Using the incidence matrix in an evolutionary algorithm for Computing minimal siphons in Petri net models

Fernando Tricas* José Manuel Colom* Juan Julián Merelo†
*Depto de Informática e Ingeniería de Sistemas
Universidad de Zaragoza
{ftricas,jm}@unizar.es
†Depto. ATC/CITIC
Universidad de Granada
jmerelo@geneura.ugr.es

***Abstract*—Petri nets are graph based tools to model and study concurrent systems and their properties; one of them is *liveness*, which is related to the possibility of every part of the system to be activated eventually. Siphons are sets of places that are related to liveness properties. When we need to deal with realistic problems its computation is hard or even impossible and this is why in this paper we are approaching it using evolutionary computation, a meta-heuristic that has proved it can successfully find solutions when the search space is big. In a previous work a formulation of the siphon property based on linear constraints and a genetic algorithm was proposed for general Petri Nets. Here we propose to adapt an algebraic method based on the selection of rows of the matrix that cancel in an adequate way input and output transitions so the resulting selection is a siphon. We will also present an evaluation for a family of resource allocation systems (RAS). The proposed solution is based on a genetic algorithm (GA); we can see how siphons can be computed using this genetic algorithm, with experiments showing that in some cases they are able to find a few solutions in less time than previous deterministic algorithms.**

***Keywords*—Siphons, genetic algorithms, incidence matrix, algebraic methods, computing, deadlock prevention**

I. Introduction

A Resource Allocation System (RAS) can be seen as a finite set of concurrent processes which share a finite set of resources in a competitive way; it can be modelled by means of a discrete event system. RAS are usually complex enough to take advantage of the use of formal methods, which can be used to improve its understanding, providing tools for the analysis and implementation steps. They also help in the dialog between people involved in the design, construction and system management. Our proposal is to use Petri (or Place/Transition) nets as a tool for this purpose. They are used to visualize and, through formal analysis, describe structural properties of the system they represent [1].

The competition for resources makes possible the existence of deadlocks; a deadlock occurs when some processes are waiting for resources that have been allocated to other processes, that are also waiting for resources that have been allocated to the former ones (maybe in a more complicated ways, the dependence does not need to be direct). RAS have been widely used when synthesizing deadlock avoidance and prevention policies, and many of the published work relies on minimal siphons for this [2], [3], [4], [5], [6], [7], [8]. A minimal siphon is a set of places such that the existence of any edge from a transition t to a place of D implies that there is an edge from some place of D to t. When a siphon reaches a state with no tokens, it will never become marked again; for this reason they are related to liveness properties. In consequence, some (efficient) methods to compute these structural components are needed. In [8] some work has been done in the field of Flexible Manufacturing Systems (FMS) trying to reduce the number of siphons to be considered for deadlock prevention, but not avoiding the computation of the whole set of minimal siphons. In most cases siphon enumeration cannot be avoided and this is the reason for trying to obtain efficient implementations ([2], [9], [5], [7].)

In this work we are going to propose a genetic algorithm implementation that uses a formulation of the siphon property by means of linear algebra, using the incidence matrix of the Petri Net model. This implementation has been tested in a well-known family of RAS. We will show how we can compute siphons using a genetic algorithm with an existing generic package.

The rest of this paper follow the this scheme: Section II provides an introduction to Petri Nets, Section III presents the Genetic Algorithm. In Section IV we presents the adapted method, and Section V is devoted to our experimental setup and the experimental results, together with some discussion about them. Finally, some conclusions are presented.

II. Petri Nets

A Petri net (or Place/Transition net) is a 3-tuple $\mathcal{N} = \langle P, T, W \rangle$ where P and T are two non-empty disjoint sets whose elements are called *places* and *transitions*, respectively. In a generic way, elements belonging to $P \cup T$ are called *nodes*. $W : (P \times T) \cup (T \times P) \to \mathbb{N}$ defines the *weighted*

Accepted Manuscript to appear in IEEE ICSTCC 2014 Proceedings (http://www.ace.tuiasi.ro/icstcc2014/)

