`. If more than one rule is applicable, then one rule is selected randomly. If `b->bc` is selected, `abc` will be transformed to `abcc`. If `c->aδ` is selected, `abc` will be transformed to `aba` and membrane 4 will be dissolved. `aba` is then considered to be in membrane 3. If membrane 4 is dissolved, the rules in membrane 4 will no longer be used and the rules in membrane 3 will be used instead. The symbol δ is a directive to dissolve a membrane. Transformation in the system continues until no further transformation can happen. When that happens, the system is halted.

Membrane computing has been used successfully to solve various computation problems such as the satisfiability problem [23] and robot path planning [24]. Tools such as P-Lingua [25], [26] have been developed to facilitate simulation of membrane systems.

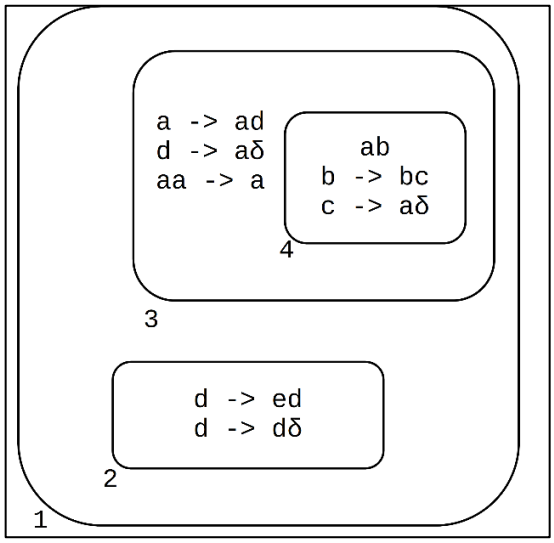

Fig. 2: A simple membrane computing system.

Cellular automata [27] consist of a lattice of interconnected finite state machines called cells, a set of allowable states and a transition function. The alteration of cell states occurs synchronously governed by a local transition function and the states of neighboring cells. Artificial chemistry can be thought of as cellular automata where the cells can move around [9]. Cellular automata have been used successfully in solving problems in nature and computer science such as simulating mangroves rehabilitation [28], robot path planning [29] and edge detection [30].

Evolutionary computation [31], [32] is a computing method inspired by the reproduction, mutation and crossover of cell DNAs. The chromosomes consisting of genes of a DNA are used to encode fitness parameters. The fitness parameters are evaluated using a fitness function in which an evolutionary computing method will attempt to find a global minimum or a global maximum.

Starting with an initial population of DNAs, the search is conducted until a target is found or a certain preset number of generations have been explored.

2.3. *Comments*

From the literature reviewed above, it can be seen that there are two different approaches in the current research of artificial chemistry and artificial cells. One is to model chemistry and cells for analysis and prediction, and another is to compute using concepts in chemistry and cells. Research work related to the first approach is Synthon, Biodrive, ECELL, tile based AChem, string based AChem and the Organic Builder. Research work related to the second approach is cellular automata, evolutionary computation and membrane computing. aCell proposed in this research work represents an attempt fill in the gap between these two approaches by bringing them closer.

The description below explains why the Organic Builder from the first approach and membrane computing from the second approach are chosen as two starting points of aCell and highlights how aCell differs from existing research work.

Among the current research, the Organic Builder shows potential for simulating cell processes using molecules explicitly. Therefore, its approach is adopted in aCell while mathematical simulation which is used in BioDrive and E-CELL is not adopted. aCell extends the approach of the Organic Builder further by simulating artificial molecules and cells in a 3-D space instead of a 2-D space. The synthon approach is too detailed for aCell because aCell needs to strike a balance between serving the needs of AChem and computation. String based AChem does not support detailed simulation of molecule movement from one location to another in a simulation space. Although tile based AChem supports that, the way the Organic Builder simulates movement is still closer to what aCell intends to achieve.

The main difference between aCell and E-Cell is the focus on the simulation of individual molecules and the use of uniform timescale. Although this is not be efficient enough for the simulation of a large number of molecules, it is useful for investigating every detail at the molecular level. This level of investigation is still needed when there are still gaps in our understanding of biological cells.

Following the research reported in [19], aCell uses the concept of morphogenesis and cell reproduction. However, the shape formed by cells in aCell is a simple one following just the shape of the simulation space. The concept of food is used by aCell, but food input is carried to cells by an input flow of molecules instead of being placed in advance in a simulation space. As cell death encourages more renewal and exploration, aCell uses an age limit to simulate programmed cell death.

aCell uses the concept of membrane found in membrane computing to enhance the efficiency of computation and to compartmentalize certain chemical reactions. However, it only supports one membrane per cell and does not support a hierarchy of membranes per cell as in membrane computing. As the concepts of rewriting rule of membrane computing and the Organic Builder are closely related, aCell uses the features of their rewriting rules with some modifications to specify chemical reactions.

As membrane computing and the Organic Builder are two directly related work to aCell, a detailed feature list is presented in Table 1 to show the features adopted by aCell. How aCell implements the features will be described in the following two sections of this paper.

Table 1: Features adopted by aCell.

	Features	Membrane Computing	The Organic Builder	aCell
	AChem			
1	Molecules	√	√	√
2	Molecular bonds		√	
3	Coordinates of molecules and cells		√	√
4	An explicit flow of molecules into and out of the simulation space	√		√
5	3D simulation space			√
6	Reactions triggered by collisions or proximity		√	√
7	Endothermic and exothermic reactions	√		√
8	Rewriting rules for reactions	√	√	√
	Artificial Cell			
9	Cell mutation	√	√	√
10	Aging of cells	√		√

11	Changes to membrane permeability based on the energy levels of cells	√		√
12	Single layer of cell membrane	√	√	√
13	Multiple layers of cell membrane	√		
14	Cell DNA	√	√	√
15	Cell reproduction	√	√	√

3. THE BUILDING BLOCKS OF ACELL

The aCell model has two major parts: the building blocks and the processes of the model. The building blocks will be described first in the following subsections followed by the last subsection which explains the reasons for the design decisions made. The next section will describe the processes.

There are four building blocks in the model. They are the definition of a molecule or an atom, the definition of a cell, the reaction among a group of molecules and the simulation space.

3.1. *The Definition of a Molecule or an Atom*

A sphere with a radius of one unit length is used to represent a molecule or an atom. The radius is a constant regardless of the actual size of the molecule represented. A molecule in this model can have one to n atoms.

Each molecule has an ID, a name and a state number. The ID is a unique identifier in the system and the name can be a chemical name (H_2O), a common name (water) or a generic name $(A, B, C, \ldots)$. The state number accounts for the shape of the molecule and the electrical charges held by the molecule. The range of the state number is from zero to the maximum positive integer value supported by the programming language used to implement the model.

Fig. 3(a) shows a simulation space of aCell with ten molecules and four cells. In this example, the molecules are colored green, and the cells are colored brown. The colors can be set to any colors as needed. The simulation space is 40-unit length at each dimension.

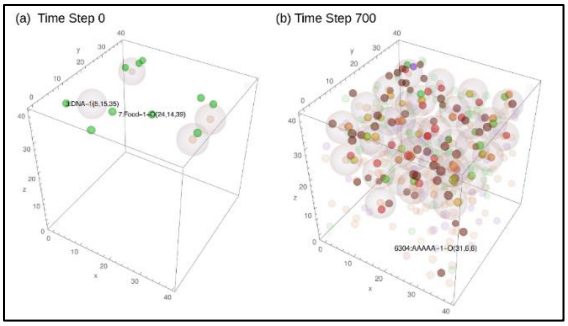

Fig. 3: An example of aCell simulation space at time step 0 and time step 700.

