Computing with Metabolic Machines

Claudio Angione[1], Giovanni Carapezza[2], Jole Costanza[2], Pietro Lió[1] and Giuseppe Nicosia[2]

[1] Computer Laboratory, University of Cambridge - UK
{claudio.angione, pietro.lio}@cl.cam.ac.uk
[2] Department of Maths & Computer Science, University of Catania - Italy
{carapezza, costanza, nicosia}@dmi.unict.it

Abstract

If Turing were a first-year graduate student interested in computers, he would probably migrate into the field of computational biology. During his studies, he presented a work about a mathematical and computational model of the morphogenesis process, in which chemical substances react together. Moreover, a protein can be thought of as a computational element, i.e. a processing unit, able to transform an input into an output signal. Thus, in a biochemical pathway, an enzyme reads the amount of reactants (substrates) and converts them in products. In this work, we consider the biochemical pathway in unicellular organisms (e.g. bacteria) as a living computer, and we are able to program it in order to obtain desired outputs. The genome sequence is thought of as an executable code specified by a set of commands in a sort of ad-hoc low-level programming language. Each combination of genes is coded as a string of bits $y \in \{0,1\}^L$, each of which represents a gene set. By turning off a gene set, we turn off the chemical reaction associated with it. Through an optimal executable code stored in the "memory" of bacteria, we are able to simultaneously maximise the concentration of two or more metabolites of interest. Finally, we use the Robustness Analysis and a new Sensitivity Analysis method to investigate both the fragility of the computation carried out by bacteria and the most important entities in the mathematical relations used to model them.

1 Introduction: From Turing to Bray

The key role that computation plays in the bio-inspired science was first discovered by Turing in 1952 [22], who proposed computational processes in the morphogenesis. Turing states that an organism, most of the time, develops from one pattern into another. In order to follow this general process without simplifying assumptions (which are often made when conducting theoretical analyses), one needs not only to have a theoretical approach, but also to treat particular cases in detail with the aid of a digital computer. According to Bray [3], a single protein is able to transform one or multiple input signals into an output signal, thus it can be viewed as a computational or information carrying element. The cellular behaviour, controlled by complex regulatory circuits, strongly depends on the signal transduction proteins (e.g. N-WASP and Cdk2 [19]), which integrate multiple input signals responding simultaneously to all of them [14]. Although DNA computing is a young field of research, it seems that computation in organisms took place million years ago. Indeed, Landweber and Kari [13] provided a model to view the unscrambling of genes in ciliates as a computational process. In particular, the correct gene assembly in ciliates requires the joining of many DNA sequences controlled by sequence repeats, and this has much in common with the Adleman's algorithm [1] in graph theory. Hence, a guided genome recombination system can simulate a Turing Machine (TM), and therefore a functional macronuclear gene can be viewed as the output of a computation carried out on the micronuclear sequence [2].

Following this line of thought, we provide a framework to show that bacteria could have computational capability and act as molecular machines. Remarkably, as the biological system grows larger, reaching the desired multiple input/output performance becomes a difficult task, thus some sort of machine optimisation is required. To this end, we provide a novel algorithm called Genetic Design through Multi-Objective optimisation (GDMO), with the aim of programming molecular machines to maximise the yield of a desired metabolite. By giving to a bacterium a certain amount of input metabolites, its metabolism follows a given pattern and produces output metabolites. Therefore, by changing these input quantities, one could force it to move to another development pattern [22]. For instance, most bacteria can selectively use substrates from a mixture of different carbon sources. The presence of preferred carbon sources prevents the expression, and often also the activity, of catabolic systems that enable the use of secondary substrates. The ability of bacteria to prefer one carbon source to another is termed carbon catabolite repression (CCR) [12].

It is well known that a von Neumann architecture is composed of a processing unit, a control unit, a memory to store both data and instructions, and input-output mechanisms. The bacterium takes as input chemicals (substrates) necessary for his growth and duplication, and thanks to its biochemical network (coded by the genes of its genome), produces small metabolites as output. We show that by providing an analogy between the von Neumann architecture and a bacterium, its metabolism could be framed as a TM.

Through an effective formalism, we describe the whole behaviour of the bacterial cells in terms of the von Neumann architecture. In particular, the genome sequence is thought of as an executable code specified by a set of commands in a sort of ad-hoc low-level programming language. Each combination of genes is coded as a string of bits $y \in \{0,1\}^L$, each of which represents a gene set. By turning off a gene set, we turn off the chemical reactions associated with it. The memory unit contains the string y, which is a program written in an ad-hoc low-level programming language. The control unit is a function g_Φ that defines a partition of the string, and is uniquely determined by the pathway-based clustering of the chemical reaction network. We model the processing unit of the bacterium as the collection of all its chemical reactions. This way, we associate the chemical reaction network of bacteria with a TM.

By investigating the whole metabolism of bacteria considering pathways of many proteins, we extend the above mentioned Bray's idea, i.e. thinking of a protein as a computational element. Likewise, the cell as a computational element receives, processes, and responds to inputs from the environment. However, since the question *'what does this cell do?'* has often more than one correct answer, we program molecular machines using GDMO. GDMO acts on the genetic level of the organism to find which are the genetic strategies in order to obey control signals; then, it executes the optimisation of multiple biological functions. By exploring effectively the whole space of gene knockouts, GDMO optimises acetate and succinate production, as well as other multiple biological functions. In this paper, we test it on the model *E. coli iAF1260* and we compare it with state-of-the-art methods, e.g. GDLS, OptFlux, OptGene, OptKnock. Each point of the Pareto front provided by GDMO is a molecular machine to execute a particular task. Pareto optimality allows to obtain not only a wide range of Pareto optimal solutions, but also the *best trade-off design*. Finally, we propose a solution for the problem of making the sensitivity analysis pathway-dedicated: we develop the Pathway oriented Sensitivity Analysis (PoSA) to investigate the functional components of the molecular machine and detect the most sensitive ones. The robustness analysis supports GDMO and PoSA in that it indicates the components of the molecular machine that are likely to "fail".