flow relation: if $W(x,y) > 0$, then we say that there is an arc from x to y, with weight or multiplicity $W(x,y)$. *Ordinary nets* are those where $W : (P \times T) \cup (T \times P) \to \{0,1\}$. Given a net $\mathcal{N} = \langle P, T, W \rangle$ and a node $x \in P \cup T$, ${}^\bullet x = \{y \in P \cup T \mid W(y,x) > 0\}$ is called the *pre-set* of x, with $x^\bullet = \{y \in P \cup T \mid W(x,y) > 0\}$ being the *post-set* of x. This notation is extended to a set of nodes as follows: given $X \subseteq P \cup T$, ${}^\bullet X = \bigcup_{x \in X} {}^\bullet x$, $X^\bullet = \bigcup_{x \in X} x^\bullet$. A Petri net is *self–loop free* when $W(x,y) \neq 0$ implies that $W(y,x) = 0$. The Pre–incidence matrix $\mathbf{Pre} : P \times T \to \mathbb{N}$ of $\mathcal{N}$ is $\mathbf{Pre}[p,t] = W(p,t)$. The Post–incidence matrix $\mathbf{Post} : P \times T \to \mathbb{N}$ of $\mathcal{N}$ is $\mathbf{Post}[p,t] = W(t,p)$. A self–loop free Petri net $\mathcal{N} = \langle P, T, W \rangle$ can be alternatively represented as $\mathcal{N} = \langle P, T, \mathbf{C} \rangle$ where $\mathbf{C}$ is the incidence matrix: a $P \times T$ indexed matrix such that $\mathbf{C}[p,t] = W(t,p) - W(p,t) = \mathbf{Post}[p,t] - \mathbf{Pre}[p,t]$. A *marking* is a mapping $\mathbf{m} : P \to \mathbb{N}$; in general, markings are represented in vector form. A transition $t \in T$ is *enabled* for a marking $\mathbf{m}$ if and only if $\forall p \in {}^\bullet t \,.\, \mathbf{m}[p] \geq W(p,t)$; this fact will be denoted as $\mathbf{m} \xrightarrow{t}$ (or $\mathbf{m}[t>$). If t is enabled at $\mathbf{m}$, it can *occur*; when it occurs, this gives a new marking $\mathbf{m}' = \mathbf{m} + \mathbf{C}[P,t]$; this will be denoted as $\mathbf{m} \xrightarrow{t} \mathbf{m}'$ (or $\mathbf{m}[t > \mathbf{m}'$), and we say that $\mathbf{m}$' is reached from $\mathbf{m}$ by the occurrence of t. The *state equation* of a marked net is an algebraic equation that gives a necessary condition for the reachability of a marking from the initial marking: a markings $\mathbf{m} \in \mathbb{N}^{|P|}$ such that $\exists \boldsymbol{\sigma} \in \mathbb{N}^{|T|} \,.\, \mathbf{m} = \mathbf{m_0} + \mathbf{C} \cdot \boldsymbol{\sigma}$ is said to be *potentially reachable*. The *potentially reachability set* of a net is the set of solutions for the *state equation*. *Flows* (*Semiflows*) are integer (natural) annullers of matrix $\mathbf{C}$ (That is, a vector, $\mathbf{y} \neq \mathbf{0}$ such that $\mathbf{y} \cdot \mathbf{C} = \mathbf{0}$). Right and left annullers are called *T–(Semi)flows* and *P–(Semi)flows*, respectively. The *support* of P–(Semi)flows is given by: $\|\mathbf{y}\| = \{p \in P \mid \mathbf{y}[p](>) \neq 0\}$. Let $\mathcal{PS}$ be the set of minimal P–Semiflows of $\mathcal{N}$. (Semi)flows are called *minimal* when its support is not a strict super-set of the support of any other, and the greatest common divisor of its elements is one. A P–Semiflow $\mathbf{y}$ defines the following invariant property: $\forall \mathbf{m_0} \,.\, \forall \mathbf{m} \in \mathrm{RS}(\mathcal{N}, \mathbf{m_0}) \,.\, \mathbf{y} \cdot \mathbf{m} = \mathbf{y} \cdot \mathbf{m_0}$ (cyclic behavior law).

Given a Petri net we will call $\mathcal{N}$ a subset of places $D \subseteq P$ is a *siphon* ($E \subseteq P$ is a *trap*) of the net $\mathcal{N}$ if, and only if, ${}^\bullet D \subseteq D^\bullet$ ($E^\bullet \subseteq {}^\bullet E$). A siphon (trap) is minimal if, and only if, it does not properly contain another siphon (trap). Siphons have the important property that, if at a given marking the siphon is unmarked, it will remain unmarked. Researchers have considered and studied different methods for finding siphons and traps. Among them let us present the main types, that we will classify based on the underlying techniques used for their computation: *Algebraic methods* compute families of siphons by means of the solution of a set of linear equations or inequalities. They use the net incidence–matrix or a transformation of it. Methods using this approach can be found in [10]. *Methods based on graph theory* directly use the graph representation of the Petri net to compute siphons: methods using this approach can be found in [11]. *Methods using logic formulas* are based on characterizing siphons by means of boolean variables, which typically represent places or transitions and their relations [12].

III. Genetic Algorithms

Genetic algorithms [13] (GAs) are bioinspired methods that take their inspiration from Darwin's evolutionary and its genetic-molecular basis. More technically the genetic algorithm is a search and optimization algorithms based on a population of solutions and using a method that is similar to natural selection. A random population of candidate solutions is evolved trying to explore the search space looking for better solutions. The sketch of the canonical genetic algorithm is as follows [14]:

1) (Start) Initialize the population with n random solutions coded using a data structure generally called *chromosome*
2) (Loop) Repeat until a required set of solutions are found or the evaluation budget (number of evaluations set in advance) is exhausted.
3) Evaluate every member of the population on the basis of its ability to solve the problem, that *ability* is called *fitness*.
4) (New population) Generate n new solutions
   a) (Selection) Create a pool of solutions where the former population is represented according to its fitness.
   b) (Reproduction) Extract randomly solutions from these pools and combine them (in a process called *crossover*) or change them (*mutation*). In some cases, both operations are performed sequentially
   c) (Accepting) Insert these new solutions into the former population, in some cases replacing it completely.

