molecules

MDPI

Article

Small Universal Bacteria and Plasmid Computing Systems

Xun Wang [1], Pan Zheng [2], Tongmao Ma [1] and Tao Song [1,3,*]

[1] College of Computer and Communication Engineering, China University of Petroleum, Qingdao 266580, China; wangsyun@upc.edu.cn (X.W.); matongmao@163.com (T.M.)
[2] Department of Accounting and Information Systems, University of Canterbury, Christchurch 8041, New Zealand; panzheng@ieee.org
[3] Departamento de Inteligencia Artificial, Universidad Politcnica de Madrid (UPM), Campus de Montegancedo, 28660 Boadilla del Monte, Spain
* Correspondence: tsong@upc.edu.cn or t.song@upm.es; Tel.: +86-150-532-587-69

Received: 25 April 2018; Accepted: 21 May 2018; Published: 29 May 2018

Abstract: Bacterial computing is a known candidate in natural computing, the aim being to construct "bacterial computers" for solving complex problems. In this paper, a new kind of bacterial computing system, named the bacteria and plasmid computing system (BP system), is proposed. We investigate the computational power of BP systems with finite numbers of bacteria and plasmids. Specifically, it is obtained in a constructive way that a BP system with 2 bacteria and 34 plasmids is Turing universal. The results provide a theoretical cornerstone to construct powerful bacterial computers and demonstrate a concept of paradigms using a "reasonable" number of bacteria and plasmids for such devices.

Keywords: bacterial computing; bacteria and plasmid system; Turing universality; recursively enumerable function

1. Introduction

In cell biology, bacteria, despite their simplicity, contain a well-developed cell structure that is responsible for some of their unique biological structures and pathogenicity. The bacterial DNA resides inside the bacterial cytoplasm, for which transfer of cellular information, transcription, and DNA replication occurs within the same compartment [1,2]. Along with chromosomal DNA, most bacteria also contain small independent pieces of DNA called plasmids, which can be conveniently obtained and released by a bacterium to act as a gene delivery vehicle between bacteria in the form of horizontal gene transfer [3].

Bacterial computing was coined with the purpose of building biological machines, which are developed to solve real-life engineering and science problems [4]. Practically, bacterial computing proves mechanisms and the possibility of using bacteria for solving problems in vivo. If an individual bacterium can perform computation work as a computer, this envisions a way to build millions of computers in vivo. These "computers", combined together, can perform complicated computing tasks with efficient communication via plasmids. Using such conjugation, DNA molecules, acting as information carriers, can be transmitted from one cell to another. On the basis of the communication, information in one bacteria can be moved to another and can be used for further information processing [5,6].

Bacterial computing models belong to the field of bio-computing models, such as DNA computing models [7–9] and membrane computing models [10–12]. Because of the computational intelligence and parallel information processing strategy in biological systems, most of the bio-computing models have been proven to have the desired computational power. Most of these can do what a Turing machine

can do (see, e.g., [13–19]). The proposed bacterial computing models can provide powerful computing models at the theoretical level but a lack of practical results. Current bacterial computing models are designed for solving certain specific biological applications, such as bacteria signal pathway detecting, but give no result for computing power analysis.

In general bacterial computing models, information to be processed is encoded by DNA sequences, and conjugation is the tool for communicating among bacteria. The biological process is shown in Figure 1.

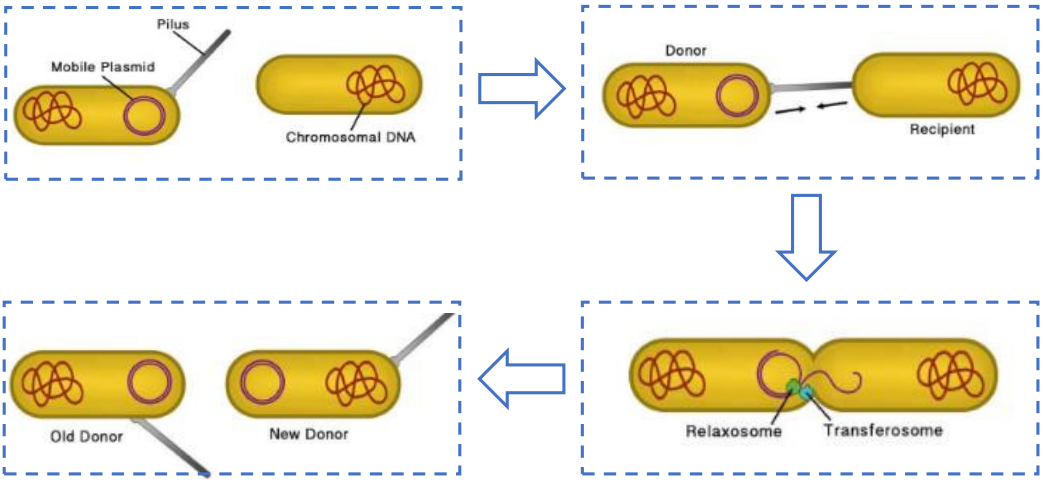

Figure 1. Bacteria conjugation from biological point of view.

Looking for small universal computing devices, such as small universal Turing machines [20,21], small universal register machines [22], small universal cellular automata [23], small universal circular Post machines [24], and so on, is a natural and well-investigated topic in computer science. Recently, this topic started to be considered also in the framework of bio-computing models [25–31].

In this work, we focus on designing small universal bacteria and plasmid computing systems (BP systems); that is, we construct Turing universal BP systems with finite numbers of bacteria and plasmids. Specifically, we demonstrate that a BP system with 2 bacteria and 34 plasmids is universal for computing recursively enumerable functions and families of sets of natural numbers. In the universality proofs, 2 bacteria are sufficient, as in [32], but the numbers of plasmids needed are reduced to about 10 from a possible infinite number. The results provide a theoretical cornerstone to construct powerful "bacterial computers" and demonstrate a concept of paradigms using a "reasonable" number of bacteria and plasmids for these devices.

2. The Bacteria and Plasmid System

In this work, as for automata in automata theory, the BP system is formally designed and defined. In general, the system is composed of three main components:

- a set of bacteria;
- a set of plasmids;
- a set of evolution rules in each bacterium, including conjugation rules and gene-editing (inserting/deleting) rules.

The evolution rules are in the form of productions in formal language theory, which are used to process and communicate information among bacteria. Such a system is proven to be powerful for a number of computing devices; that is, they can compute the sets of natural numbers that are Turing

computable. However, in the universality proof, the number of plasmids involved is not limited. It is possible to use an infinite number of plasmids for information processing and exchanging. Such a feature is acceptable (as for the infinite tape in Turing machines) in mathematic theory but is not feasible with the biological facts.