Each molecule in Fig. 3(a) has an ID number, a name and a state number. However, to avoid clutter, only one molecule is shown with its description label. At the location (24, 14, 39), there is a molecule with the ID number 7. Its name is Food, and it has a state number of 1. The character '`-`' is just a text separator to separate the name from the state number. For visualization only, a character of either '`I`' or '`O`' is displayed on the right side of the state number to indicate whether the molecule is "Inside" or "Outside" the membrane of a cell. It is not part of the name of a molecule. The coordinates of the molecule shown are also for the purpose of visualization only. Fig. 3(b) shows how the simulation space can look like at time step 700 after new molecules are inserted and reactions have occurred.

3.2. *The Definition of a Cell*

A sphere with a radius of one unit length and with the name `DNA` is a special molecule used to represent the DNA of a cell. It is always stationary and is positioned at the center of a cell and enclosed within a cell membrane. The name of the DNA molecule can be changed to other more descriptive names if necessary. The distance from a DNA molecule to its membrane is fixed at 3-unit length. The value 3 is an arbitrarily set number and it can be changed to a different distance if needed. It is set to be the same as the distance required for molecules to be within close enough proximity for a reaction to occur. This enables all molecules within a cell to be close enough for them to react among themselves. More details about reaction and close proximity will be explained in the next section. In Fig. 3(a), there is a DNA molecule with the ID number 3 and a state number of 1 at location (5, 15, 35).

The center of each cell, which has a `DNA` molecule, is placed 10-unit length apart from the `DNA` molecules of its neighboring cells. This rule of placement is followed

when a new cell is reproduced. This predefined distance for the placement of a new cell prevents the cells in a simulation space from being packed too tightly together and blocking molecules from moving between two cells. The inter-cell distance can be changed depending on the needs of a simulation.

The membrane of a cell resists movement of molecules into and out of a cell. This resistance allows a cell to accumulate inside its membrane the molecules it needs, such as food molecules. It also prevents molecules which a cell does not need, such as waste molecules from entering. This mechanism of impediment will be described further in the next section together with the description of the movement of a molecule.

For each type of molecules used in a simulation, two membrane resistance values need to be specified. One is when the molecule is entering a cell and another is when the molecule is exiting a cell. The value can be an integer value between 0 and 100 where 0 means no resistance and 100 means total resistance. When a molecule is going to move though a membrane, a random integer between 1 to 100 is generated to assess whether the move succeeds. If the integer is greater than the resistance value, the move succeeds, and fails if it is otherwise.

3.3. *Reactions Among Molecules*

A reaction in the aCell model is specified with a statement using the following syntax.

`ID =` n
`Energy =` e
$x_1 + x_2 + .. + x_m \text{ -> } y_1 + y_2 + .. + y_n$

n is a positive integer used to specify the unique identifier of a reaction. e is the amount of energy needed for the reaction to occur or the amount of energy released by the reaction. A positive energy value indicates the amount of energy added to the simulation space (for an exothermic reaction) while a negative value indicates the amount of energy subtracted from the simulation space (for an endothermic reaction). x_1, x_2, …, x_m and y_1, y_2, …, y_m each represents a molecule or an atom. Some examples are shown in Table 2.

There are only two operators in a reaction statement. The operator '+' indicates that the reactants (on the left-hand side) are all in close proximity among themselves and the products (on the right-hand side) are all in close proximity among themselves. Close proximity is defined as within a distance of 3-unit length apart. The operator '->' indicates the direction of the reaction. When the energy requirement is fulfilled and the reactants are in close proximity, the reactants are transformed into the products which are also placed in close proximity. The character '-' is not an operator. It is just a text separator.

Table 2: Two examples of the reaction syntax.

Reaction	$H_2O \rightleftharpoons H^+ + OH^-$
Syntax	ID = 1 Energy = 1 H2O-1 -> H-2 + OH-3. ID = 2 Energy = -1 H-2 + OH-3 -> H2O-1.
Reaction	$C_6H_{12}O_6 + 6O_6 \rightarrow 6CO_2 + 6H_2O$
Syntax	ID = 1 Energy = 5 C6H12O6-1 + O2-1 + O2-1 + O2-1 + O2-1 + O2-1 + O2-1 -> CO2-1 + CO2-1 + CO2-1 + CO2-1 + CO2-1 + CO2-1 + H2O-1 + H2O-1 + H2O-1 + H2O-1 + H2O-1 + H2O-1.

Although the amount of energy consumed or released by a reaction should ideally be mapped appropriately to reality, it is currently set arbitrarily. A correct mapping scheme is considered as future research work to be done together with biochemists. Similarly, the state number of a molecule is also set arbitrarily.

For a reaction to occur, sufficient local energy and the presence of all the molecules on the left-hand side in close proximity are needed. Based on the total amount of global energy E, specified before a simulation run, the amount of local energy at each coordinate is calculated as E/V where V is the volume of the simulation space.

3.4. *The Simulation Space*

The length of each dimension of the simulation space can be preset to any value and needs not be the same for all dimensions. The six sides of the simulation space can each have a wall or nothing. The top side and the bottom side are usually left open to allow the flow of molecules into and out from the simulation space as shown in Fig. 4. Without a wall, if a molecule moves beyond a

boundary of a side, the molecule is removed from the simulation. If there is a wall, the molecule stops moving and stays in place.

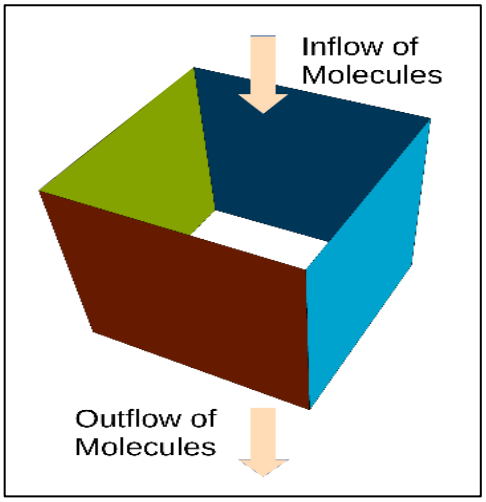

Fig. 4: A simulation space walled up at four sides.

3.5. *Simplifications Adopted by aCell*

By keeping the size of every molecule and atom the same in aCell, aCell sacrifices the modeling of physical dimensions. This means whether it is a big molecule or a small one, it will still fit into the same spot in a simulation space resulting in inaccuracies in the location and volume of molecules. However, this design decision has its advantage. The movement of molecules will be much easier to simulate because rotation of molecules need not be taken into the consideration.

It is common in nature for a large molecule to have reactions occurring simultaneously with various other molecules at different parts of the large molecule. With the reaction rule of aCell, simultaneous reactions can still be simulated regardless of whether size is taken into consideration, although the actual physical locations where the reactions occur simultaneously will be inaccurate. As long as all the molecules reacting with the large molecule are on the left-hand side of a reaction rule, the reaction will be processed as a reaction occurring at the same time.

Furthermore, when there is insufficient or imprecise knowledge in biochemistry, abstract molecules will often be used in aCell. In this case, the correct size of an abstract molecule will be inaccurate anyway.

