The Foundation of Evolutionary Petri Nets*

Marco S. Nobile[1], Daniela Besozzi[2], Paolo Cazzaniga[3], Giancarlo Mauri[1]

[1] Università degli Studi di Milano-Bicocca, Dipartimento di Informatica, Sistemistica e Comunicazione
Viale Sarca 336, 20126 Milano (Italy)
E-mail: {nobile,mauri}@disco.unimib.it
[2] Università degli Studi di Milano, Dipartimento di Informatica
Via Comelico 39, 20135 Milano (Italy)
E-mail: besozzi@di.unimi.it
[3] Università degli Studi di Bergamo, Dipartimento di Scienze Umane e Sociali
Piazzale S. Agostino 2, 24129 Bergamo (Italy)
E-mail: paolo.cazzaniga@unibg.it

Abstract. Evolutionary Computation (EC) mimics evolution processes to solve burdensome computational problems, like the design, optimization and reverse engineering of complex systems, and its effectiveness is tied to a proper formalization of the candidate solutions. Petri Net (PN) formalism is extensively exploited for the modeling, simulation and analysis of the structural and behavioral properties of complex systems. Here we introduce a novel evolutionary algorithm inspired by EC, the Evolutionary Petri Net (EPN), which is based on an extended class of PNs, called Resizable Petri Net (RPN), provided with two genetic operators: mutation and crossover. RPN includes the new concept of hidden places and transitions, that are used by the genetic operators for the optimization of PN-based models. We present a potential application of EPNs to face one of the most challenging problems in Systems Biology, the reverse engineering of biochemical reaction networks.

1 Introduction

Evolution is the process whereby the inherited characteristics of a population are modified over successive generations; these modifications help the offspring of the best fitting individuals in their competition for sustenance resources and survival. In other words, the evolution process is the way a population adapts itself in order to better survive in a complex and ever changing environment.

Evolutionary Computation (EC) is the application of these principles to solve complex computational problems [1], when classic deterministic approaches may results unfeasible due to a prohibitive computational effort, for instance in the case of high-dimensional combinatorial problems. In the EC field, the individuals forming a population correspond to a set of candidate solutions (smaller than the whole set of possible solutions) which are iteratively improved by means of an evolutive pressure, driven by a fitness function that quantifies the quality of

* Partially supported by Research Infrastructure SYSBIO Centre of Systems Biology

G. Balbo and M. Heiner (Eds.): BioPPN 2013, a satellite event of PETRI NET 2013,
CEUR Workshop Proceedings Vol. 988, 2013.

each individual. The improvement is determined by the application of genetic operators (e.g., mutation, crossover), which modify the structure of each individual, allowing the exploration of the search space and the convergence toward an optimal solution.

Intuitively, the structure of the individuals, and the way genetic operators are implemented, have a deep impact on the evolutionary process. In the case of distributed, asynchronous and concurrent systems, one of the most common modeling approach consists in the use of Petri Nets (PNs), that is, bipartite graphs that allow the analysis of the structural properties of the system and of its dynamic behavior [2]. PNs have been coupled to EC to solve problems in a plethora of different fields, e.g., job scheduling optimization [3], development of robust and flexible manufacturing systems [4], learning and classification [5]. PNs have also been exploited to solve complex biological problems, such as the discovery of PN models of biochemical systems consistent with population level genetic models [6] or the automatic reverse engineering of kinetic metabolic pathways [7], that is, the automatic creation of a model of a biological system able to explain some experimentally observed phenomena. EC have been widely applied to the latter problem [8–11]. Nevertheless, only a few works have exploited a PN representation of individuals in EC methods [12], probably because of the difficulty in defining proper genetic operators and the intrinsic limitations of standard PNs.

In this work we introduce the *Evolutionary Petri Net* (EPN), which is based on an extended class of PNs, called *Resizable Petri Net*, and two genetic operators, crossover and mutation, that altogether give a foundation for the development of a robust evolutionary algorithm for the optimization, the automatic simplification and the reverse engineering of PNs. Due to space limits, we briefly introduce a potential application of EPNs to face one of the most challenging problems in Systems Biology, i.e., the reverse engineering of biochemical reaction networks.

2 Evolutionary Computation

Evolutionary Computation (EC) exploits the Darwinian evolution theory to solve complex problems [1]. Many bio-inspired EC methods have been proposed (e.g., Genetic Algorithms (GA) [13], Particle Swarm Optimization [14]), all sharing the following common traits: *(i)* they exploit a population $\mathcal{P}$ of randomly generated individuals, i.e., the candidate solutions; *(ii)* $\mathcal{P}$ evolves, generation after generation, thanks to an iterative process that employs random modifications of the individuals; *(iii)* the individuals able to solve the problem better than the other candidate solutions are conserved and promoted during the evolutive process; *(iv)* to the aim of discriminating the best solutions in $\mathcal{P}$, a *fitness function* quantifies the quality of each individual; *(v)* the process is executed iteratively until a termination criterion is met.

In the particular case of GAs, the individuals are encoded as fixed-length strings (the *genome*), composed by concatenating symbols from a finite alphabet, and the evolutionary process is driven by three operators: selection, crossover and mutation. The selection mechanism introduces an evolutionary pressure on the population, in order to make the individuals compete and adapt according to the fitness function that characterizes the problem in hand; selection methods are beyond the scope of the present paper, a comprehensive review is available in [15]. The crossover operator simulates the exchange of genetic material occurring during biological reproduction, by creating two offspring from the union of substructures belonging to two selected parents, aiming at the recombination of two promising parents, in order to improve the average quality of the population. The individuals undergo the crossover mechanism with a certain probability; alternatively, individuals are identically copied into $\mathcal{P}$. In both cases, the mutation operator can randomly modify these individuals – with a low probability – by changing, for instance, symbols in its genome. The mutation mechanism is used for the exploration of the search space, being the only operator able to introduce new genetic material in the population.

