Computing Minimal Siphons in Petri Net Models of Resource Allocation Systems: An Evolutionary Approach

Fernando Tricas[1], José Manuel Colom[1], and Juan Julián Merelo[2]

[1] Depto de Informática e Ingeniería de Sistemas
Universidad de Zaragoza
{ftricas,jm}@unizar.es
[2] 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 have been 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 this work a formulation of the siphon property using linear constraints is presented for general Petri Nets. 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 will show how siphons can be computed using it, 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, computing, deadlock prevention

1 Introduction

A Resource Allocation System (RAS) is a discrete event system in which a finite set of concurrent processes shares in a competitive way a finite set of resources. RAS are usually complex enough to take advantage of the use of formal methods, which can help 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].

Software systems are also complex systems that can be seen as a set of processes sharing (and competing for) resources. There is some recent work in this

area, such as [2] where a more detailed discussion of similarities and differences with Flexible Manufacturing Systems (FMS) as the archetypal example of RAS can be seen. In [3, 4] there is some work related to software systems and special classes of Petri Nets for concurrency problems. The competition for resources implies the existence of deadlocks; they occur when some processes are waiting for the evolution of other processes, that are also waiting for the former ones to evolve (the dependence does not need to be direct). RAS have proved to be specially useful when synthesizing deadlock avoidance and prevention policies, and many of the published work relies on minimal siphons for this [5–10]. A minimal siphon is a set of places such that 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 [10] some promising work has been done in the field of Flexible Manufacturing Systems. They propose to reduce the number of siphons to be considered for deadlock prevention, but they do not avoid the computation of the whole set of minimal siphons. In most cases siphon enumeration cannot be avoided, and this makes interesting to obtain better methods to find them ([5, 11, 12, 9]).

In this work we are going to propose a genetic algorithm (GA). that uses a formulation of the siphon property by means on linear constraints. 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. This approach opens the door to adapt another siphon-based techniques for deadlock prevention.

The contents of this paper are organized as follows. Section 2 provides an introduction to Petri Nets and the main concepts related to the problem, Section 3 presents the standard Genetic Algorithm. There is also some information about methods existing in the literature for solving the same problem, Section 4 presents the adapted method, Section 5 shows our experimental setup and the experimental results, together with some discussion about them. Finally, some conclusions are presented.

2 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 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 the *pre-set* of x, while $x^\bullet = \{y \in P \cup T \mid W(x, y) > 0\}$ is 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: $\|\mathbf{y}\| = \{p \in P \mid \mathbf{y}[p] > 0\}$. Let $\mathcal{PS}$ be the set of minimal P–Semiflows of $\mathcal{N}$. A (Semi)flow is 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: $\forall \mathbf{m_0} \,.\, \forall \mathbf{m} \in \mathrm{PRS}(\mathcal{N}, \mathbf{m_0}) \,.\, \mathbf{y} \cdot \mathbf{m} = \mathbf{y} \cdot \mathbf{m_0}$ (cyclic behavior law).

Given $\mathcal{N}$ an ordinary Petri net, 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 never be marked. 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 [13]. 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 [14, 15]. Methods based on logic formulas are based on characterizing siphons by means of boolean variables, which typically represent places or transitions and their relations. Methods using this approach can be found in [16, 17].

3 Genetic Algorithms

Genetic algorithms [18] are inspired by Darwin's theory about evolution and its genetic-molecular basis. More technically the genetic algorithm is a search heuristic that mimics the process of natural selection. A random population

of candidate solutions is evolved trying to explore the search space looking for better solutions. The sketch of the basic genetic algorithm is [19]:

1. (Start) Generate random population of n chromosomes (suitable solutions)
2. (Fitness) Evaluate the fitness of each chromosome in the population
3. (New population) Create a new population by repeating the following steps until the new population is complete
   (a) (Selection) Select two parent chromosomes from a population according to their fitness (the better fitness, the bigger chance to be selected)
   (b) (Crossover) With some probability cross over the parents to form a new offspring (children). If no crossover was performed, offspring is an exact copy of parents.
   (c) (Mutation) With a mutation probability mutate new offspring at each locus (position in chromosome).
   (d) (Accepting) Place new offspring in a new population
4. (Replace) Use new generated population for a further run of algorithm
5. (Test) If the end condition is satisfied, stop, and return the best solution in current population
6. (Loop) Go to step 2

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` [20] 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++.