A BP system of degree m is a construct of the following form:

$$\Pi = (O, b_1, b_2, \ldots, b_m, P, b_{out}), \text{where the following are true.}$$

- $O = \{g_1, g_2, \ldots, g_n\}$ is a set of genes in the chromosomal DNA of bacteria.
- $P = P_{crispr} \cup P_{temp} \cup \{p_{null}\}$ is a set of plasmids.
  - Plasmids in P_{crispr} are of the form $(cas9, gRNA_{g_i}^{\alpha})$ with $\alpha \in \{insert, delete\}$, which is used for cutting specific genes.
  - Plasmids in P_{temp} are of the form $(gRNA_{g_i}^{template})$, which takes templates of genes to be inserted.
  - Plasmid p_{null} is of the form (Pro_{Rap}^{Rel}) for bacteria conjugation.
- Variables $b_1, b_2, \ldots, b_m$ are m bacteria of the form $b_i = (w_i, R_i)$, where
  - w_i is a set of genes over O initially placed in bacterium b_i;
  - R_i is a set of rules in bacterium b_i of the following forms:
    (1) **Conjugation rule** is of the form $(\text{ATP-}P_c, b_i/b_j, \text{ATP-}P_c')$, by which ATP in bacterium b_i is consumed and a set of plasmids $P_c' \subseteq P$ associated with ATP is transmitted into bacterium b_j.
    (2) **CRISPR/Cas9 gene inserting rule** is of the form $p_i p_{s_i} \times (g_j, g_k)$, where $p_i \in P_{crispr}$, $\alpha = insert$, $p_{s_i} \in P_{temp}$, and g_j and g_k are two neighboring genes. The insertion is operated if and only if g_j and g_k are neighboring genes and plasmids $p_i p_{s_i}$ are present in the bacterium.
    (3) **CRISPR/Cas9 gene deleting rule** is of the form $p_i \times (g_j, g_k)$ with $p_i p_{null} \in P_{crispr}$, $\alpha = detele$, and g_j and g_k being two neighboring genes. The rule can be used if and only there exists gene g_i placed between the two neighboring genes.
- Variable b_{out} is the output bacterium.

It is possible to have more than one enabled conjugation rule at a certain moment in a bacterium, but only one is non-deterministically chosen for use. This is due to the biological fact that ATP can support the transmission of one plasmid but not all of the plasmids. If a bacterium has more than one CRISPR/Cas9 operating rule associated with a certain common plasmid, only one of the rules is non-deterministically chosen for use; if the enabled CRISPR/Cas9 operating rules are associated with different plasmids, all of them will be used to edit the related genes.

The configuration of the system is described by chromosomal DNA encoding the information in each bacterium. Thus, the initial configuration is $\langle (w_1, w_2, \ldots, w_m) \rangle$. Using the conjugation and CRISPR/Cas9 rules defined above, we can define the transitions among configurations. Any sequence of transitions starting from the initial configuration is called a computation. A computation is called successful if it reaches a halting configuration, that is, no rule can be used in any bacterium. The computational result is encoded by the chromosomal DNA in bacterium b_{out} when the system halts, where $b_{out} \in \{b_1, b_2, \ldots, b_m\}$ denotes the output bacterium. There are several ways to encode numbers by the chromosomal DNA. We use the number of genes in the chromosomal DNA to encode different numbers computed by the system.

The set of numbers computed by system Π is denoted by $N(\Pi)$. We denote by $NBP(bact_j, plas_k)$ the family of sets of numbers computed/generated by BP systems with m bacteria and k plasmids (if no limit is imposed on the values of parameters m and k, then the notation is replaced by $*$).

We need an input bacterium to receive genetic signals in the form of short DNA segments from the environment or certain bacteria, as well as an output bacteria, with which the system can compute functions. The input bacterium is denoted by b_{in} with $b_{in} \in \{b_1, b_2, \ldots, b_m\}$. Input bacterium b_{in} can read/receive information from the environment, where information is encoded by DNA segments or a string of genes. When a BP system has both input and output bacteria, it starts by reading/receiving information from the environment through input bacterium b_{in}. After reading the input information, the system starts its computation by using the conjugation and CRISPR/Cas9 gene inserting/deleting rules; it then finally halts. The computational result is stored in the output bacterium b_{out} encoded by a number of certain genes.

Mathematically, if the input information is x, which is encoded by DNA segments composed of x genes, when the system halts, bacterium b_{out} holds y genes. It is said that the BP system can compute the function $f(x) = y$. In general, if the inputs are $x_1, x_2, \ldots, x_n$ in the form of DNA strands containing x_i copies of gene g_i with $i = 1, 2, \ldots, n$, when the system halts, we obtain the computational result y in the output bacterium in the form of y copies of genes. The system is said to compute the function $f(x_1, x_2, \ldots, x_n) = y$.

3. Universality Results

In this section, we construct two small universal BP systems. Specifically, we construct a Turing universal BP system with 2 bacteria and 34 plasmids to compute recursively enumerable functions. As a natural-number computing device, a universal BP system with 2 bacteria and 34 plasmids is achieved.

In the following universality proofs, the notion of a register machine is used. A register machine is a construct of the form $M = (m, H, l_0, l_h, R)$, where m is the number of registers, H is the set of instruction labels, l_0 is the start label, l_h is the halt label (assigned to instruction HALT), and R is the set of instructions; each label from H labels only one instruction from R, thus precisely identifying it. The instructions are of the following forms:

- $l_i : (\text{ADD}(r), l_j, l_k)$ (add 1 to register r and then go to one of the instructions with labels l_j and l_k);
- $l_i : (\text{SUB}(r), l_j, l_k)$ (if register r is non-zero, then subtract 1 from it, and go to the instruction with label l_j; otherwise, go to the instruction with label l_k);
- l_h : HALT (the halt instruction).

A register machine M generates a set $N(M)$ of numbers in the following way: it starts with all registers being empty (i.e., storing the number zero) and then applies the instruction with label l_0; it continues to apply instructions as indicated by the labels (and made possible by the contents of registers). If the register machine finally reaches the halt instruction, then the number n present in specified register 0 at that time is said to be generated by M. If the computation does not halt, then no number is generated. It is known (e.g., see [33]) that register machines generate all sets of numbers that are Turing computable.

A register machine can also compute functions. In [22], register machines are proposed for computing functions, with the universality defined as follows: Let $\varphi_x(y)$ be a fixed admissible enumeration of the unary partial recursive functions. A register machine M is said to be universal if there is a recursive function g such that for all natural numbers x and y, it holds $\varphi_x(y) = M(g(x), y)$; that is, with input $g(x)$ and y introduced in registers 1 and 2, the result $\varphi_x(y)$ is obtained in register 0 when M halts.