GAs have shown the ability of efficiently solving a plethora of complex problems, but all these results represent specific solutions to particular instances of a generic problem. In contrast, Genetic Programming (GP) [16] is an extension of GA in which the individuals are computer programs, generally encoded as tree data structures where the inner nodes are functions, and the leaves (or terminal nodes) are constant values or variables. Because of this peculiar representation, GP can identify the optimal solution for whole classes of problems, instead of specific instances. The functioning of GP is conceptually identical to GA and shares all the common traits of evolutionary algorithms, but it presents some slight difference with respect to GAs. Since the individuals are represented as trees, they are not fixed in height and they are exposed to *bloating*, that is, the uncontrolled growth of individuals, generation after generation. A maximum height D for the individuals can be defined, but this might negatively affect the evolution of the population. Moreover, the (unknown) actual height of the tree corresponding to the optimum might be greater than D.

GP has been extended to support individuals encoded as graphs [17], leading to the definition of many crossover operators, which are supposed to work with individuals whose number of nodes is different. For some applications, the individuals of GP may be represented by even more complex data structures like bipartite graphs, which describe in a more natural way the elements of a complex system and their mutual interactions. Anyway, the development of robust genetic operators, able to perform the evolution of such graphs, is challenging.

The set of terminal nodes of GP is generally defined at the beginning of the evolutionary process, meaning that the interacting elements of the complex system under investigation are well known: a condition that, sometimes, is not realistic. In this work we propose a novel methodology, inspired by GP, in which individuals are encoded as multi-partite graphs which, at the end of the evolutionary process, represent putative bipartite graphs, namely Petri Nets. Our methodology is able to automatically determine the best fitting size for the inferred network, by modifying the number of nodes of the candidate solutions during the evolutionary process by means of a special mutation operator.

3 Evolutionary Petri Nets

Petri Net (PN) is a modeling formalism for distributed, asynchronous and concurrent systems [2], which is defined as a weighted, directed, bipartite graph consisting of two kinds of nodes: the nodes representing the state (or conditions) of the system, called *places* and denoted by circles, and the nodes representing *transitions* (or events) between places, denoted by bars. PNs are extensively exploited for the simulation and analysis of the structural and behavioral properties of complex systems. Basic notions and notations of PNs can be found in [18]. To the aim of developing an evolutionary methodology whose candidate solutions are based on PNs, we hereby propose an extension of the conventional PN formalism, called *Evolutionary Petri Net* (EPN). EPNs provide a conceptual framework for the representation of candidate solutions, and embed robust and consistent genetic operators.

Before introducing the EPN formalism, we define a *Resizable Petri Net* (RPN) as a 9-tuple $\xi = (P, P^h, T, T^h, F, W, M_0, O_{pre}, O_{post})$ where:

- $P = \{p_1, \ldots, p_m\}$ is a finite set of places;
- P^h is the set of hidden places, such that $P \cap P^h = \emptyset$;
- $T = \{t_1, \ldots, t_n\}$ is a finite set of transitions;
- T^h is the set of hidden transitions, such that $T \cap T^h = \emptyset$ and $(P \cup P^h) \cap (T \cup T^h) = \emptyset$;
- $F \subseteq \big((P \cup P^h) \times (T \cup T^h)\big) \bigcup \big((T \cup T^h) \times (P \cup P^h)\big)$ is the set of arcs;
- $W : F \to \mathbb{N}$ is a weight function, which associates a non-negative integer value to each arc;
- $M_0 : P \to \mathbb{N}$ is the initial marking of non-hidden places of the net (all hidden places have zero tokens, initially);
- $O_{pre} \in \mathbb{N}$ is the maximum pre-order allowed in the RPN, that is, for each $t \in (T \cup T^h)$

$$\sum_{\substack{p \in {}^\bullet t \\ p \in (P \cup P^h)}} W(p, t) \leq O_{pre}; \tag{1}$$

- $O_{post} \in \mathbb{N}$ is the maximum post-order allowed in the RPN, that is, for each $t \in (T \cup T^h)$

$$\sum_{\substack{p \in t^\bullet \\ p \in (P \cup P^h)}} W(t, p) \leq O_{post}. \tag{2}$$

Differently from a traditional PN, a RPN is composed of a fixed number of places ($|P| = m$) and transitions ($|T| = n$), together with a variable number of hidden places and transitions in the sets P^h and T^h, respectively, whose cardinalities can change during the evolutionary process because of the application of the genetic operators. Two examples of RPN are depicted in Figure 1a; unlike other similar works on dynamically reconfigurable PNs [18], or on virtual PNs [19], RPNs are modified by "exogenous" mutations that can arbitrarily introduce new hidden places and new hidden transitions, where hidden transitions can represent events involving both elements from P and P^h. In RPNs we explicitly separate the sets P and T from the hidden (modifiable) sets P^h and T^h in order to make use of the available, consolidated domain knowledge of the system under investigation as static (i.e., non modifiable) places and transitions. In the case of zero-knowledge on the interaction of the elements of the net, the set P is populated with those elements that are known to be part of the system and whose data can be exploited by the fitness function, while $T = \emptyset$.

Conversely, hidden places and transitions are exploited by the genetic operators of EPN in order to explore alternative and more complex topologies of the net. Since hidden (i.e., modifiable) sets are dynamically modified and evaluated by the EC algorithm, EPNs allow the optimization of an existent PN according to some given constraints; moreover, they allow the automatic discovery of simplified or more efficient alternative models. These tasks can be accomplished by initializing the system of interest as a "fully-hidden" RPN (that is, $P = T = \emptyset$), and letting the evolutionary algorithm explore the space of alternative models.

The RPN formalism includes the pre- and post-order conditions (Equations 1 and 2) for multiple reasons. First, they help to reduce the bloating phenomenon of GP by limiting the number of arcs. Secondly, they avoid the possibility of a convergence to degenerate or overfitting solutions, represented by completely connected PNs, by limiting the weights of in- and out-going arcs of transitions. This can also be used to limit the search space, by excluding *a priori* unfeasible topologies. However, both pre- and post-order conditions are optional and can be excluded from the RPN by setting $O_{pre} = O_{post} = \infty$.