4 The Proposed Approach

As far as we know, there are no approaches using genetic algorithms to compute structural properties of Petri Net models. Some work has been done on process mining and scheduling [21–24]. A siphon is a special set of places, as defined above. In [25] the method presented in [26] (algebraic based) was selected, taking advantage of a parallel approach. Here we will explore a logic formula based approach: with the formulation for siphons presented in [27] we will explore the space state by means of the use of a genetic algorithm in its more classic way.

It is straightforward to try to use the standard GA without much difficulty: each place p of the Petri net will be represented by means of a binary variable v_p. The siphon property can be represented as follows:

$$\forall p \in P, \forall t \in {}^\bullet p, v_p \leq \sum_{q \in {}^\bullet t} v_q, \text{with } v_q, v_p \in \{0, 1\} \tag{1}$$

The siphon would be composed of the set of places whose corresponding variable equals to one, $v_p = 1$. The meaning of each equation is that if place p is in the set (it belongs to the siphon) it must contain, at least, one of the

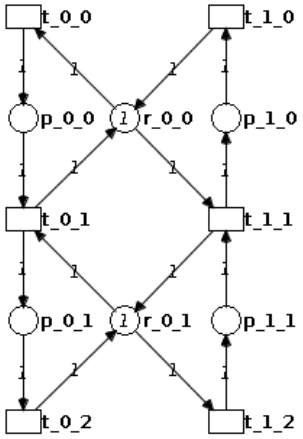
places that are in the pre–set of each of its entry transitions. This needs to be completed with some restrictions that avoid undesired situations:

$$\sum_{p \in P \setminus P_0} v_p < |P \setminus P_0| \tag{2}$$

That is, we are not interested in the whole set of places since it is a siphon but it is an uninteresting one. Finally,

$$\forall \mathbf{Y} \in \mathcal{PS}, \sum_{p \in \mathbf{Y}} v_p < \|\mathbf{Y}\| \tag{3}$$

In this case, the selected set of places cannot be a P-Semiflow, since they are uninteresting siphons. P-Semiflows cannot be emptied because of the cyclic behavior law described above. Moreover, they are much less expensive from a computational point of view. For the Figure 1 and the set of equations shown there the assignment $v_{p\_0\_1} = v_{p\_1\_0} = v_{r\_0\_0} = v_{r\_0\_1} = 1$ is a solution and the set of places defined by them is a minimal siphon ($\{p\_0\_1$, p_1_0, r_0_0, $r\_0\_1\}$). It is easy to see that if we add $v_{p\_0\_0} = 1$ to the previous solution the equations remain true. This is one of the problems of this method: these equations can describe siphons, but they do not need to be minimal.



Set of equations:
Related to Equation 1:
$v_{p\_0\_0} \leq v_{r\_0\_0}$
$v_{p\_0\_1} \leq v_{p\_0\_0} + v_{r\_0\_1}$
$v_{p\_1\_0} \leq v_{p\_1\_1} + v_{r\_0\_0}$
$v_{p\_1\_1} \leq v_{r\_0\_1}$
$v_{r\_0\_0} \leq v_{p\_0\_0} + v_{r\_0\_1}$
$v_{r\_0\_0} \leq v_{p\_1\_0}$
$v_{r\_0\_1} \leq v_{p\_1\_1} + v_{r\_0\_0}$
$v_{r\_0\_1} \leq v_{p\_0\_1}$
Related to Equation 2:
$v_{p\_0\_0}+v_{p\_0\_1}+v_{p\_1\_0}+v_{p\_1\_1}+v_{r\_0\_0}+ v_{r\_0\_1} < 6$
Related to Equation 3:
$v_{p\_0\_0} + v_{p\_1\_0} + v_{r\_0\_0} < 3$
$v_{p\_0\_1} + v_{p\_1\_1} + v_{r\_0\_1} < 3$

Fig. 1. A very simple Petri net and the equations that represent its siphons. The idle places of the system have not been represented for the shake of brevity.

