Fundamenta Informaticae XXI (2018) 1001–1023
DOI 10.3233/FI-2016-0000
IOS Press

Computational Intelligence for Life Sciences

Daniela Besozzi[1], **Mauro Castelli**[2], **Paolo Cazzaniga**[3,4], **Luca Manzoni**[1,12], **Marco S. Nobile**[1,4], **Stefano Ruberto**[5,6], **Leonardo Rundo**[7,8,1], **Simone Spolaor**[1], **Andrea Tangherloni**[9,10,11,1], **Leonardo Vanneschi**[2]

[1]*Department of Informatics, Systems and Communication, University of Milano - Bicocca, Milano, Italy*

[2]*NOVA Information Management School (NOVA IMS), Universidade Nova de Lisboa, Campus de Campolide, Lisboa, Portugal*

[3]*Department of Human and Social Sciences, University of Bergamo, Bergamo, Italy*

[4]*SYSBIO.IT Centre of Systems Biology, Milano, Italy*

[5]*Penn, University of Pennsylvania, PA, USA*

[6]*GSSI, Gran Sasso Science Institute, INFN, L'Aquila, Italy*

[7]*Department of Radiology, University of Cambridge, Cambridge, UK*

[8]*Cancer Research UK Cambridge Centre, Cambridge, UK*

[9]*Department of Haematology, University of Cambridge, Cambridge, UK*

[10]*Wellcome Trust Sanger Institute, Wellcome Trust Genome Campus, Hinxton, UK*

[11]*Wellcome Trust – Medical Research Council Cambridge Stem Cell Institute, Cambridge, UK*

Abstract. Computational Intelligence (CI) is a computer science discipline encompassing the theory, design, development and application of biologically and linguistically derived computational paradigms. Traditionally, the main elements of CI are Evolutionary Computation, Swarm Intelligence, Fuzzy Logic, and Neural Networks. CI aims at proposing new algorithms able to solve complex computational problems by taking inspiration from natural phenomena. In an intriguing

Address for correspondence: Department of Informatics, Systems and Communication, University of Milano-Bicocca, Viale Sarca 336, 20126 Milano, Italy. E-mail: daniela.besozzi@unimib.it

turn of events, these nature-inspired methods have been widely adopted to investigate a plethora of problems related to nature itself. In this paper we present a variety of CI methods applied to three problems in life sciences, highlighting their effectiveness: we describe how protein folding can be faced by exploiting Genetic Programming, the inference of haplotypes can be tackled using Genetic Algorithms, and the estimation of biochemical kinetic parameters can be performed by means of Swarm Intelligence. We show that CI methods can generate very high quality solutions, providing a sound methodology to solve complex optimization problems in life sciences.

Keywords: Computational Intelligence, Evolutionary Computation, Swarm Intelligence, Genetic Programming, Genetic Algorithm, Particle Swarm Optimization, Protein Folding, Haplotype Assembly, Parameter Estimation

1. Introduction

The longstanding synergy between computer science and life sciences has always been fruitful and plenty of novel ideas. Since its inception, computer science has taken inspiration from biology and nature to define more powerful and flexible computational models. Already in the 1940s and 1950s, the observation of the natural phenomena of pattern formation and self-reproduction allowed two "fathers of computer science", Alan M. Turing and John von Neumann, to formulate the basis of the computational model known as Cellular Automata [1, 2]. The inner working of neurons in the brain inspired the creation of the perceptron [3], which is the basic component of artificial neural networks that were popularized during the current "deep learning revolution" [4]. The Darwinian theory of evolution [5] was the inspiration for the development of efficient and effective optimization methods, such as Genetic Algorithms (GAs) [6] and Genetic Programming (GP) [7]. The coordinated movement of flocks of birds and schools of fish inspired another important optimization algorithm, called Particle Swarm Optimization (PSO) [8, 9]. The process of taking inspiration from nature also considered the behavior of social insects, leading to the definition of optimization methods such as Ant Colony Optimization (ACO) [10] and Artificial Bee Colony (ABC) [11].

Noteworthy, the interplay between computer science and nature has generated a virtuous circle, in which natural phenomena are used to inspire Computational Intelligence (CI) methods that are then used to solve complex problems related to different fields of life sciences. In this survey, we focus on this two-way interaction by describing the application of different CI methods for the solution of three problems in biology, characterized by an increasing level of complexity. Namely, we move from the inference of the three-dimensional structure of single molecules (specifically, proteins), to the assembly of large sets of data related to the genome, up to the estimation of kinetic parameters that drive the temporal evolution of biochemical reaction networks. Although they are characterized by very different features and deal with distinct types of data, these problems can be stated as optimization problems and hence solved by means of nature-inspired CI methods.

As a first example of application, in Section 2 we show how GP, an evolutionary method that can produce symbolic expressions or even entire programs, can be employed to tackle the complex problem of protein folding. Protein folding consists in determining the native structure of a protein—that is, the conformation corresponding to the functional state of the molecule—by considering as input the

sequence of its aminoacids. This problem can be formulated as an optimization problem, where the best solution corresponds to the folding characterized by the minimum value of the Gibbs free energy [12]. We show here that a new GP variant based on semantic equivalence classes, that was first introduced in [13], can be exploited to derive better solutions for the protein folding problem with respect to the state-of-the-art GP.

Section 3 is dedicated to the haplotype assembly problem. We show that GAs, a search heuristic inspired by the processes of selection, mutation and genetic crossover, can be exploited to reconstruct the two distinct copies of each chromosome[1], called haplotypes. GenHap [14, 15], a novel computational method based on GAs specifically developed for the haplotyping problem, has the relevant advantage of taking into account huge datasets produced by the latest sequencing technologies. Haplotype assembly represents indeed an important problem in bioinformatics and genome analysis, since it allows for the characterization of the genome of an individual [16].

Finally, in Section 4 we show how Swarm Intelligence can be effectively applied to the parameter estimation problem, by comparing the performances of five techniques and discussing the role of reboot strategies [17, 18, 19, 20]. In the field of Systems Biology, simulating the dynamics of models of cellular processes represents one of the most effective methodologies to understand their functioning in physiological or perturbed conditions. However, the lack of kinetic rates, necessary to accurately mimic the occurrence of biochemical reactions over time, strongly limits the scope of these analyses. Parameter estimation, which consists in identifying a proper model parameterization, is a non-linear, non-convex and multi-modal optimization problem that is typically tackled by means of CI.

The examples presented in this survey provide an evidence of the effectiveness of various bio-inspired CI methods in solving hard problems in life sciences.

2. Genetic Programming for Protein Folding

Genetic Programming (GP) belongs to the CI research field called Evolutionary Computation: it consists of the automated learning of computer programs by means of a process inspired by Charles Darwin's theory of biological evolution. In the context of GP, one can interpret the word *program* in general terms, and therefore, GP can be applied to the particular cases of learning expressions, functions and, as in this work, data-driven predictive models. In GP, programs are typically encoded by defining a set, $\mathcal{F}$, of primitive functional operators and a set, $\mathcal{T}$, of terminal symbols. GP's objective is to navigate the space of all possible programs that can be constructed by composing symbols in $\mathcal{F}$ and $\mathcal{T}$, looking for the most appropriate ones to solve the problem at hand. Generation by generation, GP stochastically transforms populations of programs into new, hopefully improved, populations. The appropriateness of a solution in solving the problem (i.e., its quality) is expressed by using an objective function (the fitness function).

In order to transform a population into a new population of candidate solutions, GP selects the most promising programs that belong to the current population and applies to those programs some particular search operators called genetic operators, typically crossover (to exchange genetic material between individuals) and mutation (to add new genetic material). In their traditional form, crossover

[1] Somatic diploid human cells contain 22 different pairs of homologous chromosomes and a pair of sex chromosomes, where one copy is inherited from the mother and one from the father.

and mutation operators create new candidate solutions by performing a transformation that works at the syntactical level, where the syntax corresponds to the structure of a GP individual: new solutions are generated by exchanging subtrees among the parent solutions (crossover), or by replacing an existing subtree with a randomly generated one (mutation). Thus, standard crossover and mutation operators perform a blind transformation of the parent individuals, without taking into account the information about the *behavior* of the parent solutions. This is an important limitation because the behavior, or *semantics*, is actually what defines the output produced by a GP model. Given two parent solutions, the use of syntax-based genetic operators makes impossible, or at least complicated, to get a prediction about the behavior of the newly generated solutions. This, in turn, results in an inherently difficulty in understanding the dynamic of the underlying evolutionary process [21]. To counteract the aforementioned limitations associated with the use of syntax-based genetic operators, in recent years the scientific community started to investigate and develop genetic operators that are able to directly work on the semantics of the individuals [21]. In particular, the use of semantic methods [21, 22, 23] is one of the hottest topics in GP and has recently attracted a significant attention from researchers, particularly in the applied domain of symbolic regression [24, 25, 26, 27, 28, 29].

Let $\mathcal{X} = \{\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n\}$ be the set of input data (each of them a vector of values), or fitness cases, of a symbolic regression problem, and $\mathbf{t} = [t_1, t_2, \ldots, t_n]$ the vector of the respective expected output or target values (in other words, for each $1 \leq i \leq n$, the value t_i is the expected output corresponding to input $\mathbf{x}_i$). A GP individual (or program) P can be seen as a function that, for each input vector $\mathbf{x}_i$, returns the scalar value $P(\mathbf{x}_i)$. We call the *semantics* of P the vector $\mathbf{s}_P = [P(\mathbf{x}_1), P(\mathbf{x}_2), \ldots, P(\mathbf{x}_n)]$. This vector can be represented as a point in an n-dimensional space, which is also called the *semantic space*. It can be counterpoised to the *syntactic* or *genotypic space*, where individuals are represented by programs. The target vector $\mathbf{t}$ itself is a point in the semantic space and, in general, the objective of GP is to find at least one program in the genotypic space that maps into $\mathbf{t}$ in the semantic space.