The modifiers of RPNs, exploited by the evolutionary algorithm, are two classic genetic operators: crossover and mutation. Given the space Ξ of all possible RPN topologies, we can define an *Evolutionary Petri Net* (EPN) as a triple $E = (\xi, \chi, \mu)$ where:

- $\xi \in \Xi$;
- $\chi : (\Xi \times \Xi) \to (\Xi \times \Xi)$ is the crossover operator which modifies two RPNs ξ and $\bar{\xi}$, where ξ and $\bar{\xi}$ are such that $P_\xi = P_{\bar{\xi}}$;
- $\mu : \Xi \cup \{p_{in}, t, p_{out}\} \to \Xi$ is the mutation operator, where $\{p_{in}, t, p_{out}\}$ is a triple consisting of two places p_{in} and p_{out} and a transition t, namely $p_{in}, p_{out} \in (P \cup P^h) \cup P^\infty$ and $t \in (T \cup T^h) \cup T^\infty$, where P^∞ and T^∞ are infinite sets of places and transitions, such that $P^\infty \cap (P \cup P^h) = \emptyset$ and $T^\infty \cap (T \cup T^h) = \emptyset$.

The functioning of χ and μ is described in the next sections, where we denote by $\xi(\tau)$, $\tau \in \mathbb{N}$, the RPN at the τ-th generation of the evolutionary process.

3.1 The Crossover Operator

The crossover mechanism implements the exchange of genetic material between two RPNs, in order to generate new offspring which inherit the best substructures of the parents. Various crossover mechanisms specifically designed for graphs have been proposed in literature [20, 21], in particular to tackle the complex case of two networks with a different number of nodes [17] but, to the best of our knowledge, no specific work exists about the crossover between bipartite graphs or, more specifically, PNs. Indeed, the crossover between two PNs is supposed to identify some substructures in each graph, "detach" them from one parent graph and "attach" them into the other, and viceversa, while keeping the consistency of both graphs. The difficulties arise in: *(i)* how to characterize a substructure; *(ii)* how to "detach" it from a graph; *(iii)* how to "attach" it to a new graph, considering that there is not a direct correspondence between the elements belonging to different sets of hidden places. The last issue is extremely relevant, because it determines the ability of EPNs to transferring a precise functionality from a RPN to its offspring.

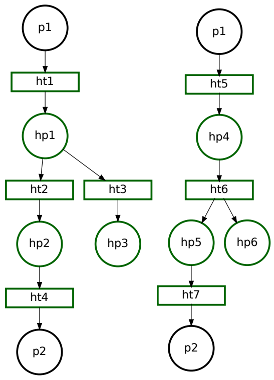
Our proposal for a crossover mechanism of two RPNs $\xi(\tau), \bar{\xi}(\tau) \in \Xi$, is named *sticky crossover* (SC). SC works on hidden transitions as follows: a transition $t_\chi \in T^h$ ($\bar{t}_\chi \in \bar{T}^h$, respectively) is randomly selected in the RPN ξ ($\bar{\xi}$) (the red nodes in Figure 1b), and the substructures consisting of the preset and postset of t_χ ($\bar{t}_\chi$) (dotted lines) are exchanged between ξ and $\bar{\xi}$. We denote by $P^+ = (^\bullet t_\chi \cup t_\chi^\bullet) \cap P^h$ ($\bar{P}^+$, respectively) the set of hidden places contained in the substructure connected to t_χ ($\bar{t}_\chi$), that is added to $\bar{\xi}$ (ξ); $F^+ = \{(t_\chi, p), (p, t_\chi) \in F \mid p \in P^+\}$ ($\bar{F}^+$) denotes the set of arcs belonging to this substructure.

To the aim of determining the attachment points for the moving substructures, we randomly select two input places $p_b \in {^\bullet t_\chi}$ and $\bar{p}_b \in {^\bullet \bar{t}_\chi}$, and two output places $p_e \in t_\chi{^\bullet}$ and $\bar{p}_e \in \bar{t}_\chi^\bullet$ (Figure 1c); p_b, p_e, $\bar{p}_b$, and $\bar{p}_e$ will be used as "attachment" points for the incoming substructure, that is, the substructure identified by transition t_χ is attached to $\bar{\xi}$, in such a way that $\bar{p}_b$ is replaced by p_b, $\bar{p}_e$ is replaced by p_e, and viceversa (see Figure 1d).

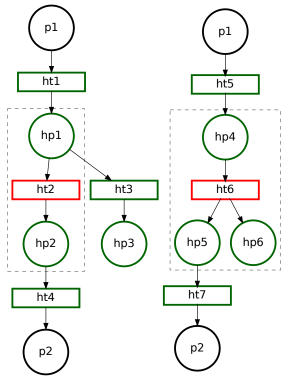
Formally, the crossover mechanism acts on the sets of places, transitions and arcs of the RPN as follows:

$$
\begin{aligned}
\bar{P}^h =& \bar{P}^h \cup P^+ \setminus \{\bar{p}_b, \bar{p}_e\}; \quad P^h = P^h \cup \bar{P}^+ \setminus \{p_b, p_e\};\\
\bar{T}^h =& \bar{T}^h \cup \{t_\chi\} \setminus \{\bar{t}_\chi\}; \quad T^h = T^h \cup \{\bar{t}_\chi\} \setminus \{t_\chi\};\\
\bar{F} =& \bar{F} \cup F^+ \setminus \bar{F}^+ \cup \{(\bar{t}, p_b) | \bar{t} \in {^\bullet \bar{p}_b}\} \cup \{(p_e, \bar{t}) | \bar{t} \in \bar{p}_e^\bullet\} \cup\\
& \cup \{(p_b, \bar{t}) | \bar{t} \in \bar{p}_b^\bullet\} \cup \{(\bar{t}, p_e) | \bar{t} \in {^\bullet \bar{p}_e}\} \setminus \{(\bar{t}, \bar{p}_b) | \bar{t} \in {^\bullet \bar{p}_b}\} \setminus\\
& \setminus \{(\bar{p}_e, \bar{t}) | \bar{t} \in \bar{p}_e^\bullet\} \setminus \{(\bar{p}_b, \bar{t}) | \bar{t} \in \bar{p}_b^\bullet\} \setminus \{(\bar{t}, \bar{p}_e) | \bar{t} \in {^\bullet \bar{p}_e}\};\\
F =& F \cup \bar{F}^+ \setminus F^+ \cup \{(t, \bar{p}_b) | t \in {^\bullet p_b}\} \cup \{(\bar{p}_e, t) | t \in p_e{^\bullet}\} \cup\\
& \cup \{(\bar{p}_b, t) | t \in p_b^\bullet\} \cup \{(t, \bar{p}_e) | t \in {^\bullet p_e}\} \setminus \{(t, p_b) | t \in {^\bullet p_b}\} \setminus\\
& \setminus \{(p_e, t) | t \in p_e{^\bullet}\} \setminus \{(p_b, t) | t \in p_b^\bullet\} \setminus \{(t, p_e) | t \in {^\bullet p_e}\}.
\end{aligned}
$$