With this formulation we can construct a fitness function for the genetic algorithm that can guide the system towards a solution. As each variable can have a value of 0 or 1, this approach is well-suited to be formulated as a genetic algorithm. A final remark is that the genetic algorithm is an optimization algorithm

so some objective function is needed. We have decided to minimize the number of active variables. Since we cannot state by means of a simple logical formula the minimality property, 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, the number of process places, or some more complex measurements. The complete system would be:

$$\begin{array}{l} min \sum_{p \in P} v_p \\ \forall p \in P, \forall t \in {}^\bullet p, v_p \leq \sum_{q \in {}^\bullet t} v_q, v_p \in \{0,1\} \\ \sum_{p \in P \setminus P_0} v_p < |P \setminus P_0| \\ \forall \mathbf{Y} \in \mathcal{PS}, \sum_{p \in \mathbf{Y}} v_p < \|\mathbf{Y}\| \end{array} \tag{4}$$

Since we want to obtain a result that minimizes the function and that satisfies the restrictions we need to combine this information. When we have an individual which represents an empty siphon or a siphon composed by all the places of the net, we can return a negative number, equivalent to twice the number of restrictions (the idea is to help de GA to avoid these solutions). For the other restrictions, we can just count the number of places in the siphon when they are met. When there are unmet restrictions, we just return the difference between the number of such restrictions and the total number of restrictions (this is a negative number, that grows when more restrictions are met). We have tried several configurations giving more weight to the number of places in the siphon or to the number of satisfied restrictions but not significant differences appear.

5 The Experiments

We have compared the nets used in [25] as a benchmark of the performance of the methods. These nets belong to S^4PR class. It is a well–know 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 PN, 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 [29, 25]. The first and second classes of systems are obtained by means of the composition of a set of sequential processes: each process, at each processing step, has attached a single (and different) resource. An instance of the Petri net representing two of such sequential processes of length two would follow the structure of the net in Figure 1 (only the resources for the first process are shown). There are two ways to study size variations in this family of systems: one of them is changing the length of the process; that is, the number of processing steps (two in the figure). The second one is changing the number of processes to be composed (in the figure two processes are shown). For the experiment, the sequential processes are composed with other processes according to the following rules: The first process shares its resources with the second one in reverse order: the resource used at the first step in the first process is used at the last step of the second

Table 1. Computing the minimal siphons with the algebraic methods

Name	Size	Number of siphons	[28]	[26]
FMSAD	3	42	0	0.07
	4	78	0.04	0.17
	5	150	0.69	0.72
	6	250	13.83	6.27
	7	490	466.95	84.54
	8	906	11127.44	1169.57
FMSLD	3	24	0	0.01
	4	54	0	0.02
	5	116	0.08	0.06
	6	242	2.27	0.12
	7	496	71.78	0.29
	8	1006	2570.88	0.87
Phil	3	10	0	0.02
	4	17	0.02	0.03
	5	26	0.34	0.04
	6	37	6.88	0.07
	7	50	291.58	0.08
	8	65	6930.71	0.13

Column 1: Name as in [25].
Column 2: Size of the problem (number of processes in FMSAD; length of two parallel processes in FMSLD; number of philosophers in Phil).
Column 3: Number of minimal siphons (computed by means of an algebraic algorithm).
Column 4: Time needed to compute all the siphons (in seconds) with the method described by Lautenbach in [28].
Column 5: Time needed to compute all the siphons (in seconds) with the method described by Boer and Murata in [26].

process; the resource used at the second step of the first process is used by the one that is previous to the last step in the second process, and so on. The second process is composed with the third one in a similar way and so on, until we reach the total number of composed processes. The last process is composed with the first one, in a similar way. Using the previous ideas two different families of S^4PR nets have been generated, labeled as *FMSAD* and *FMSLD* in the tables.

FMSAD nets are obtained by means of the composition of a variable number of sequential processes as the ones depicted in Figure 1, with a fixed length of 3. The number of processes to be composed in parallel is the parameter. In the experiments, the number of composed processes is varying from 3 to 8. *FMSLD* nets are obtained by means of the composition of a fixed number of two sequential processes as the ones depicted in Figure 1, with a variable length which is the parameter. In the experiments, this length is varying from 3 to 8.

The last one corresponds to an implementation of the well known dining philosophers problem. The parameter corresponds to the number of philosophers.

