# A RECURSIVE METHOD FOR MINIMAL SIPHON ENUMERATION IN PETRI NETS

**Arianna Benigno[†], Roberto Cordone[‡], Luca Ferrarini[†] and Luigi Piroddi[†]**

[†] *Dip. di Elettronica e Informazione, Politecnico di Milano, P.za L. Da Vinci, 32, 20133, Milano, Italy*
[‡] *Dip. di Tecnologie dell'Informazione, Università degli Studi di Milano, Polo Didattico e di Ricerca di Crema, Via Bramante, 65, 26013, Crema, Italy*

Abstract: The enumeration of minimal siphons in ordinary Petri nets is fundamental in the assessment of behavioral properties and a crucial step in the development of deadlock prevention algorithms. A novel recursive algorithm is proposed in the paper for this purpose, based on theoretical results that extend well known properties described in the literature. The algorithm uses already found solutions to progressively decompose the search problem into smaller sub-problems. This recursive decomposition approach is combined with the LTUR procedure, which efficiently computes a siphon, based on a logical clause description approach. A prototypical version of the search algorithm has been developed and an experiment has been carried out on a large set of random test instances to evaluate the efficiency of the method. 

Keywords: Petri-nets, Deadlock, Search methods, Recursive algorithms, Computational methods.

## 1. INTRODUCTION

In Petri net (PN) modeling of discrete production systems (FMS, batch processes, communication systems, etc.), the assessment of behavioral properties, such as reachability, reversibility and liveness is crucial for their correct characterization. Siphons (Murata, 1989) are structures intimately related to these properties (Jeng, and Peng, 1997), and their computation is important in PN analysis. In addition, deadlock prevention algorithms are typically based on siphon control policies (Moody, and Antsaklis, 1998).

Briefly, a siphon is a set of places whose input transitions are also output transitions: this implies that, once a siphon becomes empty of tokens, it will permanently remain empty. Siphons can be characterized with many different approaches, e.g. inequalities (Murata, 1989), logic equations (Kinuyama, and Murata, 1986; Minoux, and Barkaoui, 1990), algebraic approaches (Lautenbach, 1987), linear equations with slack variables (Cordone, *et al.*, 2002; Ezpeleta, and Couvreur, 1991), structural properties (Barkaoui, and Minoux, 1992; Boer, and Murata, 1994; Jeng, and Peng, 1996). Besides, the problem of finding *all* minimal siphons requires an efficient search algorithm to deal with the exponential growth of the search space with the size of the PN. For example, in (Jeng, and Peng, 1997) a constructive algorithm is proposed based on a depth-first search of different combinations of places. This algorithm builds a branching tree in which the single paths from the root to each leaf correspond to the subsequent addition of places to a sub-set which is candidate to be a siphon.

In the present work, an innovative algorithm based on a completely different branching approach is proposed. The algorithm exploits the following facts:

1. A generic siphon of a PN can be rapidly obtained by means of the LTUR procedure (Minoux, 1988), which exploits the Horn logical clauses (Horn, 1956) in the problem formulation.
2. A minimal siphon contained in a non minimal siphon can be found by applying the LTUR pro-

cedure to a suitably generated set of smaller sub-problems. These are obtained from the original search problem, by means of a decomposition technique, which imposes simple constraints on the solutions, such as including or removing some specific places. These constraints can be described in terms of Horn clauses as well.

3. When a minimal siphon is obtained, the original search problem can be further decomposed in sub-problems, that globally exclude the given siphon, as well as other siphons strictly containing it, from the search space.

All the minimal siphons of the net can be determined by repeatedly applying the decomposition procedure.

Notice that place constraints have also been used in other related papers (Yamauchi, *et al.*, 1998; Yamauchi, and Watanabe, 1999), but with a different perspective, i.e. the search for siphons which are minimal *and also* contain a given set of places. Here, a generic sub-problem aims at the extraction of siphons which are minimal with respect to the set of all siphons containing a given set of places. In the former case, place constraints are a part of the problem, whereas in the proposed approach they are generated on purpose, to reduce the search effort.