Robust genetic interventions in cells, framed as optimal programs to be run in a molecular machine, can be exploited to extend and modify the behaviour of cells and cell aggregates. For

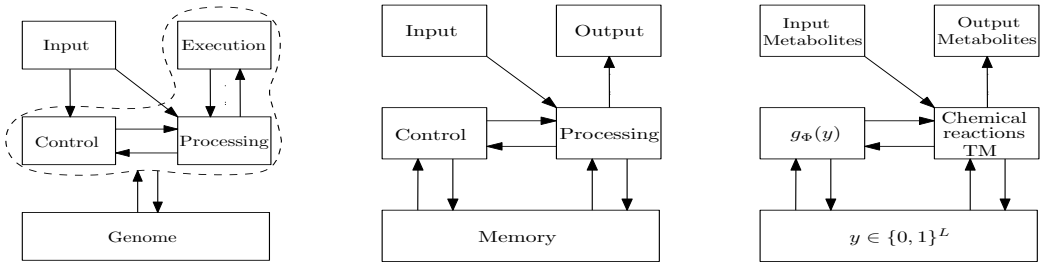

Figure 1: Comparison among biological systems (a), von Neumann architecture (b), and bacteria (c). The string y is a program stored in the RAM. The function g_Φ represents the control unit: it interprets the binary string y and turns gene sets off. The processing unit is the metabolism of the bacteria, composed of all the chemical reactions that take place in it. The goal is to produce desired metabolites as output of the molecular machine.

instance, programs can instruct cells to make logic decisions according to environmental factors and current cell state. A program embedded in a cell could allow its metabolic network to work with a specific user-imposed aim. The metodology proposed in this paper aims at addressing the challenge pointed out by Weiss [23], i.e. to provide a robust computation in cells, with reliable and reproducible results. This would lead to effectively modify and harness biological organisms for our purposes.

2 Bacteria as von Neumann architectures

Inspired by Brent and Bruck [4], who studied similarities and differences between biological systems and von Neumann computers, in Figure 1 we propose a mapping between the von Neumann architecture and bacteria. This mapping suggests thinking of the metabolism as a Turing Machine (TM). The bacterium takes as input the substrates required for its growth and, thanks to its chemical reaction network, produces desired metabolites as output. The string y acts as a program stored in the RAM.

Let us consider the multiset Y of the bits of y. A partition Π of the multiset $Y = \{y_1, y_2, ..., y_L\}$ is a collection $\{b_1, b_2, ..., b_p\}$ of submultisets of Y that are nonempty, disjoint, and whose union equals Y. The elements $\{b_s\}_{s=1,...,p}$ of a partition are called blocks. We denote by $P(Y;p)$ the set of all partitions of Y with p blocks. $P(Y;p)$ has a cardinality equal to the Stirling number, namely $|P(Y;p)| = S_{L,p}$ [20]. In order to formalise the control unit behaviour, let us define the function:

$$g_\Phi : \{0,1\}^L \longrightarrow \bigcup_{y \in \{0,1\}^L} P(Y;p)$$

$$\bar{y} \in \{0,1\}^L \longmapsto \Pi \in P(\bar{Y};p),$$

where the partition Π is uniquely determined by the pathway-based subdivision of the chemical reaction network. This subdivision, in turn, can be formalised as a p-blocks partition Φ of the set of the bit indexes in the string y. In particular, if we denote by $[L]$ the set of the first L natural numbers, we have $\Phi \in P([L];p)$. According to the partition Φ, the control function g_Φ partitions the multiset Y associated with the string y (see Figure 2).

The function g_Φ plays the role of the control unit, since it interprets the binary string y and turns gene sets on and off, according to the pathway-based partitioning of the reactions occurring in the bacterium. Each element of the partition Π is the submultiset b_s of all the gene sets that play a role in the reactions belonging to the s-th pathway. In other words, g_Φ turns syntax into semantics. The processing unit of the bacterium could be modelled as the collection of all its chemical reactions. In this regard, a TM can be associated with the chemical reaction network of bacteria [11].

Let us consider the *Minsky's register machine* (RM), i.e., a finite state machine augmented with a finite number of registers. Formally, a Minsky machine $\mathcal{M} = (D, i_0, i_1, \varphi)$ is a finite set D of states, along with a finite set $H = \{H_r\}_r$ of registers, and a multivalued mapping $\varphi : D\backslash\{i_0\} \longrightarrow \{(H_r, i), (H_r, j, k) \mid H_r \in H,\ j, k \in D\}$. The set D has two distinguished elements $i_0, i_1 \in D$ representing the initial state and the halting state respectively. Each register H_r of the RM holds a non-negative integer. The instruction $inc(i, r, j)$ increments register r by 1 and causes the machine to move from state i to state j through the mapping $\varphi(i) = j$. Conversely, the instruction $dec(i, r, j, k)$, given that $H_r > 0$, decrements register r by 1 and causes the machine to move from state i to state j ($\varphi(i) = j$); if $H_r = 0$, the machine moves from state i to state k ($\varphi(i) = k$). The Minsky's RM has been proven to be equivalent to the TM [16].

In order to map the chemical reaction network to the RM, we define [11]: (i) the set of state species $\{D_i\}$, where each D_i is associated with the state i of the RM; (ii) the set of register species $\{H_r\}$, where each H_r is associated with the register r of the RM, and therefore represents the molecular count of species r. The instruction $inc(i, r, j)$ can be viewed as the chemical reaction $D_i \to D_j + H_r$. The instruction $dec(i, r, j, k)$ can be viewed as either $D_i + H_r \to D_j$ or $D_i \to D_k$ depending on whether $H_r > 0$ or $H_r = 0$ respectively. It is noteworthy that the molecular machine performs the reaction $D_i \to D_k$ only when there are no molecules of H_r, since the r-th register cannot be decreased and the reaction $D_i + H_r \to D_j$ cannot take place (this is equivalent to the "test for zero" in the RM). In our FBA approach, the variables are the fluxes of the reactions in the network, therefore a high flux corresponds to both a high rate of reaction and a high mass of products. Consequently, given the increment reaction $inc(i, r, j)$, the value of H_r is positively correlated with the reaction flux; conversely, in the decrement reaction $dec(i, r, j, k)$, when $H_r > 0$ the value of H_r is negatively correlated with the reaction flux.