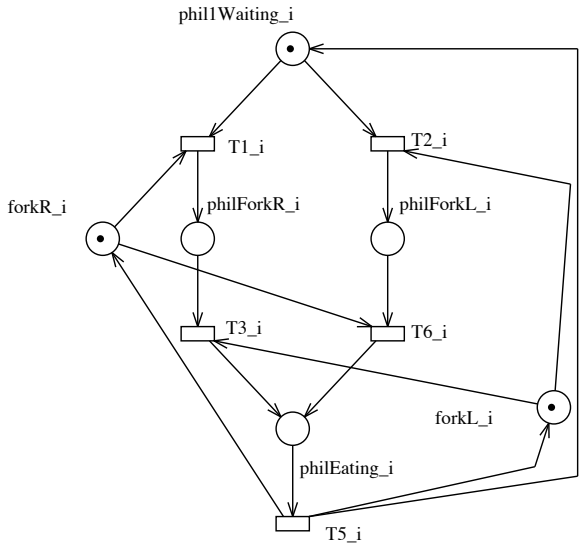


Fig. 2. Petri net model of the i–th philosopher

Figure 2 shows the model of the ith philosopher. Places $forkR\_i$ and $forkL\_i$ model, respectively, the state (free/engaged) of its right and left forks. The fork $forkR\_i$ will be shared with the philosopher on his right, and the $forkL\_i$ will be shared with the philosopher on his left. The results obtained for this family of nets are entitled *Phil* in the tables.

We have recomputed the values shown in [25] to include time results obtained with the same computer for all the experiments. This has been done with a desktop computer, Intel(R) Core(TM) i5-2400 CPU @ 3.10GHz, with 4Gb of RAM. These results can be seen in Table 1. They are shown in a graphical way in the first column of Figure 3. Notice also that for each type of problem the next size takes more than 24 hours to finish the computation 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) and we also measured this. 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 bitblip operation with 33% of probability and we have selected a two-point crossover operator with probability of 66%. Then, we have concentrated on the size of the

initial population and the number of 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).

We have tested two approaches for the initial population: First, introducing the P–Semiflows (when there are less P–Semiflows than the size of the initial population we add the needed individuals at random; when there are more, we add all of them and we complete the population until we reach a multiple of eight individuals). Second, using a fully random initial population. The reason for trying the first approach is that P–Semiflows could guide the algorithm toward interesting places in the net (in some classes of nets it is possible to construct them as a seed [30]). Notice that they cannot be part of the solution (they are explicitly forbidden, see equation (3) in Section 4).

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 2. They also can be seen in a graphical way in the second column of Figure 3 and in Figure 4. In the Figure 4 we have also included the standard deviation of the thirty runs of the program. We have included in the Figure 3 the graphics for the algebraic methods as a baseline. The times provided in the table and in the figures for the genetic algorithms are the average of the thirty runs of each experiment with the smaller acceptable initial population for each size of each problem.

We can see that the genetic algorithm is slower, in general, than the algebraic methods except for the case of FMSAD example, where the genetic algorithm seems to obtain its solution in less time (and it grows slowly if we compare with the algebraic method, which seems to grow exponentially).

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 (an interpreted language). Nevertheless, putting the results together helps us to see that they are not so far away and that the approach can be adequate for some types of problems or when the size grows in such a way that it cannot be managed with deterministic methods. The second thing to note is that the genetic algorithm does not obtain all the siphons but a number of them (as the best fitted members of the final population).

In Table 2 we can see that there are no relevant differences in time when using the P–Semiflows as the initial population and when we use a random initial population. In the FMSAD example there is a small difference in the number of evaluations, which tend to be bigger (but the differences are small and there are cases when there are less evaluations with the random population -sizes 4, 5-). In the FMSLD example the number of evaluations tends to be lower for the initial random population (except for sizes 5,7). Finally, in the Phil example, the cases

Table 2. Times for siphon computation with the proposed method

			Initial Population			
			P–Semiflows		Random	
	Size	Pop.	Time	Eval.	Time	Eval.
FMSAD	3	64	0.93 (0.22)	912 (214.83)	0.96 (0.18)	938 (169.27)
	4	64	1.97 (0.33)	1,102 (185.03)	1.92 (0.32)	1,082 (185.82)
	5	64	3.97 (1.01)	1,448 (361.33)	3.33 (0.41)	1,222 (152.79)
	6	128	10.16 (1.19)	2,627 (305.12)	12.61 (12.70)	2,657 (394.46)
	7	256	31.07 (4.65)	5,877 (785.53)	31.71 (3.47)	5,954 (644.96)
	8	256	43.18 (4.65)	6,299 (677.50)	50.95 (17.79)	7,009 (1,231.68)
FMSLD	3	32	0.19 (0.03)	408 (72.73)	0.50 (1.73)	379 (73.60)
	4	32	0.37 (0.09)	458 (112.10)	0.36 (0.07)	447 (84.49)
	5	32	0.65 (0.10)	519 (79.26)	0.65 (0.10)	526 (79.82)
	6	32	1.01 (0.22)	571 (124.39)	1.00 (0.21)	562 (117.14)
	7	32	1.59 (0.34)	654 (137.96)	1.59 (0.85)	661 (343.29)
	8	64	4.14 (1.18)	1,349 (385.48)	4.05 (0.78)	1,310 (248.29)
Phil	3	64	0.33 (0.05)	790 (106.88)	0.34 (0.03)	812 (64.70)
	4	64	0.64 (0.06)	887 (92.53)	0.62 (0.07)	863 (91.17)
	5	128	2.02 (0.20)	1,852 (195.65)	2.03 (0.23)	1,859 (195.90)
	6	128	3.08 (0.35)	1,988 (217.09)	3.18 (0.38)	2,042 (240.61)
	7	128	4.77 (0.52)	2,260 (237.10)	4.60 (0.41)	2,185 (183.80)
	8	128	6.61 (0.60)	2,454 (221.54)	6.40 (0.67)	2,362 (234.06)