## 2. PETRI NETS AND SIPHONS

### *2.1 Basic notions on Petri nets*

A PN is a bipartite graph, where nodes are classified either as *places* or *transitions*, and directed arcs connect nodes of different type. More formally, a PN structure is defined as a triplet $(P, T, F)$, where $P$ is a set of $n$ places and $T$ is a set of $m$ transitions. $F \subseteq P\times T \cup T\times P$ represents the *flux relation*, i.e. the relation between places and transitions. Alternatively, the flux relation can be given in the form of matrices, namely the input ($I$), output ($O$) and incidence ($C = O–I$) matrices. The generic element $i_{kj}$ of the $I_{m\times n}$ matrix represents the *weight* of the arc from place $p_k$ to transition $t_j$ (0 or 1 for ordinary PNs). Correspondingly, the generic element $o_{kj}$ of matrix $O_{m\times n}$ represents the weight of the arc from transition $t_j$ to place $p_k$. A *marking* $M$: $P \rightarrow N = \{0, 1, 2, ...\}$ defines the distribution of *tokens* in places. $M_0$: $P \rightarrow N$ is the initial marking. A transition $t_j \in T$ is said to be *enabled* in a marking $M$ if $M \geq I_j$, where $I_j$ is the $j$-th column of input matrix $I$. An enabled transition may *fire*, yielding the marking $M^* = M + O_j – I_j = M + C_j$, where $O_j$, $I_j$ and $C_j$ are the $j$-th columns of the $O$, $I$ and $C$ matrices, respectively.

### *2.2 Siphons*

Let $S$ be a set of places in a PN. Then, its pre-set $\bullet S = \{t_j \in T \mid o_{kj} \neq 0$, with $p_k \in S\}$, is the set of input transitions for the places in $S$, while its post-set, $S\bullet = \{t_j \in T \mid i_{kj} \neq 0$, with $p_k \in S\}$, is the set of output transitions for the places in $S$. A *siphon* is a set $S$ of places such that $\bullet S \subseteq S\bullet$.

The number of siphons of a PN grows exponentially with size (Boer, and Murata, 1994). However, only the *minimal* siphons or the *basis* siphons are generally of interest. A siphon is said to be a *basis* siphon if it cannot be obtained by the union of other siphons. All siphons in a net can be generated operating the union of some suitable basis siphons. A siphon is said to be a *minimal* siphon if it does not contain any other siphon. A minimal siphon is also a basis siphon, but not all basis siphons are minimal. We also introduce the notion of *subset-minimal siphon*, that is a siphon including all places in a sub-set $P_{in}$ of the set of places $P$, and not strictly containing any other such siphon. In particular, a *place-minimal* siphon with respect to a place $p$ is a subset-minimal siphon with respect to a sub-set $P_{in} = \{p\}$. A siphon is a basis siphon if and only if it is place-minimal with respect to some place (Boer, and Murata, 1994).

## 3. SEARCHING FOR MINIMAL SIPHONS

The proposed algorithm basically operates as follows: when a minimal siphon is found, several multiple sub-problems are generated which have all the other minimal siphons as solutions. The procedure is recursively applied to each sub-problem, generating new minimal siphons. These sub-problems are obtained by introducing specific place constraints on the original net: a place may be required to belong to the computed siphons or, conversely, not to belong to them. Both types of place constraints determine a simplification of the search problem, as showed in the following definitions and theoretical results.

Let $P_{out} \subset P$ and limit the search to the siphons which do not contain any place in $P_{out}$, *i.e.* the siphons contained in $\tilde{P} = P–P_{out}$. In such a constrained siphon search problem, all the places not in $\tilde{P}$ (and the arcs connected with them) can be discarded from the net, thus greatly simplifying the problem.