A specific universal register machine M_u shown in Figure 2 is used here, which was modified by a universal register machine from [22]. Specifically, the universal register machine from [22] contains a separate check for zero of register 6 of the form l_8 : (SUB(6), l_0, l_{10}); this instruction was replaced in M_u by l_8 : (SUB(6), l_9, l_0), l_9 :(ADD(6), l_{10}) (see Figure 2). Therefore, in the modified universal register machine, there are 8 registers (numbered from 0 to 7) and 23 instructions (hence 23 labels),

the last instruction being the halting instruction. The input numbers are introduced in registers 1 and 2, and the result is obtained in register 0.

l_0 : (SUB(1), l_1, l_2), l_1 : (ADD(7), l_0),
l_2 : (ADD(6), l_3), l_3 : (SUB(5), l_2, l_4),
l_4 : (SUB(6), l_5, l_3), l_5 : (ADD(5), l_6),
l_6 : (SUB(7), l_7, l_8), l_7 : (ADD(1), l_4),
l_8 : (SUB(6), l_9, l_0), l_9 : (ADD(6), l_{10}),
l_{10} : (SUB(4), l_0, l_{11}), l_{11} : (SUB(5), l_{12}, l_{13}),
l_{12} : (SUB(5), l_{14}, l_{15}), l_{13} : (SUB(2), l_{18}, l_{19}),
l_{14} : (SUB(5), l_{16}, l_{17}), l_{15} : (SUB(3), l_{18}, l_{20}),
l_{16} : (ADD(4), l_{11}), l_{17} : (ADD(2), l_{21}),
l_{18}: (SUB(4), l_0, l_h), l_{19} : (SUB(0), l_0, l_{18}),
l_{20} : (ADD(0), l_0), l_{21} : (ADD(3), l_{18}),
l_h : HALT

Figure 2. The universal register machine for computing Turing-computable functions [22].

3.1. A Small Universal BP System as Function Computing Device

Theorem 1. *There exists a Turing universal BP system with 2 bacteria and 34 plasmids that can compute Turing-computable recursively enumerable functions.*

Proof. To this aim, we construct a BP system Π with 2 bacteria and 34 plasmids to simulate the register machine M_u shown in Figure 2. The system Π is of the following form:

$$\Pi = (O, b_1, b_2, P, b_{in}, b_{out}), \text{where the following are true.}$$

- $O = \{g_0, g_1, \ldots, g_7, g_m\}$ is set of genes in chromosomal DNA of bacteria.
- $P = P_{crispr} \cup P_{temp} \cup \{p_{null}\}$ is a set of plasmids shown in Table 1, where
  - $P_{crispr} = \{p_1, p_2, \ldots, p_{22}, p_h\}$, whose elements associated with the labels of instructions are used for gene cutting;
  - $P_{temp} = \{p_{s_i} \mid i = 1, 2, 5, 7, 9, 16, 17, 20, 21\}$ are plasmids taking templates of genes to be inserted, which are used for simulating ADD instructions;
  - plasmid p_{null} for bacteria conjugation is used for simulating SUB instructions.
- $b_1 = (w_1, R_1)$, where $w_1 = \lambda$, meaning no initial chromosomal DNA is placed in bacteria b_1; the set of rules R_1 is shown in Table 2.
- $b_2 = (w_2, R_2)$, where $w_2 = g_0 g_m g_1 g_m g_2 g_m g_3 g_m g_4 g_m g_5 g_m g_6 g_m g_7 g_m$, indicating the initially placed chromosomal DNA in bacterium b_2; the set of rules R_2 is shown in Table 2.
- $b_{in} = b_{out} = b_2$, which means bacterium b_2 can read signals from the environment, and when the system halts, the computational result is stored in bacterium b_2.

In general, for each add instruction l_i acting on register $r \in \{0, 1, 2, 3, 4, 5, 6, 7\}$, plasmids $p_i = (cas9, gRNA_{g_r}^{insert})$ and $p_{s_i} = (gRNA_{g_r}^{template})$ are associated; for any SUB instruction l_i acting on register $r \in \{0, 1, 2, 3, 4, 5, 6, 7\}$, a plasmid $p_i = (cas9, gRNA_{g_r}^{delete})$ is associated in system Π. The numbers stored in register r are encoded by the number of copies of gene g_r with $r \in \{0, 1, 2, 3, 4, 5, 6, 7\}$ in chromosomal DNA of bacterium b_2. Specifically, if the number stored in register r is $n \geq 0$, then bacterium b_2 contains $n + 1$ copies of gene g_r.

During the simulation of register machine M_u by system Π, when bacterium b_1 holds a pair of plasmids $p_i p_{s_i}$ (respectively $p_i p_{null}$) and ATP, the system starts to simulate an ADD instruction

(respectively a SUB instruction) l_i of M_u: plasmids $p_i p_{s_i}$ (respectively $p_i p_{null}$) are transmitted to bacterium b_2 by the conjugation rule; then one copy of gene g_r between neighboring genes g_r and g_m is inserted (respectively deleted) to simulate increasing (respectively decreasing) the number in register r by 1; after this, bacterium b_2 sends ATP and plasmids $p_j p_{null}$ to bacterium b_1 if the proceeding instruction l_j is a SUB instruction or plasmids $p_j p_{s_j}$ if the proceeding l_j is an ADD instruction.

Table 1. Plasmids in system Π.

Plasmid	Forms of Plasmids	Plasmid	Forms of Plasmids
p_0	$p_0 = (cas9, gRNA_{g_1}^{delete})$	p_{16}	$p_{16} = (cas9, gRNA_{g_4}^{insert})$
p_1	$p_1 = (cas9, gRNA_{g_7}^{insert})$	p_{17}	$p_{17} = (cas9, gRNA_{g_2}^{insert})$
p_2	$p_2 = (cas9, gRNA_{g_6}^{insert})$	p_{18}	$p_{18} = (cas9, gRNA_{g_4}^{delete})$
p_3	$p_3 = (cas9, gRNA_{g_5}^{delete})$	p_{19}	$p_{19} = (cas9, gRNA_{g_3}^{delete})$
p_4	$p_4 = (cas9, gRNA_{g_6}^{delete})$	p_{20}	$p_{20} = (cas9, gRNA_{g_0}^{insert})$
p_5	$p_5 = (cas9, gRNA_{g_5}^{insert})$	p_{21}	$p_{21} = (cas9, gRNA_{g_3}^{insert})$
p_6	$p_6 = (cas9, gRNA_{g_7}^{delete})$	p_h	$p_h = (cas9, gRNA_{g_h}^{delete})$
p_7	$p_7 = (cas9, gRNA_{g_1}^{insert})$	p_{s_1}	$p_{s_1} = (gRNA_{g_7}^{template})$
p_8	$p_8 = (cas9, gRNA_{g_6}^{delete})$	p_{s_2}	$p_{s_2} = (gRNA_{g_6}^{template})$
p_9	$p_9 = (cas9, gRNA_{g_6}^{insert})$	p_{s_5}	$p_{s_5} = (gRNA_{g_5}^{template})$
p_{10}	$p_{10} = (cas9, gRNA_{g_4}^{delete})$	p_{s_7}	$p_{s_7} = (gRNA_{g_1}^{template})$
p_{11}	$p_{11} = (cas9, gRNA_{g_5}^{delete})$	p_{s_9}	$p_{s_9} = (gRNA_{g_6}^{template})$
p_{12}	$p_{12} = (cas9, gRNA_{g_5}^{delete})$	$p_{s_{16}}$	$p_{s_{16}} = (gRNA_{g_4}^{template})$
p_{13}	$p_{13} = (cas9, gRNA_{g_2}^{delete})$	$p_{s_{17}}$	$p_{s_{17}} = (gRNA_{g_2}^{template})$
p_{14}	$p_{14} = (cas9, gRNA_{g_5}^{delete})$	$p_{s_{20}}$	$p_{s_{20}} = (gRNA_{g_0}^{template})$
p_{15}	$p_{15} = (cas9, gRNA_{g_3}^{delete})$	$p_{s_{21}}$	$p_{s_{21}} = (gRNA_{g_3}^{template})$
p_{null}	(Pro_{Rap}^{Rel})	p_{s_h}	$p_{s_h} = (Pro_{Rap}^{Rel})$