Column 1: Name (as in [25])
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). P–Semiflows in the initial population.
Column 6: Average time. Random initial population.
Column 7: Average number of evaluations (rounded). Random initial population.
In all the cases, in parentheses, the standard deviation.

where the number of evaluations is better is the same for both initial types of initial population.

The results obtained show that the approach is suitable: we can compute (minimal) siphons with the proposed method. Comparing with traditional methods the genetic approach does not provide better time computation except for one example (but they are implementations in different languages) and the behavior is better with more complex problems (as one would expect). 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, as we have done with P–Semiflows) and apply again the GA.

As another way to evaluate the approach we computed Table 3 where we can see the total number of different siphons obtained with the proposed method compared to the total number of siphons for each system. For this we have used

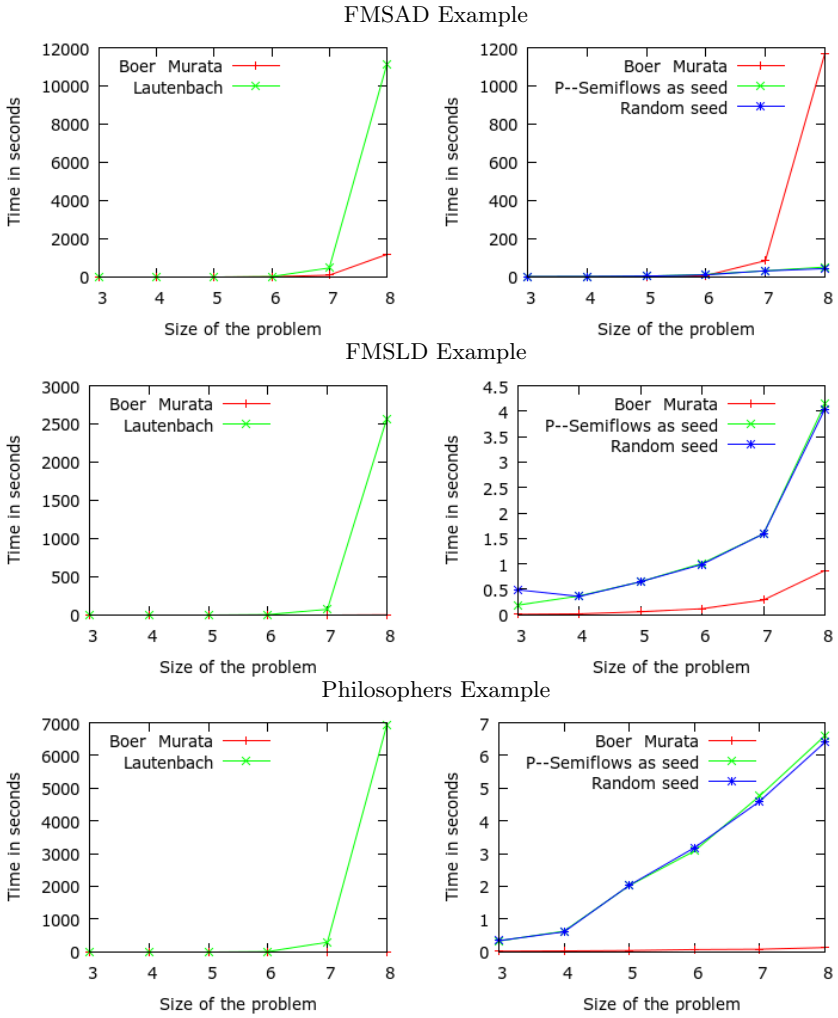


Fig. 3. Comparison of times for different examples and sizes

the same experiments as in the previous table: we can count the number of different siphons for each size of each problem in the 30 runs of the experiment. With this we can show that the genetic algorithm has a good behavior (different runs examine different parts of the solutions space) but we are not measuring what would happen with the addition of new restrictions to forbid siphons that have been computed previously. Moreover, when the size of the problem increases the method computes less siphons. Our feeling is that this is due to the size of the population (the size is small compared to the number of total siphons when