The main task of a genetic algorithms designer is to find good parameter settings (population size, encoding, selection criteria, genetic operator probabilities, fitness evaluation, ...). We have used the `Algorithm::Evolutionary` [15] implementation following the example `tide_bitstring.pl` for the experiments. There are many other available implementations, but this one is known by the authors, is written in Perl and needs just a few lines of code to be adapted to new problems. Since it is written in an interpreted scripting language it can be, in general, slower than other libraries written in Java or C++.

IV. The proposed approach

The method presented by Boer and Murata in [16] uses an algebraich approach, where the underlying graph of the Petri net is represented by means of the signi-incidence matrix, using the input and output arcs without weights.

For example, in Fig. 1 we can see a very simple Petri net (which belongs to S^4PR [17], [12]) and in Table I its sign incidence matrix. In this matrix, the rows represent the set of input/output transitions of places (or set of places): positive entries for input transitions, negative ones for output transitions, and $\pm$–signed ones for transition that are both input and output of the set. Adequate arithmetic operations that allow to cancel any positive entry with a negative or a '$\pm$' valued one can be defined in order to represent the siphon property. We will need to find adequate combinations of rows such that positive signs can be cancelled to produce '$\pm$' and negative ones. For example, if we use the sign incidence matrix of Table I, we can sum the rows corresponding to places $P1\_1$ and $R1$, obtaining:

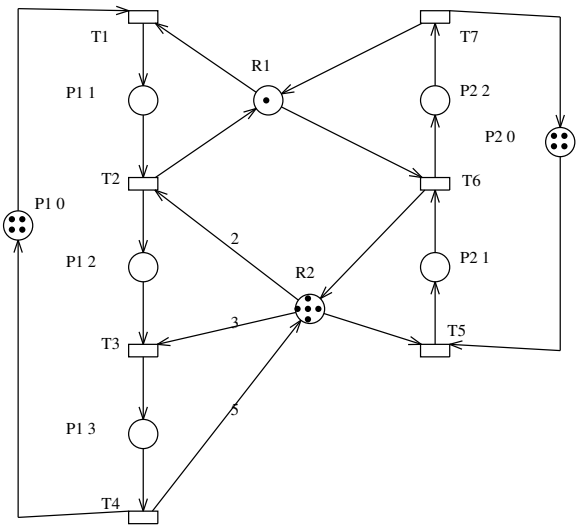

Fig. 1. A S^4PR with deadlock problems.

TABLE I. SIGN INCIDENCE MATRIX OF NET OF FIG. 1

	T1	T2	T3	T4	T5	T6	T7	
P1_1	+	−	0	0	0	0	0	1
P1_2	0	+	−	0	0	0	0	2
P1_3	0	0	+	−	0	0	0	3
P2_1	0	0	0	0	+	−	0	4
P2_2	0	0	0	0	0	+	−	5
P1_0	−	0	0	+	0	0	0	6
P2_0	0	0	0	0	−	0	+	7
R1	−	+	0	0	0	−	+	8
R2	0	−	−	+	−	+	0	9

	T1	T2	T3	T4	T5	T6	T7	
$P1\_1$	+	−	0	0	0	0	0	1
$R1$	−	+	0	0	0	−	+	8
$P1\_1 \oplus R1$	±	±	0	0	0	−	+	

The '±' symbols in the first and the second columns represent that, in the net, ${}^\bullet\{P1\_1 \cup R1\} \cap \{P1\_1 \cup R1\}^\bullet = \{T1, T2\}$. Moreover, the '−' symbol in the sixth column represents the fact that $T6 \in \{P1\_1 \cup R1\}^\bullet$, but it is not an input transition of this set; finally, the '+' symbol in the seventh column represents that $T7$ is an input transition of this set of places, but it is not an output transition.

The following theorem can be used for the computation of the siphons. It helps us with the selection of a set of adequate rows (places) of the *sign incidence matrix*.

Theorem 1 ([16]): Let $\mathcal{N} = \langle P, T, \mathbf{C}\rangle$ be a Petri net with $|T| = n$ and $|P| = m$. A subset of places, $S = \{p_1, p_2, ..., p_k\} \subseteq P$, is a siphon if and only if the addition of the k row vectors of the sing incidence matrix of $\mathcal{N}$, $A_1 \oplus A_2 \oplus ... \oplus A_k$, contains no '+' entries, where A_j denotes the row vector corresponding to place $p_j, j = 1, 2, ..., k$. $\diamondsuit$

Assuming the previous results hold, if the sum of these rows does not contain '+' entries, this means that each '+' in a row is cancelled with a '−', or a '±' (that is, for each place in the output of a transition, there exists a place of the net that is the input of this transition).