*Definition 1* — Let $G = (P, T, F)$ be a PN and $\tilde{P} \subset P$. The reduction function *red* is defined as follows: $\tilde{G} = red(G, \tilde{P})$, where the reduced net $\tilde{G} = (\tilde{P}, \tilde{T}, \tilde{F})$ is defined by

*i.* $\tilde{T} = \{t \in T \mid (\bullet t \cup t\bullet) \cap \tilde{P} \neq \varnothing\}$,

*ii.* $\tilde{F}(p,t) = F(p,t)$, $\tilde{F}(t,p) = F(t,p)$, $\forall p \in \tilde{P}$, $\forall t \in \tilde{T}$. ❑

*Lemma 1* — Let $G = (P, T, F)$ be a PN and $\tilde{P} \subset P$. The set of siphons of $G$ contained in $\tilde{P}$ coincides with the set of siphons of the reduced net $\tilde{G} = red(G,\tilde{P})$. ❑

*Proof.* Consider a set of places $S \subseteq \tilde{P}$ in $G$ and let $\tilde{S}$ be the corresponding set of places in $\tilde{G}$. Now, $\bullet S = \bullet\tilde{S}$ and $S\bullet = \tilde{S}\bullet$, since all the transitions connected to places in $\tilde{P}$ are preserved by the *red* operator in net

$\tilde{G}$, as well as the corresponding arcs. ❑

On the other hand, a siphon search problem may be stated, in which only the siphons containing specific places ($P_{in}$) of the net are sought. This problem may be further simplified, by restricting the search space as explained in the following lemma.

*Lemma 2* — Let $G = (P, T, F)$ be a PN and $P_{in} \subset P$, $P_{in} \neq \emptyset$. Assume that there exists a place $p \in P - P_{in}$ and a transition $t \in T$ such that $\bullet P_{in} - P_{in}\bullet \supseteq \{t\}$ and $\bullet t = \{p\}$. Then, all the siphons of $G$ containing $P_{in}$ contain $p$ as well. ❑

*Proof.* The hypothesis implies that $t \in \bullet P_{in}$ and $t \notin P_{in}\bullet$. Therefore, $\bullet P_{in} \not\subseteq P_{in}\bullet$ and any siphon containing $P_{in}$ must contain at least another place with $t$ in its post-set. This property holds only for place $p$ and the thesis follows. ❑

Given a siphon $S$ of a PN $G$, all the siphons of $G$ not containing $S$ can be found by exploration of suitable sub-nets of $G$, each of which is obtained by removing one of the places of $S$. In each sub-net only siphons containing a specific sub-set of $S$ must be sought. The decomposition introduced in the following Lemma increases the number of nets to be explored, but the individual net size is decreased, turning a complicated problem into several simpler sub-problems.

*Lemma 3* — Let $G = (P, T, F)$ be a PN and $\tilde{P} = \{p_1, p_2, ..., p_n\} \subset P$. The set of siphons of $G$ not containing $\tilde{P}$ is equal to $\Sigma_1 \cup \ldots \cup \Sigma_n$, where $\Sigma_i$ is the set of siphons of the reduced net $\tilde{G}_i = red(G, P - \{p_i\})$, $i = 1, ..., n$, containing $\{p_1, p_2, ..., p_{i-1}\}$. ❑

*Proof.* The set of siphons of $G$ not containing $\tilde{P}$ can be divided as $\Sigma_1 \cup \ldots \cup \Sigma_n$, where $\Sigma_i$ is the set of siphons of $G$, containing $\{p_1, p_2, ..., p_{i-1}\}$ and not $p_i$. By Lemma 1, the generic $\Sigma_i$ can be computed as the set of siphons of the reduced net $\tilde{G}_i = red(G, P - \{p_i\})$, containing $\{p_1, p_2, ..., p_{i-1}\}$. ❑

Notice that, when looking for all subset-minimal siphons with respect to a sub-set $P_{in}$, the subset $\Sigma_i$ for which $p_i$ belongs to $P_{in}$ is trivially empty, and can be neglected.