Initially, there is no chromosomal DNA initially placed in bacterium b_1, but bacterium b_2 has genes $w_2 = g_0 g_m g_1 g_m g_2 g_m g_3 g_m g_4 g_m g_5 g_m g_6 g_m g_7 g_m$. At the beginning, the system receives $g(x)$ copies of gene g_1 and y copies of gene g_2 from the environment through input bacterium b_2, which simulates the numbers $g(x)$ and y being introduced in registers 1 and 2 for register machine M_u. In this way, the chromosomal DNA of bacterium b_2 becomes

$$g_0 g_m g_1^{g(x)+1} g_m g_2^{y+1} g_m g_3 g_m g_4 g_m g_5 g_m g_6 g_m g_7 g_m.$$

Once completing the reading of information from the environment, a pair of plasmids $p_0 p_{s_0}$ and one unit of ATP is placed in bacterium b_1 to trigger the computation; meanwhile no plasmid or ATP is initially contained in bacterium b_2. The transition of system Π by reading input signals encoded by $g(x)$ copies of genes g_1 and y copies of gene g_2 through input bacterium b_2 is shown in Figure 3.

In what follows, we explain how system Π simulates ADD instructions and SUB instructions and outputs the computational result.

Simulating the ADD instruction: l_i : (DD(r), l_j).

We assume at a certain moment that system Π starts to simulate an ADD instruction l_i of M_u, acting on register $r \in \{0, 1, 2, \ldots, 7\}$. At that moment, bacterium b_1 holds two plasmids $p_i p_{s_i}$ and ATP, such that the conjugation rule $(\text{ATP-}p_i p_{s_i}, b_1/b_2, \text{ATP-}p_i p_{s_i})$ is used. By using the conjugation rule, plasmids $p_i p_{s_i}$ and ATP are transmitted to bacterium b_2. In system Π, plasmids p_i and p_{s_i} are associated with the ADD instruction l_i, where plasmid p_i is of the form $p_i = (cas9, gRNA_{g_r}^{insert})$ for

cutting a certain site of chromosomal DNA, and p_{s_i} is of the form $p_i = (gRNA_{g_r}^{template})$ carrying the gene to be inserted.

In bacterium b_2, the CRISPR/Cas9 inserting rule $p_i \times (g_r, g_m)$ is used to insert gene g_r between neighboring genes g_r and g_m. In this way, the number of gene g_r of bacterium b_2 is increased by 1, which simulates the number in register r being increased by 1. We note that there is a unique position at which gene g_r can be inserted with the context of neighboring g_r and g_m.

Table 2. Rules in each bacterium of system Π.