For example, if we use the sign incidence matrix of Table I, we can sum the rows corresponding to places $P1\_1$, $P2\_2$ and $R1$, obtaining the desired result,

	T1	T2	T3	T4	T5	T6	T7	
$P1\_1 \oplus R1$	±	±	0	0	0	−	+	
$P2\_2$	0	0	0	0	0	+	−	5
$P1\_1 \oplus P2\_2 \oplus R1$	±	±	0	0	0	±	±	

showing that they are a siphon, since ${}^\bullet\{P1\_1, P2\_2, R1\} = \{P1\_1, P2\_2, R1\}^\bullet$ (in this case, they are also a trap).

The method proposed in [16] was based on the iterative application of the previous 'cancellation' of '+' signs by means of the addition of matrix rows. Any resulting grouping of places such that the sum has no '+' signs corresponds to a siphon. In that approach the algorithm started selecting a matrix row, and then searching for some rows that could cancel the '+' signs. In our case we can avoid selecting a concrete row and let the genetic algorithm select a combination of rows: when their sum does not contain a '+' sign, the associated set of places is a siphon.

For each individual the variables will represent places (rows of the sign incidence matrix), and the algorithm will use as fitness function the number of places represented by active bits. In the example previously shown, the resulting individual would be: $(1, 0, 0, 0, 1, 0, 0, 1, 0)$, representing the siphon $\{P1\_1, P2\_2, R1\}$. The objective will be to minimize the number of active variables, in order to try to find the minimal siphons. The reason for this is that we cannot state by means of the proposed method the minimality, so we need to direct the GA to some interesting solution. We have chosen to compute the smaller siphons. We can imagine alternative objective functions that would take into account just the number of resource places, or the number of process places, or some more complex measurements. There are some situations that should be avoided: having an empty siphon is one of them (all the variables set to zero); other undesirable situation is to have all the places included (all the variables set to one); finally, we do not want to reach a solution such that the columns have a '+' sign in a variable since it would not be a siphon. In all of these cases, we penalize these types of solutions in the fitness function. In [18] we excluded the P–Semiflows from the solution for the set-based approach. Here we have not considered P–Semiflows, so they can be a solution of the genetic algorithm. In the previous example, the set $\{P1\_1, P2\_2, R1\}$ is a siphon, a trap and a P–Semiflow.

Since we want to obtain a result that minimizes the number of active variables (remember that they represent rows in the sign incidence matrix) in such a way that the corresponding sum has no postive signs, we need to combine this information in some way. For this, given a chromosome we can do the operations (the sum of the rows) and then we can see if there is some '+' sign in the result. In this case, we return a negative value. When we have an individual which represents an empty siphon or a siphon composed by all the places of the net, we can do the same.

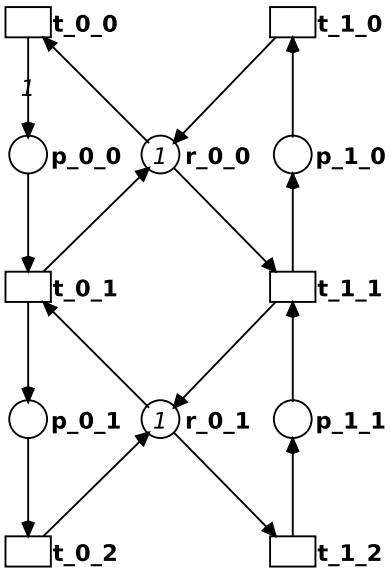

Fig. 2. Two processes with two stages sharing resources as in FMSAD and FMSLD models modeled as a Petri net.

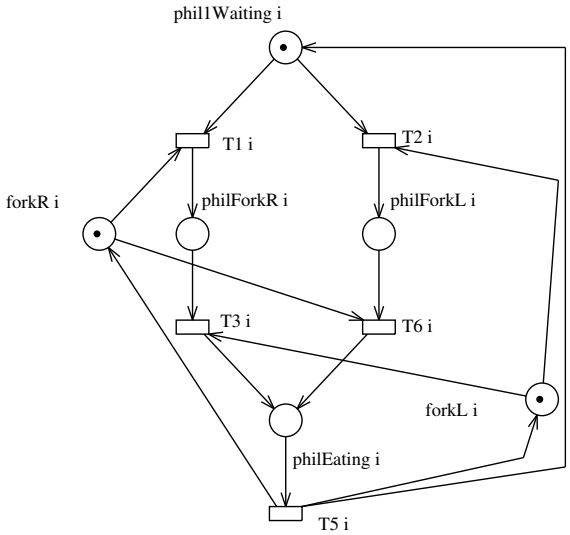

Fig. 3. The i–th philosopher problem modeled as a Petri net

V. THE EXPERIMENTS

We have compared the results for the nets used in [17], [18] as a benchmark for the performance of the methods. These nets belong to S^4PR class. It is a well–known subclass for the modeling of a wide set of RAS with a well-defined and easy to understand structure. Even the proposed method should allow us to look for siphons in any general Petri net, our previous work has concentrated in this class of nets and our examples belong to it. S^4PR nets allow the modeling of concurrent sequential processes with routing decisions and a general conservative use of resources. There is a more detailed presentation of some of these models in [19], [17], [12].