## 4. A MINIMAL SIPHON ENUMERATION ALGORITHM

Consider the problem of finding all minimal siphons of a PN $G = (P, T, F)$. The general structure of an algorithm which computes the collection $\Sigma$ of all such siphons can be summarized as follows:

Step 0) $\tilde{G} = G$, $\tilde{P}_{in} = \emptyset$, $\Sigma = \emptyset$.
Step 1) Search $G$ for a generic siphon $S$, such that $S \supseteq P_{in}$. If none exists, stop.
Step 2) Find a siphon $S'$, such that $P_{in} \subseteq S' \subseteq S$ and $S'$ is minimal with respect to all siphons containing $P_{in}$. Add $S'$ to $\Sigma$.
Step 3) All other solutions must not include $S$. Apply Lemma 3 to find them, that is define a suitable number of sub-nets $\tilde{G}_i$ of $\tilde{G}$ and the corresponding place constraints $\tilde{P}_{in,i}$. For each couple $(\tilde{G}_i, \tilde{P}_{in,i})$ let $G = \tilde{G}_i$, $P_{in} = \tilde{P}_{in,i}$ and goto Step 1.

This structure results into a recursive algorithm. A call to the procedure either returns a siphon which is minimal with respect to the current place constraints $P_{in}$ or terminates at Step 1. When the algorithm ends, the collection $\Sigma$ contains all the requested minimal siphons. The solution set may also include some non-minimal siphons, since subset-minimal siphons are not necessarily minimal. Actually, the decomposition technique ensures that all siphons strictly containing any of the already obtained solutions will be excluded from the search space, but it may allow for even smaller solutions to be found in the search process. Two simple strategies can be applied to rule out non minimal siphons: either the *FindMinimal-Siphon* function (see below) is used on each obtained solution (setting $P_{in} = \emptyset$), so that the existence of smaller siphons can be ascertained, or all the solutions are collected and screened *a posteriori*.

### *4.1 Searching for a generic siphon*

The implementation of Step 1 exploits the relation between siphons and predicative logic, which is well known in the literature (Murata, 1989). All the siphons of a PN $G = (P, T, F)$ must verify a set of logical conditions obtained with the following rationale. Suppose that $S \subseteq P$ is a siphon and $p_i \in S$. Then, $S \cap \bullet t_{ik} \neq \emptyset$, $\forall t_{ik} \in T \cap \bullet p_i$. This amounts to the following set of implications:

IF $p_i \in S$ THEN ($W_1$ AND ... AND $W_{M_i}$), $i = 1, .., n$,

where $W_k = ((p_{k1} \in S)$ OR ... OR $(p_{kN_k} \in S))$, $\bullet t_{ik} = \{p_{k1}, p_{k2}, ..., p_{kN_k}\}$, $k = 1, ..., M_i$, $\bullet p_i = \{t_{i1}, t_{i2}, ..., t_{iM_i}\}$. By associating a Boolean variable $x_i$ to each place $p_i$, the preceding set of relations can be transformed in a set of $\sum_{i=1}^{n} M_i$ logical clauses such that siphons correspond one-by-one to the truth assignments $x_i$ which satisfy them:

$$\bar{x}_i \vee ((x_{11}) \vee (x_{12}) \vee \ldots \vee (x_{1N_1})) = 1,$$
$$\bar{x}_i \vee ((x_{21}) \vee (x_{22}) \vee \ldots \vee (x_{2N_2})) = 1, \ldots,$$
$$\bar{x}_i \vee ((x_{M_i1}) \vee (x_{M_i2}) \vee \ldots \vee (x_{M_iN_{Mi}})) = 1, \; i = 1, ..., n.$$