The most significant contribution for the definition of semantics-based genetic operators for GP was presented in the work by Moraglio and coauthors [30], where genetic operators that have a direct effect on the semantics of the newly created individuals were proposed. These operators are known as Geometric Semantic Operators (GSOs), and the GP algorithm that uses these operators is known in the literature as Geometric Semantic Genetic Programming (GSGP). The main property of GSOs is that they induce a unimodal fitness landscape for any problem consisting of finding a match between outputs and known targets. Thus, GSGP can be used to explore a semantic search space that contains no local optima. An implementation of GSGP was proposed in [22, 31], and its generalization ability—an important and widely studied concept in GP [32, 33, 34, 35]—was discussed in the work by Gonçalves [36]. In particular, GSGP is characterized by a better generalization ability. Despite its ability in outperforming traditional syntax-based genetic operators over several domains and its theoretical properties [37, 38, 39, 40], GSGP is characterized by a rapid growth of the individuals when crossover and mutation are applied [30]. This is an important issue because it prevents GP practitioners to understand the final model produced by GSGP. Thus, the technique presented in Section 2.1 exploits the idea of direct semantic manipulation like GSGP, but traditional genetic operators are used. While this approach, based on *semantic equivalence classes*, results in losing the property of exploring a unimodal fitness landscape, it allows obtaining human-readable solutions that are compact in size. This is an important feature for a predictive model. The technique is a generalization of the work by Ruberto

and coauthors [41], where two optimally aligned individuals in the error space are combined into a new one that approximates the target, using a method called Error Space Alignment Genetic Programming.

In Section 2.1 we introduce the notion of GP with equivalence classes, and how it relates to operations on the semantic space. Then, two GP systems that use different definitions of equivalence class are defined. Finally, GP with equivalence classes is used for solving the biologically important problem of protein folding, comparing its performance with the one achieved by other existing GP systems.

2.1. Genetic Programming with Semantic Equivalence Classes

One recent approach to improve the search in the semantic space by means of GP is to actually perform some kind of partition of the individuals in the semantic space. The main idea is that even if two individuals might not have a good quality, there might be an automated and deterministic way to combine them to produce a higher quality individual. The easiest way to define GP with semantic equivalence classes is probably to start with a general definition of our idea of *equivalence function* (*EF*). Each of them is a function that takes two semantics as arguments and returns a vector of the same cardinality, that is, $EF : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}^n$. We say that two GP individuals P and Q are *equivalent* (i.e., they belong to the same *Equivalence Class* – EQC) if $EF(\mathbf{s}_P, \mathbf{s}_Q)$ (i.e., the result of *EF* applied to the semantics of the two individuals) is a constant vector (i.e., a vector whose components are all identical).

Starting from this definition, we are interested in understanding when a GP individual P is equivalent to a globally optimal solution (i.e., an individual whose semantics is exactly equal to the target vector $\mathbf{t}$). This happens when

$$EF(\mathbf{t}, \mathbf{s}_P) = \mathbf{k}, \tag{1}$$

where $\mathbf{k}$, for $k \in \mathbb{R}$, is the constant vector $[k, k, \ldots, k]$ of length n. Once we have a GP individual equivalent to $\mathbf{t}$, we can reconstruct an individual whose semantics is identical to $\mathbf{t}$ (i.e., a globally optimal solution) analytically. The operation is possible if the function *EF* is invertible, and it becomes trivial if *EF* is easy to invert, like for instance in the case of a linear function. In order to reconstruct a globally optimal solution, we first have to obtain $\mathbf{t}$ from Equation (1). This can be done as in Equation (2):

$$\mathbf{t} = EF^{-1}(\mathbf{k}, \mathbf{s}_P). \tag{2}$$

At this point, reconstructing the genotype of an optimal solution is straightforward: it is sufficient to combine the genotype of P and the constant k by means of the operator EF^{-1}, which represents the root node of the genotype of the optimal individual.

With this in mind, it now makes sense to investigate the possibility of defining a new GP system whose objective is finding a solution that belongs to the same EQC as the global optimum. In order to obtain this, we need to define a new fitness function, to be minimized, able to quantify the *dissimilarity* between the EQC of an individual and the EQC of the global optimum. A simple possibility is to measure the *dispersion* of the components of vector $\mathbf{k}$ in Equation (1). For this purpose, we can, for instance, use the *variance* of the components of $\mathbf{k}$.

Here, these general concepts will be instantiated by defining two different concrete *EF* functions. An important observation is that, for continuous values, it is unlikely for two individuals to be *exactly* in the same EQC. For this reason, a threshold is used to establish the EQCs. We also point out that in none of the above phases we have considered an error function (as, for instance, the Root Mean Square

Error (RMSE)) to drive the evolutionary process: the evolved individuals can have a very poor value of the error function. Finally, it is worth noticing that once we find an individual that belongs to the same EQC as another individual that is already in the population, it may be useful to reject it. After all, if we consider our system as exploring the space of EQCs, then we do not need an EQC to be represented by more than one individual. In what follows, we propose a filtering process to implement this rejection: when a new individual is generated, it is checked against all other individuals in the population and rejected if it belongs to an EQC that is already represented by at least one other individual. In that situation, a new individual is immediately generated in substitution and checked again.

GPPLUS: GP by Translation. Here, we propose the first instance of the general framework discussed so far. In this case, given two GP individuals P and Q, the equivalence function *EF* is defined as in Equation (3):

$$EF(\mathbf{s}_Q, \mathbf{s}_P) = \mathbf{s}_P + \mathbf{s}_Q. \tag{3}$$

Let us assume that one among P and Q is a globally optimal solution (say, without lost of generality, Q). Then, $\mathbf{s}_Q = \mathbf{t}$. So, Equation (3) becomes:

$$EF(\mathbf{t}, \mathbf{s}_P) = \mathbf{s}_P + \mathbf{t}. \tag{4}$$

Now, by replacing Equation (4) in Equation (1) we obtain:

$$\mathbf{s}_P + \mathbf{t} = \mathbf{k}, \tag{5}$$

where $\mathbf{k}$ is the constant vector of value $k \in \mathbb{R}$. This means that we are able to analytically reconstruct a globally optimal solution if we are able to find a solution P whose semantics is directly proportional to the target, with a constant of proportionality k. In that case, from Equation (5) we have that $\mathbf{t} = \mathbf{k} - \mathbf{s}_P$, and so the genotype of the globally optimal solution is simply a tree that has a "$-$" binary function at the root, a single node with constant k as the left subtree, and the genotype of P as the right subtree.

GPMUL: GP by proportions. In this case, for any two GP individuals P and Q, we define:

$$EF(\mathbf{s}_P, \mathbf{s}_Q) = \frac{\mathbf{s}_P}{\mathbf{s}_Q}, \tag{6}$$

and so, if Q is a globally optimal solution, we have:

$$EF(\mathbf{s}_P, \mathbf{t}) = \frac{\mathbf{s}_P}{\mathbf{t}}, \tag{7}$$

which allows us to reconstruct the target using $\mathbf{t} = \mathbf{s}_P / \mathbf{k}$, where $\mathbf{k}$ is the constant vector of value $k \in \mathbb{R}$. In this case, if GP is able to evolve an individual P such that the target multiplied by a constant k is equal to the semantics of P, then a globally optimal solution can be reconstructed by building a genotype that has the symbol of binary division as root, the genotype of P as the left subtree, and a single node with constant k as the right subtree. It is worth noticing that the event of accidentally introducing a division by zero can be simply avoided. In fact, Equation (7) contains a division by zero only if the semantics of P or the target are constant zero vectors, which, besides being an extremely rare situation, can easily be prevented with a simple test.

2.2. Protein Folding

Protein folding is a part of the problem of understanding the physicochemical properties of proteins. Proteins are polymers composed of aminoacids, which are joined together by amide links, called peptide bonds. A protein can then be considered as a sequence of different aminoacids. While it is important to know the properties of the single aminoacids, the properties of the proteins are not simply a sum of the properties of their components. An important part of the physicochemical properties of a protein derives from their three-dimensional structure. The aim of the protein folding problem is to derive the native structure of a protein, given its sequence of aminoacids.

The dataset used in this work to test the use of semantic GP for protein folding is available from the UCI machine learning repository (`http://archive.ics.uci.edu/ml/`). It consists of approximately 45000 instances, each of them with 9 attributes (total surface area, non polar exposed area, fractional area of exposed non polar residue, fractional area of exposed non polar part of residue, molecular mass weighted exposed area, average deviation from standard exposed area of residue, average deviation from standard exposed area of residue, Euclidean distance, secondary structure penalty, and spatial distribution constraints) and a single target value, defined as the root-mean-square deviation of atomic positions (RMSD).

2.3. Results

Usually GP has the advantage, with respect to many other machine learning methods, to provide models that are not complete black-boxes, whose predictions are impenetrable and cannot be understood. The individuals resulting from the GP evolutionary process can be complex but still understandable (a "grey box" in some sense). This means that GP can be applied to problems like the one of estimating the RMSD in such a way to find not only the distance between a model of a protein structure and a target model, but also to understand why the prediction works, what are the interactions between the different input attributes, and possibly gaining additional insights. Therefore, in this section we compare GPPLUS and GPMUL to their filtered counterparts (called FGPPLUS and FGPMUL, respectively). Furthermore, in order to have a more general vision on the usefulness of filters, we include in the comparison also a well-known GP system, i.e., GP with Linear Scaling (LS), and its filtered counterpart (FLS). We compare the results also with one of the most recent trends of GP, i.e., Geometric Semantic GP (GSGP), considered as the state-of-the-art GP technology in several application areas [42, 43, 44, 45, 46].