Given a fixed volume V in which the reactions occur, and given two reactions *inc* and *dec* with fluxes v_1 and v_2 respectively, the metabolism of the bacterium has a probability of error per step equal to $\epsilon = v_2/(v_1/V + v_2)$. Since it is well known that the simulated TM can be universal, the mapping between metabolism and TM allows to perform any kind of computation, through a set of species and chemical reactions characterised by their flux. In principle, this means that bacteria can carry out at least any computation performed by a computer.

3 Genetic design of (living) computers

In light of the correspondence between bacteria and computers proposed in the previous section, our aim is to program bacteria to optimise their yield. We program molecular machines using a novel algorithm called Genetic Design through Multi-Objective optimisation (GDMO), which is able to process the information, decide and execute. Through a specific optimal code stored in the "memory" of the organism, we are able to simultaneously maximise the concentration of two or more metabolites of interest. In particular, GDMO acts on the genetic level of the organism to find which are the genetic strategies in order to obey control signals; then, it executes the

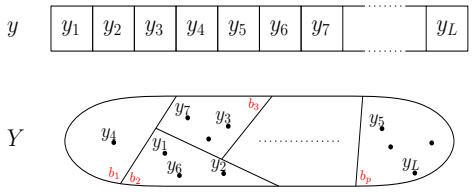

Figure 2: The multiset Y associated with y is partitioned by Π in p blocks. The elements of Π are submultisets of Y, since y is a string of bits, thus 0 and 1 may occur more than once in the same subset. In this example, $\Pi = \{\{y_4\}, \{y_1, y_6, y_2\}, \ldots, \{y_5, ..., y_L\}\}$, $\Phi = \{\{4\}, \{1, 6, 2\}, \ldots, \{5, ..., L\}\}$

optimisation of multiple biological functions. The molecular machine we take into account is the *Escherichia Coli* in the Flux Balance Analysis (FBA) framework. In this context, the multiple biological functions are represented by desired productions, e.g., vitamins, proteins, biofuel, antibodies, electron productivity, or the energetic yield of the organism. The genetic code, i.e. the "computation instructions" given to the machine, is represented by a string of bits $y \in \{0,1\}^L$. Each bit in y is a gene set that distinguishes between single and multifunctional enzymes, isozymes, enzyme complexes and enzyme subunits; this way, it captures the complexity and diversity of the biological relationships through a Boolean approach.

In the past years, a variety of methods has been implemented based on evolutionary algorithms [9], simulated annealing [10], bi-level optimisation framework [8], and mixed-integer linear programming (MILP) [5, 18]. All these methods have been tested in FBA organism machines. However, they require high computational efforts: the execution times grow exponentially [5, 10, 9], or linearly [8] as the number of manipulations allowed in the final designs increases. Because of the large number of reactions occurring in cellular metabolism, the dimension of the solution space is very large, and finding genetic manipulations is very expensive. Our method is able to explore effectively the whole space of knockouts. We tested the performance of GDMO by optimising acetate, succinate, and other multiple biological functions in *E. coli,* iAF1260, and comparing it with state-of-the-art methods.

Additionally, we propose the Pathway oriented Sensitivity Analysis (PoSA), which represents a solution for the problem of making the sensitivity pathway-dedicated. This method interrogates the structural components of the machine to reveal the most sensitive ones. PoSA is able to find the genetic manipulations that have a greater influence on the outputs of the model. Unlike other Sensitivity Analysis methods applied in biological modelling, such as RoSA or SoSA (respectively Reactions- and Species- oriented Sensitivity Analysis) whose inputs (reactions or species) are valued in a *real* domain, PoSA is applied when inputs are valued in a *discrete* domain Ω. Indeed, each input of the model is represented by a set of binary variables (bits). PoSA computes two sensitive indices $\mu*$ and $\sigma*$ for each pathway. Large measures of these indices indicate an input with an important influence on the outputs.

The ability of the molecular machine to adapt to perturbations due to internal or external agents, aging, temperature, and environmental changes is one of evolutionism guidelines and should also be a fundamental design principle. By evaluating the fragility of the components of the metabolic network, e.g. genes, fluxes and reactions, we evaluate the variables and subroutines of the molecular machine that are prone to failure. In fact, we want to assess the ability of a strain to adapt to small perturbations that can occur at any stage of the biochemical processes, within the bacterium or caused by the environment in which it reproduces itself. To this aim, we perform a statistical post-processing analysis inspired by the well-known robustness

analysis used in Electronic Design Automation (EDA) ([17]). The local robustness evaluates genes, fluxes and reactions, and returns a numerical value that indicates their "fragility", thus ensuring the robustness of the solutions proposed by GDMO and PoSA.

4 Algorithmic complexity

GDMO is a combinatorial global search method that finds the genetic manipulation strategies to simultaneously optimise multiple cellular functions (i.e., objective functions) in metabolic networks. The multi-objective optimisation aims at exploiting the computational capabilities of bacteria, in order to allow the maximum production of metabolites of practical or industrial interest. Simultaneous optimisation of multiple objectives differs from single-objective optimisation because the solution is not unique when the objectives are in conflict with each other. The solution of a multi-objective problem is a potentially infinite set of points, called Pareto optimal solutions or *Pareto front*. A point y^* in the solution space is said to be Pareto optimal if there does not exist a point y such that $Z(y)$ dominates $Z(y^*)$, where Z is the vector of r objective functions. The variable space, (i.e., the domain of y) is defined in a binary space.

The method we present implements a genetic algorithm inspired by NSGA-II [6] and is composed of 4 key steps. We start with the initialisation of the population *Pop* and the computation of the fitness score. The population *Pop* is represented by a $I \times (L + F)$ matrix, where I is the number of individuals and F is the number of the objective functions. Each individual represents a feasible solution, composed by the proposed knockout strategy $\tilde{y}$ (L dimensional) and by the corresponding objective function values. The fitness score is computed after sorting each individual according to the level of non-domination. Each individual is assigned a rank, thus between two solutions with differing non-domination ranks we prefer the point with the lower rank [6]. The individuals of the initial population can be initialised in different ways: randomly, or assigning present status to all genes, or selecting a set of knocked out genes.