Notice that the number of logical clauses actually coincides with the number of arcs connecting transitions to places, *i.e.* the number of non-zero elements in the output matrix $O$. On the other hand, the number $N_k$ of affirmed literals in each clause is equal to the number of arcs connecting places to the transition $t_{ik}$ originating the clause, *i.e.* the number of non-zero

elements in the corresponding column of the input matrix $I$. Observe that, if both $O_{ij} = 1$ and $I_{ij} = 1$ for some $i$ and $j$ (self-loop), the resulting clause for arc $(i, j)$ has the same variable both affirmed and negated. This makes the clause trivially satisfied. Therefore, such clauses are immediately discarded.

At most one variable in each clause is negated: such clauses are denoted *Horn clauses* and are well known in the literature (Horn, 1956): efficient algorithms exist for finding solutions to sets of Horn clauses in linear time (Minoux, 1988). In this view, it is important to notice that the introduction of reduced sub-nets and place constraints modifies the set of clauses defined above, but preserves their Horn structure:

- when considering a sub-net some place variables are set to 0; these can be eliminated from all the clauses in which they appear as affirmed literals ($x \vee 0 = x$); in addition, all the clauses where the variable set to 0 is the negated one are trivially verified and can be eliminated ($1 \vee x = 1$);
- if a specific place $p_i$ belongs to $P_{in}$, the corresponding variable $x_i$ must be set to 1; all the clauses containing $x_i$ are trivially verified and can be eliminated, whereas literal $\overline{x}_i$ can be removed from all clauses in which it appears.

The solution of the resulting (and possibly reduced) set of Horn clauses is simple. If any of the clauses has a single literal, its value is fixed and is simplified from the other clauses. When all unitary clauses have been eliminated and no further reduction is possible, a trivial solution is obtained by setting all the remaining variables to 1. A solution always exists unless the simplification process results in inconsistencies of the type 0 = 1, in which case no siphon with the requested characteristics exists. An algorithm to solve the set of Horn clauses can be implemented by means of the following data structures:

- $X$ is the set of all variables ($X = \{x_i: p_i \in P\}$);
- $C = \{c_1, ..., c_{NC}\}$ is the set of all clauses;
- each clause $c_j \in C$ is a set of literals, i.e. affirmed ($x_i$) or negated variables ($\overline{x}_i$);
- $F$ is the set of all literals fixed to 1 ($F = \{x_i: p_i \in P_{in}\} \cup \{\overline{x}_i: p_i \in P_{out}\}$).

Obviously, $F$ may not contain complementary literals. The procedure returns a siphon $S$ or an empty set if none exists. In the first case, the set $F$ may also be enlarged. The function *FindSiphon*, here listed in a pseudo-code version, implements such an algorithm. The sub-procedure *fix* is used to constrain the value of a specific literal.

FUNCTION $(S,F)$ = *FindSiphon*$(C,F)$
FOR ALL $l$ IN $F$, $C$ = *fix*$(C,l)$, END
IF $\exists\ c_j : |c_j| = 0$, $S = \varnothing$, RETURN, END
WHILE $\exists\ c_j : |c_j| = 1$
  $l$ = *extract*$(c_j)$, $C$ = *fix*$(C,l)$
  IF $\exists\ c_j : |c_j| = 0$, $S = \varnothing$, RETURN, END
  $F = F \cup \{l\}$
END

$S = P$
FOR ALL $x_i$ IN $X$, IF $\overline{x}_i \in F$, $S = S - \{p_i\}$, END, END

FUNCTION $C$ = *fix*$(C,l)$
FOR ALL $c_j$ IN $C$
  IF $l \in c_j$, $C = C - \{c_j\}$, END
  IF $\overline{l} \in c_j$, $c_j = c_j - \{\overline{l}\}$, END
END