The first and second classes of systems are compositions of a set of sequential processes which have attached at each processing step a single (and different) resource. We can see and sketch of the Petri net representing two of such sequential processes (with the resources for the first process only) in Fig. 2. We can change the size for this family of systems in two ways: changing the length of the process; that is, the number of places for each process (two in the Figure). We can also change the number of processes (two in the Figure). Then, for a given number of processes of a fixed length we can compose them: Each process shares its resources with the following one in reverse order. So, if process i uses resources following the sequence $i_1, i_2, ..., i_m$, the next process will use them following the sequence $i_m, i_{m-1}, ..., i_1$ (m is the length and resources are associated to process i). The last process is composed with the first one following the same rules. With these rules, two different families of S^4PR nets can be generated, labeled as $FMSAD$ and $FMSLD$ in the tables. *FMSAD* nets have a variable number of sequential processes with a fixed length of 3. This variable number of processes is the parameter. In our case, we are composing from 3 to 8 processes. *FMSLD* have two sequential processes of variable length which is the parameter. We have made experiments with a length from 3 to 8. The last one corresponds to a model of the well known dining philosophers problem. In this case we can change the number of philosophers. Fig. 3 shows the model of the ith philosopher. Places $forkR\_i$ and $forkL\_i$ represent the right and left forks, respectively. Each fork will be shared with the philosopher that is on the corresponding place on the table. The results obtained for this family of nets are entitled *Phil* in the tables.

In [18] we recomputed the values shown in [17] to include time results obtained with the same computer for all the experiments. The results of the best one are shown in the comparison of Fig. 4. All the computations have been done with a desktop computer, Intel(R) Core(TM) i5-2400 CPU @ 3.10GHz, with 4Gb of RAM. Notice also that for each type of problem the next size takes more than 24 hours to compute the set of siphons with the first method.

For these new experiments, we have measured the time used by the genetic algorithm to obtain at least one siphon; the GA should be able to compute more than one (just selecting the adequate set of best fit individuals). In any case, it would be difficult to predict the number of good siphons, so let us use this time as a conservative measure.

The genetic algorithm has several parameters that need to be adjusted. We have used elitism and rank-based selection. For mutation we have used a bitflip operation with 33% of probability and we have selected a two-point crossover operator with probability of 66%. These are the parameters used by default in the `tide_bitstring.pl` example, and we did not modify them. We have then focused on finding the right size for the initial population and the correct number of

TABLE II. TIMES FOR SIPHON COMPUTATION WITH THE PROPOSED METHOD

	Size	Population	Time	Evaluations
FMSAD	3	16	0,03 (0,01)	247 (83,95)
	3	32	0,05 (0,01)	479 (76,77)
	3	64	**0,08** (0,01)	834 (141,33)
	4	16	0,04 (0,01)	226 (80,94)
	4	32	0,09 (0,02)	565 (145,35)
	4	64	**0,18** (0,03)	1179 (215,43)
	5	16	0,05 (0,02)	205 (98,40)
	5	32	0,11 (0,04)	483 (209,11)
	5	64	**0,27** (0,08)	1225 (395,43)
	6	32	0,14 (0,08)	457 (268,78)
	6	64	0,31 (0,12)	1036 (463,36)
	6	128	**0,85** (0,28)	2964 (955,49)
	7	32	0,16 (0,08)	407 (228,09)
	7	64	0,34 (0,15)	909 (443,33)
	7	128	0,93 (0,42)	2474 (1.205,99)
	7	256	**1,84** (0,77)	5016 (2.368,35)
	8	64	0,40 (0,19)	822 (463,71)
	8	128	0,88 (0,44)	1842 (1.025,05)
	8	256	**2,04** (1,20)	4248 (2.669,90)
	9	64	0,41 (0,12)	664 (214,16)
FMSLD	3	8	0,01 (0,00)	107 (29,72)
	3	16	0,01 (0,00)	208 (50,96)
	3	32	**0,03** (0,00)	405 (64,78)
	4	8	0,01 (0,00)	118 (31,09)
	4	16	0,02 (0,00)	228 (35,98)
	4	32	**0,04** (0,01)	462 (75,54)
	5	16	0,03 (0,01)	272 (51,70)
	5	32	**0,05** (0,01)	491 (74,13)
	6	16	0,04 (0,01)	270 (68,93)
	6	32	**0,07** (0,01)	535 (105,53)
	7	16	0,05 (0,01)	323 (74,41)
	7	32	**0,10** (0,01)	637 (84,94)
	8	16	0,07 (0,02)	336 (106,79)
	8	32	0,13 (0,01)	661 (83,84)
	8	64	**0,23** (0,04)	1222 (208,84)
Phil	3	16	0,02 (0,01)	200 (60,89)
	3	32	0,05 (0,01)	445 (99,29)
	3	64	**0,08** (0,01)	834 (141,33)
	4	32	0,06 (0,02)	391 (118,45)
	4	64	**0,13** (0,04)	887 (232,26)
	5	64	0,16 (0,06)	790 (271,09)
	5	128	**0,38** (0,11)	1876 (528,81)
	6	64	0,19 (0,06)	707 (205,14)
	6	128	**0,42** (0,15)	1559 (587,77)
	7	64	0,23 (0,05)	661 (137,34)
	7	128	**0,44** (0,07)	1281 (167,60)
	8	64	0,27 (0,01)	625 (0,00)
	8	128	**0,58** (0,14)	1325 (285,92)

Column 1: Name (as in [17])
Column 2: Size of the problem.
Column 3: Population of the instance.
Column 4: Average time. P–Semiflows in the initial population. Random initial population.
Column 5: Average number of evaluations (rounded). Random initial population.
In all the cases, in parentheses, the standard deviation.
In **boldface** we have marked the cases with the initial population of the same size as in Table III for comparison.