Proximity, not collision, is used to trigger a reaction in aCell. Therefore, mass, velocity and momentum are not modeled in aCell. As what will be described in the next section about movement, every molecule and atom in aCell only moves to a location next to the original location in one time step of a simulation run. This further simplifies the simulation of movement in aCell. Similarly, the reason for cells in aCell to be stationary is also for the sake of simplicity in simulation. The mechanism of cell movement is complex, and it sometimes needs inter-cell co-ordination which is even more complex.

Reaction distance directly affects the number of possible reactions which can occur. This is because the longer the distance, the larger the number of molecules which can participate in a reaction. The distance is currently arbitrarily set at 3. The distance from a DNA molecule to the membrane surrounding it is also set at 3. Setting both to the same value enables the membrane to act as a capsule to cause reactions involving the DNA molecule to be more likely to occur. This technique is also used in membrane computing to facilitate computation.

In nature, the reaction distance may vary depending on the types of reactions and the types of cells. Thus, support for these complex features is omitted in aCell at this stage. They are considered as possible future extensions to aCell when the implications of different reaction distances, different cell sizes and different cell types are further studied. At the current stage, the size of each cell is uniform in aCell and only one type of cells is supported in a simulation run.

The aCell model aims to strike a balance among three factors; closeness to reality, fast simulation speed and high flexibility to model the nature of a cell. By explicitly modeling each molecule in aCell instead of modeling collectively using differential equations, each molecule can be individually specified, and the actions of each molecule can be individually tracked. This provides aCell the high flexibility needed in future work to model each element of a cell using only molecules. To counter the slowness in explicit simulation, a highly realistic representation used in the synthon model is not used and the physical aspects are simplified. The current aCell model simulates molecules and cells at a higher abstract level than that of the synthon model.

4. The Processes of aCell

In this section, the eight major processes of the aCell model are described. They are 1) cell energy update, 2) cell membrane permeability adaptation, 3) cell death, 4) cell reproduction, 5) cell fitness evaluation, 6) input of

new molecules, 7) reactions among molecules and 8) the movement of a molecule.

4.1. *Overview of the Processes*

Before a simulation run of aCell, the characteristics of each type of molecules and the reactions have to be specified. In our implementation of an aCell simulator, they are specified in two text files, `MoleculeWorld.txt` and `Reactions.txt`. The color, the name, the state number and the membrane resistance values of each type of molecules are specified in `MoleculeWorld.txt`. The reactions are specified in `Reactions.txt`. Details about the format of the text files can be found in (Goh et al., 2019).

Fig. 5 shows an overview of the aCell model components and how they interact. Except for the files `MoleculeWorld.txt` and `Reactions.txt`, all the files in Fig. 5 specify or log the states of a simulation at each time step. `MoleculeInput.csv` specifies the types of molecules and the number of molecules to inject into the simulation space at each time step. The rest of the files shown on the right side of the simulator are log files, which are the output of the simulator.

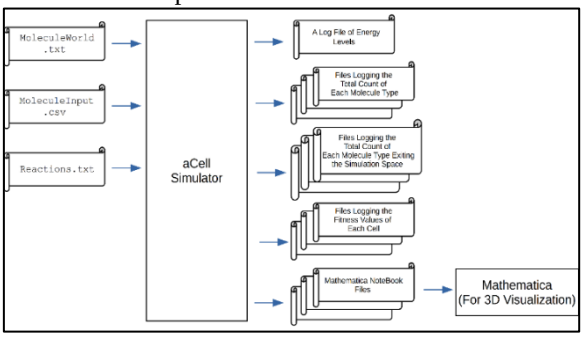

Fig. 5: An overview of the model components and how they interact.

A simulation run of aCell starts with either an empty simulation space or a simulation space with some initial cells. When cells are not involved in a simulation run, for example in a run to simulate reversible chemical reactions, the simulation run starts with an empty space. When cells are needed, for example for solving an optimization problem where the chromosomes consisting of genes of a DNA are needed to encode fitness parameters of the optimization problem, some initial cells are inserted at the beginning. The number of initial cells can be specified according to the needs of a simulation run.

aCell supports direct insertion of new molecules at specified or random locations in a simulation space at each time step of a simulation run. However, new molecules are normally inserted from the top of the simulation space at random locations.

When a molecule moves, it can move to randomly one of its neighboring locations or stays put at each time step. The probability to move downwards is designed to be greater so that a flow of molecules moving into and out of the simulation space can be simulated. This is because cells which are stationary in nature often rely on flow currents to bring nutrients to them. If the target location where a molecule is going to move to is already occupied, the molecule stays in place.

All cells are stationary in aCell. If sufficient resources such as food are provided, the cells can reproduce. Normally a simulation run involving cells will let the simulation space be filled up with cells first before inserting molecules for an intended experiment.

To simulate chemical reactions, every molecule is evaluated at each time step to see if it is involved in a reaction. The selection of molecules for evaluation is random. Furthermore, if a molecule can be involved in more than one reaction, one reaction is randomly decided. DNA molecules are treated in the same way as other molecules for chemical reactions. After a reaction, the reacting molecules are eliminated, and the resulting molecules are produced. If a molecule does not fulfill the conditions of any reactions, the molecule is moved as described above.

The reproduction process, the aging process and the death of a cell are not simulated as chemical reactions in aCell yet because their reaction pathways are complex and have not been fully understood yet. Thus, they are simulated at high abstract levels and the use of programming constructs is necessary.

For a cell to reproduce, the cell must reach a certain age and must have a certain minimum amount of energy. Age in aCell is calculated based on the number of metabolism reactions which has occurred in a cell from its introduction to the current point in time. Mutation occurs in the chromosomes of a new cell of the next generation during reproduction. The degree of mutation is dependent on the difference between the current fitness value of a parent cell and the optimum fitness value. The bigger the difference, the more aggressive the mutation.

The evaluation of the fitness value of a cell is also implemented at a high abstract level using programming constructs instead of chemical reactions. This is because

it is not known yet how a number system exists in nature with molecules and how a comparison of two numbers can be made using chemical reactions.

A cell will die when it ages beyond a certain age limit, has less than 0 amount of energy or has more energy than a certain overheat energy level. The age limit and the overheat level are parameters which can be specified. When a cell dies, the cell and its DNA molecule are removed from the simulation space.

aCell currently has a basic membrane permeability adaptation scheme. When the energy level in a cell is more than half the cell overheat level, the membrane of the cell will prevent food molecules from entering the cell, thus preventing metabolism reactions from occurring. This is meant to ensure that the cell does not kill itself by overheating.

4.2. *Details of the Processes*

The main procedure of aCell is shown in Fig. 6. The details of the processes will be explained according to the sequence in the main procedure to show how the processes come together in a simulation loop.

```
1: Input: Molecule definitions as in MoleculeWorld.txt, reactions as in
   Reactions.txt, molecule input as in MoleculeInput.csv and simulation
   parameters (shown in capital letters)
2: Output: Log files and a Mathematica notebook file for each time step
3: procedure MAIN
4:     Initialize parameters and set up a new simulation run;
5:     timeStep ← 0;
6:     while timeStep ≤ MAX_TIME do
7:         Deduct energy from each cell;
8:         Process adaptation of cell membrane permeability to food molecules;
9:         Process cell death;
10:        Process cell reproduction;
11:        Calculate the fitness value of each cell and update the best-value variable;
12:        Generate and insert new molecules into the simulation space (new molecules
   are inserted into aftetrList);
13:        while there are molecules in beforeList do
14:            Select randomly a molecule m from beforeList and delete m from be-
   foreList;
15:            Process a reaction m participates in if there is one;
16:            if a reaction related to m did not occur then
17:                Move molecule m;
18:                Insert m into afterList;
19:            end if
20:        end while
21:        Write a Mathematica notebook file for visualization for the current timeStep;
22:        Update molecule log files;
23:        Update the energy log file
24:        Move every molecule in afterList from afterList to beforeList;
25:        Increment timeStep;
26:    end while
27: end procedure
```

Fig. 6: The main procedure of aCell.