Successively, three steps are iteratively carried out. In a *binary tournament selection* process, two individuals are selected at random, and their fitness is compared. The individual with the best fitness is selected as a parent. The algorithm selects a number of parents (i.e. the best individuals) equal to the half of the population. Parents are mutated using a *combinatorial mutation operator* convenient to create an offspring population. Mutation represents a switch, from 0 to 1, or from 1 to 0 for a gene set. The process is randomly executed and for each parent individual are formed 10 offsprings and only the best is chosen. Mutation can achieve the maximum knockouts number equal to the parameter C (fixed at 50 by default), as reported in equation (1). A novel population of size *Pop* is formed selecting the best individuals from the parents of the previously generation and the current offspring. Finally, a *selection* operator is performed in order to reach the last front. Then, for each generation, Pareto optimal solutions are provided.

This cycle is repeated until the solutions set does not improve, or until an individual with a desired phenotype is achieved or when the number of generation is bounded out. The number of generations D and individuals of population I are parameters chosen by the user. The time-complexity of the genetic algorithm when the objective number is 2 (e.g., when acetate production and biomass formation are chosen as objectives), is $O(2DI^2)$, where D is the number of generations and I is the population size.

We tested our method for the genome-scale FBA model of *E. coli* K-12 MG1655, iAF1260. This model consists of an $m \times n$ stoichiometric matrix S, where m is the number of metabolites involved in the network, while n is the number of reactions. The *(i,j)*-th element S_{ij} is the

stoichiometric coefficient of the *i*-th metabolite in the *j*-th reaction. Flux v_j through reaction j is constrained by a lower bound vector v_j^L and an upper bound vector v_j^U. In order to allow the algorithm to work at the genetic level, we used gene-protein-reaction (GPR) mappings. GPR mappings provide the links between each geneset and the reactions that depend on it, and defines how certain genetic manipulations affect reactions in the metabolic network. For a set of L genetic manipulations, the GPR mappings are represented by a $L \times n$ matrix G, where the *(l,j)*-th element is 1 if the *l*-th genetic manipulation maps onto reaction j, and is 0 otherwise. The genetic manipulations are performed by a string of bits represented by the vector y, whose *l*-th element y_l is equal to 1 if the geneset embroiled in manipulation l is knocked out, and 0 otherwise. This string can be thought of as the program stored in the RAM of the machine. The multi-objective optimisation is performed to find the program that, running in the molecular machine, gives the optimal output. We evaluate the value of the objective functions by solving the following bi-level problem:

$$\begin{aligned}
&\text{maximise} \quad c'v \\
&\text{such that} \quad \sum_{l=1}^{L} y_l \leq C \\
&\qquad y_l \in \{0,1\} \\
&\qquad \text{maximise} \quad f'v \\
&\qquad \text{such that} \quad Sv = 0 \\
&\qquad\qquad (1-y)'G_j v_j^L \leq \ v_j \leq \ (1-y)'G_j v_j^U, \\
&\qquad\qquad j = 1, \ldots, n,
\end{aligned} \tag{1}$$

where f is a vector of weights (n dimensional), and f' is its transpose. The elements in f_z are either 0 or 1. f_z is equal to 1 if v_z is the flux of the natural objective. v_z is the core biomass and is constrained to values greater than 0.05 h^{-1}. v_j^L and v_j^U are the lower and upper bound values (thermodynamic constraints) of the generic flux v_j. (In our analysis, we consider $v_j^U = 100$ and $v_j^L = -100$ for the fluxes that represent reversible reactions). c is a vector of weights (n dimensional) associated with the synthetic objectives. The weights c_j and c_h are equal to 1, when the synthetic objectives v_j and v_h have to be maximised. y is the knockout vector (L dimensional) and contains only 0 when the metabolic network is complete. When y_l is equal to 1, the gene set embroiled in the l-th manipulation is turned off, and the corresponding reactions are in the absent status (their lower and upper bounds are set to zero). C is an integer and stands for the maximum number of knockouts allowed. We used the GLPK solver to solve the problem 1.

4.1 Pathway-oriented Sensitivity Analysis

In PoSA, the knockout vector y used to represent the genetic manipulations is partitioned in submultisets of bits $Y = \{b_1, b_2, \ldots, b_s, \ldots, b_p\}$. Each submultiset b_s includes the genetic manipulations linked to the reactions involved in the s-th metabolic functional pathway of the network. Each submultiset b_s has a length W_s, where $W_s < \ L, s = 1, \ldots, p$. We clustered in each submultiset all the genes that are involved in a single functional pathway, e.g. the *Citric Acid Cycle*, *Oxidative Phosphorylation*, *Pentose Phosphate Pathway*, and so on.

We generated the Gene-Pathway mappings (GP), defined by the $p \times L$ matrix P, where the *(s,l)*-th element of P is 1 if the l-th genetic manipulation is linked to the reactions involved in the s-th functional pathway, and 0 otherwise. We also formalised the Reaction-Pathway

mappings (RP), mathematically described by the $p \times n$ matrix R, where the *(s,j)*-th element of P is 1 if the *j*-th reaction is part of the *s*-th functional pathway, and 0 otherwise. The *E. coli* model used for our analysis is partitioned in $p = 36$ functional pathways. For the combinatorial problem described above, we defined the "Elementary Effect" [15] for the input b_s as:

$$EE_s = \frac{\left[f(b_1, b_2, \ldots, b_{s-1}, \tilde{b}_s, b_{s+1}, \ldots, b_p) - f(\tilde{y})\right]}{\Delta_s} \tag{2}$$

where $\tilde{b}_s$ is the mutation on the input b_s, and consists of the *switch* of bits chosen randomly in b_s: if a bit is equal to 0 (or 1), the permutation turns it into 1 (or 0). Δ_s is a scale factor defined as:

$$\Delta_s = \frac{1}{W_s} \sum_{i=1}^{W_s} \tilde{b}_s(i), \ s = 1, \ldots, p. \tag{3}$$

The output $f(y)$ considered in our analysis is the vector v of fluxes. $\tilde{y}$ is the mutation carried on the knockout vector y defined in the Boolean region of interest $\Omega = \{0,1\}^L = \{(y_1, \ldots, y_l, \ldots, y_L) | y_l \in \{0,1\}\}$. The pseudo-code in Algorithm 1 and Algorithm 2 explains the PoSA method in details. The parameters β and K of PoSA establish respectively the allowed knockouts in the whole network and in the pathway b_s. In our analysis we chose $\beta = 0.1$ (default value), and we recommend to set $0.02 \leq \beta \leq 0.2$. K is selected by the user or set by default to 4.