TABLE III. TIMES FOR SIPHON COMPUTATION WITH THE METHOD PROPOSED IN [18]

			Initial Population	
			Random	
	Size	Pop.	Time	Eval.
FMSAD	3	64	0.96 (0.18)	938 (169.27)
	4	64	1.92 (0.32)	1,082 (185.82)
	5	64	3.33 (0.41)	1,222 (152.79)
	6	128	12.61 (12.70)	2,657 (394.46)
	7	256	31.71 (3.47)	5,954 (644.96)
	8	256	50.95 (17.79)	7,009 (1,231.68)
FMSLD	3	32	0.50 (1.73)	379 (73.60)
	4	32	0.36 (0.07)	447 (84.49)
	5	32	0.65 (0.10)	526 (79.82)
	6	32	1.00 (0.21)	562 (117.14)
	7	32	1.59 (0.85)	661 (343.29)
	8	64	4.05 (0.78)	1,310 (248.29)
Phil	3	64	0.34 (0.03)	812 (64.70)
	4	64	0.62 (0.07)	863 (91.17)
	5	128	2.03 (0.23)	1,859 (195.90)
	6	128	3.18 (0.38)	2,042 (240.61)
	7	128	4.60 (0.41)	2,185 (183.80)
	8	128	6.40 (0.67)	2,362 (234.06)

Column 1: Name (as in [17])
Column 2: Size of the problem.
Column 3: Population of the instance.
Column 4: Average time. Random initial population.
Column 5: Average number of evaluations (rounded). Random initial population.
In all the cases, in parentheses, the standard deviation.

evaluations. We start the experiment for each example with an initial population of size 8 and we run the program thirty times; if it fails (does not compute a siphon) more than once, we double the size of the initial population and repeat until we can reach thirty iterations with at most one failed result. We also established a maximum number of evaluations: if no solution is found after this number of evaluations the algorithm stops (and we consider this run a failure). The initial population is fully random. In [18] we tested the inclusion of the initial population with the P–Semiflows but not significant differences appeared, so we discarded that approach for this work. When the algorithm stops, we can check whether the solution with best fitness is a siphon or not: if it has not positive fitness it won't be a siphon.

The results obtained can be seen in Table II. They also can be seen (with the standard deviation) in a graphical way in the second column of Fig. 4 and a comparison with the algebraic algebraic method which serves as inspiration for the GA in the first column. The times shown for the genetic algorithm are the average of the thirty runs of each experiment with the smaller acceptable initial population for each size of each problem. The time needed to reach a solution has been included in TableeII; these results have been computed using the same population size as our previous work (they are highlighted with boldface in Table II) and some intermediate cases (filling the gap by duplicating the initial population size). We can see that the incidence matrix-based genetic algorithm is faster than the previous set-based genetic one presented in [18]. The results of the experiments of this set-based genetic method can be seen in Table III for comparison. It requires smaller populations and less evaluations to complete its task. In the comparison with the classical algebraic approach this genetic approach is faster for the FMSxD models but slower in the case of Philosophers example (column 1 of Fig. 4).There are two things to remark here: the results should not be compared directly, since the algebraic implementations where done in C, and the genetic algorithm has been programmed using Perl. The second thing to note is that the genetic algorithm does not obtain all the siphons. If we were interested in computing all the siphons, we could add the computed ones as negative restrictions (this set of places cannot be a solution) and apply again the GA. There we saw that bigger initial populations where better for obtaining more siphons running the algorithm several times. We should expect a similar behaviour here.