- Cell Energy Update, Cell Membrane Permeability Adaptation, Cell Death and Cell Reproduction

In the main simulation loop, the first cell process updates the energy level of every cell by deducting a predefined amount of energy from each cell. Thereafter, the membrane permeability of every cell towards `Food-1` molecules is updated. `Food-1` molecules are molecules which can release energy when they react with the DNA molecule of a cell in a metabolic reaction. If the energy of a cell is greater than or equal to half of `CELL_OVERHEAT_LEVEL`, its membrane resistance to `Food-1` entering is set to 100. Otherwise, the resistance is set to 0.

There are various theories on how aging occurs in nature. In aCell, a metabolism counter is used to determine age. Whenever a metabolic reaction occurs in a cell, the metabolism counter and thus the age of the cell is incremented by one. When a cell dies, the cell and its DNA molecule are removed from the simulation space. However, other molecules contained within the cell membrane remain. The conditions for cell death are shown in Fig. 7.

```
1: if ( the metabolism counter in a cell > AGE_LIMIT ) ∨ ( the energy in a cell ) < 0 )
   ∨ ( energy in a cell > CELL_OVERHEAT_LEVEL ) then
2:     the cell dies;
3: end if
```

Fig. 7: The conditions of cell death.

```
1: if ( the metabolism counter in a cell > MATURITY_AGE ) ∧ ( energy in a cell ≥
   MINIMUM_ENERGY_FOR_REPRODUCTION ) ∧ ( there is an empty location next to
   the cell) then
2:     deduct INITIAL_REPRODUCED_CELL_ENERGY from the cell;
3:     if fitnessDifference of the cell ≤ FITNESS_DIFF_EXCELLENT then
4:         mutationStep ← MUTATION_STEP_EXCELLENT;
5:     else if ( fitnessDifference of the cell > FITNESS_DIFF_EXCELLENT ) ∧ (
   fitnessDifference of the cell ≤ FITNESS_DIFF_NORMAL ) then
6:         mutationStep ← MUTATION_STEP_NORMAL;
7:     else if fitnessDifference of the cell > FITNESS_DIFF_NORMAL then
8:         mutationStep ← MUTATION_STEP_BAD;
9:     end if
10:    generate a new cell;
11:    the energy in the new cell ← INITIAL_REPRODUCED_CELL_ENERGY;
12:    for each gene in the new cell do
13:        the value of the gene ← value of its parent's gene + a random value between
   -mutationStep and mutationStep;
14:        if the value of the gene > MAXIMUM_FITNESS_PARAMETER then
15:            the value of the gene ← MAXIMUM_FITNESS_PARAMETER;
16:        end if
17:        if the value of the gene < MINIMUM_FITNESS_PARAMETER then
18:            the value of the gene ← MINIMUM_FITNESS_PARAMETER;
19:        end if
20:    end for
21: end if
```

Fig. 8: The pseudo-code of cell reproduction.

The pseudo-code of cell reproduction is shown in Fig. 8. When a cell is above a predefined maturity age and contains energy above a predefined threshold value, it duplicates itself to produce a new cell nearby if there is an empty spot next to the cell for the new cell to come into existence. The DNA of a cell is a stationary molecule which can react with other molecules, and it also contains an array of genes. A gene is currently represented by a

floating-point number. It can easily be changed to any suitable data type depending on the purpose of a cell simulation. The reproduction process transfers energy from a parent cell to a new cell. The amount of energy transferred is equivalent to the amount set in `INITIAL_REPRODUCED_CELL_ENERGY`.

In order to determine the value of each gene in the new cell, the difference between the fitness value of the parent cell and the parameter `OPTIMUM_FITNESS` is calculated, and the difference is used to determine the level of mutation to introduce to each gene of the new cell. The bigger the difference, the bigger the mutation step. For each gene in the new cell, the gene value is set to the gene value of the parent cell plus a random value between the negative value of the mutation step and the positive value of the mutation step.

- **Cell Fitness Evaluation**

The value of each gene of a cell is a fitness parameter. The fitness of a cell is calculated by fitting the value of each gene into a fitness function specified for a simulation run. From the results of fitness evaluation of every cell, the cell with the best fitness is determined and logged at each time step. The fitness value and the fitness parameters of each cell are also logged.

- **Input of New Molecules**

New molecules are generated according to a specification file. In our implementation, it is named `MoleculeInput.csv`. The file specifies the number of new molecules, the types of molecules to be inserted and the locations of insertion for each time step of a simulation run. The location of insertion is normally set to random locations at the top of the simulation space. Details about the format of `MoleculeInput.csv` and how the input can be specified can be found in [33].

The new molecules are regarded as having moved at the time step when they are introduced and therefore are stored in a list called `afterList`. `afterList` is a linked list used internally by the simulator of aCell. It is introduced here to provide context to the description of processes related to molecules. A molecule which has moved or reacted with other molecules is put into `afterList` while a molecule who has not, remains in another linked list called `beforeList`.

- **Reactions Among Molecules**

Every molecule in `beforeList` is processed at every time step. Molecules in `beforeList` are shuffled randomly first before selection for processing begins to ensure that the order of molecule selection from `beforeList` is random.

A molecule can only either participate in a reaction once or move once per time step. While there are still molecules in `beforeList`, a molecule is selected and removed from `beforeList` and processed as described below. We shall designate the selected molecule as `m` for the purpose of the description.

After `m` is selected from `beforeList`, it is checked to see whether it can react with other molecules. Every molecule within close proximity from `m` is considered and every possible reaction is considered. Only one out of the possible reactions is selected randomly to occur. All the reacting molecules of that reaction are removed from `beforeList` and the new molecules which are the results of the reaction are inserted into `afterList`. A molecule cannot react with another molecule through a cell membrane. If the search fails to find a possible reaction or if there are insufficient number of empty locations within close proximity from the location of `m` for the resulting molecules to be placed, then no reaction will occur.

If the matching reaction is a metabolism reaction, which can only occur in a cell, 50% of the energy released from the reaction is added to global energy and the other 50% is added to the cell where `m` is in. Thereafter, the age which is the metabolism count of the cell is incremented by one.

- **The Movement of a Molecule**

If a reaction involving `m` does not occur, `m` will move. To process movement, whether `m` is within a cell membrane or outside needs to be determined first. If it is within a cell membrane, the destination of its movement is determined without a downward flow. If it is not, it will be determined with a downward flow. If a cell in the natural world is stationary, it often positions itself or lives in a spot where there is a stream of molecules flowing through it in order for it to capture nutrients. aCell simulates this behavior by simulating a tendency of a molecule to move downward.

When a molecule in a location (x, y, z) moves, it can move randomly up to a location $(x_1, y_1, z + 1)$, sideways to a location (x_1, y_1, z) or downward to a location $(x_1, y_1, z - 1)$ where $x_1 \in \{x - 1, x, x + 1\}$ and $y_1 \in \{y - 1, y, y + 1\}$. Including the current location of the molecule, there are 27 possible neighboring locations for the molecule to move to.