The parameters for all the problems were selected after a first tuning phase using a grid search. The selected parameters configuration was selected among the ones with the best performances and, among them, the one minimizing the running time. In particular, the population size was 200, the maximum number of generations 200, the trees were initialized with a maximum depth of 6 using the *Ramped Half-and-Halt* method (no limit was imposed on tree depth at runtime). Selection was performed by means of a tournament with a size of 4, the crossover rate was 0.9, and the mutation rate 0.1. Elitism was realized by preserving the best individual in the population. The functional set consisted of $\{+, -, \times, /\}$, where the division was protected by returning 1 in the presence of a denominator equal to 0. Finally, the terminal set consisted of n variables corresponding to the number of features in the dataset. One of the parameters was not shared among all tested methods: the filter

parameter. A large extensive study was performed to understand its impact on the performances. After this initial study, the values of the filter parameter were set to 10^{-5} for FGPPLUS, 10^{-3} for FGPMUL, and 10^{-7} for FLS, in order to obtain the best result for each of the considered methods. For all test problems 100 independent runs were performed, each one with a randomized partition into training (70%) and test (30%) data. Even if the values reported are the RMSE on the training and test data, it is important to stress that only GSGP was using the minimization of the RMSE as its target. That is, the objective is to minimize the error between our prediction and the target, the RMSD.

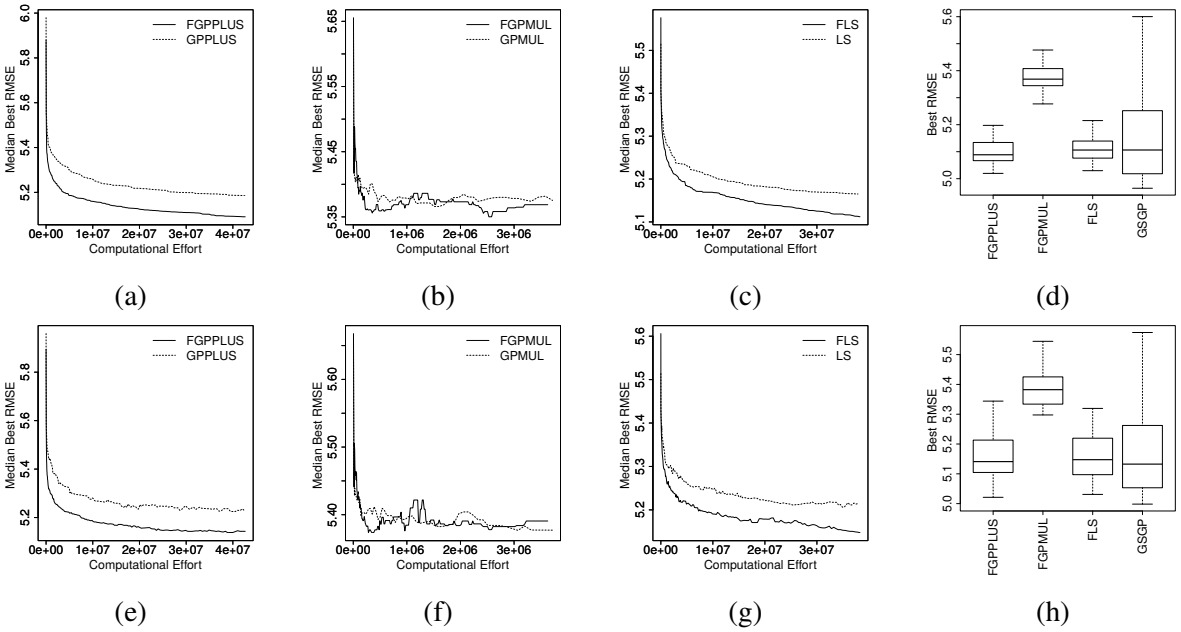

Figure 1. Plots (a)-(d) report the results on the training set. Plots (e)-(h) report the results on the test set. Plots (a) and (e) report the results of GPPLUS. Plots (b) and (f) report the results of GPMUL. Plots (c) and (g) report the results of LS. Plots (d) and (h) report the performance of GSGP, as well as those achieved by the 3 best variants (with or without filters) of the proposed methods based on equivalence classes.

The experimental results are shown in Fig. 1. It is possible to observe that for GPPLUS and LS, and their filtered counterparts, the behaviors on the training set are similar, with a gradual decrease in the error (therefore, a better fitness is obtained). In both cases the filtered version obtains better results. It is important to notice that with the increase in the computational effort the difference between a method and its filtered variant is preserved. The behavior is different for the case of GPMUL and its filtered variant. The fitness rapidly improves for both GPMUL and FGPMUL, but then further improvements are not as significant and there are some large fluctuations in the error value. Contrarily to the GPPLUS and LS cases, there is not a clear gap between GPMUL and its filtered variant. The same behaviors are observed in the training set. As it is possible to observe from the boxplots presented in Fig. 1, both GPPLUS and LS have a small variance with an average fitness comparable to GSGP that, however, has a larger variance with respect to all other methods. GPMUL has a small variance but the higher error shows that it is not able to reach the same quality of the solutions of the other methods. The filtered

versions of the methods have the same behaviors of the unfiltered ones.

The Wilcoxon rank-sum test for pairwise data comparison with a significance value of $\alpha = 0.05$ confirms that the differences between GPPLUS and FGPPLUS are statistically significant, both on the training and on the test set, while the ones between GPMUL and FGPMUL are not statistically significant. Also, the differences between LS and FLS are statistically significant. Finally, concerning plots (d) and (h) of Fig. 1, where Bonferroni correction was applied, the differences between FGPPLUS, FLS and GSGP are not statistically significant[2].

From these results, we can conclude that (1) filters are useful: both GPPLUS and LS, with filters, are competitive with the most advanced GP system, GSGP, obtaining models of comparable accuracy, but of incomparably smaller size; (2) GPPLUS, although it implements a very simple preliminary idea of equivalence function, is competitive with the state-of-the-art GP systems. In the future, we plan to define more complex equivalence functions to further improve the presented results.

3. Genetic Algorithms for Haplotype Assembly

Genetic Algorithms (GAs) are population-based optimization strategies that mimic Darwinian processes [47], where a population $\mathcal{P}$ of randomly generated individuals undergoes a selection mechanism and is iteratively modified by means of genetic operators (i.e., crossover and mutation, applied with probability p_c and p_m, respectively). In the most common and widespread formulation, each individual c_p of the population (with $p \in \{1, \ldots, |\mathcal{P}|\}$) encodes a possible solution of the given optimization problem as a fixed-length string of characters taken from a finite alphabet. The individuals characterized by better fitness values have a higher probability to be selected for the next iteration. Then, the selected individuals undergo crossover and mutation operators to possibly improve offspring and introduce new genetic material in the population.

Several methods exploiting heuristic and meta-heuristic strategies have been proposed to solve the haplotype assembly problem. Among the meta-heuristic approaches, GAs were effectively applied thanks to the discrete structure of the candidate solutions [48], which is well-suited to efficiently address the intrinsic combinatorial nature of this problem.

3.1. The Haplotype Assembly Problem

Haplotyping is the inference process that consists in assigning all the heterozygous Single Nucleotide Polymorphisms (SNPs) to exactly one of the two chromosome copies in order to fully leverage the available haplotype information. SNPs are one of the most studied genetic variations since they play a fundamental role in many biomedical applications (e.g., drug-design, disease susceptibility studies), as well as in the characterization of the expression of phenotypic traits [49]. Nowadays, computational approaches are largely used to face this problem since direct experimental reconstructions are not cost-effective [50] due to the huge number of required sequencing experiments. During the years, two completely different classes of computational methods gained ground to address this intensive task [51]. One class consists in statistical methods that infer the haplotypes of a given individual by combining the frequency by which the SNPs are usually correlated in different populations and genotypes sampled

[2]The p-values of these statistical tests are not reported due to space limits.

in a population. The other class is composed of methods that directly leverage the sequencing data; in this case, the two different haplotypes are reconstructed by partitioning the entire set of reads into two sub-sets, taking advantage of the partial overlap among them [52].

Solving the Minimum Error Correction (MEC) problem is one of the most successful approaches belonging to the second class, allowing to obtain haplotypes characterized by a low percentage of SNPs erroneously assigned. MEC computes the two haplotypes that partition the sequencing reads into two disjoint sets with the least number of corrections to the SNP values [53] but, unfortunately, it was proven to be NP-hard [54]. Afterwards, Greenberg *et al.* [55] proposed a weighted variant of MEC, named weighted MEC (wMEC), to take into account the weights representing the confidence for the presence of a sequencing error. The weights associated with each SNP value of each read are then exploited during the correction process to reduce the probability of assigning an SNP to the wrong copy. These weights are generally correlated to Phred quality score [56], which represents the probability that a given base is called incorrectly by the sequencer. However, the effectiveness and success of these methods were limited by the length of the reads produced by second-generation sequencing technologies: indeed, these sequencers produce reads that are not long enough to span over a relevant number of SNP positions. Since these short reads do not cover adjacent SNP positions adequately, the major issue was the reconstruction of short haplotype blocks [57], preventing the correct reconstruction of the full haplotypes. During the latest years, the third-generation of sequencing platforms, such as PacBio RS II (Pacific Biosciences of California Inc., Menlo Park, CA, USA) [58] and Oxford Nanopore MinION (Oxford Nanopore Ltd., Oxford, United Kingdom) [59], stood out thanks to their capacity of generating reads covering several hundreds of kilobases and spanning different SNP loci at once.

Among the different heuristics and meta-heuristics that have been developed for the haplotype assembly problem [60, 15], GenHap [14] makes use of GAs to efficiently manage large instances of the wMEC problem, obtained by taking into account the data produced by third-generation sequencing technologies. In order to deal with the computational complexity of the wMEC problem, GenHap also relies on a *divide-et-impera* approach to solve in a parallel fashion the sub-problems that are obtained by partitioning the entire problem into smaller and manageable sub-problems.