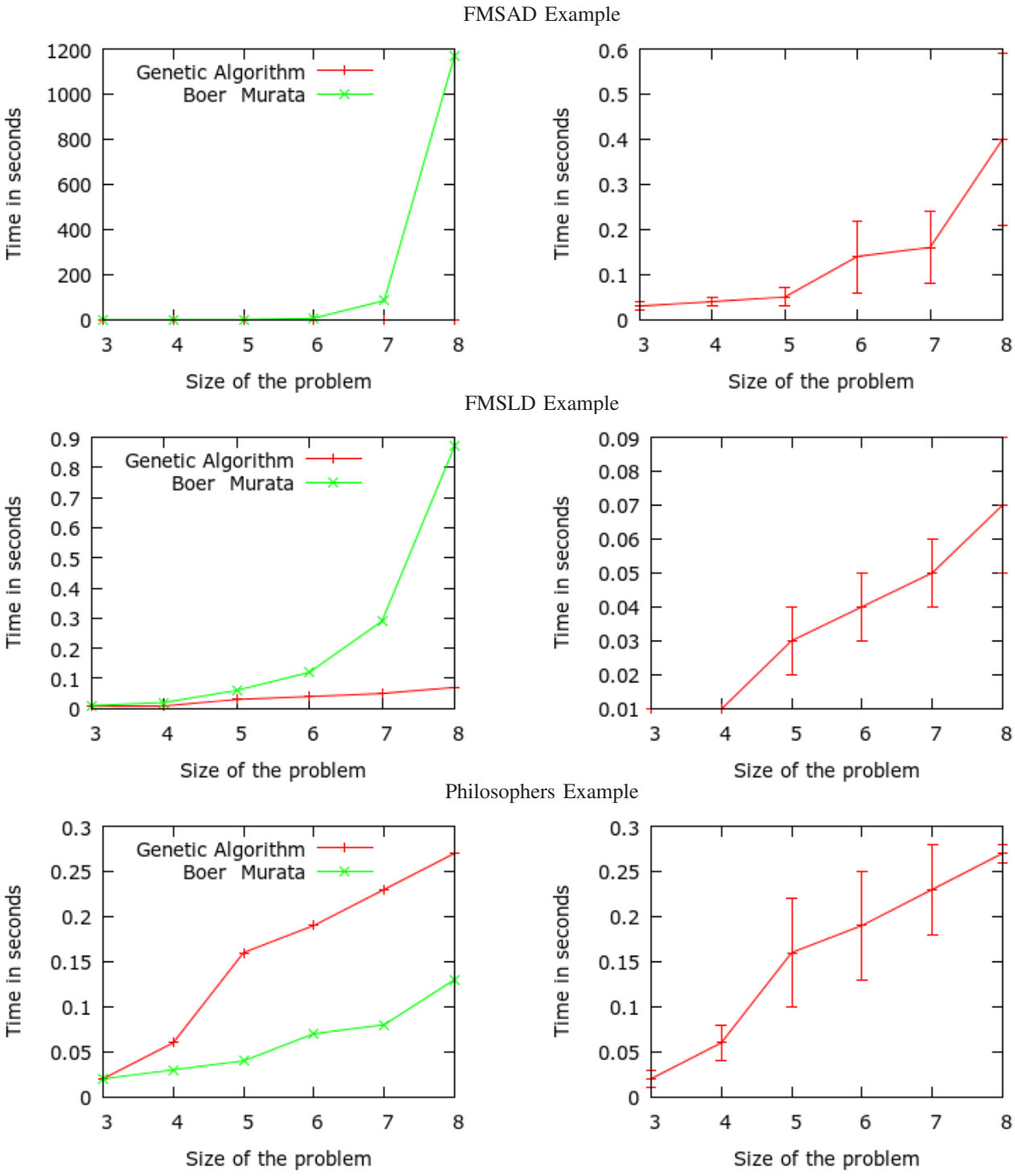

Fig. 4. Comparison of times for different examples and sizes

VI. Conclusions and Further Work

In order to avoid control systems going into a situation of deadlock, some policies require the computation of a set of configurations named minimal siphons, which has been proved to be a complex endeavor since the number of these siphons can be very high even in systems with a small size. This paper has further worked on a previous attempt to study the problem.

An adaptation of a method based on the incidence matrix

has been introduced in order to try to explore the use of genetic algorithms to search for structural components of the net (siphons).

The method seems to have better performance than the one presented in a previous work but the question of whether it is adequate when we need to compute all the siphons is still open.

Moreover, when the computation of all the siphons becomes prohibitively expensive, the genetic algorithm can still deal with bigger problems if it is acceptable for us to have a partial set of the siphons instead of the whole set provided by algebraic approaches.

In this sense, our proposal for further work will follow several ideas: First of all, the genetic algorithm is well suited for parallelization as in [17]. Second, the problem can be formulated not only in terms of siphon computation but in terms of a problem with more information. In the last years some ideas have been proposed in order to avoid the computation of all the minimal siphons. The methods rely on the computation of some special bad siphons together with bad markings (structural objects and bad states information is merged): if we introduce the state equation the genetic algorithm will have more information and, hopefully, it will be an alternative method to the one proposed in previously published work [20].

ACKNOWLEDGMENT

The authors are indebted to the anonymous referees and the PC who contributed to improve the quality and presentation of this paper from its first version. This work is supported in part by project ANYSELF (TIN2011-28627-C04-02) by the Spanish Mineco, and TIN2013-40809-R by the Spanish Ministry of Science and Innovation; P08-TIC-03903 awarded by the Andalusian Regional Government, and by Group of Discrete Event Systems Engineering (GISED) awarded by Aragonese Government.

REFERENCES

[1] T. Murata, "Petri nets: properties, analysis and applications," *Proceedings of the IEEE*, vol. 77, no. 4, pp. 541–580, Apr. 1989.

[2] J. Ezpeleta, J. Colom, and J. Martínez, "A Petri net based deadlock prevention policy for flexible manufacturing systems," *IEEE Trans. Rob. Aut.*, vol. 11, no. 2, pp. 173–184, 1995.