The distribution of effects EE_s is obtained permuting y by randomly sampling KQ points from Ω and permuting b_s by randomly sampling KQN points from Ω. If the procedure was performed for each input, the result would be a random sample at a total cost of KQ for calculating $f(\tilde{y})$ and KQN for $f(b_1, b_2, \ldots, \tilde{b}_s, \ldots, b_p)$, with a total cost of $pKQ(N+1)$ evaluates of function. The estimation of the mean μ^* and standard deviation σ^* will be used as indicator of which inputs should be considered important. A large (absolute) measure of central tendency for EE_s indicates an input with an important overall influence on the output. A large measure of spread indicates an input whose influence is highly dependent on the values of the inputs [15].

4.2 Robustness Analysis

The basic principle of this analysis is as follows. Firstly, we define the perturbation as a function $\tau = \gamma(\Psi, \sigma)$ where γ applies a stochastic noise σ to the system Ψ and generates a trial sample τ. The γ-function is called γ-perturbation. Without loss of generality, we assume that the noise is defined by a random distribution. In order to make statistically meaningful the calculation of robustness, we generate a set T of trial samples τ. Each element τ of the set T is considered robust to the perturbation, due to stochastic noise σ, for a given property (or metric) ϕ, if the following condition is verified:

$$\rho(\Psi, \tau, \phi, \epsilon) = \begin{cases} 1, & \text{if } \ |\phi(\Psi) - \phi(\tau)| \leq \epsilon \\ 0, & \text{otherwise} \end{cases} \tag{4}$$

where Ψ is the *reference system*, ϕ is a *metric* (or property), τ is a *trial sample* of the set T and ϵ is a *robustness threshold*. The definition of this condition makes no assumptions about the function ϕ. It can be anything (not necessarily related to properties or characteristics of the system); however, it is implicitly assumed that it is quantifiable.

Algorithm 1 PoSA Algorithm

Require: $[f, y, Q, N, K, \alpha, \beta]$
/* f output of the model */
/* y knockout vector */
/* $[Q, N, K, \alpha, \beta]$ parameters of PoSA*/

Given $Y = \{b_1, b_2, \ldots, b_s, \ldots, b_p\}$
for $s \leftarrow 1$ to p **do**
  Select the pathway b_s
  for $q \leftarrow 1$ to Q **do**
    for $\alpha \leftarrow 1$ to K **do**
    /* perform Mutation Operator on y */
      $\tilde{y}(q, \alpha) \leftarrow \text{Mutation}(y, \beta)$
      for $h \leftarrow 1$ to N **do**
      /* perform Mutation Operator on b_s */
        $\tilde{b}_s(\alpha, q, h) \leftarrow \text{Mutation}\Big(b_s, \frac{\alpha}{K}\Big)$
      /* evaluate scale factor Δ */
        $\Delta_s(h, \alpha, q) = \dfrac{\sum \tilde{b}_s(\alpha, q, h)}{W_s}$
      /* evaluate an Elementary Effect on pathway b_s */
        $EE_s(h, \alpha, q) = \dfrac{f(\tilde{y}(i, \alpha)) - f(b_1, b_2, \ldots, \tilde{b}_s, \ldots, b_p)}{\Delta_s(h, \alpha, i)}$
      end for
    end for
  end for
  $\mu_s{}^* \leftarrow \text{mean}(EE_s)$ /* evaluate the $\mu_s{}^*$ sensitivity index */
  $\sigma_s{}^* \leftarrow \text{var}(EE_s)$ /* evaluate the $\sigma_s{}^*$ sensitivity index */
end for
return $[\mu*\, , \sigma*\,]$

Algorithm 2 Mutation Operator

Require: $[x, a]$
/* $a \in \{0, 1\}$ is a real constant and defines the maximum number of mutations in the Boolean vector x of W_x elements*/

/* Select randomly an integer value A in $[1, a \cdot M_x]$ */
$A \leftarrow \text{random } (1, aW_x)$

/* Select randomly A bits on x vector */
$ind \leftarrow \text{random } (1, W_x, A)$

$\tilde{x} \leftarrow \text{not}(x\,[ind]\,)$
return $\tilde{x}$

The robustness of a system Ψ is the number of robust trials of T, with respect to the property ϕ, over the total number of trials ($|T|$). In formal terms:

$$\Gamma\left(\Psi, T, \phi, \epsilon\right) = \frac{\sum_{\tau \in T} \rho\left(\Psi, \tau, \phi, \epsilon\right)}{|T|} \tag{5}$$

where Γ is a dimensionless quantity that states, in general, the robustness of a system and, in this case, of a strain.

Γ is a function of ϵ, so the choice of this parameter is crucial and not a trivial task. Generally, if a system is robust, all trials must differ the less possible from it. In EDA, designers choose ϵ taking into account the manufacturing processes, the physical properties of the materials and the adherence to the original design. In the area of robust cell design, setting a general and reasonable threshold has required to perform extensive computational experiments [21]. This way, we detected 3% of $\phi(\Psi)$ for local analysis (local robustness) and 6% of $\phi(\Psi)$ for global analysis (global robustness).

5 Results

In Table 2, we report the genetic strategies obtained by GDLS ([8]), OptFlux ([10]), OptGene ([9]), OptKnock ([5]) and GDMO to optimise succinate and acetate productions in the *E. coli* metabolic network. For all methods, we used the *E. coli* K-12 MG1655 iAF1260 model ([7]), in anaerobic conditions with 10 mmol h^{-1} gDW^{-1} available glucose.

OptKnock ([5]) is a bi-level optimisation framework, that uses CPLEX 7.0 and that makes inoperative fluxes in order to maximise a specific bioengineering objective as acetate, subject to the maximisation of a cellular objective (e.g., biomass yield). OptKnock, as OptGene and OptFlux, does not use the FBA reductions and does not implement the GPR mappings, thus it identifies and turns off reactions that uncouple the objective functions. We allow at most 50 reaction knockouts. OptFlux and OptGene implement respectively an evolutionary algorithm and a simulated annealing meta-heuristics, while GDLS employs a local search method through a principled heuristic approach and integrates the GPR mappings and the FBA reductions. GDLS, OptGene and OptFlux solutions are extracted by published data in [8].