3.2. GenHap

The structure of the individuals composing the population $\mathcal{P}$ in GenHap is very simple but effective, since a partition of the fragment matrix $\mathbf{M}$ is encoded as a binary string. $\mathbf{M}$ is obtained by considering two homologous sequences belonging to a diploid organism, consisting in n positions and m reads obtained after a sequencing experiment. Each read can be easily reduced to a fragment vector $\mathbf{f} \in \{0, 1, -\}^n$, where 0 codifies the positions equal to the reference sequence (i.e., wild-type), 1 indicates the positions in which a SNP occurred, and $-$ denotes the positions that are not covered by the read. A haplotype is defined as a vector $\mathbf{h} \in \{0, 1\}^n$, representing the combination of SNPs and wild-type positions belonging to one of the two chromosomes. Hence, $\mathbf{M}$ is the $m \times n$ matrix containing all these fragments. Two distinct fragments $\mathbf{f}$ and $\mathbf{g}$ are in conflict if there exists at least a position j (with $j \in \{1, \ldots, n\}$) such that $f_j \neq g_j$ and $f_j, g_j \neq -$, otherwise they are said to be in agreement. Similarly, $\mathbf{M}$ is conflict-free if there are two different haplotypes $\mathbf{h}_1$ and $\mathbf{h}_2$ such that each row M_i (with $i \in \{1, \ldots, m\}$) is in agreement with $\mathbf{h}_1$ or $\mathbf{h}_2$. The two haplotypes $\mathbf{h}_1$ and $\mathbf{h}_2$ characterized by

the least number of corrections to the SNP values among the 2^n candidate haplotypes are reconstructed by solving the wMEC problem, that is, by partitioning the matrix $\mathbf{M}$ into two disjoint matrices $\mathbf{M}_1$ and $\mathbf{M}_2$. These two partitions are obtained by evaluating each individual of P following this simple idea: if the i-th (with $i \in \{1, \ldots, n\}$) bit is equal to 0, then the read i belongs to $\mathbf{M}_1$; otherwise, the read i belongs to $\mathbf{M}_2$. Starting from these sub-matrices, GenHap infers the two candidate haplotypes $\mathbf{h}_1$ and $\mathbf{h}_2$ for each individual of P, calculating the number of errors for each pair of candidate haplotypes. The described procedure iterates until a termination criterion is met.

Since the required execution time, as well as the problem difficulty, increase with the number of reads and SNPs, the fragment matrix $\mathbf{M}$ can be divided into sub-matrices consisting of a subset of reads. Notice that this strategy can be applied thanks to the long reads with higher coverage produced by the latest sequencing technologies. These overlapping reads allow for partitioning the initial problem into easier sub-problems, avoiding the possibility of obtaining incorrect reconstructions during the merging phase. Each sub-matrix, which corresponds to a sub-problem, is solved by an independent GA execution that converges to a solution representing the two sub-haplotypes characterized by the least number of corrections to the SNP values. Once all the sub-problem results are gathered, the pairs of sub-haplotypes are combined to obtain the entire haplotypes $\mathbf{h}_1$ and $\mathbf{h}_2$. Following this *divide-et-impera* approach, the computational complexity and burden are strongly reduced.

3.3. Results

In order to test the effectiveness of GenHap and show how the read length affects the number of haplotype blocks, we generated a set of synthetic (yet realistic) instances by using two different sequencing platforms, namely, NovaSeq (Illumina Inc., San Diego, CA, USA) [61] and Oxford Nanopore MinION [59]. The former generates short reads (here we set 150bp) while the latter produces long reads (here we set 6000bp). For both sequencing technologies, we generated different instances to collect statistically sound results varying the following parameters: (*i*) $\#\text{SNPs} \in \{500, 1000, 5000\}$; (*ii*) $\text{cov} \in \{\sim 30\times, \sim 60\times, \sim 90\times\}$; (*iii*) average $f_{\text{SNPs}} = 200$ (i.e., one SNP every 200bp exists [62, 63]). To be more precise, 10 different instances were generated for each combination of cov and $\#$SNPs. The results are evaluated considering the number of SNPs erroneously assigned ($\#\text{SNPs}_{\text{err}}$) with respect to the provided ground truth. We calculated the accuracy of GenHap as $\left(100 - 100 \cdot \frac{\#\text{SNPs}_{\text{err}}}{\#\text{SNPs}}\right)$.

As shown in Table 1, GenHap obtains better results when applied to infer the pair of haplotypes on the MinION instances, achieving an accuracy always higher than 99% with a negligible standard deviation (less than 0.5%). As a matter of fact, GenHap was conceived to deal with the data generated by future generation sequencing technologies, which produce longer reads with higher coverage with respect to the previous generations. Notice that these instances are always characterized by a single haplotype block. The results obtained by GenHap on the instances generated by the NovaSeq sequencer show that the accuracy is lower, ranging from 94% to 96% with a standard deviation up to $\sim 2\%$. Due to the short reads characterizing this sequencing platform (i.e., ~ 150bp), a high number of haplotype blocks is produced. This number increases along with the $\#$SNPs, reaching ~ 2000 blocks when $\#\text{SNPs} = 5000$ are analyzed. The coverage is generally capable of mitigating this problem: indeed, increasing the coverage allows for decreasing the number of haplotypes blocks affecting the instances generated by this technology. Finally, also the running time is heavily interested by the

Table 1. Results achieved by GenHap on the NovaSeq and MinION datasets, by considering cov $\in \{\sim 30\times, \sim 60\times, \sim 90\times\}$ and #SNPs $\in \{500, 1000, 5000\}$. The performances were evaluated both in terms of accuracy and number of detected haplotypes blocks. The obtained values are expressed as average $\pm$ standard deviation over the 10 model instances for each configuration.

		NovaSeq		MinION	
cov	#SNPs	Accuracy	Blocks	Accuracy	Blocks
$\sim 30\times$	500	$95.08\% \pm 0.99\%$	195 ± 6	$99.98\% \pm 0.07\%$	1 ± 0
	1000	$94.38\% \pm 1.39\%$	388 ± 9	$99.95\% \pm 0.07\%$	1 ± 0
	5000	$95.42\% \pm 0.51\%$	1940 ± 13	$99.85\% \pm 0.44\%$	1 ± 0
$\sim 60\times$	500	$94.94\% \pm 1.26\%$	185 ± 6	$99.94\% \pm 0.10\%$	1 ± 0
	1000	$96.58\% \pm 0.99\%$	382 ± 12	$100.00\% \pm 0.00\%$	1 ± 0
	5000	$95.34\% \pm 0.37\%$	1862 ± 21	$99.99\% \pm 0.01\%$	1 ± 0
$\sim 90\times$	500	$95.75\% \pm 1.48\%$	187 ± 4	$100.00\% \pm 0.00\%$	1 ± 0
	1000	$95.68\% \pm 1.05\%$	370 ± 9	$100.00\% \pm 0.00\%$	1 ± 0
	5000	$95.08\% \pm 0.54\%$	1843 ± 28	$99.98\% \pm 0.02\%$	1 ± 0

length of the reads, as previously shown in [15]. The shorter the reads, the higher the number of reads required to obtain the same coverage. As a matter of fact, GenHap is able to solve the instances with #SNPs $= 5000$ and cov $\simeq 90\times$ produced exploiting the MinION platform in less than 3 seconds running on 16 cores of an Intel processor based on the Skylake architecture. When the same instances generated by using the NovaSeq platform are considered, the required running time is always greater than 12 minutes.

4. Swarm Intelligence for Parameter Estimation

Differently from Evolutionary Computation techniques, Swarm Intelligence (SI) takes its inspiration from the emergent behavior of groups of living organisms. According to sociobiological investigations, some animals and social insects, like those belonging to the Hymenoptera order (e.g., ants, bees) [64], have behavioral patterns that, collectively, allow groups to self-organize, share information and perform complex tasks that a single individual would not be able to carry out [65]. SI techniques are inspired by such behaviors, and are exploited to design nature-inspired holistic optimization techniques.

Ant Colony Optimization (ACO) [66] is a SI method based on the mechanism of *stigmergy*, that is, the indirect communication between agents typical of pheromone-based ants signaling [67]. Foraging ants, indeed, tend to deposit a pheromone trail along a route leading to food. When other ants meet the pheromone trail, they tend to follow that route to reach food, reinforcing the signal. Eventually, the optimal route emerges from the collective movement of the colony, while sub-optimal trails slowly evaporate. In ACO, simulated pheromone trails are used to stochastically generate and iteratively improve a set of candidate solutions. Convergence theorems were proposed for this powerful combinatorial optimization algorithm [68], which was proven to be effective in tackling problems belonging to the NP complexity class [69, 70]. $\text{ACO}_{\mathbb{R}}$ was proposed in [71] as an extension of ACO for real-valued optimization problems, which generates new ants at each iteration by means of a probabilistic distribution. This generative distribution, based on Gaussian kernels, is calculated

according to the characteristics of α ants stored in an external archive. The archive is updated at each iteration, using the best individuals found during the fitness evaluations. A selection pressure on the ants stored in the archive can be used to influence the new generations towards the best individuals. This bias can be balanced by means of an additional parameter β. A third parameter ξ can be used to simulate pheromone evaporation of ACO, to determine how fast the low quality solutions are "forgotten".

Similarly to ACO, Artificial Bee Colony (ABC) exploits the emergent behavior of a population of virtual insects, taking inspiration from the collective behavior of foraging honey bees. This optimization method exploits a number n_e of virtual employed, n_o of onlooker, and n_s of scout bees [72], which cooperate in identifying the best food resources (i.e., solutions with the best fitness value). In particular, scouts are responsible for the exploration of the search space and become employed when they identify a promising food source (i.e., a region with good fitness). During the iterative process, onlookers are partitioned into groups that randomly exploit every food source assigned to an employed bee. ABC is a global optimization method, designed to explore real-valued search spaces, which was shown to be competitive with respect to other SI and Evolutionary Computation techniques [73].