Sim.	Rules	Bac.
l_0	(ATP-p_0p_{null}, b_1/b_2, ATP-p_0p_{null})	b_1
	$p_0 \times (g_1, g_m)$, (ATP-p_{null}, b_2/b_1, ATP-$p_1p_{s_1}$), (ATP-p_0p_{null}, b_2/b_1, ATP-$p_2p_{s_2}$)	b_2
l_1	(ATP-$p_1p_{s_1}$, b_1/b_2, ATP-$p_1p_{s_1}$)	b_1
	$p_1 \times (g_7, g_m)$, (ATP-p_{s_1}, b_2/b_1, ATP-p_0p_{null})	b_2
l_2	(ATP-$p_2p_{s_2}$, b_1/b_2, ATP-$p_2p_{s_2}$)	b_1
	$p_2 \times (g_6, g_m)$, (ATP-p_{s_2}, b_2/b_1, ATP-p_3p_{null})	b_2
l_3	(ATP-p_3p_{null}, b_1/b_2, ATP-p_3p_{null})	b_1
	$p_3 \times (g_5, g_m)$, (ATP-p_{null}, b_2/b_1, ATP-$p_2p_{s_2}$), (ATP-p_3p_{null}, b_2/b_1, ATP-p_4p_{null})	b_2
l_4	(ATP-p_4p_{null}, b_1/b_2, ATP-p_4p_{null})	b_1
	$p_4 \times (g_6, g_m)$, (ATP-p_{null}, b_2/b_1, ATP-$p_5p_{s_5}$), (ATP-p_4p_{null}, b_2/b_1, ATP-p_3p_{null})	b_2
l_5	(ATP-$p_5p_{s_5}$, b_1/b_2, ATP-$p_5p_{s_5}$)	b_1
	$p_1 \times (g_5, g_m)$, (ATP-p_{s_5}, b_2/b_1, ATP-p_6p_{null})	b_2
l_6	(ATP-p_6p_{null}, b_1/b_2, ATP-p_6p_{null})	b_1
	$p_6 \times (g_7, g_m)$, (ATP-p_{null}, b_2/b_1, ATP-$p_7p_{s_7}$), (ATP-p_6p_{null}, b_2/b_1, ATP-p_8p_{null})	b_2
l_7	(ATP-$p_7p_{s_7}$, b_1/b_2, ATP-$p_7p_{s_7}$)	b_1
	$p_7 \times (g_1, g_m)$, (ATP-p_4p_{null}, b_2/b_1, ATP-p_4p_{null})	b_2
l_8	(ATP-p_8p_{null}, b_1/b_2, ATP-p_8p_{null})	b_1
	$p_8 \times (g_6, g_m)$, (ATP-p_{null}, b_2/b_1, ATP-$p_9p_{s_9}$), (ATP-p_0p_{null}, b_2/b_1, ATP-p_0p_{null})	b_2
l_9	(ATP-$p_9p_{s_9}$, b_1/b_2, ATP-$p_9p_{s_9}$)	b_1
	$p_9 \times (g_6, g_m)$, (ATP-$p_{10}p_{null}$, b_2/b_1, ATP-$p_{10}p_{null}$)	b_2
l_{10}	(ATP-$p_{10}p_{null}$, b_1/b_2, ATP-$p_{10}p_{null}$)	b_1
	$p_{10} \times (g_4, g_m)$, (ATP-p_0p_{null}, b_2/b_1, ATP-p_0p_{null}), (ATP-$p_{10}p_{null}$, b_2/b_1, ATP-$p_{11}p_{null}$)	b_2
l_{11}	(ATP-$p_{10}p_{null}$, b_1/b_2, ATP-$p_{11}p_{null}$)	b_1
	$p_{11} \times (g_5, g_m)$, (ATP-p_{null}, b_2/b_1, ATP-$p_{12}p_{null}$), (ATP-$p_{11}p_{null}$, b_2/b_1, ATP-$p_{13}p_{null}$)	b_2
l_{12}	(ATP-$p_{12}p_{null}$, b_1/b_2, ATP-$p_{12}p_{null}$)	b_1
	$p_{12} \times (g_5, g_m)$, (ATP-p_{null}, b_2/b_1, ATP-$p_{14}p_{null}$), (ATP-$p_{12}p_{null}$, b_2/b_1, ATP-$p_{15}p_{null}$)	b_2
l_{13}	(ATP-$p_{13}p_{null}$, b_1/b_2, ATP-$p_{13}p_{null}$)	b_1
	$p_{13} \times (g_2, g_m)$, (ATP-p_{null}, b_2/b_1, ATP-$p_{18}p_{null}$), (ATP-$p_{13}p_{null}$, b_2/b_1, ATP-$p_{19}p_{null}$)	b_2
l_{14}	(ATP-$p_{14}p_{null}$, b_1/b_2, ATP-$p_{14}p_{null}$)	b_1
	$p_{14} \times (g_5, g_m)$, (ATP-p_{null}, b_2/b_1, ATP-$p_{16}p_{s_{16}}$), (ATP-$p_{14}p_{null}$, b_2/b_1, ATP-$p_{17}p_{s_{17}}$)	b_2
l_{15}	(ATP-$p_{15}p_{null}$, b_1/b_2, ATP-$p_{15}p_{null}$)	b_1
	$p_{15} \times (g_3, g_m)$, (ATP-p_{null}, b_2/b_1, ATP-$p_{18}p_{null}$), (ATP-p_{15}, b_2/b_1, ATP-$p_{20}p_{s_{20}}$)	b_2
l_{16}	(ATP-$p_{16}p_{s_{16}}$, b_1/b_2, ATP-$p_{16}p_{s_{16}}$)	b_1
	$p_{16} \times (g_4, g_m)$, (ATP-$p_{s_{16}}$, b_2/b_1, ATP-$p_{11}p_{null}$)	b_2
l_{17}	(ATP-$p_{17}p_{s_{17}}$, b_1/b_2, ATP-$p_{17}p_{s_{17}}$)	b_1
	$p_{17} \times (g_2, g_m)$, (ATP-$p_{s_{17}}$, b_2/b_1, ATP-$p_{21}p_{s_{21}}$)	b_2
l_{18}	(ATP-$p_{18}p_{null}$, b_1/b_2, ATP-$p_{18}p_{null}$)	b_1
	$p_{18} \times (g_4, g_m)$, (ATP-p_{null}, b_2/b_1, ATP-p_0p_{null}), (ATP-$p_{18}p_{null}$, b_2/b_1, ATP-$p_hp_{s_h}$)	b_2
l_{19}	(ATP-$p_{19}p_{null}$, b_1/b_2, ATP-$p_{19}p_{null}$)	b_1
	$p_{19} \times (g_3, g_m)$, (ATP-p_{null}, b_2/b_1, ATP-p_0p_{null}), (ATP-p_{15}, b_2/b_1, ATP-$p_{18}p_{null}$)	b_2
l_{20}	(ATP-$p_{20}p_{s_{20}}$, b_1/b_2, ATP-$p_{20}p_{s_{20}}$)	b_1
	$p_{20} \times (g_0, g_m)$, (ATP-$p_{s_{20}}$, b_2/b_1, ATP-p_0p_{null})	b_2
l_{21}	(ATP-$p_{21}p_{s_{21}}$, b_1/b_2, ATP-$p_{21}p_{s_{21}}$)	b_1
	$p_0 \times (g_3, g_m)$, (ATP-$p_{s_{21}}$, b_2/b_1, ATP-$p_{18}p_{null}$)	b_2
l_h	(ATP-$p_hp_{s_h}$, b_1/b_2, ATP-$p_hp_{s_h}$)	

By using the CRISPR/Cas9 inserting rule, plasmid p_i is consumed, and plasmid p_{s_i} and ATP remain in bacterium b_2. The conjugation rule in bacterium b_2 is designed by the operation of the proceeding instruction l_j. One of the following two cases occurs in bacterium b_2.

- If instruction l_j is an ADD instruction, then bacterium b_2 has the conjugation rule $(\text{ATP-}p_{si}, b_2/b_1, \text{ATP-}p_j p_{s_j})$. By using the rule, plasmids $p_j p_{s_j}$ and ATP are conjugated to bacterium b_1. In this case, system Π starts to simulate the proceeding ADD instruction l_j.
- If instruction l_j is a SUB instruction, then bacterium b_2 has the conjugation rule $(\text{ATP-}p_{si}, b_2/b_1, \text{ATP-}p_j p_{null})$, by which plasmids $p_j p_{null}$ and ATP are transmitted to bacterium b_1. In this case, system Π starts to simulate the proceeding SUB instruction l_j.

Therefore, system Π can correctly simulate the ADD instruction of M_u. The system starts from bacterium b_1 having plasmid $p_i p_{s_i}$ and ATP, which are transmitted to bacterium b_2 by the conjugation rule. In bacterium b_2, the number of gene g_r in chromosomal DNA is increased by 1 using the CRISPR/Cas9 gene inserting rule, and plasmids $p_j p_{s_j}$ (if the proceeding instruction l_j is an ADD instruction) or $p_j p_{null}$ (if the proceeding instruction l_j is a SUB instruction) are transmitted to bacterium b_1, which means that system Π starts to simulate instruction l_j.

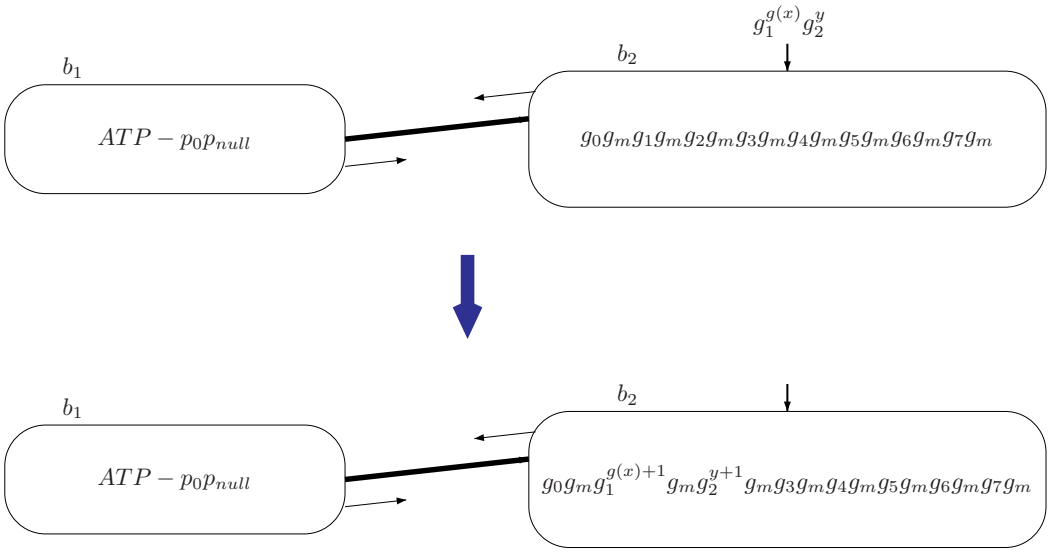

Figure 3. The transition of system Π by reading input information encoded by $g(x)$ copies of genes g_1 and y copies of gene g_2 through input bacterium b_2.