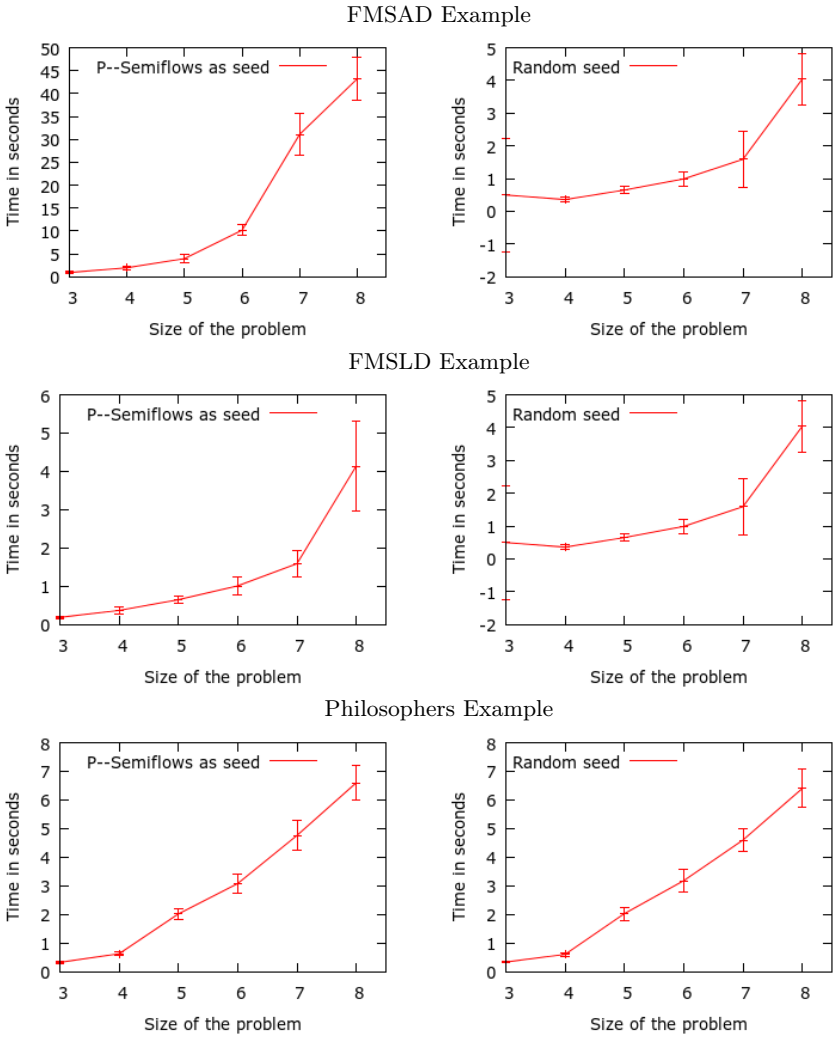


Fig. 4. Comparison of times for different examples and sizes

the size of the problem grows). For this reason we have added columns 6 and 7. There we can see that the number of different siphons computed increases with a bigger initial population. We can also see in that Table that the random initial population tends to produce more different siphons across different experiments.

Table 3. Number of siphons with different methods and populations

		Number of siphons	P	R	P	R	Percentage			
FMSAD	3	42	22	28	25	28	52.38%	66.67%	59.52%	66.67%
	4	78	29	32	34	42	37.18%	41.03%	43.59%	53.85%
	5	150	42	48	44	49	28%	32%	29.33%	32.67%
	6	250	48	53	55	68	19.20%	21.20%	22%	27.20%
	7	490	75	70	90	83	15.31%	14.29%	18.37%	16.94%
	8	906	59	67	110	78	6.51%	7.40%	12.14%	8.61%
FMSLD	3	24	14	11			58.33%	45.83%		
	4	54	28	32	32	36	51.85%	59.26%	59.26%	66.67%
	5	116	34	31	38	45	29.31%	26.72%	32.76%	38.79%
	6	242	31	37	43	48	12.81%	15.29%	17.77%	19.83%
	7	496	35	36	48	49	7.06%	7.26%	9.68%	9.88%
	8	1006	38	48	64	58	3.78%	4.77%	6.36%	5.77%
Phil	3	10	4	6	2	2	40%	60%	20%	20%
	4	17	5	5	5	8	29.41%	29.41%	29.41%	47.06%
	5	26	6	9	7	10	23.08%	34.62%	26.92%	38.46%
	6	37	10	12	10	12	27.03%	32.43%	27.03%	32.43%
	7	50	12	15	11	11	24%	30%	22%	22%
	8	65	11	15	10	13	16.92%	23.08%	15.38%	20%