Particle Swarm Optimization (PSO) is a SI population-based optimization meta-heuristic, inspired by the social behavior of bird flocking or fish schooling [8]. In PSO, a population (the *swarm*) of n candidate solutions (the *particles*) moves in a M-dimensional search space and cooperates to identify an optimal solution. Each particle i ($i = 1, \ldots, n$) is characterized by a position $\boldsymbol{x}_i \in \mathbb{R}^M$, and by a velocity $\mathbf{v}_i \in \mathbb{R}^M$ that is used to update its position. The PSO concept consists in changing, at each iteration, the velocity of each particle towards some attractor, typically its best position $\mathbf{b}_i \in \mathbb{R}^M$ found so far, and the global best position $\mathbf{g} \in \mathbb{R}^M$ found by the swarm. The update procedure continues until some termination criterion is met (e.g., when a maximum number of iterations $IT_{\texttt{MAX}}$ is reached). The behavior of the swarm is influenced by two parameters: the social attraction $c_{\text{soc}} \in \mathbb{R}^+$ and the cognitive attraction $c_{\text{cog}} \in \mathbb{R}^+$. These parameters control the global exploration and local exploitation of the search space, and they are weighted by two vectors $\mathbf{r}_1, \mathbf{r}_2 \in \mathbb{R}^M$ of random numbers, sampled with uniform probability in $[0, 1]$, to prevent particles from prematurely converging to local minima. The velocity of particles is also limited to a maximum magnitude $v_{\texttt{MAX}} \in \mathbb{R}^+$, and weighted by an inertia factor $w \in \mathbb{R}^+$ to avoid chaotic behaviors of the swarm. This leads to the following definition of the velocity update function for a generic i-th particle:

$$\mathbf{v}_i(t+1) = w \cdot \mathbf{v}_i(t) + c_{\text{soc}} \cdot \mathbf{r}_1 \odot (\boldsymbol{x}_i(t) - \mathbf{g}(t)) + c_{\text{cog}} \cdot \mathbf{r}_2 \odot (\boldsymbol{x}_i(t) - \mathbf{b}_i(t)), \tag{8}$$

where $\odot$ denotes the component-wise multiplication operator. Then, the position of the particle is updated by calculating $\mathbf{x}_i(t+1) = \mathbf{x}_i(t) + \mathbf{v}_i(t)$, for all $i = 1, \ldots, n$, and $0 \leq t \leq IT_{\texttt{MAX}}$. The value of w may be kept constant throughout the optimization process, or change according to some update function. Several works analyzed the performances of PSO with different settings, since they have a relevant impact on the optimization performances. For instance, some works focused on the optimal choice of parameters c_{soc} and c_{cog} [74, 75, 76, 77], or w [78]. The performances of PSO strongly depend on a proper selection of the functioning settings, in particular in the case of the parameter estimation problem [79] considered in this section. The choice of the best functioning settings is generally driven by the problem under investigation and the characteristics of the fitness landscape. Thus, to solve this issue, a dynamic tuning of PSO's settings during the optimization process by means of fuzzy reasoning is usually exploited [80, 81, 82]. Generally, the PSO versions hybridized with fuzzy

logic update the settings of *all* particles by using a common set of values determined by means of a fuzzy rule-based system. Alternative approaches—like PPSO [17] and FST-PSO [19, 18]—use fuzzy logic to automatically infer at run-time a different set of PSO settings for *each* particle of the swarm.

Despite the lack of a proper convergence theorem, PSO was successfully applied to a plethora of problems in many different disciplines [83], including parameter estimation in Systems Biology [84, 79, 85]. In particular, FST-PSO$_{\texttt{nmv}}$, a version of FST-PSO where the fuzzy rules governing minimum velocity throttling are disabled, was shown to outperform the other methods [18].

4.1. The Problem of Parameter Estimation in Systems Biology

Systems Biology aims at a system-level investigation of biological processes, by considering the complex interactions among biomolecules [86]. In this context, mathematical models and computational methods represent valuable and integrative tools to experimental biology, thanks to their capability to simulate the emergent behavior of biological processes and to elucidate the mechanisms governing their functioning [87]. A precise assignment of the kinetic parameters involved in these models is mandatory to perform accurate simulations of the dynamics, since their values determine the rate of the reactions and ultimately drive the temporal evolution of the system. Unfortunately, these parameters are generally difficult—or often impossible—to measure by means of laboratory experiments; therefore, parameters need to be estimated by using, for instance, CI methods [88]. Specifically, parameter estimation (PE) is an optimization problem that consists in the identification of the (unknown) vector $\mathbf{k}$ of kinetic parameters able to minimize the distance between any available target time series (given by, e.g., the concentration of some biochemical species that can be measured experimentally) and the simulated dynamics obtained by using the vector $\mathbf{k}$.

In what follows, we consider the mechanistic reaction-based modeling (RBM) approach for the description of biochemical reaction networks [89]. Assuming the law of mass-action [90], it is possible to derive the system of coupled Ordinary Differential Equations (ODEs) corresponding to an RBM. ODEs describe the variation over time of the species concentrations, whose dynamics can be efficiently obtained using deterministic simulators [91].

4.2. Results

Several CI algorithms have been exploited for the PE of biochemical systems [79, 88, 92]. In this section, we will focus on the SI methods mentioned above. In order to compare their performances, we generated a set of RBMs consisting in 25 species and 25 biochemical reactions. For each model size we created 6 different RBMs, and we repeated the PE 15 times with each SI method to achieve statistically sound results. ABC was implemented using the SwarmPackagePy library. We exploited Victor O. Costa's implementation of ACO$_{\mathbb{R}}$ available on GitHub[3]. We implemented PSO from scratch, using the Python programming language (v. 2.7.13) and the NumPy library (v. 1.13.3). FST-PSO and FST-PSO$_{\texttt{nmv}}$ (v. 1.4.8) were downloaded from the PyPI repository. The settings used in the PE tests are summarized in Table 2. In order to reduce the computational effort of the methodology, we leveraged

[3] `https://github.com/vctrop/ant_colony_for_continuous_domains`

Table 2. Settings of the SI methods used in the PE tests.

ABC	**ACO$_{\mathbb{R}}$**	**PSO**	**FST-PSO**	**FST-PSO$_{\texttt{nmv}}$**
$n_o = \lfloor 0.5 \cdot n \rfloor$	$\alpha = n$	$w = 0.729$	–	–
$n_e = \lfloor 0.4 \cdot n \rfloor$	$\beta = 0.1$	$c_{\text{soc}} = 1.496$		
$n_s = n - n_o - n_e$	$\xi = 0.85$	$c_{\text{cog}} = 1.496$		

the GPU-powered deterministic biochemical simulator cupSODA to execute the simulations needed to evaluate the fitness function [93, 94, 95].

Fig. 2 shows the distribution of the fitness values of the best individuals at time $t = IT_{\texttt{MAX}}$ found during each optimization run. These results show that FST-PSO$_{\texttt{nmv}}$ outperforms the other methods. ABC is also characterized by good performances in the case of Models 2, 4 and 6. Interestingly, in our tests ACO$_{\mathbb{R}}$ showed mixed performances (e.g., boxplots related to Models 3 and 4) and, with respect to the other algorithms, it also shows a small variance in the fitness value of the solution found.

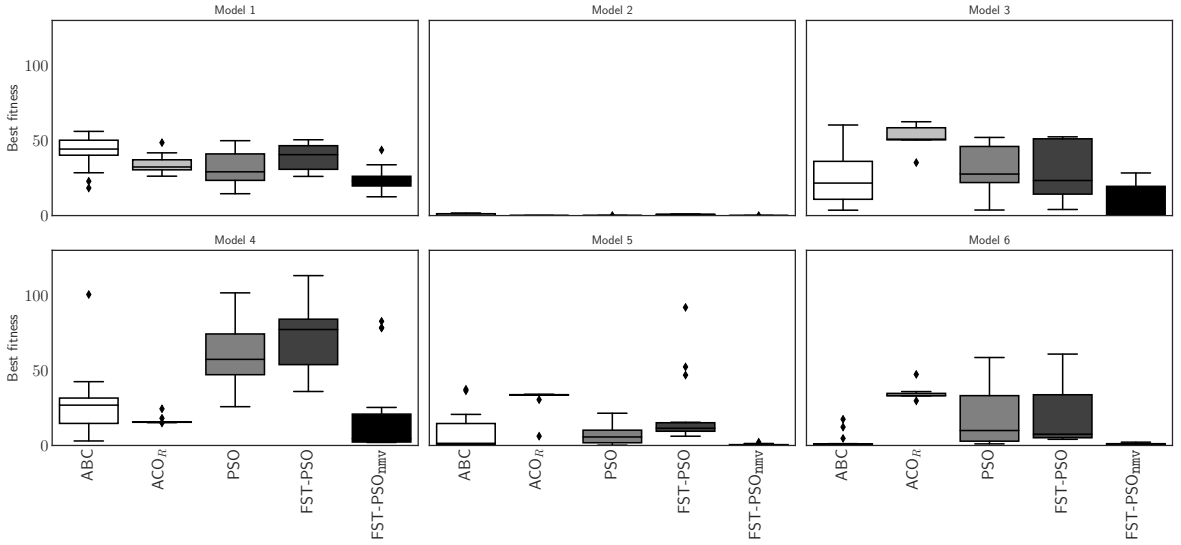

Figure 2. The box-plots show the performances of the SI methods applied to the PE of 6 RBMs consisting in 25 reactions and 25 species.