Simulating the SUB instruction: l_i : (SUB(r), l_j, l_k).

We suppose at a certain computation step that system Π has to simulate a SUB instruction l_i : (SUB(r), l_j, l_k). For any SUB instruction l_i, plasmid p_i of the form $p_i = (cas9, gRNA_{g_r}^{delete})$ is associated in system Π. In bacterium b_1, there are plasmids $p_i p_{null}$ and ATP such that the conjugation rule $(\text{ATP-}p_i p_{null}, b_1/b_2, \text{ATP-}p_i p_{null})$ can be used. In bacterium b_2, it has the following two cases.

– If there is at least one gene g_r existing between neighboring genes g_r and g_m in chromosomal DNA of bacterium b_2 (corresponding to the case that the number stored in register r is $n > 0$), then the CRISPR/Cas9 deleting rule $p_i \times (g_1, g_m)$ is used to delete one copy of gene g_r from chromosomal DNA. This simulates the number stored in register r being decreased by 1. By consuming plasmid p_i, bacterium b_2 retains plasmid p_{null} and ATP such that a conjugation rule $(\text{ATP-}p_{null}, b_2/b_1, \text{ATP-}p_j p_{s_j})$ or $(\text{ATP-}p_{null}, b_2/b_1, \text{ATP-}p_j p_{null})$ is used, which depends on

whether the proceeding instruction would be an ADD or a SUB instruction. In this way, plasmids $p_j p_{s_j}$ or $p_j p_{null}$) and ATP are transmitted to bacterium b_1. The system starts to simulate instruction l_j.

– If there is no gene g_r existing between neighboring genes g_r and g_m in chromosomal DNA of bacterium b_2 (corresponding to the case that the number stored in register r is 0), then the CRISPR/Cas9 deleting rule $p_i \times (g_1, g_m)$ cannot be used, but a conjugation rule $(\text{ATP-}p_i p_{null}, b_2/b_1, \text{ATP-}p_k p_{s_k})$ or $(\text{ATP-}p_i p_{null}, b_2/b_1, \text{ATP-}p_k p_{null})$ is able to be used. Plasmids ($p_k p_{s_k}$ or $p_k p_{null}$) and ATP are conjugated to bacterium b_1, which means the system starts to simulate instruction l_k.

We note that when plasmids $p_i p_{null}$ are conjugated to bacterium b_2 from bacterium b_1, it may happen that both the CRISPR/Cas9 deleting rule $p_i \times (g_1, g_m)$ and $(\text{ATP-}p_i p_{null}, b_2/b_1, \text{ATP-}p_k p_{s_k})$ (or $(\text{ATP-}p_i p_{null}, b_2/b_1, \text{ATP-}p_k p_{null})$) can be used. In this case, the CRISPR/Cas9 deleting rule $p_i \times (g_1, g_m)$ will be applied because of the fact that it has priority over the plasmid transferring rule.

The simulation of a SUB instruction is correct: System Π starts from bacterium b_1 having plasmid $p_i p_{null}$ and ATP and ends with plasmid $p_j p_{s_j}$ or $p_j p_{null}$ and ATP (if the number stored in register r is $n > 0$) to start the simulation of instruction l_j; otherwise it ends with plasmid $p_k p_{s_k}$ or $p_k p_{null}$ and ATP (if the number stored in register r is 0) to start the simulation of instruction l_k.

Simulating the halt instruction: l_h : HALT.

When register machine M_u reaches the halt instruction l_h : HALT, the computation of register machine M_u halts. At that moment, bacterium b_1 in system Π holds plasmids $p_h p_{s_h}$ and ATP, and the conjugation rule $(\text{ATP-}p_h p_{sh}, b_1/b_2, \text{ATP-}p_h p_{s_h})$ can be used. By using the rule, plasmids $p_h p_{s_h}$ and ATP are transmitted to bacterium b_2; no gene can be edited by plasmid p_h, and no rule can be used. Hence, the computation of system Π finally halts.

The number of gene g_0 in chromosomal DNA of bacterium b_2 encodes the number stored in register 0 of M_u. If the number stored in register 0 is $n > 0$, then there are $n + 1$ copies of gene g_0 in chromosomal DNA of bacterium b_2. The computational result can be obtained by counting the number of gene g_0 in chromosomal DNA of bacterium b_2.

From the above description of system Π and its work, it is clear that system Π can simulate each computation of M_u. We can check that the constructed system Π has

- 2 bacterium for conjugation with each other;
- 22 plasmids p_i for the 22 ADD and SUB instructions with $i = 0, 1, 2, \ldots 21$;
- 9 plasmids p_{s_i} for 9 ADD instructions with $i = 1, 2, 5, 7, 9, 16, 17, 20, 21$;
- 1 plasmid p_{null} for the 13 SUB instructions;
- 2 plasmids p_h and p_{s_h} for the HALT instruction;
- 8 genes g_i for encoding numbers in registers i with $i = 0, 1, 2 \ldots 7$;
- 1 gene g_m for separating gene g_i in chromosomal DNA.

This gives, in total, 2 bacteria, 34 plasmids, and 9 genes.
This concludes the proof. □

3.2. A Small Universal BP System as a Number Generator

In this section, we construct a small universal BP system as a number generator. A BP system Π_u is universal if, given a fixed admissible enumeration of the unary partial recursive functions $(\varphi_0, \varphi_1, \ldots)$, there is a recursive function g such that for each natural number x, whenever we input the number $g(x)$ in Π_u, the set of numbers generated by the system is equal to $\{n \in N | \varphi_x(n) \text{ is defined}\}$. In other words, after introducing the "code" $g(x)$ of the partial recursive function φ_x in the form of $g(x)$ copies of certain genes in chromosomal DNA of the input bacterium, the BP system generates all numbers n for which $\varphi_x(n)$ is defined.

System Π_u has the same topological structure, plasmids, and evolution rules as system Π constructed in Section 3.1, but the input bacterium is b_2 and the output bacterium is b_1. Differently from the universal computing devices considered in Section 3.1, the strategy to simulate a universal register machine as a number generator is as follows.

Step 1. The output bacterium b_1 initially has n copies of gene g_m.