To simulate a downward flow, the nine possible locations on top of the molecule `m` are designated by nine numbers, 1 to 9. The nine possible locations at the same level are designated by 10 to 18 and each of the nine possible locations below are designated by two numbers from 20 to 36. By generating a random integer between 1 to 36 to determine the destination location, the probability of `m` moving to a location downward is 2/ 36 and the probability of it moving to a location sideways or upward is 1/ 36.

When a downward flow is not needed, the 27 possible destination locations are designated by 27 numbers from 1 to 27. A random number between 1 to 27 is generated to determine the destination location. After determining the destination location, the movement mode of `m` is considered. There are four modes of movement. They are 1) movement from space to space, 2) movement from space to cell, 3) movement from cell to space and 4) movement from one cell to another cell.

The first mode of movement does not need to consider the membrane resistance against `m`. The other three modes have to consider the membrane resistance. Each molecule has two membrane resistance values, one against it when it attempts to enter a cell and another against it when it attempts to exit a cell. The resistance values are specified for each molecule type in the `MoleculeWorld.txt` specification file in our implementation. To overcome the resistance, a random number between 1 to 100 is generated and that number must be greater than the resistance value.

If `m` moves from space to cell, it will need to overcome the membrane resistance which prevents it from entering a cell. If `m` moves from cell to space, it will need to overcome the membrane resistance which prevents it from exiting a cell. If `m` moves from one cell to another cell, it will need to overcome both the membrane resistance which prevents it from exiting the first cell and the membrane resistance which prevents it from entering the second cell.

If `m` fails to overcome a resistance value, it will remain in place. If the destination location is already occupied by another molecule, `m` will also remain in place. The six sides of the simulation space can each have a wall or nothing. If `m` moves beyond a side without a wall, `m` will be removed from the simulation. If there is a wall, `m` will stop moving and remains in place.

At the end, if `m` can successfully move, it will be considered as acted and will be placed in `afterList`.

- **Other Supporting Processes by the Simulator**

After processing every molecule in `beforeList`, a display text file of the current time step is generated to enable the visualization of molecules and cells in the simulation space. The text file contains commands of Mathematica to display the molecules and the cells in 3-D as shown in Fig. 3. The 3-D visualization facility of Mathematica allows panning, rotating and zooming operations.

At the end of a time step, the total count of each type of molecules in the simulation space and the total count of each type of molecules that have exited the simulation space are logged. The final amount of global energy at that time step is also logged. Before moving to the next time step, all molecules in `afterList` are moved back to `beforeList` and the time step counter is incremented by one.

5. Experiments and the Results

To test the capabilities of aCell, seven experiments were conducted using aCell to simulate biochemical reactions (1 and 2), perform fundamental computing operations (3 to 5) and solve optimization problems (6 and 7). They were

1. Formation of larger structures from small molecules
2. Reversible chemical reactions
3. Simulation of an AND gate
4. Simulation of a NOT gate
5. Simulation of a 1-bit memory element
6. Searching for the global minimum of the Restriping function when n = 2. The global minimum is $f(0,0) = 0$, the search domain was $-5.12 \leq x_i \leq 5.12$ and $A = 10$.
7. Searching for the global minimum of the Rosenbrock function when n = 2. The global minimum is $f(1,1) = 0$ and the search domain was $-5 \leq x_i \leq 5$.

By using the same framework throughout the experiments, the experiments attempted to demonstrate that aCell is a model suitable to serve as the groundwork for the study of the relationship between biochemical reactions in cells and computation using cells in the future.

There are four key aspects of chemistry essential to cellular processes [4]. They are 1) covalent and noncovalent interactions among molecules, 2) small

molecules serving as building blocks for larger structures, 3) reversible chemical reactions and 4) reactions which can store and release energy. The first aspect is already supported by the nature of the rewriting rules used by aCell. The fourth aspect is also supported by allowing each reaction in aCell to be specified as either endothermic or exothermic. Thus, the second aspect and the third aspect were examined with experiments 1 and 2 respectively.

The basic building blocks of a computing system are the AND gate, the OR gate and the NOT gate. Experiments 3 and 4 were meant to verify that aCell can simulate them. As a simulation of an AND gate is similar to the simulation of an OR gate, that experiment and its result is not described in this paper. Experiment 5 to show that aCell can simulate a 1-bit memory element was meant to demonstrate that a complex digital circuit can be simulated using aCell reactions without following the usual way of connecting logic gates together.

Experiments 6 and 7 were experiments to use aCell to solve two typical benchmark fitness functions normally used in evolutionary computing.

5.1. *Common Setup for the Experiments*

A typical simulation run of aCell consists of four steps.

1) Specification of the details of the simulation space, molecules, cells, reactions, the input of new molecules into the simulation space and the number of time steps to run.
2) Running the simulation where reactions occur, and cells reproduce and die.
3) Termination of the simulation when the specified number of time steps to run is reached.
4) The log files are studied to obtain the results of the run.

In the first step, there are two types of specifications. One is the specification in text files, namely, `MoleculeWorld.txt`, `Reactions.txt` and `MoleculeInput.csv`. Another is the setting up of parameter values in the aCell simulator.

`Reactions.txt` of each experiment is shown together with the result in each figure from Fig. 9 to Fig. 14. `MoleculeWorld.txt` and `MoleculeInput.csv` files are not shown in this paper due to the limitation of space. Reading the files are not needed to examine the results because the charts show the input molecules and the output molecules of the experiments. Nevertheless, the files are available for download for the details at `https://github.com/clgoh3221/Specifications`.

aCell parameters were set to values as shown in Table 3. They were set to reasonable arbitrary values to provide a fixed baseline environment. The input of molecules specified in `MoleculeInput.csv` was then calibrated to suit the baseline and the requirements of the experiments.

Table 3: Values of aCell parameters.

Parameter	Value
LENGTH_OF_X_DIMENSION	40
LENGTH_OF_Y_DIMENSION	40
LENGTH_OF_Z_DIMENSION	40
NUMBER_OF_INITIAL_CELLS	4
SPACE_BETWEEN_TWO_NUCLEI	10
INITIAL_CELL_ENERGY	100.0
ENERGY_USED_BY_A_CELL_PER_TIME_STEP	1.0
MINIMUM_ENERGY_FOR_REPRODUCTION	150.0
INITIAL_REPRODUCED_CELL_ENERGY	100.0
CELL_OVERHEAT_LEVEL	500.0
AGE_LIMIT	10
MATURITY_AGE	1
GLOBAL_ENERGY	64100
NUMBER_OF_GENES_IN_A_CELL	2

For the experiments, the reaction distance was set to 3-unit length and a simulation space of 40-unit length x 40-unit length x 40-unit length was used. The space could contain 4 x 4 x 4 artificial cells. The energy released by a metabolic reaction in a cell was fixed at an arbitrary 150 units.

5.2. *Formation of Larger Structures from Small Molecules*

In experiment 1 (Fig. 9), the goal is to form `AAAAA-1` molecules from `A-1` molecules. The figure shows the average number of molecules by type detected in the simulation space of ten simulation runs with ten different random seeds.

Reactions 1 and 2 were reactions of cell metabolism. Reaction 1 represented a reaction which occurred when a food molecule was detected. It produced a `P-1` molecule which could react with a food molecule to release energy as in reaction 2. Reaction 2 was specified as a metabolic

reaction and made known to the aCell simulator. Every time reaction 2 occurred in a cell, the age counter of the cell was incremented by one. The presence of a `DNA-1` molecule in a reaction was to ensure that the reaction only occurred inside a cell. Reaction 3 was used to get rid of excess of `P-1` molecules in a cell.