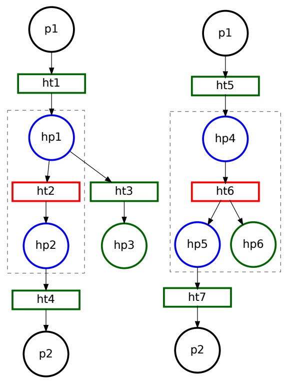
The weights of the arcs exchanged during the crossover process are not modified. Only hidden places are moved between RPNs during the crossover process, because each RPN contains an identical set of fixed places, so that there is a biunivocal correspondence between the elements in P and P^h. For this reason, for each place $p \in (^\bullet t_\chi \cup t_\chi^\bullet) \cap P$, where t_χ is the transition selected for crossover,



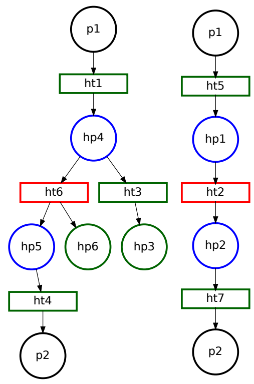
(a) Parents RPNs



(b) Step #1: selection of the transitions t_χ and $\bar{t}_\chi$



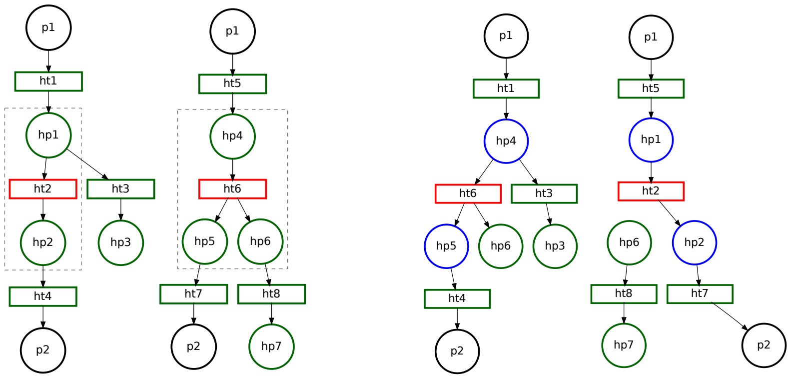
(c) Step #2: random selection of p_b, p_e, $\bar{p}_b$ and $\bar{p}_e$ nodes



(d) Step #3: exchange of substructures

Fig. 1. Example of *sticky crossover* between two RPNs. (a) From each parent RPN (b) a hidden transition is randomly selected (red nodes ht_2 and ht_6), identifying the substructures that will be exchanged between the RPNs (dotted gray line). (c) One random place from the preset and one from the postset of ht_2 and ht_6 are selected (the blue nodes hp_1, hp_2, hp_4 and hp_5 that correspond to p_b, p_e, $\bar{p}_b$ and $\bar{p}_e$, respectively). (d) The substructures are exchanged between the RPNs and attached by replacing the respective blue nodes, thus yielding the two offspring. For the sake of compactness we do not report the weights of arcs when they are equal to 1.

we let $\bar{F} = \bar{F} \cup \{(\bar{p}, t_\chi)\} \cup \{(t_\chi, \bar{p})\}$ and viceversa. It is important to clarify that the elements in P^h and T^h are "anonymous", that is, they do not share any semantics between different RPNs: if a hidden element is transferred from a RPN to another during a crossover, it is considered a new unknown element with respect to the already existing elements.



(a) Parents RPNs (b) RPNs after crossover

Fig. 2. Example of *sticky crossover* breaking up one RPN, thus creating a separate component (the hidden transition ht_8 in Subfigure (b), and its pre- and postsets).

If the SC involves $p \in P^h$ and $\bar{p} \in \bar{P}$ as the selected "attachment" places of the crossover then, in the exchange of the substructure from ξ to $\bar{\xi}$, place $\bar{p} \in \bar{P}$ in $\bar{\xi}$ remains unchanged, while in the exchange from $\bar{\xi}$ to ξ, place $p \in P^h$ in ξ is replaced by $\bar{p}$. So doing, ξ will result in a consistent RPN, since $P = \bar{P}$.

SC is a convenient crossover operator for many reasons: *(i)* it allows the crossover between two arbitrary RPNs, regardless the cardinality of the sets $P^h, \bar{P}^h$, which can vary during the genetic evolution; *(ii)* the pre- and post-order of the transitions of the offspring (as defined in Equations 1 and 2) are automatically conserved; *(iii)* the "directionality" of transitions is preserved, since SC swaps substructures whose presets are still presets, and elements of postsets are still postsets. Nevertheless, the SC has two drawbacks: *(i)* considering a substructure identified by a transition t_χ, if $\tilde{t} \neq t_\chi$ is connected to a place $p \in ({}^\bullet t_\chi \cup t_\chi^\bullet), p \notin \{p_e, p_b\}$, then the SC breaks the RPN leaving a separate component (see Figure 2); *(ii)* the SC allows the exchange of a single transition between two RPNs, which could have a little impact on large networks.