Step 2. System Π_u starts by loading $g(x)$ copies of gene g_1 and n copies of gene g_2 in the input bacterium b_2.

Step 3. The computation of Π_u is activated by using plasmid $p_0 p_{null}$ to simulate the register machine M_u from Figure 2, with $g(x)$ stored in register 1, and number n stored in register 2.

If the computation in register machine M_u halts, instruction σ_{l_h} can finally be activated. To simulate register machine M_u reaching the HALT instruction, system Π_u holds plasmids $p_h p_{s_h}$ and transmits them to bacterium b_2. After this, system Π_u halts, as no rule can be used in bacterium b_2. When the system halts, the number of gene g_m in the output bacterium b_1 is the computational result, which is exactly the number n. Hence, the number n can be computed/generated by system Π_u.

The difference between systems Π and Π_u is the loading input information process. The initial configuration and transition of system Π_u by reading input signals encoded by $g(x)$ copies of genes g_1 and n copies of gene g_2 through input bacterium b_2 are shown in Figure 4.

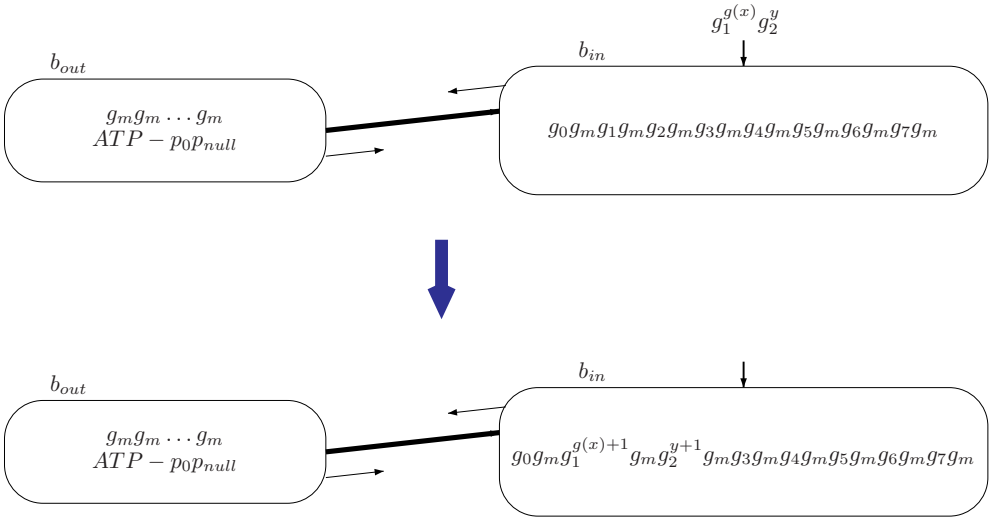

Figure 4. The initial configuration and transition of system Π_u by reading input information encoded by $g(x)$ copies of genes g_1 and n copies of gene g_2 through input bacterium b_2.

We can check that the constructed system Π_u has

- 2 bacterium for the conjugation with each other;
- 22 plasmids p_i for the 22 ADD and SUB instructions with $i = 0, 1, 2, \ldots 21$;
- 9 plasmids p_{s_i} for 9 ADD instructions with $i = 1, 2, 5, 7, 9, 16, 17, 20, 21$;
- 1 plasmid p_{null} for the 13 SUB instructions;
- 2 plasmids p_h and p_{s_h} for the HALT instruction;
- 8 genes g_i for encoding numbers in registers i with $i = 0, 1, 2 \ldots 7$;
- 1 gene g_m for separating gene g_i in chromosomal DNA.

This gives, in total, 2 bacteria, 34 plasmids, and 9 genes.
Therefore, we have the following theorem.

Theorem 2. *There is a Turing universal BP system with 2 bacteria and 34 plasmids that can compute a Turing-computable set of natural numbers.*

4. Conclusions

In this work, we construct two small universal BP systems. Specifically, it is obtained that a BP system with 2 bacteria, 34 plasmids, and 9 genes is universal for both computing recursively enumerable functions and computing/generating a family of sets of natural numbers. It is obtained that 34 plasmids are sufficient for constructing Turing universal BP systems. This provides theoretical support as well as paradigms using a reasonable number of bacteria and plasmids to construct powerful bacterial computers.

Following the research line, finding smaller universal BP systems deserves further research. A possible way to slightly decrease the number of plasmids used in small universal BP systems is using code optimization, exploiting some particularities of the register machine M_u. For example, as considered in [25], for the sequence of two consecutive ADD instructions l_{17}: $(\mathrm{ADD}(2), l_{21})$ and l_{21}: $(\mathrm{ADD}(3), l_{18})$, without any other instruction addressing the label l_{21}, the two ADD modules can be combined. However, a challenging problem regards what the minimum size of a universal BP system is—in other words, what the borderline between universality and non-universality is. Characterization of universality by BP systems is expected. A balance between the number of bacteria and plasmids in universal BP systems can be considered, that is, using more bacteria to reduce the number of plasmids.

It is worth developing the applications of BP systems. Bio-inspiring computing models perform well in computations, particularly in solving computational complex problems in feasible time [34–36]. It is of interest to use BP systems to solve computationally hard problems. Some specific applications using BP systems would be of interest to researchers from biological fields.

In artificial intelligence, there are many bio-inspired algorithms (see, e.g., [37,38]). It is worth designing bacteria-computing-inspired algorithms or introducing bacteria computing operators in classical algorithms. Additionally, it would be meaningful to construct powerful bacterial computers or computing devices in biological labs.

Author Contributions: Conceptualization, X.W. and T.S.; Methodology, X.W.; Software, T.M.; Validation, P.Z., T.S. and X.W.

Acknowledgments: This work was supported by the National Natural Science Foundation of China (Grant Nos. 61502535, 61572522, 61572523, 61672033, and 61672248), PetroChina Innovation Foundation (2016D-5007-0305), Key Research and Development Program of Shandong Province (No. 2017GGX10147), Natural Science Foundation of Shandong Province (No. ZR2017MF004), Talent introduction project of China University of Petroleum (No. 2017010054), Research Project TIN2016-81079-R (AEI/FEDER, Spain-EU) and Grant 2016-T2/TIC-2024 from Talento-Comunidad de Madrid, Project TIN2016-81079-R (MINECO AEI/FEDER, Spain-EU), and the InGEMICS-CM Project (B2017/BMD-3691, FSE/FEDER, Comunidad de Madrid-EU).