Column 1: Name (as in [25])
Column 2: Size of the problem.
Columns 3: Number of siphons obtained by means of an algebraic algorithm.
Columns 4: Number of siphons obtained with the proposed method. P–Semiflows as initial population.
Column 5: Number of siphons obtained with the proposed method. Random initial population.
Columns 6-7: The same as columns 4-5 but doubling the size of the initial population.
Columns 8-11: The results of columns 4-7 as a percentage of Column 3.

6 Conclusions and further work

Some deadlock prevention control policies need the set of minimal siphons to be computed. It is well known that this is a very hard task because the number of such components can be very high. This paper has concentrated on the study of such question.

An adaptation of a set of logical formulas has been provided in order to try the genetic algorithm to search for a structural component of the net (siphons).

Even when it is not clear if the method has a good enough performance, it opens the door to further work. It is our intention to try to apply it to some deadlock prevention methods proposed in the past, where some special siphons need to be computed at each step.

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 [25]. 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. [31]. Finally, we feel that adding more information about the siphon properties to the method (siphonosity?) it would work better. Other improvements for the GA need to be tested.

Acknowledgments

The authors are indebted to the anonymous referees and the PC who have helped us to improve the quality and presentation of this paper. This work is supported in part by project ANYSELF (TIN2011-28627-C04-02) by the Spanish Mineco and TIN2011-27479-C04-01 by the Spanish Ministry of Science and Innovation; P08-TIC-03903 awarded by the Andalusian Regional Government, project 83, Campus CEI BioTIC, and by Group of Discrete Event Systems Engineering (GISED) awarded by Aragonese Government.

References

1. Murata, T.: Petri nets: properties, analysis and applications. Proceedings of the IEEE **77**(4) (April 1989) 541–580
2. López-Grao, J.P., Colom, J.M.: A Petri Net Perspective on the Resource Allocation Problem in Software Engineering. Transactions on Petri Nets and Other Models of Concurrency V. Springer-Verlag, Berlin, Heidelberg (2012) 181–200
3. Liao, H., Wang, Y., Stanley, J., Lafortune, S., Reveliotis, S., Kelly, T., Mahlke, S.: Eliminating Concurrency Bugs in Multithreaded Software: A New Approach Based on Discrete-Event Control. Control Systems Technology, IEEE Trans. **PP**(99)
4. Liao, H., Lafortune, S., Reveliotis, S., Wang, Y., Mahlke, S.: Optimal Liveness-Enforcing Control for a Class of Petri Nets Arising in Multithreaded Software. Automatic Control, IEEE Transactions on **58**(5) (May 2013) 1123–1138
5. Ezpeleta, J., Colom, J., Martínez, J.: A Petri net based deadlock prevention policy for flexible manufacturing systems. IEEE Trans. Rob. Aut. **11**(2) (1995) 173–184
6. Barkaoui, K., Pradat-Peyre, J.: On Liveness and Controlled Siphons in Petri Nets. In Billington, J., Reisig, W., eds.: Proceedings of the 1996 International Conference on Aplications and Theory of Petri Nets, Springer Verlag (June 1996)
7. Tricas, F., García-Vallés, F., Colom, J., Ezpeleta, J.: An Iterative Method for Deadlock Prevention in FMS. In Boel, R., Stremersch, G., eds.: Discrete Event Systems: Analysis and Control. Proc. of WODES, Ghent, Belgium (2000) 139–148
8. Huang, Y., Jeng, M.D., Xie, Z., Chung, S.: Deadlock prevention policy based on Petri nets and siphons. Int. Journal of Production Research **39**(2) (2001) 283–305