Successively, we focused on PSO and investigated the effects of different reboot strategies, that is, the possibility of resetting the position of particles in case of stagnation, thus preventing an early convergence of the swarm to local minima. A reboot strategy should accurately and efficiently identify those particles that are no longer exploring effectively the search space, and re-initialize them to increase diversity in the swarm. In the context of continuous optimization problems in large search spaces, different reboot strategies were integrated in PSO [96, 97], which consist in "restarting" the algorithm by re-initializing all particles in the swarm except for the global best position found so far. We tested three re-initialization methods on a biochemical model of Heat Shock Response in yeast

[98]: *global*, *local*, and *distance* (see [20] for further information). In these tests, we used swarms consisting in $n = 512$ particles. According to our results (Fig. 3), these reboot strategies are able to outperform the standard version of PSO, however the *local* and *global* strategies obtain considerably better performances.

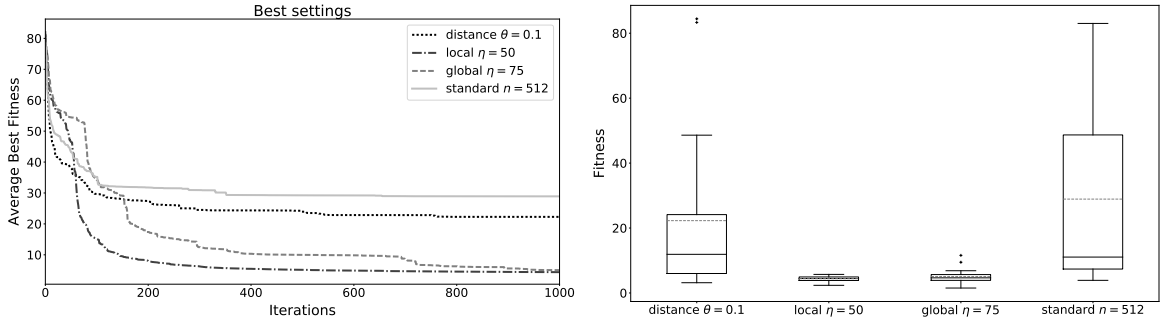

Figure 3. *Left*: Average Best Fitness achieved by the standard PSO and PSO with the three reboot strategies, performed with their respective best threshold values θ and η (see [20] for an extensive overview of the settings). *Right*: box-plots obtained considering the best fitness values reached at the last iteration of 30 optimizations, with the same settings specified above. The solid (dashed) line corresponds to the median (mean).

Due to the advantage provided by these reboot methods, we argue that the performances of FST-PSO$_{\texttt{nmv}}$ could be further improved by integrating the *local* reboot strategy. Unfortunately, the implementation of such method is not straightforward, since resetting the position of a particle could hinder the effectiveness of the fuzzy reasoner. We plan to investigate possible solutions to this problem and implement a reboot version of FST-PSO$_{\texttt{nmv}}$ in the near future, which will provide the foundation for the PE of large-scale biochemical systems as those defined, for instance, by rule-based models [99].

5. Final Remarks

The practice of watching closely the occurrence of natural phenomena, and trying to understand the general laws behind them, has always been a source of inspiration in physics, mathematics and computer science. For instance, the question of how natural selection and the laws of evolution drive the "survival of the fittest" organisms and, over millennia, have shaped the emergence of life on earth—sometimes in beautiful, astonishing ways [100]—led to the definition of efficient optimization methods that can be effectively exploited to solve complex problems in a plethora of real life applications. Similarly, the emergent behavior of social insects, which are naturally able to determine the best solution to daily jobs to ensure the viability of the whole colony (see, e.g., [101, 102]), provided several clues to computer scientists on how to translate the biological principles of communication among simple living beings, into more general foundations that might govern the interaction among abstract algorithmic agents.

In this paper we have shown three significant examples of how biologically-inspired computer science methods can help the discipline that brought them on: (1) Genetic Programming, inspired by Darwin's evolutionary theory, can help in tackling the protein folding problem; (2) the similarly inspired

Genetic Algorithms can be used in the context of the haplotype assembly problem, an interesting topic within the area of genome analysis; (3) Swarm Intelligence, inspired by the movement of flocks of birds, school of fish or social insects, can help in exploring and finding the best kinetic parameters for the simulation of biochemical systems.

Indeed, the CI methods presented here, and the life science problems that we have explored to show their effectiveness, do not represent an isolated case of how well many real world applications can take advantage of nature-inspired methods. This is only a very brief overview of how computer science and the "wet" sciences (biology, in particular) can benefit one another.

However, Evolutionary Computation and Swarm Intelligence methods share a common drawback: being iterative and population-based algorithms, they require a massive amount of fitness evaluations. This issue often originates a computational challenge, which prevents the use of CI in several scientific and engineering domains. There are two potential directions to mitigate this problem. On the one hand, since the fitness evaluations of the individuals in a population during each generation are mutually independent, CI algorithms can be accelerated by means of a distributed or parallel architecture [92, 85, 95, 103, 104], or leveraging high-performance simulators to reduce the running times of fitness evaluations [105, 106]. On the other hand, surrogate models could be used in place of actual fitness evaluations. Surrogate models are created by using sparse samples of the real fitness function and are supposed to be orders of magnitude faster to be calculated, providing a clear performance advantage. Interestingly, surrogate models can be implemented using Genetic Programming [107], originating a *weird embedding* of CI methodologies pursuing radically different goals.

Acknowledgements

This work was partially supported by national funds through FCT (Fundação para a Cîencia e a Tecnologia) Portugal, under projects DSAIPA/DS/0022/2018 (GADgET) and PTDC/CCI-INF/29168/2017 (BINDER).

References

[1] Turing AM. The chemical basis of morphogenesis. *Phil. Trans. R. Soc. Lond. B*, 1952. **237**(641):37–72.

[2] von Neumann J, Burks AW. Theory of self-reproducing automata. University of Illinois Press, Urbana, IL, USA, 1966.

[3] Rosenblatt F. The perceptron, a perceiving and recognizing automaton (Project Para). Cornell Aeronautical Laboratory, 1957.

[4] Goodfellow I, Bengio Y, Courville A, Bengio Y. Deep learning. The MIT Press, Cambridge, MA, USA, 1 edition, 2016.

[5] Darwin C. The Origin of Species by means of Natural Selection; or, the Preservation of Favoured Races in the Struggle for Life. John Murray, London, UK, 6th edition, 1872.

[6] Holland JH. Adaptation in Natural and Artificial Systems. The University of Michigan Press, Ann Arbor, MI, USA, 1975.

[7] Koza J. Genetic Programming: On the Programming of Computers by means of Natural Selection. The MIT Press, Cambridge, MA, USA, 1992.

[8] Kennedy J, Eberhart R. Particle swarm optimization. In: Proc. Int. Conf. Neural Networks, volume 4. IEEE, 1995 pp. 1942–1948.

[9] Shi Y, Eberhart RC. Parameter selection in particle swarm optimization. In: Porto VW, Saravanan N, Waagen D, Eiben AE (eds.), Evolutionary Programming VII, volume 1447 of *LNCS*. Springer-Verlag, 1998 pp. 591–600.

[10] Dorigo M, Birattari M. Ant colony optimization. In: Encyclopedia of Machine Learning, pp. 36–39. Springer, 2011.

[11] Pham DT, Ghanbarzadeh A, Koç E, Otri S, Rahim S, Zaidi M. The bees algorithm–a novel tool for complex optimisation problems. In: Intelligent Production Machines and Systems, pp. 454–459. Elsevier, 2006.

[12] Nelson DL, Cox MM. Lehninger Principles of Biochemistry. WH Freeman & Company, New York, NY, USA, 4th edition, 2004.

[13] Ruberto S, Vanneschi L, Castelli M. Genetic programming with semantic equivalence classes. *Swarm Evol. Comput.*, 2019. **44**:453–469.

[14] Tangherloni A, Spolaor S, Rundo L, Nobile MS, Cazzaniga P, Mauri G, Liò P, Merelli I, Besozzi D. GenHap: a novel computational method based on genetic algorithms for haplotype assembly. *BMC Bioinform.*, 2019. **20(Suppl 4)**:172.

[15] Tangherloni A, Rundo L, Spolaor S, Nobile MS, Merelli I, Besozzi D, Mauri G, Cazzaniga P, Liò P. High Performance Computing for Haplotyping: models and Platforms. In: Aldinucci M, Padovani L, Torquati M (eds.), Euro-Par 2018 Workshops, volume 11339 of *LNCS*. Springer, 2019 p. 650–661.

[16] Levy S, Sutton G, Ng P, Feuk L, Halpern A, Walenz B, Axelrod N, Huang J, Kirkness E, Denisov G, et al. The diploid genome sequence of an individual human. *PLoS Biol.*, 2007. **5**(10):e254.

[17] Nobile MS, Pasi G, Cazzaniga P, Besozzi D, Colombo R, Mauri G. Proactive particles in swarm optimization: a self-tuning algorithm based on fuzzy logic. In: Proc. Int. Conf. Fuzzy Systems. IEEE, 2015 pp. 1–8.

[18] Tangherloni A, Rundo L, Nobile MS. Proactive Particles in Swarm Optimization: a settings-free algorithm for real-parameter single objective optimization problems. In: Proc. Congr. Evolutionary Computation. IEEE, 2017 pp. 1940–1947.

[19] Nobile MS, Cazzaniga P, Besozzi D, Colombo R, Mauri G, Pasi G. Fuzzy Self-Tuning PSO: A settings-free algorithm for global optimization. *Swarm Evol. Comput.*, 2018. **39**:70–85.

[20] Spolaor S, Tangherloni A, Rundo L, Nobile MS, Cazzaniga P. Reboot strategies in particle swarm optimization and their impact on parameter estimation of biochemical systems. In: Proc. Conf. Computational Intelligence in Bioinformatics and Computational Biology. IEEE, 2017 pp. 1–8.