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Gitai, Z. The new bacterial cell biology: Moving parts and subcellular architecture. *Cell* **2005**, *120*, 577–586. [CrossRef] [PubMed]
2. Goldstein, E.; Drlica, K. Regulation of bacterial dna supercoiling: Plasmid linking numbers vary with growth temperature. *Proc. Natl. Acad. Sci. USA* **1984**, *81*, 4046–4050. [CrossRef] [PubMed]
3. Summers, D. *The Biology of Plasmids*; John Wiley & Sons: Hoboken, NJ, USA, 2009.
4. Poet, J.L.; Campbell, A.M.; Eckdahl, T.T.; Heyer, L.J. Bacterial computing. *XRDS Crossroads ACM Mag. Stud.* **2010**, *17*, 10–15. [CrossRef]
5. Gupta, V.; Irimia, J.; Pau, I.; Rodríguez-Patón, A. Bioblocks: Programming protocols in biology made easier. *ACS Synth. Biol.* **2017**, *6*, 1230–1232. [CrossRef] [PubMed]

6. Gutierrez, M.; Gregorio-Godoy, P.; del Pulgar, G.P.; Munoz, L.E.; Sáez, S.; Rodríguez-Patón, A. A new improved and extended version of the multicell bacterial simulator gro. *ACS Synth. Biol.* **2017**, 6, 1496–1508. [CrossRef] [PubMed]
7. Adleman, L.M. Molecular computation of solutions to combinatorial problems. *Sciences* **1994**, *266*, 1021–1024. [CrossRef]
8. Carell, T. Molecular computing: Dna as a logic operator. *Nature* **2011**, *469*, 45–46. [CrossRef] [PubMed]
9. Xu, J. Probe machine. *IEEE Trans. Neural Netw. Learn. Syst.* **2016**, *27*, 1405–1416. [CrossRef] [PubMed]
10. Păun, G.; Rozenberg, G.; Salomaa, A. *The Oxford Handbook of Membrane Computing*; Oxford University Press: Oxford, UK, 2010.
11. Păun, G.; Rozenberg, G. A guide to membrane computing. *Theor. Comput. Sci.* **2002**, *287*, 73–100. [CrossRef]
12. Păun, G. Computing with membranes. *J. Comput. Syst. Sci.* **2000**, *61*, 108–143. [CrossRef]
13. Freund, R.; Kari, L., Păun, G. DNA computing based on splicing: The existence of universal computers. *Theory Comput. Syst.* **1999**, *32*, 69–112. [CrossRef]
14. Kari, L.; Păun, G.; Rozenberg, G.; Salomaa, A.; Yu, S. DNA computing, sticker systems, and universality. *Acta Inform.* **1998**, *35*, 401–420. [CrossRef]
15. Martın-Vide, C.; Păun, G.; Pazos, J.; Rodrıguez-Patón, A. Tissue P systems. *Theor. Comput. Sci.* **2003**, *296*, 295–326. [CrossRef]
16. Ionescu, M.; Păun, G.; Yokomori, T. Spiking neural P systems. *Fundam. Inform.* **2006**, *71*, 279–308.
17. Song, T.; Pan, L.; Păun, G. Asynchronous spiking neural p systems with local synchronization. *Inform. Sci.* **2013**, *219*, 197–207. [CrossRef]
18. Ezziane, Z. DNA computing: Applications and challenges. *Nanotechnology* **2005**, *17*, R27. [CrossRef]
19. Chen, X.; Pérez-Jiménez, M.J.; Valencia-Cabrera, L.; Wang, B.; Zeng, X. Computing with viruses. *Theor. Comput. Sci.* **2016**, *623*, 146–159. [CrossRef]
20. Rogozhin, Y. Small universal turing machines. *Theor. Comput. Sci.* **1996**, *168*, 215–240. [CrossRef]
21. Baiocchi, C. Three small universal turing machines. In *Machines, Computations, and Universality*; Springer: Berlin/Heidelberg, Germany, 2001; pp. 1–10.
22. Korec, I. Small universal register machines. *Theor. Comput. Sci.* **1996**, *168*, 267–301. [CrossRef]
23. Iirgen Albert, J.; Culik, K., II. A simple universal cellular automaton and its one-way and totalistic version. *Complex Syst.* **1987**, *1*, 1–16.
24. Kudlek, M.; Yu, R. Small universal circular post machines. *Comput. Sci. J. Mold.* **2001**, *9*, 25.
25. Păun, A.; Păun, G. Small universal spiking neural P systems. *BioSystems* **2007**, *90*, 48–60. [CrossRef] [PubMed]
26. Zhang, X.; Zeng, X.; Pan, L. Smaller universal spiking neural P systems. *Fundam. Inform.* **2008**, *87*, 117–136.
27. Pan, L.; Zeng, X. A note on small universal spiking neural P systems. *Lect. Notes Comput. Sci.* **2010**, *5957*, 436–447.
28. Păun, A.; Sidoroff, M. Sequentiality induced by spike number in SNP systems: Small universal machines. In *Membrane Computing*; Springer: Berlin/Heidelberg, Germany, 2012; pp. 333–345.
29. Song, T.; Jiang, Y.; Shi, X.; Zeng, X. Small universal spiking neural P systems with anti-spikes. *J. Comput. Theor. Nanosci.* **2013**, *10*, 999–1006. [CrossRef]
30. Song, T.; Xu, J.; Pan, L. On the universality and non-universality of spiking neural P systems with rules on synapses. *IEEE Trans. Nanobiosci.* **2015**, *14*, 960–966. [CrossRef] [PubMed]
31. Song, T.; Pan, L. Spiking neural P systems with request rules. *Neurocomputing* **2016**, *193*, 193–200. [CrossRef]
32. Song, T.; Rodríguez-Patón, A.; Gutiérrez, M.; Pan, Z. Computing with bacteria conjugation and crispr/cas9 gene editing operations. *Sci. Rep.* **2018**, sumitted.
33. Minsky, M.L. *Computation: Finite and Infinite Machines*; Prentice-Hall, Inc.: Upper Saddle River, NJ, USA, 1967.
34. Valencia-Cabrera, L.; Orellana-Martín, D.; Martínez-del Amor, M.A.; Riscos-Nú nez, A.; Pérez-Jiménez, M.J. Computational efficiency of minimal cooperation and distribution in polarizationless p systems with active membranes. *Fundam. Inform.* **2017**, *153*, 147–172. [CrossRef]
35. Song, B.; Pérez-Jiménez, M.J.; Pan, L. An efficient time-free solution to qsat problem using p systems with proteins on membranes. *Inf. Comput.* **2017**, *256*, 287–299. [CrossRef]

36. Macías-Ramos, L.F.; Pérez-Jiménez, M.J.; Riscos-Nú nez, A.; Valencia-Cabrera, L. Membrane fission versus cell division: When membrane proliferation is not enough. *Theor. Comput. Sci.* **2015**, *608*, 57–65. [CrossRef]
37. Ma, X.; Sun, F.; Li, H.; He, B. Neural-network-based sliding-mode control for multiple rigid-body attitude tracking with inertial information completely unknown. *Inf. Sci.* **2017**, *400*, 91–104. [CrossRef]
38. Alsaeedan, W.; Menai, M.E.B.; Al-Ahmadi, S. A hybrid genetic-ant colony optimization algorithm for the word sense disambiguation problem. *Inf. Sci.* **2017**, *417*, 20–38. [CrossRef]

Sample Availability: Samples of the compounds are not available.