We run GDMO initialising the *E. coli* network with an empty set of knockout, i.e., in wild type configuration, and setting the population size I and the number of generations D as $I = 1000$ and $D = 1500$. Table 2 reports the best solutions obtained by the previously methods and our proposed solutions. It is clear that GDMO overcomes previous methods, proposing strategies with a lower knockout cost. In particular, we obtained solutions in terms of succinate (strain B_2, in Table 1) better than the ones achieved by GDLS and OptKnock. Our strategy achieves a slightly higher biomass turning off only eight genes. As a matter of fact, solutions of past methods have a high k cost; in particular, OptKnock reaches k cost=54, that is very hard to reproduce in laboratory. For acetate production, we also show the solution reported in the third and second last column of Table 2: we are able to find, respectively, a grater level of biomass and acetate through knockout costs of value 5 and 10.

As regards the Global Robustness of a strain, we performed the perturbation with Gaussian noise with zero mean and sigma equal to 5% of the perturbed variable. In particular, the perturbed variables are the upper and lower bounds that regulate the metabolic fluxes. Hence, a trial τ is created by perturbing each of the upper v_j^U and lower bounds v_j^L, $j = 1, \ldots, n$, of the metabolic flux. After N trials, for each of them we evaluate the property $\phi(\tau)$ (by flux balance analysis), which in our case can be the value of acetate, succinate, biomass or a combination

Table 1: Proposed solutions to maximise acetate (Ac) and succinate (Suc) productions [mmolh^{-1} gDW^{-1}]. For each strain we report the biomass formation (Bm) [h^{-1}], the knockout cost (k cost) and the calculated Robustness indices: the Global index (GR) and the relative minima of the local robustness results (LR), associated with the related reactions and the corresponding genes. For all other fluxes not reported in the Table, we obtained 100% of local robustness.

Strain	**Ac**	**Bm**	**k cost**	**GR**(%)	**LR**(%)	**Reaction**	**Gene(s)**
A_1	19.198	0.052	12	0.43	17.5	D-glucose exchange (Ex-glc)	
					24.0	Glyceraldehyde-3-phosphate dehydrogenase (GAPD)	(b1179)
					46.5	Triose-phosphate isomerase (TPI)	(b3919)
					47.0	Formate C-acetyl transferase (PFL)	(b0902) (b0903)
							(b2579) (b3114)
							(b3951) (b3952)
					70.5	Glucose-permease IIA component (GLCptsspp)	(b1101) (b2415)
							(b2416) (b2417)
							(b1621) (b1817)
							(b1818) (b1819)
							(b2415) (b2416)
A_2	19.150	0.053	10	1.75	12.5	D-glucose exchange	
					14.5	Glyceraldehyde-3-phosphate dehydrogenase	(b1179)
					43.0	Triose-phosphate isomerase	(b3919)
					52.5	Formate C-acetyl transferase	(b0902) (b0903)
							(b2579) (b3114)
							(b3951) (b3952)
					71.0	Glucose-permease IIA component	(b1101) (b2415)
							(b2416) (b2417)
							(b1621) (b1817)
							(b1818) (b1819)
							(b2415) (b2416)
A_3	18.532	0.096	9	13.55	28.5	D-glucose exchange	
					47.5	Glyceraldehyde-3-phosphate dehydrogenase	(b1179)
					72.0	Triose-phosphate isomerase	(b3919)
					73.0	Formate C-acetyl transferase	(b0902) (b0903)
							(b2579) (b3114)
							(b3951) (b3952)
A_4	14.046	0.104	5	43.88	24.5	D-glucose exchange	
GDLS	15.914	0.05	14	0.14	0.0	D-glucose exchange	
OptKnock	12.565	0.1165	53	49.65	25.5	D-glucose exchange	
Strain	**Suc**	**Bm**	**k cost**	**GR**(%)	**LR**(%)	**Reaction**	**Gene(s)**
B_1	12.012	0.055	15	16.55	31.5	D-glucose exchange (Ex-glc)	
					34.0	Glyceraldehyde-3-phosphate dehydrogenase (GAPD)	(b1179)
					50.0	Triose-phosphate isomerase (TPI)	(b3919)
B_2	11.530	0.070	10	19.58	28.0	D-glucose exchange	
					28.0	Glyce1raldehyde-3-phosphate dehydrogenase	(b1179)
					52.5	Triose-phosphate isomerase	(b3919)
B_3	10.610	0.087	8	18.99	30.0	D-glucose exchange	
					32.0	Glyceraldehyde-3-phosphate dehydrogenase	(b1179)
					47.5	Triose-phosphate isomerase	(b3919)
B_4	9.284	0.093	5	22.17	18.5	D-glucose exchange	
					27.0	Glyceraldehyde-3-phosphate dehydrogenase	(b1179)
					29.0	Triose-phosphate isomerase	(b3919)
GDLS	9.727	0.05	26	0.27	0.0	D-glucose exchange	
Optknock	9.069	0.1181	54	51.92	32	D-glucose exchange	

of them, and then we calculate the function ρ. Once a value of ρ is obtained for each of the trials, we compute the value of robustness (the function Γ) which, in this case, we call *Global Robustness*. The values are shown in Table 1.

Conversely, in the Local Robustness analysis, we perturb again the upper and lower bounds of a metabolic flux, but in this case we create N trials perturbing a single flux and so we calculate the robustness for each metabolic flux. The indicative value of the *Local Robustness*, which we utilise and show in Table 1, is the absolute minimum of these values. Figure 3 highlights the local robustness results for each flux. Most of them are equal to 100. The values

Table 2: The table reports the best solutions obtained by OptFlux ([10]), OptGene ([9]), GDLS ([8]), OptKnock ([5]) and our multi-objective optimisation algorithm for maximising acetate and succinate production [mmolh^{-1} gdW^{-1}]. The last two rows provide the biomass [h^{-1}] and the knockout cost (k cost). Details are shown in Table 1.