The second issue can be solved with two strategies: the first is to exploit some graph visiting algorithm (i.e., breadth- or depth-first) and extend the substructure accordingly; the second strategy is to determine a value $\rho \in \mathbb{N}$ such that $1 \leq \rho \leq \min\{|T^h|, |\bar{T}^h|\}$, and to repeat the crossover on ρ different transitions, that is, to randomly create two vectors of indexes $i_1, \ldots, i_\rho$ and $j_1, \ldots, j_\rho$ (with $i_k, j_k \in \mathbb{N}$, $k = 1, \ldots, \rho$), and then apply the crossover operator on each couple $(t_{i_k}, \bar{t}_{j_k})$. The first strategy allows to exchange multiple transitions that have been causally connected according to the chosen graph visiting algorithm, whilst the second one allows to exchange multiple independent transitions.

In the implementation of EPNs for real case applications, the computational complexity of applying the crossover operator is, in the worst case, $O(|F|^2 \cdot |P|)$. However, the use of a hash function might yield a reduction of the computational cost, in the average case, to $O(|P|)$.

3.2 The Mutation Operator

The mutation operator modifies the structure of a RPN $\xi(\tau)$ in Ξ, or the properties of its places and arcs (i.e., capacity and weight), by acting on a single, randomly chosen hidden transition $t \in T^h$. In particular, the mutation operator associates $\xi(\tau)$ to a new consistent RPN $\xi(\tau+1)$, according to a specified triple $\{p_{in}, t, p_{out}\}$. The rationale behind this triple is to provide new genetic material, that is, to modify the topology of the RPN; the functioning of the mutation operator for all the possible cases of $\{p_{in}, t, p_{out}\}$ is summarized in Table 1. After the application of any mutation case (except #8), we set $F = F \cup (p_{in}, t) \cup (t, p_{out})$ and $W(p_{in}, t) = W(t, p_{out}) = 1$. Cases #7 and #8 are particular and deserve a detailed explanation.

Case #7 introduces a brand new transition which is "disconnected" from the rest of the network, since it exploits new places that are not used by any other transition. So doing, the dynamics of these places, that is, the succession of their markings as a consequence of the firings, is independent from the rest of the RPN. In other words, Case #7 is a "silent" modification of the topology of the RPN, that will not have a direct impact on its behavior. Nevertheless, a further application of a genetic operator to the mutated RPN may connect this latent transition to the main net component, thus conditioning the behavior of the whole RPN.

Case #8 does not introduce any new genetic material. This operator is used to change the capacities $K(p)$ of randomly selected places $p \in P$, i.e. $K(p) = rnd$, where rnd is a random number sampled from the uniform distribution $(0, K_{max}]$ (where K_{max} is the maximum capacity for the places in the RPN). Alternatively, the mutation can modify the weight of an arc; for instance, given the arc weight $W(p,t)$, its value can be updated as $W(p,t) = W(p,t) \pm 1$. It is worth noting that also Case #8 can even lead to a modification in the structure of the RPN, whenever an arc weight is set to zero (i.e., the arc is removed). As a consequence, isolated hidden places or hidden transitions, which represent sources or sinks, can be introduced in the RPN after Case #8 is applied and must be removed. Therefore, after the application of this operator, the consistency of the RPN must be verified, regarding each place p and transition t involved in the mutation: *(i)* if $p \in P^h$ and $\nexists \{(p,t),(t,p)\}\ \forall t \in T \cup T^h$, then $P^h = P^h \setminus \{p\}$; *(ii)* if $t \in T^h$ and $\nexists \{(t,p),(p,t)\}\ \forall p \in P \cup P^h$, then $T^h = T^h \setminus \{t\}$.

As a final step of the evolution process, pre- and post-order conditions of the offspring need to be verified, since the described mutations might produce a putative RPN whose topology does not satisfy Equations 1 and 2. In such a case, a further modification of the weights of the ingoing and/or outgoing arcs of a transition t is required. In particular, a randomly selected input (output, respectively) place of transition t is modified so that $W(\tilde{p}, t) = W(\tilde{p}, t) - 1$ (and/or $W(t, \tilde{p}) = W(t, \tilde{p}) - 1$), where $\tilde{p} \in {}^\bullet t$ ($\tilde{p} \in t^\bullet$, respectively). It is clear that if $W(x,y) = 0$, then the corresponding arc (x,y) is removed from F. This operation is repeated until the RPN respects all the pre- and post-order conditions.

It is worth noting that these two mechanisms implicitly allow mutation to delete places, thus reducing the size of the RPN.

Table 1. Effect of the mutation operator on a Resizable Petri Net $\xi(\tau)$.

No.	*Condition*	$\xi(\tau+1)$	*Semantics of the mutation*
1.	$p_{in} \notin P \cup P^h$ $t \in T \cup T^h$ $p_{out} \in P \cup P^h$	$P^h = P^h \cup \{p_{in}\}$ $P^\infty = P^\infty \setminus \{p_{in}\}$	A new hidden place p_{in} is created and added to P^h; transition t is extended to have a new input place p_{in}
2.	$p_{in} \in P \cup P^h$ $t \in T \cup T^h$ $p_{out} \notin P \cup P^h$	$P^h = P^h \cup \{p_{out}\}$ $P^\infty = P^\infty \setminus \{p_{out}\}$	A new hidden place p_{out} is created and added to P^h; transition t is extended to have a new output place p_{out}
3.	$p_{in} \in P \cup P^h$ $t \notin T \cup T^h$ $p_{out} \in P \cup P^h$	$T^h = T^h \cup \{t\}$ $T^\infty = T^\infty \setminus \{t\}$	A new hidden transition t is created and added to T^h; transition t is connected to the existing input and output places p_{in} and p_{out}, respectively
4.	$p_{in} \notin P \cup P^h$ $t \notin T \cup T^h$ $p_{out} \in P \cup P^h$	$P^h = P^h \cup \{p_{in}\}$ $T^h = T^h \cup \{t\}$ $P^\infty = P^\infty \setminus \{p_{in}\}$ $T^\infty = T^\infty \setminus \{t\}$	A new hidden transition t is created and added to T^h; a new hidden place p_{in} is created and added to P^h; transition t is connected to new input place p_{in} and to an existing output place p_{out}
5.	$p_{in} \in P \cup P^h$ $t \notin T \cup T^h$ $p_{out} \notin P \cup P^h$	$T^h = T^h \cup \{t\}$ $P^h = P^h \cup \{p_{out}\}$ $P^\infty = P^\infty \setminus \{p_{out}\}$ $T^\infty = T^\infty \setminus \{t\}$	A new hidden transition t is created and added to T^h; a new hidden place p_{out} is created and added to P^h; transition t is connected to an existing input place p_{in} and to the new output place p_{out}
6.	$p_{in} \notin P \cup P^h$ $t \in T \cup T^h$ $p_{out} \notin P \cup P^h$	$P^h = P^h \cup \{p_{in}, p_{out}\}$ $P^\infty = P^\infty \setminus \{p_{in}, p_{out}\}$	Two new hidden places p_{in} and p_{out} are created and added to P^h; transition t is connected to p_{in} as input place and to p_{out} as output place
7.	$p_{in} \notin P \cup P^h$ $t \notin T \cup T^h$ $p_{out} \notin P \cup P^h$	$P^h = P^h \cup \{p_{in}, p_{out}\}$ $T^h = T^h \cup \{t\}$ $P^\infty = P^\infty \setminus \{p_{in}, p_{out}\}$ $T^\infty = T^\infty \setminus \{t\}$	A new hidden transition t is created and added to T^h; two new hidden places p_{in} and p_{out} are created and added to P^h; transition t is connected to the new input place p_{in} and to the new output place p_{out}
8.	$p_{in} \in P \cup P^h$ $t \in T \cup T^h$ $p_{out} \in P \cup P^h$	$P^h = P^h$ $T^h = T^h$ $P^\infty = P^\infty, T^\infty = T^\infty$	No new genetic material is introduced. Either p_{in} or p_{out} is randomly chosen; then, its capacity or the weight of the arc connecting it to t is modified