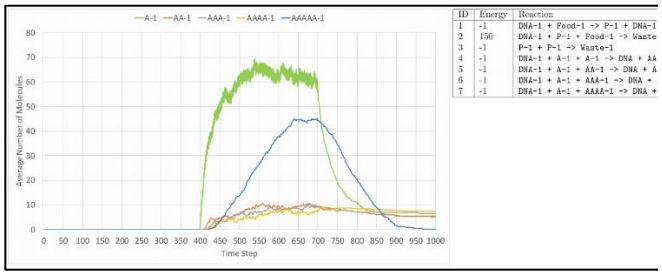

Fig. 9: Formation of larger structures.

Reactions 4 to 7 were the reactions which formed larger molecules in a cell. `A-1` represented the smallest molecule type and subsequently larger molecules type were represented by `AA-1`, `AAA-1`, `AAAA-1` and `AAAAA-1`. The permeability of the cell membrane of each cell was set to trap molecules of type `A-1` to type `AAAA-1` within to facilitate the formation of molecule type `AAAAA-1`. This trapping of molecules enabled an efficient process to form larger molecules, similar to how cell membranes are used in nature.

The experiment started at time step 0 with four initial cells. Food molecules were inserted throughout the experiment to enable the cells to reproduce and to sustain themselves. From time step 401 to time step 700 `A-1`, molecules were injected. `AAAAA-1` molecules started to form inside the simulation space at time step 450 showing the formation of larger structures from small molecules.

5.3. *Reversible Chemical Reactions*

In experiment 2 (Fig. 10), molecule types `A-1` to `D-1` were used to represent the general form of a simple reversible reaction. Artificial cells were not used in this experiment because reversible reactions can occur without cells.

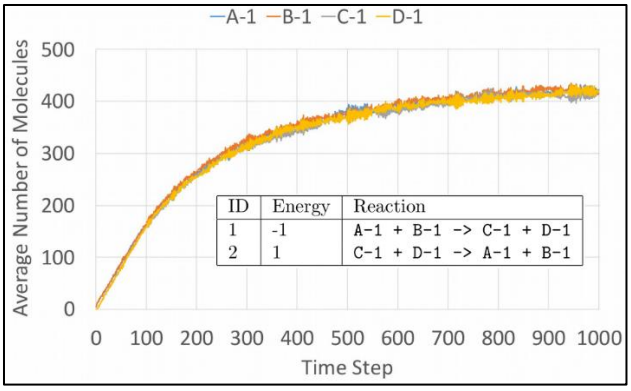

Fig. 10: Simulation of a reversible chemical reaction.

Throughout the experiment, molecules of type `A-1` and `B-1` were injected at each time step. Fig. 10 shows the average number of molecules by type detected in the simulation space of ten simulation runs with ten different random seeds. The result shows that the reactants and the products could reach an equilibrium and a reversible reaction can be simulated in aCell.

5.4. *Simulation of an AND Gate*

The result of experiment 3 is shown in Fig.11. Reactions 1 to 3 performed the same functions as reactions 1 to 3 in experiment 1. Reactions 4 to 7 defined the AND operation. `InATrue-1` and `InAFalse-1` molecules were used to represent the two possible states of one input line labelled A and `InBTrue-1` and `InBFalse-1` molecules were used to represent the two possible states of another input line labelled B. `OutTrue-1` and `OutFalse-1` molecules were used to represent the two possible states of the output of the AND gate.

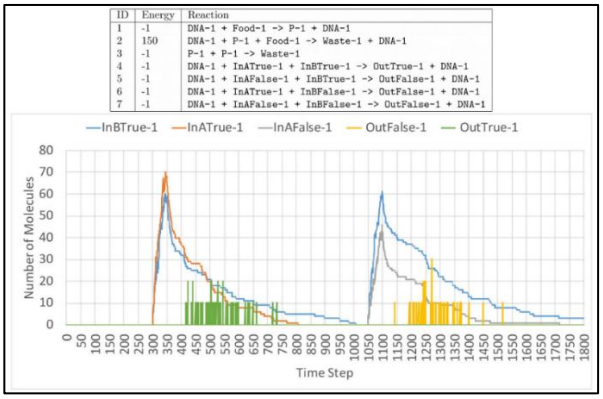

Fig. 11: Simulation of an AND gate.

Four initial cells were set up in the experiment at time step 0 and Food-1 molecules were injected at every time step of the experiment. After giving the initial cells sufficient amount of time to reproduce, from time step 300 to time step 349, InATrue-1 and InBTrue-1 molecules were injected. From time step 1050 to time step 1099, InAFalse-1 and InBTrue-1 molecules were injected, changing the input to the simulated AND gate. The time gap between the two molecule injections was needed for the molecules from the first injection to exit from the simulation space completely before the second injection began. The duration of the gap was determined by adjusting the gap iteratively based on simulation results.

The number of `InATrue-1`, `InAFalse-1` and `InBTrue-1` molecules shown was the number of molecules which existed in the simulation space at each time step. However, the number of `OutTrue-1` and `OutFalse-1`+molecules shown was the number of molecules exiting the simulation space at the bottom of the simulation space. The exiting molecules were counted this way because an AND gate is only useful when an output is detected outside the gate.

In the experiment, the number of `OutTrue-1` molecules and `OutFalse-1` molecules which exited the AND gate was small in the range of 1 to 3. To indicate clearer the number of `OutTrue-1` molecules and `OutFalse-1` molecules in Fig. 11, the number had been multiplied by 10. In other words, every ten `OutTrue-1` or `OutFalse-1` molecules shown in Fig. 11 was just one molecule.

There were not any injections of `InBFalse-1` molecules because showing that reactions 4 and 5 worked and showing that reactions 6 and 7 worked is redundant. The result shows that aCell is able to simulate an AND gate.

A slight variation to reactions 4 to 7 of this experiment can be used to show that aCell can simulate an OR gate. Therefore, that experiment and its result is not described in this paper.

5.5. *Simulation of a NOT Gate*

Fig. 12 shows the result of experiment 4. It shows that aCell is able to simulate a NOT gate. The setup of this experiment was similar to that of experiment 3.

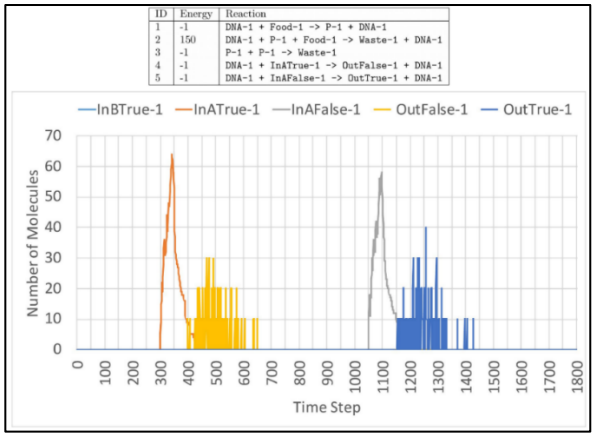

Fig. 12: Simulation of a NOT gate.

5.6. *Simulation of a 1-bit Memory Element*

Although in theory, a memory element can be constructed by devising a way to link together the simulated logic gates above, this experiment aimed to show that there is another way to do so by mapping the necessary operations related to memory to reactions that can be processed by aCell. The role of each molecule type in the reactions is described in Table 4. The result of experiment 5 is shown in Fig. 13.