	Wild Type	OptFlux	OptGene	GDLS	GDLS	OptKnock	OptKnock	GDMO	GDMO	GDMO
Acetate	8.30	15.129	15.138	15.914	-	-	12.565	13.797	19.150	-
		(82.3%)	(82.4%)	(91.7%)	-	-	(51.4%)	(66.20%)	(130.7%)	-
Succinate	0.077	10.007	9.874	-	9.727	9.069	-	-	-	10.610
		(12877%)	(12704%)	-	(12514%)	(12362%)	-	-	-	(13659%)
Biomass	0.23	n.a.	n.a.	0.0500	0.0500	0.1181	0.1165	0.1296	0.053	0.087
		n.a.	n.a.	(-78.4%)	(-78.4%)	(-77.9%)	(-49.6%)	(-43.91%)	(-77.10%)	(-62%)
k cost	n.a.	n.a.	n.a.	14	26	54	53	5	10	8

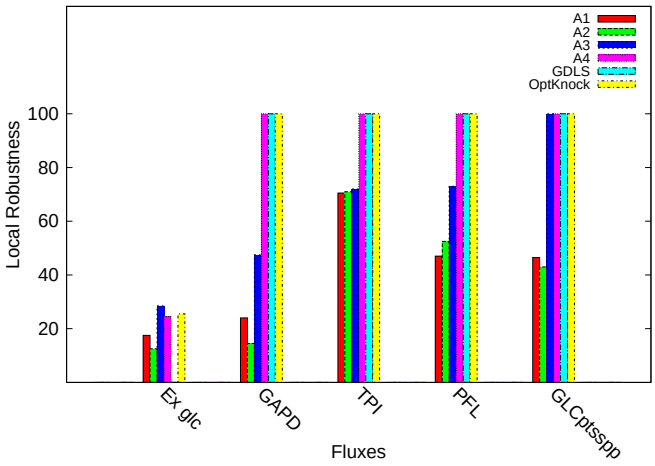

Figure 3: Local Robustness with respect to the acetate production metric. Details are shown in Table 1.

are less than 100 just for a few fluxes, which are shown in Table 1 together with the reactions and the corresponding genes.

6 Conclusions

In this paper we have highlighted a relationship between the cell and the von Neumann architecture. In particular, given a Boolean string y (considered as it is stored in the "cell memory"), a control function g_Φ translates and makes use of it to govern the cell metabolism by turning on and off the gene sets, with the aim of observing the behaviour of the chemical reaction network and optimising the cell according to multiple objective functions. This relationship is based on the mapping between the cell metabolism and a RM (equivalent to a TM). Indeed, the reactions that take place in the cell can be thought of as increment/decrement instructions of the RM, where the RM registers count the number of molecules of each metabolite.

The methodology we propose allows to optimise simultaneously several biotechnological targets, i.e. the output of the computation carried out by bacteria. This approach highlights the complex behaviour that may arise in molecular machines, and hints at future investigation of

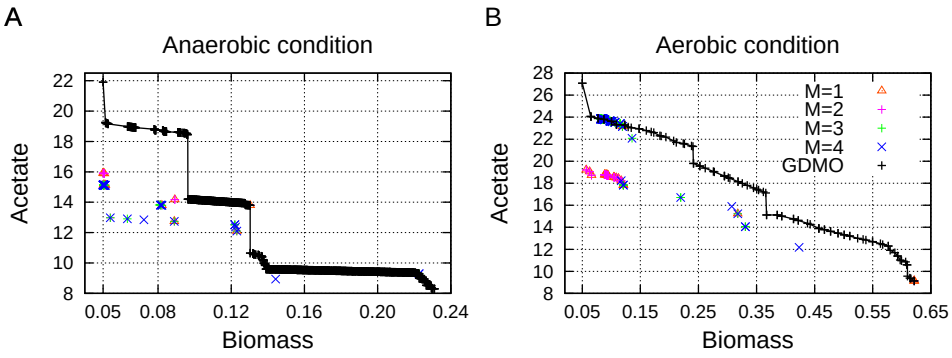

Figure 4: Maximisation of biomass formation and acetate production in anaerobic (A) and aerobic (B) conditions with glucose uptake rate 10 mmolh^{-1} gDW^{-1} in iAF1260. Pareto fronts obtained by GDMO are in black, and the results obtained by GDLS in red, purple, green and blue, set with M=1,2,3,4 and k=1,2,3,4 respectively. M and k are parameters of GDLS [8].

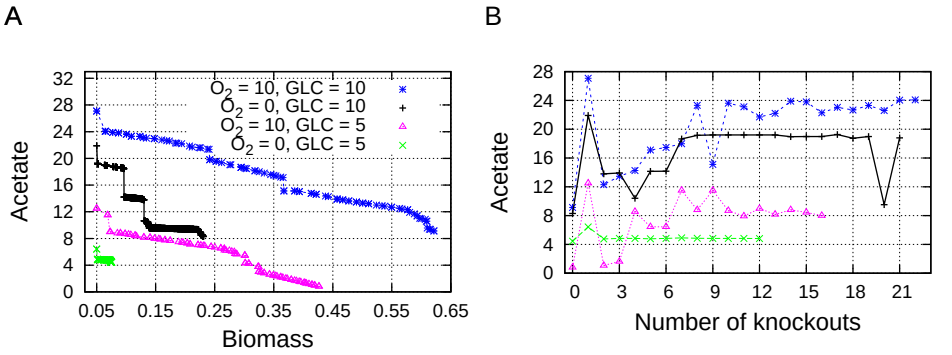

Figure 5: In A the Pareto front obtained maximising biomass [h^{-1}] and acetate production [mmolh^{-1} gdW^{-1}] in *E. coli* model, and in 4 different environmental conditions as reported in the legend. Figure B shows the number of knockouts associated to acetate production rates.

optimal multi-objective computation in bacteria. All organisms experience oscillating conditions that range from starvation to food richness. In particular, environmental changes represent the availability of different sources of food. Therefore, the simultaneous optimisation tells us what could be the capability of the organism to cope with these changes.
Although the synthetic biology is still in its infancy, we foresee the need to build and optimise synthetic organisms that produce more than one single product at a time. A program embedded in a bacterium, whose metabolism works like a TM, could be able to implement the robust knockout strategy found by GDMO and PoSA. The minimisation of the number of knockouts ensures a low-effort, reliable and reproducible result, allowing cells to become programmable manufacturers of biochemical products of interest.