The computational complexity of the mutation operator is, for the worst case, $O(|P|)$. However, as in the case of crossover, the use of a hash function allows a reduction of the computational cost of this operator, in the average case, to $O(C)$ where, in general, $C < |P|$.

4 Toward the Application of EPNs for the Reverse Engineering of Biochemical Reaction Networks

In this section we sketch the theoretical basis of a potential application of EPNs in the context of Systems Biology, for the reverse engineering (RE) of biochemical interaction networks. This problem consists in the identification of the network of reactions that describe the physical interactions among the chemical species occurring in the system (genes, proteins, metabolites, etc.). These networks can be defined thanks to human expertise, by relying on some pre-existing knowledge, although most of the times the exact molecular mechanisms occurring in living cells are not known and cannot be fully understood by means of laboratory experiments only. This problem therefore demands the development of automatic RE methods, so that a plausible network of biochemical reactions – able to reproduce some given experimental observations – can be determined in faster and inexpensive ways. Since these networks are usually kinetically parameterized, they can also be exploited to analyze the dynamics of the system under different conditions.

Here, we briefly describe the modeling of biochemical systems by means of PNs and then present a possible strategy based on EPNs for solving the RE problem, which shows the feasibility of our novel methodology. A biochemical network η can be modeled by means of a set of chemical species $\mathcal{S} = (S_1, \ldots, S_U)$, involved in a set of chemical reactions $\mathcal{R} = (R_1, \ldots, R_Z)$. Each reaction $R_\zeta \in \mathcal{R}, \zeta = 1, \ldots, Z$, is defined as $R_\zeta : a_{\zeta 1} \cdot S_1 + \ldots + a_{\zeta U} \cdot S_U \xrightarrow{k_\zeta} b_{\zeta 1} \cdot S_1 + \ldots + b_{\zeta U} \cdot S_U$, where $a_{\zeta i}, b_{\zeta i} \in \mathbb{N}$ are the stoichiometric coefficients of R_ζ, and $k_\zeta \in \mathbb{R}^+$ is the kinetic constant associated to R_ζ. The species occurring on the left-hand (right-hand) side of R_ζ are called reagents (products, respectively).

Because of their bipartite graph structure, PNs offer an ideal conceptual framework for the modeling of biochemical networks [22] defined as a set of reactions in the form of R_ζ. To this aim, a transformation of a network η into a corresponding PN (and viceversa) needs to be defined. Briefly, a mapping $f : \mathcal{S} \rightarrow P$ can be used to associate the species to the places of a PN, where the transitions represent the chemical reactions (i.e., $g : \mathcal{R} \rightarrow T$) and where the weights correspond to the stoichiometry of the reagents and products of each reaction (i.e., $v_r : a_{\zeta i} \rightarrow W(p_i, t_\zeta)$, $v_p : b_{\zeta i} \rightarrow W(t_\zeta, p_i)$). According to these mappings, it is straightforward to show that the PN on the left in Figure 1a represents the following set of reactions: $\{R_1^h : S_1 \rightarrow S_1^h, R_2^h : S_1^h \rightarrow S_2^h, R_3^h : S_1^h \rightarrow S_3^h, R_4^h : S_2^h \rightarrow S_2\}$. The number of tokens (given as discrete or continuous value) in each place represents the molecular amount (given as number of molecules or concentration, respectively) of the corresponding chemical species, so that M_0 represents the initial state of the biochemical system.

The PN representation of a biochemical reaction network allows the investigation of its structural features by means of theoretical approaches [22, 23]; in addition, PNs can be easily extended with timed delays [24], or to incorporate quantitative information (e.g., reaction rates, concentration levels) to the aim of investigating the dynamic evolution of biochemical systems [25]. Kinetic parameters can also be associated to the reactions, in order to derive the probability of each reaction to occur [26], or to convert the system into a set of coupled ordinary differential equations; in the latter case, places contain continuous values corresponding to the concentration of the chemical species associated to each place. Once that a model η of a biochemical network is defined in terms of a fully parameterized extended PN, according to the available domain knowledge, its dynamic behavior can be investigated. Anyway, when the domain knowledge is incomplete, uncertain or completely missing, a (fully parameterized) PN model of a biochemical system cannot be constructed. In such a case, it is necessary to perform the RE of the biochemical reaction network, investigating the unknown reactions and chemical species that are responsible for the observed phenomena.