### *4.2 Searching for a minimal siphon*

A minimal siphon can be easily drawn from a generic one $S$. First remove from the net all places not in $S$: $S$ is still a siphon for the remaining net, by Lemma 1. Then, remove a place belonging to $S$, and verify whether the net still includes any siphon. If it does, this has at least one place less than $S$, but it could even have several places less. Then, a second place is removed, and so on, as far as possible. If a place cannot be removed, the procedure applies to the following ones. The current siphon is minimal when all remaining places cannot be removed. Mandatory places are not tested for removal. The following pseudo-code describes the procedure:

FUNCTION $S'$ = *FindMinimalSiphon*$(C,F,S)$
$F = F \cup \{\overline{x}_i \mid x_i \in X, p_i \notin S\}$
WHILE $\{x_i \mid p_i \in S, x_i \notin F\} \neq \varnothing$
  $x_i$ = *extract*$(\{x_i \mid p_i \in S, x_i \notin F\})$, $\tilde{F} = F \cup \{\overline{x}_i\}$
  $(\tilde{S},\tilde{F})$ = *FindSiphon*$(C,\tilde{F})$
  IF $\tilde{S} = \varnothing$, $F = F \cup \{x_i\}$ ELSE $F = \tilde{F}$, $S = \tilde{S}$, END
END
$S' = S$

### *4.3 Sub-problem generation*

Each time a minimal siphon $S$ is found, it is excluded from further search together with all siphons containing it. This is obtained, thanks to Lemma 3, by building a branching tree of sub-problems with suitable place constraints. In the first sub-problem, the first free variable in $S$ is fixed to 0. In the second one, the first free variable is set to 1 and the second to 0, and so on. In other words, as many sub-problems are generated as the free places in $S$, always fixing one place to 0 and the previous ones to 1. The algorithm explores the branching tree with a *depth-first* strategy. Procedures *FindSiphon* and *FindMinimalSiphon* analyse the current sub-problem to verify whether it contains a siphon and reduce it to a minimal siphon. Whenever the current sub-problem contains no siphons, the algorithm is applied to the following sub-problem on the same tree level. If there are no more sub-problems on the same level, the algorithm *back-tracks* to the upper level. The number of levels cannot exceed the number of places in the net, since at each level at least one variable is fixed. The search stops when all of the sub-problems have been solved, that is all minimal siphons have been determined (Lemma 3).

FUNCTION $\Sigma$ = *FindAllMinimalSiphons*($C$,$F$)
$\Sigma = \varnothing$
($S$,$F$) = *FindSiphon*($C$,$F$)
IF $S \neq \varnothing$
  $S' =$ *FindMinimalSiphon*($C$,$F$,$S$), $\Sigma = \Sigma \cup \{S'\}$
  $H = \{x_i \mid p_i \in S', x_i \notin F\}$
  WHILE $H \neq \varnothing$
    $x_i =$ *extract*($H$), $H = H - \{x_i\}$, $F = F \cup \{\bar{x}_i\}$
    $\Sigma = \Sigma \cup$ *FindAllMinimalSiphons*($C$,$F$)
    $F = (F - \{\bar{x}_i\}) \cup \{x_i\}$
  END
END

Initially, function *FindAllMinimalSiphons* is called with no place constraints ($F = \varnothing$).

### *4.4 An illustrative example*

To clarify the behavior of the proposed algorithm, consider the simple PN in Fig. 1, which has the two minimal siphons $S_1 = \{p_1, p_2, p_3\}$ and $S_2 = \{p_1, p_3, p_4\}$. The set of logical clauses $C = \{c_1, c_2, c_3, c_4, c_5\}$ is readily derived as follows: $c_1 = \{\bar{x}_1, x_2, x_4\}$, $c_2 = \{\bar{x}_1, x_3, x_4\}$, $c_3 = \{\bar{x}_2, x_1\}$, $c_4 = \{\bar{x}_3, x_1\}$, $c_5 = \{\bar{x}_4, x_3\}$.