References

[1] L.M. Adleman. Molecular computation of solutions to combinatorial problems. *Science*, 266(5187):1021–1024, 1994.

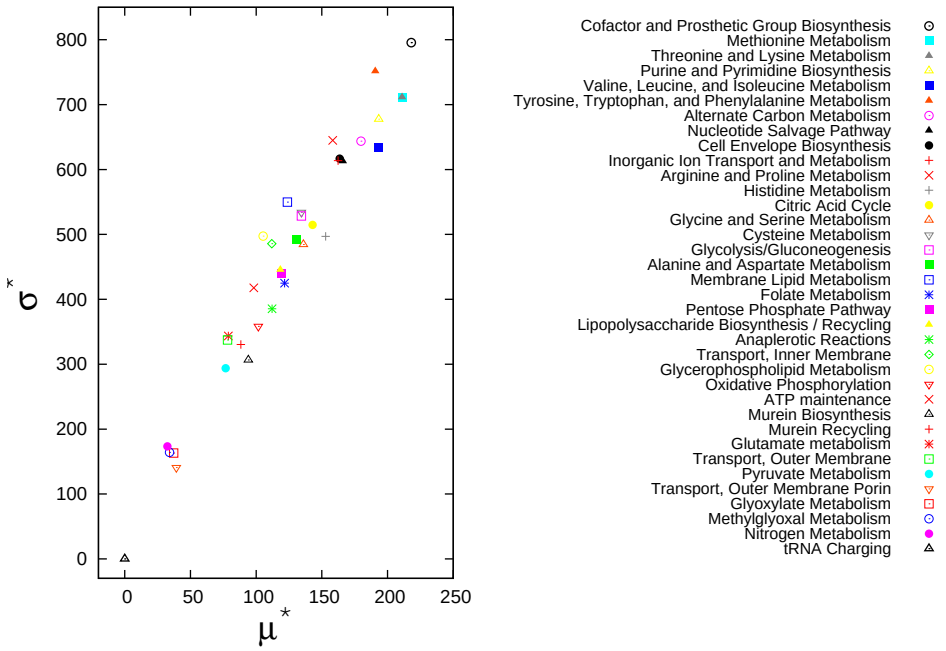

Figure 6: Sensitivity index values $[\mu*, \sigma*]$ obtained by PoSA for model of *E. coli*, iAF1260. The model is composed by 36 pathways whose reactions are clustered according to their functionality in the metabolic network.

[2] M. Amos. *Cellular Computing*. Series in Systems Biology. Oxford University Press, 2004.

[3] D. Bray et al. Protein molecules as computational elements in living cells. *Nature*, 376(6538):307–312, 1995.

[4] R. Brent and J. Bruck. 2020 computing: Can computers help to explain biology? *Nature*, 440(7083):416–417, 2006.

[5] Burgard et al. Optknock: a bilevel programming framework for identifying gene knockout strategies for microbial strain optimization. *Biotechnology and Bioengineering*, 84(6):647–657, 2003.

[6] Deb et al. A fast and elitist multiobjective genetic algorithm : NSGA-II. *Evolutionary Computation, IEEE Transactions on*, 6(2):182–197, April 2002.

[7] Feist et al. A genome-scale metabolic reconstruction for escherichia coli k-12 mg1655 that accounts for 1260 orfs and thermodynamic information. *Mol Syst Biol*, 3(121):291–301, June 2007.

[8] Lun et al. Large-scale identification of genetic design strategies using local search. *Mol Syst Biol.*, 5(296).

[9] Patil et al. Evolutionary programming as a platform for in silico metabolic engineering. *BMC Bioinformatics*, 6(1):308, 2005.

[10] Rocha et al. Natural computation meta-heuristics for the in silico optimization of microbial strains. *BMC Bioinformatics*, 9(1):499, 2008.

[11] Soloveichik et al. Computation with finite stochastic chemical reaction networks. *Natural Computing*, 7(4):615–633, 2008.

[12] B. Görke and J. Stülke. Carbon catabolite repression in bacteria: many ways to make the most out of nutrients. *Nature Reviews Microbiology*, 6(8):613–624, 2008.

[13] L.F. Landweber and L. Kari. Universal molecular computation in ciliates. *Evolution as Compu-*

tation, pages 257–274, 2003.

[14] W.A. Lim. The modular logic of signaling proteins: building allosteric switches from simple binding domains. *Current opinion in structural biology*, 12(1):61–68, 2002.

[15] Morris M.D. Factorial sampling plans for preliminary computational experiments. *Technometrics*, 33(2):161–175, May 1991.

[16] M.L. Minsky. *Computation*. Prentice-Hall, 1967.

[17] G. Nicosia, S. Rinaudo, and E. Sciacca. An evolutionary algorithm-based approach to robust analog circuit design using constrained multi-objective optimization. In Max Bramer, Frans Coenen, and Miltos Petridis, editors, *Research and Development in Intelligent Systems XXIV*, pages 7–20. Springer London, 2008. 10.1007/978-1-84800-094-02.

[18] P. Pharkya and C. Maranas. An optimization framework for identifying reaction activation/inhibition or elimination candidates for overproduction in microbial systems. *Metabolic Engineering*, 8(1):1–13, January 2006.

[19] K.E. Prehoda and W.A. Lim. How signaling proteins integrate multiple inputs: a comparison of n-wasp and cdk2. *Current opinion in cell biology*, 14(2):149–154, 2002.

[20] D. Stanton and D. White. *Constructive combinatorics*. Springer, 1986.

[21] G. Stracquadanio and G. Nicosia. Computational energy-based redesign of robust proteins. *Computers & chemical engineering*, 35(3):464–473, 2011.

[22] A.M. Turing. The chemical basis of morphogenesis. *Philosophical Transactions of the Royal Society of London. Series B, Biological Sciences*, 237(641):37–72, 1952.

[23] R. Weiss, T. Knight, and G. Sussman. Cellular computation and communication using engineered genetic regulatory networks. *Cellular computing*, pages 120–1, 2001.