Table 4: Molecule types for the experiment to simulate a 1-bit memory element

Type	Description
InitMemory-1	For the creation of StoreInit-1 in a cell
StoreInit-1	A storage element to store a state in a cell
WriteFalse-1	For writing a false state into StoreInit-1
WriteTrue-1	For writing a true state into StoreInit-1
StoreFalse-1	A true state stored in a cell
StoreTrue-1	A false state stored in a cell
Read-1	For reading a state from a cell
MemoryOutFalse-1	Output indicating a true state is stored a cell
MemoryOutTrue-1	Output indicating a false state is stored a cell
Destroy-1	For destroying the memory element in a cell

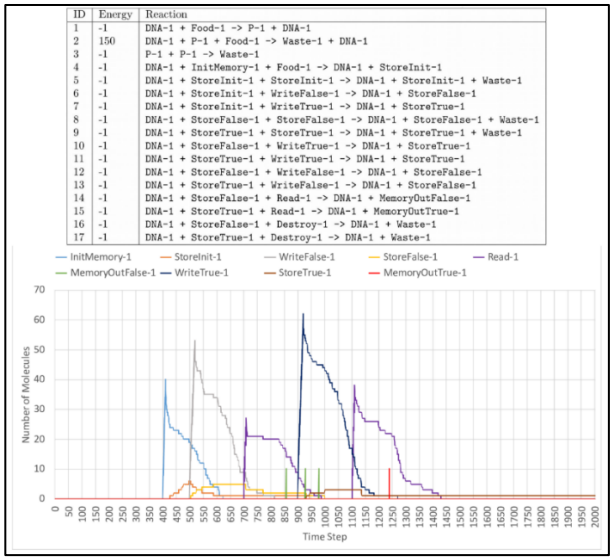

Fig. 13: Simulation of a 1-bit memory element.

Reactions 1 to 3 were similar to the reactions used before for cell metabolism. Reaction 4 initialized a cell so that it contained a molecule which could store a state. The `InitMemory-1` molecule served as a control molecule which could be used to replenish a cell with a `StoreInit-1` molecule. Stopping the injection of `InitMemory-1` molecules would stop cells from storing memory states. Reaction 5 ensured that each cell would only store one state. Reactions 6 and 7 were used to write a memory state into a cell. Reactions 8 to 13 were used to support over-writing of a memory state. Reactions 14 and 15 were used to read a memory state from a cell and reactions 16 and 17 were used to destroy a memory molecule in a cell.

From the result, it can be seen that from time step 400 to time step 650, `StoreInit-1` molecules were generated after the injection of `InitMemory-1` molecules. Then `StoreInit-1` molecules were converted to `StoreFalse-1` molecules correctly by a write operation.

Thereafter, a read operation was started at time step 700, and it worked correctly because `MemoryOutFalse-1` molecules were detected at the exit of the simulation space from time step 850 to time step 1000.

A write operation was initiated at time step 900 to change the memory state to true. When it was followed by a read operation at time step 1100, one `MemoryOutTrue-1` molecule was detected at the exit between time step 1200 and time step 1250.

For the same reasons explained before, as the number of `MemoryOutTrue-1` molecules and `MemoryOutFalse-1` molecules was small, it was multiplied by ten to enable visualization in the chart.

5.7. *Searching of the Global Maximum and the Global Minimum of Fitness Objective Functions*

The results of experiments 6 and 7 are shown in Fig. 14 and Table 5. For each experiment, the aCell simulation was run ten times with ten different random seeds. GALGO (`https://github.com/aasivas/GALGO`) was used as a baseline for comparisons with aCell to show how aCell performed relatively. It was compiled and run with its default parameters and compilation options described in its README file.

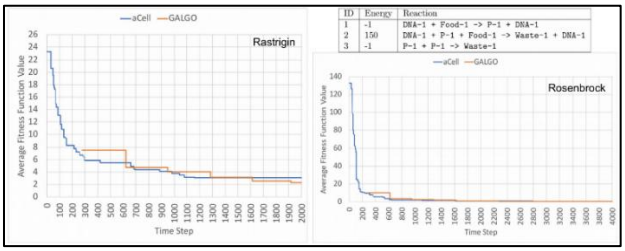

Fig. 14: Finding the global minimum of the Rastrigin function and the Rosenbrock function.

Table 5: Average final fitness values of aCell and GALGO (better results are in bold).

Function	Method	Average Fitness	Standard Deviation
Rastrigin (n = 2)	aCell	3.07985	1.94053
	GALGO	**2.29416**	1.43225
Rosenbrock (n = 2)	aCell	0.50082	0.70268
	GALGO	**0.34752**	0.45222

Only three reactions were needed for the experiments for maintaining cells. Apart from the food molecules which were needed by the cells, no other molecules were injected throughout the experiments. The cells only used mutation to search for the optimum fitness parameters.

To guide the mutation process, parameters in Table 6 were used. These parameters were determined after a series of values had been evaluated. The results show that in the most basic form using only mutation without using any selection methods and cross-over operations, aCell

can solve simple optimization problems comparable to GALGO.

Table 6: Mutation ranges used according to fitness differences.

Parameter	Experiment		
	1	2	3
OPTIMUM_FITNESS	50.00	0.00	0.00
MINIMUM_FITNESS_PARAMETER	-100.00	-5.12	-5.00
MAXIMUM_FITNESS_PARAMETER	100.00	5.12	5.00
FITNESS_DIFF_EXCELLENT	5.00	0.33	50.00
FITNESS_DIFF_NORMAL	10.00	1.00	100.00
MUTATION_STEP_EXCELLENT	5.00	0.33	0.05
MUTATION_STEP_NORMAL	10.00	1.00	0.20
MUTATION_STEP_BAD	20.00	3.00	2.00

DISCUSSION

It is hoped aCell can be used by researchers to learn computing methods from AChem and cells. It is built for future modifications and simplicity using a high-level abstraction. Many of its features are parameterized for flexibility and easy modification.

We have conducted fundamental tests to evaluate its abilities to simulate and compute. aCell can readily support many more reaction rules and many more cells in larger simulation space. However, the areas of cell chemistry to explore this is still yet to be determined because guidance from the angle of cell biology is needed.

At its current form, aCell can simulate at a high abstract level chemical reactions and basic cell processes. In addition, it can also simulate the fundamental building blocks of computation and can be used to solve simple optimization problems. All of these can be accomplished using the same method of defining molecules, reactions, cells and molecule input in the structure provided by the aCell model.

From the perspective of AChem and cell simulation, aCell can model simple to complex molecules of different configurations. It can model unlimited number of biochemical reactions, subjected only to the limitations of the underlying data structures used to implement the simulator. By chaining biochemical reactions together, complex biochemical reaction pathways can be simulated.

5.8. *Future Work*

In the area of AChem, the reactions and the molecules involved in cell processes in nature should be studied further so that they can translated to the formats usable in aCell. This will reduce the use of programming constructs and narrow the gap between biochemical processes in nature and the processes of computation. For example, current cell processes in aCell such as reproduction, mutation and the sensing of the fitness of a cell can be studied further to discover ways to specify them with only molecules and reactions in aCell. In addition, new cell processes such as genetic crossover, cell movement and inter cell communication can be introduced so that aCell can simulate biological cells better and use the new processes to mimic computation that occurs in nature. Ideally, every cell process in aCell should be specified with only reactions and molecules.

In the area of computation, although aCell can simulate the fundamental building blocks of computation, it still needs to be extended to connect the building blocks together. For example, gate connectors made up of artificial cells are needed. Circuit efficiency still needs to be measured and improved if aCell is extended and scaled up to simulate complex network of logic gates to compute.