9. Iordache, M.V., Moody, J.O., Antsaklis, P.: Synthesis of Deadlock Prevention Supervisors Using Petri Nets. IEEE Trans. Rob. Automat. **18**(1) (2002) 59–68
10. Li, Z., Zhou, M.C.: Elementary Siphons of Petri Nets and Their Applications to Deadlock Prevention in Flexible Manufacturing Systems. IEEE Trans. on Systems, Man, and Cybernetics **34**(1) (January 2004) 38–51
11. Barkaoui, K., Chaoui, A., Zouari, B.: 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) 3750–3755
12. Tricas, F., Colom, J., Ezpeleta, J.: A solution to the problem of deadlocks in concurrent systems using Petri nets and integer linear programming. In Horton, G., Moller, D., Rude, U., eds.: Proc. of the 11th European Simulation Symposium, Erlangen, Germany, The society for Computer Simulation International (oct 1999)
13. Li, S., Li, Z., Hu, H., Al-Ahmari, A., An, A.: An extraction algorithm for a set of elementary siphons based on mixed-integer programming. Journal of Systems Science and Systems Engineering **21**(1) (March 2012) 106–125
14. Jeng, M., Peng, M., Huang, Y.: An algorithm for calculating minimal siphons and traps of Petri nets. Int. J. of Intelligent Control and Systems **3**(3) (1999) 263–275
15. Barkaoui, K., Lemaire, B.: An effective characterization of minimal deadlocks and traps in petri nets based on graph theory. In: Proceedings of the 10th International Conference on Application and Theory of Petri Nets, 1989. (1989) 1–21
16. Tricas, F.: Deadlock Analysis, Prevention and Avoidance in Sequential Resource Allocation Systems, Ph.D. Thesis. Dep. Inf. e Ing. de Sist. U. Zaragoza (May 2003)
17. Cordone, R., Ferrarini, L., Piroddi, L.: Characterization of minimal and basis siphons with predicate logic and binary programming. In: IEEE Int. Symposium on Computer Aided Control System Design, 2002, IEEE (2002) 193–198
18. Holland, J.H.: Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence. Oxford, England: U Michigan Press (1975)
19. Merelo, J.J.: A Perl Primer for Evolutionary Algorithm Practitioners. SIGEVOlution **4**(4) (March 2010) 12–19
20. Merelo-Guervós, J.J., Castillo, P.A., Alba, E.: `Algorithm::Evolutionary`, a flexible Perl module for evolutionary computation. Soft Computing **14**(10) (2010) 1091–1109 Accesible at http://sl.ugr.es/000K [sl.ugr.es].
21. Prashant Reddy, J., Kumanan, S., Krishnaiah Chetty, O.V.: Application of Petri Nets and a Genetic Algorithm to Multi-Mode Multi-Resource Constrained Project Scheduling. The Int. J. of Advanced Manufacturing Tech. **17**(4) (2001) 305–314
22. Lim, A.H.L., Lee, C.S., Raman, M.: Hybrid genetic algorithm and association rules for mining workflow best practices. Expert Systems with Applications **39**(12) (September 2012) 10544–10551
23. Xing, K., Han, L., Zhou, M., Wang, F.: Deadlock-Free Genetic Scheduling Algorithm for Automated Manufacturing Systems Based on Deadlock Control Policy. Systems, Man, and Cybernetics, Part B, IEEE Trans. **42**(3) (2012) 603–615
24. Han, L., Xing, K., Chen, X., Lei, H., Wang, F.: Deadlock-free genetic scheduling for flexible manufacturing systems using Petri nets and deadlock controllers. International Journal of Production Research **52**(5) (October 2013) 1557–1572
25. Tricas, F., Ezpeleta, J.: 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 **36**(3) (2006) 532–539

26. Boer, E.R., Murata, T.: Generating basis siphons and traps of Petri nets using the sign incidence matrix. IEEE Trans. on Circuits and Systems, I – Fundamental Theory and Applications **41**(4) (1994) 266–271
27. Silva, M.: Las Redes de Petri en la Automática y la Informática. Ed. AC, Madrid (1985)
28. Lautenbach, K.: Linear algebraic calculation of deadlocks and traps. In Voss, K., Genrich, H., Rozemberg, G., eds.: Concurrency and Nets. Springer Verlag (1987) 315–336
29. Tricas, F., Ezpeleta, J.: RessAllocation Petri net Model. In Kordon, F., et al., eds.: Model Checking Contest 2013, Milano, Italy (June 2013)
30. Cano, E.E., Rovetto, C.A., Colom, J.M.: An algorithm to compute the minimal siphons in S^4PR nets. Discrete Event Dynamic Systems **22**(4) (2012) 403–428
31. Tricas, F., García-Vallés, F., Colom, J., Ezpeleta, J.: 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) 272–278