Many works perform the RE by means of evolutionary techniques [12, 27, 28]. One limitation of many of these techniques is that the individuals are modeled by means of data structures that are not ideal to describe reaction networks; in addition, many constraints and consistency controls must be introduced to control the quality and validity of the inferred network. The strongest limitation of all these methods is that the cardinality of $\mathcal{S}$, that is, the number of chemical species that are present in the system, is assumed to be known and kept fixed during the optimization. This is generally a strong assumption, that may be justified only if the biochemical system is very well known (but, in such a case, the network should be known as well) or when laboratory experiments can yield this information with a certain precision. As a matter of fact, in most cases the exact number and nature of the chemical species involved in the system – including the intermediate complexes formed by the chemical bonds among various molecules – is unknown; anyway, this is a fundamental information to properly carry out the RE of the system. Thanks to the flexibility of hidden places and transitions, the use of RPNs to represent the candidate solutions gives to EPN the possibility to explore much more possibilities than the traditional RE approaches that exploit a fixed number of chemical species, thus leading to the formulation of new hypotheses for the structure of the biochemical network that should then be validated with *ad hoc* laboratory experiments. In this context, hidden places in the RPN can be exploited to represent some molecular species or complexes which are necessary to reproduce the expected behavior of the biochemical system, but that have not been yet identified with experimental techniques; similar considerations hold for hidden transitions, which can represent molecular interactions that are not known from a biochemical point of view, but that might yield a better system functioning (in terms of the specified fitness function, and according to the available experimental data).

The application of an EPN-based methodology for the RE of biochemical systems is similar to GP. At first, a population $\mathcal{P}$ of RPNs is generated according to the available domain knowledge: when the information comes from established biological knowledge, the reactions are modeled using the sets P and T; on the contrary, when the information is uncertain, the network is modeled by means of hidden places and transitions. At generation $\tau = 0$, since the individuals are all identical, they undergo a preliminary mutation. During each generation, the best individuals are selected according to a specified fitness function, and the EPN genetic operators are applied to yield new offspring. The fitness function can be defined, for instance, by comparing a set of $\Delta \in \mathbf{N}$ experimental samples of the $|P| = m$ non-hidden chemical species (places in P) against the dynamics generated through simulations [29], exploiting the set of reactions that the RPN represents: $fitness(\eta) = \sum_{\delta=1}^{\Delta} \sum_{i=1}^{m} |\mathbf{Q}^{p_i}(t_\delta) - \mathbf{L}_\eta^{p_i}(t_\delta)|$, where $\mathbf{Q}^{p_i}(t_\delta)$ and

$\mathbf{L}_{\eta}^{p_i}(t_\delta)$ denote, respectively, the experimental and simulated quantity (concentration or molecular amount) of the p_i-th chemical species at time t_δ.

To the aim of facilitating the evolutionary process and helping the generation of biologically meaningful candidate solutions, in RE we can set $O_{pre} = 2$ to force the evolution of at most second-order reactions, since higher-order reactions have a probability to occur almost equal to zero, requiring the simultaneous collision of three or more reactant molecules. This choice helps the EPN functioning, since it strongly reduces the search space Ξ, but at the same time it does not pose any limitation to the practical applicability of the RE methodology, as higher-order reactions can be mimicked through a cascade of consecutive reactions of lower order. During the evolutionary process, the EPN explores this search space, and eventually a best individual $\mathcal{I} \in \mathcal{P}$ emerges, representing a consistent RPN that fits with all the observed phenomena: $\mathcal{I}$ is finally returned as the result of the RE. Due to space limits, we can only sketch how the crossover operator acts in determining a biochemical reaction network. Consider, for instance, the individuals ξ (left) and $\bar{\xi}$ (right) shown in Figure 1a: the crossover swaps the reactions R_2^h in ξ and R_6^h in $\bar{\xi}$, so that the chemical reaction R_2^h, renamed R_6^h after the crossover, has an additional product (i.e., S_6^h, left side of Figure 1d). On the contrary, the reaction R_6^h (renamed R_2^h after the crossover) in $\bar{\xi}$ loses a product and becomes a simple transformation from one species to another. The consequence of these modifications is that the new reaction network might have a completely different dynamic behavior, and hence a different (hopefully, better) fitness value.

5 Conclusion and Future Work

In this work we presented an extension of the PN formalism, the Resizable Petri Net, which introduces the notion of hidden places and transitions. The RPN represents the basis for the development of an evolutionary algorithm, the Evolutionary Petri Net, whose crossover and mutation operators allow the exploration of the space of all possible RPN topologies, in order to provide a powerful tool for solving complex problems whose solutions can be encoded as PNs; in particular, EPNs has the capability to evolve consistent and meaningful solutions. We described here the theoretical foundations of EPNs; a thorough analysis of this novel evolutive framework will be presented in a future work, in order to introduce and discuss different strategies for the construction of the RPN individuals (according to various real case applications), as well as specific selection mechanisms. The probability distributions that need to be associated to the mutation and crossover operators, and which have a relevant impact on the convergence of the evolutive process of RPNs, will also be discussed in a forthcoming extension of this work.

To the purpose of grounding this framework to practical problems, we provided an example of a potential application of EPNs – the RE of biochemical reaction networks – which might represent a killer-application for our evolutionary methodology. In this context, the fitness evaluation that we proposed relies on the simulation of the candidate solutions, which is possible only if a proper parameterization of the candidate network in available. Hence, the RE problem is further complicated by the need of a parameter estimation (PE) methodology for the inference of the missing kinetic parameters [30, 31]: to this aim, we are developing an integrated methodology that embeds the PE process in each generation of the RE. A further difficulty of the RE process is due to the fact that different networks can lead to the same dynamic behavior; this problem, known as *undistinguishability* [32], cannot be solved, in general, without additional knowledge. EPNs do not directly mitigate this drawback, even though pre- and post-order conditions allow the reduction of the possible topologies, thus permitting the derivation of meaningful networks that can be then discriminated by domain expertise. Finally, the initial marking of the hidden places was set to zero: a further extension of EPN, in which the initial marking co-evolves with the topology, is under investigation.