Looking at both AChem and computation, there are two possible areas for exploration. Firstly, a quantifying system based on reaction rules in nature for quantifying the number of molecules in an area. Secondly, a comparison system based on reaction rules in nature for comparing two quantities. These two areas will lead to the understanding of the number system working in nature, which is directly related to computation.

6. CONCLUSION

This study proposes aCell, a flexible artificial cell model that uses artificial chemistry to explore information processing in natural cells and bio-inspired computation. The model demonstrates capabilities in forming large molecules, modeling reversible reactions, implementing logic operations, storing state, and solving optimization problems. aCell provides a unified platform for simulating both essential cell biology and computation using the same artificial chemistry constructs. The model can be extended in future work by incorporating more realistic biochemistry and scaling up the complexity. The study contributes an accessible

modeling framework for investigating synergies between biology and computation.

7. Acknowledgment

This research was supported by the Malaysian Ministry of Higher Education (MoHE) through the Fundamental Research Grant Scheme (FRGS/1/2014/ICT02/MMU/02/3).

References

[1] L. N. Castro, "Fundamentals of natural computing: an overview," Physics of Life Reviews, vol. 4, pp. 1-36, 2007.

[2] C. Langton, "Artificial life," Artificial Life, pp. 1-47, 1989.

[3] L. Kari and G. Rozenberg, "The many facets of natural computing," Communications of the ACM, vol. 51, pp. 72-83, 2008.

[4] H. Lodish, A. Berk, C. A. Kaiser, M. Krieger, A. Bretscher, H. Ploegh, K. C. Martin, M. Yaffe, and A. Amon, Molecular Cell Biology, 6th ed., W. H. Freeman and Company, 2007.

[5] H. Suzuki and P. Dittrich, "Artificial chemistry," Artificial Life, 15, pp. 1-3, 2009.

[6] T. J. Hutton, "The organic builder: a public experiment in artificial chemistries and self-replication," Artificial Life, vol. 15, pp. 21-28, 2009.

[7] G. Păun, "From cells to computers: computing with membranes (P systems)," Biosystems, vol. 59, pp. 139-158, 2007.

[8] B. Song, K. Li, D. Orellana-Martín, M. J. Pérez-Jiménez, and I. Pérez-Hurtado, "A survey of natured-inspired computing: membrane computing," ACM Computing Surveys, vol. 54, pp. 1-31, 2021.

[9] T. J. Hutton, "Evolvable self-reproducing cells in a two-dimensional artificial chemistry," Artificial Life, vol. 13, pp. 11-30, 2007.

[10] R. Sienkiewicz and W. Jędruch, "DigiHive: artificial chemistry environment for modeling of self-organization phenomena," Artificial Life, vol. 29, pp. 235-260, 2023.

[11] T. Lenaerts and R. Bersini, "A synthon approach to artificial chemistry," Artificial Life, vol. 15, pp. 89-103, 2009.

[12] K. Kyoda, M. Muraki, and H. Kitano, "Construction of a generalized simulator for multi-cellular organisms and its application to SMAD signal transduction," in Pacific Symposium on Biocomputing, pp. 314-325, 2000.

[13] T. Yamamoto and K. Kaneko, "Tile automaton in the well-mixed medium," Physica D: Nonlinear Phenomena, vol. 181, pp. 252–273, 2003.

[14] K.Tominaga, Y. Suzuki, K. Kobayashi, K. Watanabe, T. Koizumi, and K. Kishi, "Modeling biochemical pathways using an artificial chemistry," Artificial Life, vol. 15, pp. 115-129, 2009.

[15] T. Watanabe, K. Koizumi, K. Kishi, M. Nakamura, K. Kobayashi, M. Kazuno, Y. Suzuki, Y. Asada, and K. Tominaga, "A uniform framework of molecular interaction for an artificial chemistry with compartments," in IEEE Symposium on Artificial Life, pp. 54-60, 2007.

[16] K. Takahashi, K. Yugi, K. Hashimoto, Y. Yamada, C. Pickett, and M. Tomita, "Computational challenges in cell simulation: a software engineering approach," IEEE Intelligent Systems, vol. 17, pp. 64-71, 2002.

[17] K. Hashimoto, S. Seno, P. Dhar, and M. Tomita, "Integrative modeling of gene expression and metabolism with E-CELL system," Artificial Life and Robotics, vol. 6, pp. 99-107, 2002.

[18] H. Ohno, Y. Naito, H. Nakajima, and M. Tomita, "Construction of a biological tissue model based on a single-cell model: a computer simulation of metabolic heterogeneity in the liver lobule," Artificial Life, vol. 14, pp. 3-28, 2008.

[19] S. Cussat-Blanc, H. Luga, and Y. Duthen, "From single cell to simple creature morphology and metabolism," in Artificial Life XI, pp. 134-141, 2008.

[20] N. Djezzar, N. Djedi, S. Cussat-Blanc, H. Luga, and Y. Duthen, "L-systems and artificial chemistry to develop digital organisms," in IEEE Symposium on Artificial Life, pp. 225-232, 2011.

[21] T. Schmickl, M. Stefanec, and K. Cralisheim "How a life-like system emerges from a simplistic particle motion law," Scientific Reports, vol 5, 37969, 2016.

[22] M. Stefanec and T. Schmickl, "PPS3D: a 3D variant of the primordial particle system," In ALIFE 2022, pp. 28-30, 2022.

[23] Y. Jimen and A. Fujiwara, "An asynchronous P system using branch and bound for the satisfiability problem," in the Fifth International Symposium on Computing and Networking (CANDAR), pp. 141-152, 2017.

[24] I. Pérez-Hurtado, M. J. Pérez-Jiménez, G. Zhang, and D. Orellana-Martín, "Robot path planning using rapidly-exploring random trees: a membrane computing approach," in the 7th International Conference on Computers Communications and Control (ICCC), pp. 37-46, 2018.
[25] I. Pérez-Hurtado, D. Orellana-Martín, G. Zhang, and M. J. Pérez-Jiménez, "P-Lingua in two steps: flexibility and efficiency," Journal of Membrane Computing, vol.1, pp. 93–102, 2019.
[26] I. Pérez-Hurtado, D. Orellana-Martín, M. A. Martínez-del-Amor, L. Valencia-Cabrera, and A. Riscos-Núñez, "A new P-Lingua toolkit for agile development in membrane computing," Information Sciences, vol 587, pp. 1-22, 2022.
[27] J. Schiff, Cellular Automata: A Discrete View Of The World, Wiley, 2011.
[28] H. Huynh, T. Dang, O. Mỹ Linh, H. Hoang Luong, N. Duong Trung, T. Phan, and B. Pottier, "Simulating mangroves rehabilitation with cellular automata," in the 4th International Conference on Machine Learning and Soft Computing, pp. 40-45, 2020.
[29] H. Pei, Y. Lou, and Y. Feng, "Robot path planning based on cellular automata with mixed neighborhoods," in the 11th International Symposium on Computational Intelligence and Design (ISCID), pp. 114-117, 2018.
[30] A. Enescu, A. Andreica, and L. Diosan, "Evolved cellular automata for edge detection", in Genetic and Evolutionary Computation Conference Companion, pp. 316-317, 2019.
[31] D. E. Goldberg, Genetic Algorithms In Search, Optimization, And Machine Learning, Addison-Wesley, 2002.
[32] K. A. De Jong, Evolutionary Computation: A Unified Approach, MIT Press, 2006.
[33] C. Goh, H. Ewe, and Y. Goh, "An artificial cell simulator based on artificial chemistry," in the 8th International Conference on Software and Computer Applications, pp. 338-342, 2019.