[21] Vanneschi L, Castelli M, Silva S. A survey of semantic methods in genetic programming. *Genet. Program. Evolvable Mach.*, 2014. **15**(2):195–214.

[22] Castelli M, Silva S, Vanneschi L. A C++ framework for geometric semantic genetic programming. *Genet. Program. Evolvable Mach.*, 2015. **16**(1):73–81.

[23] Castelli M, Trujillo L, Vanneschi L, Silva S, Z-Flores E, Legrand P. Geometric semantic genetic programming with local search. In: Proc. Annual Conf. Genetic and Evolutionary Computation, GECCO '15. ACM, 2015 pp. 999–1006.

[24] Popovic A, Castelli M, Vanneschi L. Predicting burned areas of forest fires: an artificial intelligence approach. *Fire Ecol.*, 2015. **11**(1):106–118.

[25] Castelli M, Castaldi D, Giordani I, Silva S, Vanneschi L, Archetti F, Maccagnola D. An efficient implementation of geometric semantic genetic programming for anticoagulation level prediction in pharmacogenetics. In: Proc. Portuguese Conf. Artificial Intelligence. Springer, 2013 pp. 78–89.

[26] Hajek P, Henriques R, Castelli M, Vanneschi L. Forecasting performance of regional innovation systems using semantic-based genetic programming with local search optimizer. *Comput. Oper. Res.*, 2019. **106**:179–190.

[27] Castelli M, Vanneschi L, De Felice M. Forecasting short-term electricity consumption using a semantics-based genetic programming framework: the South Italy case. *Energy Econom.*, 2015. **47**:37–41.

[28] Enríquez-Zárate J, Trujillo L, de Lara S, Castelli M, Z-Flores EZ, Muñoz L, Popovič A. Automatic modeling of a gas turbine using genetic programming: An experimental study. *Applied Soft Computing*, 2017. **50**:212–222.

[29] Castelli M, Vanneschi L, Popovič A. Parameter evaluation of geometric semantic genetic programming in pharmacokinetics. *International Journal of Bio-Inspired Computation*, 2016. **8**(1):42–50.

[30] Moraglio A, Krawiec K, Johnson CG. Geometric semantic genetic programming. In: International Conference on Parallel Problem Solving from Nature. Springer, 2012 pp. 21–31.

[31] Vanneschi L, Castelli M, Manzoni L, Silva S. A new implementation of geometric semantic GP and its application to problems in pharmacokinetics. In: European Conference on Genetic Programming. Springer, 2013 pp. 205–216.

[32] Vanneschi L, Castelli M, Silva S. Measuring bloat, overfitting and functional complexity in genetic programming. In: Proceedings of the 12th Annual Conference on Genetic and Evolutionary Computation. ACM, 2010 pp. 877–884.

[33] Castelli M, Manzoni L, Silva S, Vanneschi L. A comparison of the generalization ability of different genetic programming frameworks. In: IEEE Congress on Evolutionary Computation. IEEE, 2010 pp. 1–8.

[34] Castelli M, Manzoni L, Silva S, Vanneschi L. A quantitative study of learning and generalization in genetic programming. In: European Conference on Genetic Programming. Springer, 2011 pp. 25–36.

[35] Castelli M, Vanneschi L, Silva S. Semantic search-based genetic programming and the effect of intron deletion. *IEEE Transactions on Cybernetics*, 2014. **44**(1):103–113.

[36] Gonçalves I, Silva S, Fonseca CM, Castelli M. Unsure when to stop?: ask your semantic neighbors. In: Proceedings of the Genetic and Evolutionary Computation Conference. ACM, 2017 pp. 929–936.

[37] Castelli M, Trujillo L, Vanneschi L, Popovič A. Prediction of relative position of CT slices using a computational intelligence system. *Applied Soft Computing*, 2016. **46**:537–542.

[38] Vanneschi L, Castelli M, Costa E, Re A, Vaz H, Lobo V, Urbano P. Improving maritime awareness with semantic genetic programming and linear scaling: Prediction of vessels position based on AIS data. In: Mora AM, Squillero G (eds.), Applications of Evolutionary Computation. Springer International Publishing, Cham, 2015 pp. 732–744.

[39] Gonçalves I, Silva S, Fonseca CM, Castelli M. Arbitrarily close alignments in the error space: A geometric semantic genetic programming approach. In: Proceedings of the 2016 on Genetic and Evolutionary Computation Conference Companion. ACM, 2016 pp. 99–100.

[40] Castelli M, Manzoni L, Vanneschi L, Silva S, Popovič A. Self-tuning geometric semantic genetic programming. *Genetic Programming and Evolvable Machines*, 2016. **17**(1):55–74.

[41] Ruberto S, Vanneschi L, Castelli M, Silva S. ESAGP – A semantic GP framework based on alignment in the error space. In: Nicolau M, Krawiec K, Heywood MI, Castelli M, García-Sánchez P, Merelo JJ, Rivas Santos VM, Sim K (eds.), Genetic Programming. Springer Berlin Heidelberg, Berlin, Heidelberg, 2014 pp. 150–161.

[42] Castelli M, Henriques R, Vanneschi L. A geometric semantic genetic programming system for the electoral redistricting problem. *Neurocomputing*, 2015. **154**:200–207.

[43] Castelli M, Trujillo L, Vanneschi L. Energy consumption forecasting using semantic-based genetic programming with local search optimizer. *Comput. Intell. Neurosci.*, 2015. **2015**. Article ID 971908.

[44] Castelli M, Vanneschi L, Manzoni L, Popovič A. Semantic genetic programming for fast and accurate data knowledge discovery. *Swarm Evol. Comput.*, 2016. **26**:1–7.

[45] Castelli M, Silva S, Vanneschi L, Cabral A, Vasconcelos MJ, Catarino L, Carreiras JM. Land cover/land use multiclass classification using GP with geometric semantic operators. In: Proc. Eur. Conf. Applications of Evolutionary Computation. Springer, 2013 pp. 334–343.

[46] Castelli M, Vanneschi L, Silva S, Ruberto S. How to exploit alignment in the error space: two different GP models. In: Genetic Programming Theory and Practice XII, pp. 133–148. Springer, 2015.

[47] Goldberg DE. Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1 edition, 1989.

[48] Rundo L, Tangherloni A, Nobile MS, Militello C, Besozzi D, Mauri G, Cazzaniga P. MedGA: a novel evolutionary method for image enhancement in medical imaging systems. *Expert Syst. Appl.*, 2019. **119**:387–399.

[49] Hirschhorn JN, Daly MJ. Genome-wide association studies for common diseases and complex traits. *Nat. Rev. Genet.*, 2005. **6**(2):95.

[50] Kuleshov V, Xie D, Chen R, Pushkarev D, Ma Z, Blauwkamp T, Kertesz M, Snyder M. Whole-genome haplotyping using long reads and statistical methods. *Nat. Biotech.*, 2014. **32**(3):261–266.

[51] Snyder M, Adey A, Kitzman J, Shendure J. Haplotype-resolved genome sequencing: experimental methods and applications. *Nat. Rev. Genet.*, 2015. **16**(6):344–358.

[52] Patterson M, Marschall T, Pisanti N, Van Iersel L, Stougie L, Klau GW, Schönhuth A. WhatsHap: weighted haplotype assembly for future-generation sequencing reads. *J. Comput. Biol.*, 2015. **22**(6):498–509.

[53] Wang R, Wu L, Li Z, Zhang X. Haplotype reconstruction from SNP fragments by minimum error correction. *Bioinformatics*, 2005. **21**(10):2456–2462.

[54] Lippert R, Schwartz R, Lancia G, Istrail S. Algorithmic strategies for the single nucleotide polymorphism haplotype assembly problem. *Brief. Bioinform.*, 2002. **3**(1):23–31.

[55] Greenberg H, Hart W, Lancia G. Opportunities for combinatorial optimization in computational biology. *INFORMS J. Comput.*, 2004. **16**(3):211–231.

[56] Ewing B, Hillier L, Wendl MC, Green P. Base-calling of automated sequencer traces using Phred. I. Accuracy assessment. *Genome Res.*, 1998. **8**(3):175–185.

[57] Zhang K, Calabrese P, Nordborg M, Sun F. Haplotype block structure and its applications to association studies: power and study designs. *Am. J. Hum. Genet.*, 2002. **71**(6):1386–1394.

[58] Rhoads A, Au K. PacBio sequencing and its applications. *Genomics Proteomics Bioinformatics*, 2015. **13**(5):278–289.

[59] Jain M, Fiddes I, Miga K, Olsen H, Paten B, Akeson M. Improved data analysis for the MinION nanopore sequencer. *Nat. Methods*, 2015. **12**(4):351.

[60] Choi Y, Chan AP, Kirkness E, Telenti A, Schork NJ. Comparison of phasing strategies for whole human genomes. *PLoS Genet.*, 2018. **14**(4):e1007308.

[61] Quail MA, Kozarewa I, Smith F, Scally A, Stephens PJ, Durbin R, Swerdlow H, Turner DJ. A large genome center's improvements to the Illumina sequencing system. *Nat. Methods*, 2008. **5**(12):1005.

[62] Nachman MW. Single nucleotide polymorphisms and recombination rate in humans. *Trends Genet.*, 2001. **17**(9):481–485.

[63] Gabriel SB, Schaffner SF, Nguyen H, Moore JM, Roy J, Blumenstiel B, Higgins J, DeFelice M, Lochner A, Faggart M, et al. The structure of haplotype blocks in the human genome. *Science*, 2002. **296**(5576):2225–2229.

[64] Hölldobler B, Wilson EO. The Superorganism: The Beauty, Elegance, and Strangeness of Insect Societies. WW Norton & Company, New York, NY, USA, 2008.

[65] Seeley TD. The honey bee colony as a superorganism. *Am. Scientist*, 1989. **77**(6):546–553.