References

1. DeJong, K.: Evolutionary computation: a unified approach. The MIT Press (2006)
2. Murata, T.: Petri nets: Properties, analysis and applications. In: Proc. IEEE. Volume 77. (1989) 541–580
3. Thome, H., Nakamura, M., Hachiman, K.: Evolutionary Petri net approach to periodic job-shop scheduling. In: Proc. IEEE Int. Conf. on Systems, Man, and Cybernetics (SMC'99). Volume 4., IEEE Computer Society Press (1999) 441–446
4. Saitou, K., Malpathak, S., Qvam, H.: Robust design of flexible manufacturing systems using colored Petri net and genetic algorithm. Journal of Intelligent Manufacturing **13**(5) (2002) 339–351
5. Mauch, H.: Evolving Petri nets with a genetic algorithm. In: Proc. of the 2003 conference on genetic and evolutionary computation (GECCO '03). Volume 2724 of LNCS., Springer (2003) 1810–1811
6. Moore, J., L.W.Hahn: Grammatical evolution for the discovery of Petri net models of complex genetic systems. In: Proc. of the 2003 conference on genetic and evolutionary computation (GECCO '03). Volume 2724 of LNCS., Springer (2003) 2412–2413
7. Kitagawa, J., Iba, H.: Identifying metabolic pathways and gene regulation networks with evolutionary algorithms. In: Evolutionary Computation in Bioinformatics. Morgan Kaufmann (2003) 255–278
8. Koza, J.R., Mydlowec, W., Lanza, G., Yu, J., Keane, M.A.: Reverse engineering of metabolic pathways from observed data using genetic programming. In: Pacific Symposium on Biocomputing. Volume 6. (2001) 434–445
9. Sugimoto, M., Kikuchi, S., Tomita, M.: Reverse engineering of biochemical equations from time-course data by means of genetic programming. BioSystems **80**(2) (2005) 155–164
10. Ando, S., Sakamoto, E., Iba, H.: Evolutionary modeling and inference of gene network. Information Sciences **145**(3) (2002) 237–259
11. Iba, H.: Inference of differential equation models by genetic programming. Information Sciences **178**(23) (2008) 4453–4468
12. Nummela, J., Julstrom, B.A.: Evolving petri nets to represent metabolic pathways. In: Proc. of the 2005 conference on genetic and evolutionary computation (GECCO '05), ACM (2005) 2133–2139
13. Holland, J.: Adaptation in Natural and Artificial Systems. The University of Michigan Press (1975)

14. Kennedy, J., Eberhart, R.: Particle swarm optimization. In: Proc. IEEE International Conference on Neural Networks. Volume 4., IEEE (1995) 1942–1948
15. Bäck, T.: Selective pressure in evolutionary algorithms: A characterization of selection mechanisms. In: Proc. of the First IEEE Conference on Evolutionary Computation. Volume 1., IEEE (1994) 57–62
16. Koza, J.: Genetic Programming: On the Programming of Computers by Means of Natural Selection. The MIT Press (1992)
17. Katagiri, H., Hirasawa, K., Hu, J., Murata, J.: Comparing some graph crossover in genetic network programming. In: Proc. of the 41st SICE Annual Conference. Volume 2., IEEE (2002) 1263–1268
18. Llorens, M., Oliver, J.: Structural and dynamic changes in concurrent systems: Reconfigurable Petri nets. IEEE Trans. on Computers **53**(9) (2004) 1147–1158
19. Morandin Jr, O., Kato, E.: Virtual Petri nets as a modular modeling method for planning and control tasks of FMS. International Journal of Computer Integrated Manufacturing **18**(2-3) (2005) 100–106
20. Larrañaga, P., Kujipers, C., Murga, R., Inza, I., Dizdarevic, S.: Genetic algorithms for the travelling salesman problem: A review of representations and operators. Artificial Intelligence Review **13**(2) (1999) 129–170
21. Pereira, F., Machado, P., Costa, E., Cardoso, A.: Graph based crossover – a case study with the busy beaver problem. In: Proc. of the Genetic and Evolutionary Computation Conference (GECCO '99). Volume 2. (1999) 1149–1155
22. Chaouiya, C.: Petri net modelling of biological networks. Briefings in Bioinformatics **8**(4) (2007) 210–219
23. Voss, K., Heiner, M., Koch, I.: Steady state analysis of metabolic pathways using Petri nets. In Silico Biology **3**(3) (2003) 367–387
24. Li, C., Ge, Q., Nakata, M., Matsuno, H., Miyano, S.: Modelling and simulation of signal transduction in an apoptosis pathway by using timed Petri nets. Journal of Biosciences **32**(1) (2006) 113–127
25. Goss, P., Peccoud, J.: Quantitative modeling of stochastic systems in molecular biology by using stochastic Petri nets. PNAS **95**(12) (1998) 6750–6755
26. Gillespie, D.T.: Stochastic simulation of chemical kinetics. Annual Review of Physical Chemistry **58** (2007) 35–55
27. Cho, D., Cho, K., Zhang, B.: Identification of biochemical networks by S-tree based genetic programming. Bioinformatics **22**(13) (2006) 1631–1640
28. Nobile, M.S., Besozzi, D., Cazzaniga, P., Pescini, D., Mauri, G.: Reverse engineering of kinetic reaction networks by means of cartesian genetic programming and particle swarm optimization. In: Proc. IEEE Congress on Evolutionary Computation (CEC2013). (2013, in press)
29. Ghosh, S., Matsuoka, Y., Asai, Y., Hsin, K.Y., Kitano, H.: Software for systems biology: from tools to integrated platforms. Nature Reviews Genetics **12**(12) (2011) 821–832
30. Nobile, M.S., 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: Proc. EvoBIO 2012. Volume 7246 of LNCS. Springer (2012) 74–85
31. Moles, C.G., Mendes, P., Banga, J.R.: Parameter estimation in biochemical pathways: a comparison of global optimization methods. Genome Research **13**(11) (2003) 2467–2474
32. Szederkenyi, G., Banga, J.R., Alonso, A.A.: Inference of complex biological networks: distinguishability issues and optimization-based solutions. BMC Systems Biology **5**(1) (2011) 177