[3] K. Barkaoui and J. Pradat-Peyre, "On Liveness and Controlled Siphons in Petri Nets," in *Proceedings of the 1996 International Conference on Aplications and Theory of Petri Nets*, J. Billington and W. Reisig, Eds. Springer Verlag, Jun. 1996.

[4] F. Tricas, F. García-Vallés, J. Colom, and J. Ezpeleta, "An Iterative Method for Deadlock Prevention in FMS," in *Discrete Event Systems: Analysis and Control. Proc. of WODES*, R. Boel and G. Stremersch, Eds., Ghent, Belgium, 2000, pp. 139–148.

[5] F. Tricas, J. Colom, and J. Ezpeleta, "A solution to the problem of deadlocks in concurrent systems using Petri nets and integer linear programming," in *Proc. of the 11th European Simulation Symposium*, G. Horton, D. Moller, and U. Rude, Eds. Erlangen, Germany: The society for Computer Simulation International, oct 1999.

[6] Y. Huang, M. D. Jeng, Z. Xie, and S. Chung, "Deadlock prevention policy based on Petri nets and siphons." *Int. Journal of Production Research*, vol. 39, no. 2, pp. 283–305, 2001.

[7] M. V. Iordache, J. O. Moody, and P. Antsaklis, "Synthesis of Deadlock Prevention Supervisors Using Petri Nets," *IEEE Trans. Rob. Automat.*, vol. 18, no. 1, pp. 59–68, 2002.

[8] Z. Li and M. C. Zhou, "Elementary Siphons of Petri Nets and Their Applications to Deadlock Prevention in Flexible Manufacturing Systems," *IEEE Trans. on Systems, Man, and Cybernetics*, vol. 34, no. 1, pp. 38–51, Jan. 2004.

[9] K. Barkaoui, A. Chaoui, and B. Zouari, "Supervisory Control of Discrete Event Systems Based on Structure of Petri Nets," in *Proceedings of the 1997 IEEE International Conference on Systems, Man and Cybernetics. Computational Cybernetics and Simulation*. Orlando, Florida, USA: IEEE, October 1997, pp. 3750–3755.

[10] S. Li, Z. Li, H. Hu, A. Al-Ahmari, and A. An, "An extraction algorithm for a set of elementary siphons based on mixed-integer programming," *Journal of Systems Science and Systems Engineering*, vol. 21, no. 1, pp. 106–125, Mar. 2012. [Online]. Available: http://dx.doi.org/10.1007/s11518-012-5188-z

[11] M. Jeng, M. Peng, and Y. Huang, "An algorithm for calculating minimal siphons and traps of Petri nets," *Int. J. of Intelligent Control and Systems*, vol. 3, no. 3, pp. 263–275, 1999.

[12] F. Tricas, *Deadlock Analysis, Prevention and Avoidance in Sequential Resource Allocation Systems*, Ph.D. Thesis. Dep. Inf. e Ing. de Sist. U. Zaragoza, May 2003.

[13] J. H. Holland, *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*. Oxford, England: U Michigan Press.

[14] J. J. Merelo, "A Perl Primer for Evolutionary Algorithm Practitioners," *SIGEVOlution*, vol. 4, no. 4, pp. 12–19, Mar. 2010. [Online]. Available: http://dx.doi.org/10.1145/1810136.1810138

[15] J.-J. Merelo-Guervós, P.-A. Castillo, and E. Alba, "`Algorithm::Evolutionary`, a flexible Perl module for evolutionary computation," *Soft Computing*, vol. 14, no. 10, pp. 1091–1109, 2010, accesible at http://sl.ugr.es/000K[sl.ugr.es]. [Online]. Available: http://www.springerlink.com/content/8h025g83j0q68270/fulltext.pdf[www.springerlink.com]

[16] E. R. Boer and T. Murata, "Generating basis siphons and traps of Petri nets using the sign incidence matrix." *IEEE Trans. on Circuits and Systems, I – Fundamental Theory and Applications*, vol. 41, no. 4, pp. 266–271, 1994.

[17] F. Tricas and J. Ezpeleta, "Computing minimal siphons in Petri net models of resource allocation systems: a parallel solution," *Sys. Man Cyber. Part A: Systems and Humans, IEEE Trans. on*, vol. 36, no. 3, pp. 532–539, 2006. [Online]. Available: http://dx.doi.org/10.1109/TSMCA.2005.855751

[18] F. Tricas, J. Colom, and J. Merelo, "Computing minimal siphons in petri net models of resource allocation systems: An evolutionary approach," in *International Workshop on Petri Nets and Software Engineering (PNSE'14)*, D. Moldt and H. Rölke, Eds., Tunis, Tunisia, jun 2014, pp. 307–322.

[19] F. Tricas and J. Ezpeleta, "RessAllocation Petri net Model." in *Model Checking Contest 2013*, F. Kordon and et al., Eds., Jun. 2013.

[20] F. Tricas, F. García-Vallés, J. Colom, and J. Ezpeleta, "A Petri Net Structure–Based Deadlock Prevention Solution for Sequential Resource Allocation Systems," in *Proc of 2005 Int. Conf. on Robotics and Automation*, Barcelona, Spain, 2005, pp. 272–278.