[66] Dorigo M, Bonabeau E, Theraulaz G. Ant algorithms and stigmergy. *Future Gener. Comput. Syst.*, 2000. **16**(8):851–871.

[67] Theraulaz G, Bonabeau E. A brief history of stigmergy. *Artif. Life*, 1999. **5**(2):97–116.

[68] Gutjahr WJ. ACO algorithms with guaranteed convergence to the optimal solution. *Inf. Proc. Lett.*, 2002. **82**(3):145–153.

[69] Stützle T, Dorigo M. ACO algorithms for the traveling salesman problem. In: Miettinen K, Neittaanmäki P, Mäkelä MM, Periaux J (eds.), Evolutionary Algorithms in Engineering and Computer Science, pp. 163–183. Wiley, 1999.

[70] Li Y, Xul Z. An ant colony optimization heuristic for solving maximum independent set problems. In: Proc. Fifth Int. Conf. Computational Intelligence and Multimedia Applications. IEEE, 2003 pp. 206–211.

[71] Socha K, Dorigo M. Ant colony optimization for continuous domains. *Eur. J. Oper. Res.*, 2008. **185**(3):1155–1173.

[72] Karaboga D, Basturk B. A powerful and efficient algorithm for numerical function optimization: artificial bee colony (ABC) algorithm. *J. Glob. Optim.*, 2007. **39**(3):459–471.

[73] Karaboga D, Akay B. A comparative study of artificial bee colony algorithm. *Appl. Math. Comput.*, 2009. **214**(1):108–132.

[74] Cagnoni S, Vanneschi L, Azzini A, Tettamanzi AGB. A critical assessment of some variants of particle swarm optimization. In: et al MG (ed.), Proc. Workshops Applications of Evolutionary Computing, LNCS. 2008 pp. 565–574.

[75] Arumugam MS, Rao MVC. On the improved performances of the particle swarm optimization algorithms with adaptive parameters, cross-over operators and root mean square (RMS) variants for computing optimal control of a class of hybrid systems. *J. Appl. Soft Comput.*, 2008. **8**(1):324–336.

[76] Valle YD, Venayagamoorthy G, Mohagheghi S, Hernandez J, Harley R. Particle Swarm Optimization: Basic Concepts, Variants and Applications in Power Systems. *IEEE Transactions on Evolutionary Computation*, 2008. **12**(2):171–195.

[77] Rundo L, Tangherloni A, Militello C, Gilardi MC, Mauri G. Multimodal medical image registration using particle swarm optimization: a review. In: Proc. Symp. Series Computational Intelligence. IEEE, 2016 pp. 1–8.

[78] Chatterjee A, Siarry P. Nonlinear inertia weight variation for dynamic adaptation in particle swarm optimization. *Comput. Oper. Res.*, 2006. **33**(3):859–871.

[79] Dräger A, Kronfeld M, Ziller M, Supper J, Planatscher H, Magnus J. Modeling metabolic networks in *C. glutamicum*: a comparison of rate laws in combination with various parameter optimization strategies. *BMC Syst. Biol.*, 2009. **3**:5.

[80] Shi Y, Eberhart R. Fuzzy adaptive Particle Swarm Optimization. In: Proc. Congr. Evolutionary Computation, volume 1. IEEE, 2001 pp. 101–106.

[81] Abdelbar AM, Abdelshahid S, Wunsch DC. Fuzzy PSO: a generalization of particle swarm optimization. In: Proc. Int. Joint Conf. Neural Networks, volume 2. IEEE, 2005 pp. 1086–1091.

[82] Abraham A, Liu H. Turbulent particle swarm optimization using fuzzy parameter tuning. In: Abraham A, Hassanien A, Siarry P, Engelbrecht A (eds.), Foundations of Computational Intelligence, volume 3, pp. 291–312. Springer, Berlin, Germany, 2009.

[83] Poli R, Kennedy J, Blackwell T. Particle swarm optimization. *Swarm Intell.*, 2007. **1**(1):33–57.

[84] Nobile MS, Tangherloni A, Rundo L, Spolaor S, Besozzi D, Mauri G, Cazzaniga P. Computational intelligence for parameter estimation of biochemical systems. In: 2018 IEEE Congress on Evolutionary Computation (CEC). IEEE, 2018 pp. 1–8.

[85] Nobile MS, Besozzi D, Cazzaniga P, Mauri G, Pescini D. Estimating reaction constants in stochastic biological systems with a multi-swarm PSO running on GPUs. In: Soule T (ed.), Proc. 14th Int. Conf. Genetic and Evolutionary Computation Conference Companion. ACM, 2012 pp. 1421–1422.

[86] Kitano H. Systems biology: a brief overview. *Science*, 2002. **295**(5560):1662–1664.

[87] Cazzaniga P, Damiani C, Besozzi D, Colombo R, Nobile MS, Gaglio D, Pescini D, Molinari S, Mauri G, Alberghina L, et al. Computational strategies for a system-level understanding of metabolism. *Metabolites*, 2014. **4**(4):1034–1087.

[88] Moles C, Mendes P, Banga J. Parameter estimation in biochemical pathways: a comparison of global optimization methods. *Genome Res.*, 2003. **13**(11):2467–2474.

[89] Besozzi D. Reaction-based models of biochemical networks. In: Beckmann A, Bienvenu L, Jonoska N (eds.), Pursuit of the Universal. Proc. 12th Conf. Computability in Europe, CiE 2016, number 9709 in LNCS. Springer International Publishing, Switzerland, 2016 pp. 24–34.

[90] Chellaboina V, Bhat S, Haddad W, Bernstein D. Modeling and analysis of mass-action kinetics. *IEEE Control Syst.*, 2009. **29**(4):60–78.

[91] Nobile MS, Cazzaniga P, Tangherloni A, Besozzi D. Graphics processing units in bioinformatics, computational biology and systems biology. *Brief. Bioinform.*, 2017. **18**(5):870–885.

[92] Nobile MS, Besozzi D, Cazzaniga P, Mauri G, Pescini D. A GPU-based multi-swarm PSO method for parameter estimation in stochastic biological systems exploiting discrete-time target series. In: Giacobini M, Vanneschi L, Bush W (eds.), Proc. 7th Eur. Conf. Evolutionary Computation, Machine Learning and Data Mining in Bioinformatics, volume 7246 of *LNCS*. Springer-Verlag, 2012 pp. 74–85.

[93] Nobile MS, Besozzi D, Cazzaniga P, Mauri G. GPU-accelerated simulations of mass-action kinetics models with cupSODA. *J. Supercomput.*, 2014. **69**(1):17–24.

[94] Nobile MS, Besozzi D, Cazzaniga P, Mauri G, Pescini D. cupSODA: a CUDA-powered simulator of mass-action kinetics. In: Malyshkin V (ed.), Proc. 12th Int. Conf. Parallel Computing Technologies, volume 7979 of *LNCS*, pp. 344–357. Springer, 2013.

[95] Nobile MS, Tangherloni A, Besozzi D, Cazzaniga P. GPU-powered and settings-free parameter estimation of biochemical systems. In: Proc. Congr. Evolutionary Computation. IEEE, 2016 pp. 32–39.

[96] García-Nieto J, Alba E. Restart particle swarm optimization with velocity modulation: a scalability test. *Soft Comput.*, 2011. **15**(11):2221–2232.

[97] De Oca MAM, Stutzle T, Birattari M, Dorigo M. Frankenstein's PSO: a composite particle swarm optimization algorithm. *IEEE Trans. Evol. Comput.*, 2009. **13**(5):1120–1132.

[98] Petre I, Mizera A, Hyder CL, Meinander A, Mikhailov A, Morimoto RI, Sistonen L, Eriksson JE, Back RJ. A simple mass-action model for the eukaryotic heat shock response and its mathematical validation. *Nat. Comput.*, 2011. **10**(1):595–612.

[99] Harris LA, Nobile MS, Pino JC, Lubbock AL, Besozzi D, Mauri G, Cazzaniga P, Lopez CF. GPU-powered model analysis with PySB/cupSODA. *Bioinformatics*, 2017. **33**(21):3492–3494.

[100] Simon M. The Wasp that Brainwashed the Caterpillar: Evolution's Most Unbelievable Solutions to Life's Biggest Problems. Penguin Books, New York, NY, USA, 2016.

[101] Hölldobler B, Wilson EO. The Superorganism: The Beauty, Elegance, and Strangeness of Insect Societies. W. W. Norton & Company, New York, NY, USA, 2008.

[102] Tautz J. The Buzz about Bees. Biology of a Superorganism. Springer-Verlag, Berlin Heidelberg, Germany, 2008.

[103] Tangherloni A, Nobile MS, Cazzaniga P. GPU-powered bat algorithm for the parameter estimation of biochemical kinetic values. In: Proc. Conf. Computational Intelligence in Bioinformatics and Computational Biology. IEEE, 2016 pp. 1–6.

[104] Ramazzotti D, Nobile MS, Cazzaniga P, Mauri G, Antoniotti M. Parallel implementation of efficient search schemes for the inference of cancer progression models. In: Proc. Conf. Computational Intelligence in Bioinformatics and Computational Biology. IEEE, 2016 pp. 1–6.

[105] Tangherloni A, Nobile MS, Besozzi D, Mauri G, Cazzaniga P. LASSIE: simulating large-scale models of biochemical systems on GPUs. *BMC Bioinform.*, 2017. **18**(1):246.

[106] Nobile MS, Cazzaniga P, Besozzi D, Pescini D, Mauri G. cuTauLeaping: A GPU-powered tau-leaping stochastic simulator for massive parallel analyses of biological systems. *PLoS ONE*, 2014. **9**(3):e91963.

[107] Díaz-Manríquez A, Toscano G, Barron-Zambrano JH, Tello-Leal E. A review of surrogate assisted multiobjective evolutionary algorithms. *Comput. Intell. Neurosci.*, 2016. **2016**.