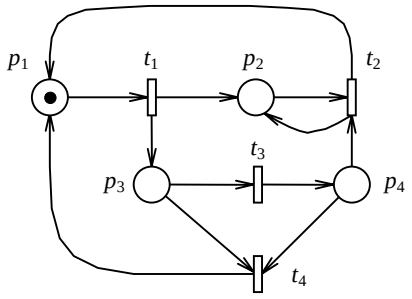


Fig. 1. The PN Example

Initially, $F = \varnothing$ and there are no unitary clauses. Therefore, function *FindSiphon* trivially finds a siphon $S = \{p_1, p_2, p_3, p_4\}$. Function *FindMinimalSiphon* computes $\{x_i \mid p_i \in S, x_i \notin F\} = \{x_1, x_2, x_3, x_4\}$ and operates the following steps:

1) $F = \{\bar{x}_1\}$; function *FindSiphon* simplifies the literals $\bar{x}_1$, $\bar{x}_2$, $\bar{x}_3$, $\bar{x}_4$ in this order from $C$; at the end of the process $F = \{\bar{x}_1, \bar{x}_2, \bar{x}_3, \bar{x}_4\}$ and no siphon is obtained ($S = \varnothing$); finally, $F = \{x_1\}$.
2) $F = \{x_1, \bar{x}_2\}$; literals $x_1$, $\bar{x}_2$, $x_4$, $x_3$ are simplified, resulting in $F = \{x_1, \bar{x}_2, x_3, x_4\}$; a siphon $S = \{p_1, p_3, p_4\}$ is found and $F$ and $S$ are reset to $F$ and $S$, respectively.
3) Since $\{x_i \mid p_i \in S, x_i \notin F\} = \varnothing$ the computed siphon is minimal and the execution of function *FindMinimalSiphon* ends.

Now $S' = \{p_1, p_3, p_4\}$ is added to the solution set $\Sigma$ and 3 sub-problems are generated. $H = \{x_1, x_3, x_4\}$ and function *FindAllMinimalSiphons* is called with $F = \{\bar{x}_1\}$, $F = \{x_1, \bar{x}_3\}$ and $F = \{x_1, x_3, \bar{x}_4\}$, respectively. A siphon ($S = \{p_1, p_2, p_3\}$) is found only with the third set of constraints. Since for this case function *FindSiphon* yields $F = \{x_1, x_2, x_3, \bar{x}_4\}$, the siphon cannot be further reduced and it is added to $\Sigma$. No further branching is performed and the algorithm ends. The algorithm evolution is summarized in Figure 2.

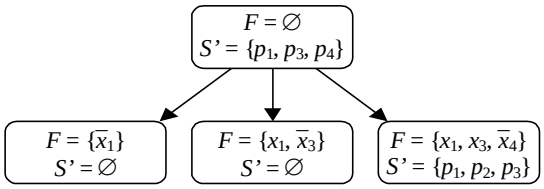


Fig. 2. Evolution of the *FindAllMinimalSiphons* function for the example

## 4. EXPERIMENTAL RESULTS

The algorithm described in Section 4 has been coded in C and tested on a 2.4 GHz Pentium computer. The test set consists of 270 randomly generated instances of PNs, with variable size and topology. In detail, six size classes have been considered, with $n = m =$ 5, 10, 15, 20, 25, 30, respectively. Each size class includes 9 sub-classes, whose connectivity has been determined by setting the density of input ($d_i$) and output ($d_o$) arcs to 0.25, 0.50, 0.75 in all possible combinations. Each sub-class consists of five instances.

The CPU time required to enumerate all the minimal siphons, averaged over the PN instances, is given in Table 1, together with the average number of minimal siphons for each class size. The results of two other algorithms, i.e. a MIP-based approach (Cordone, *et al.*, 2002) and an exhaustive method, are also reported there for comparison purposes. The former is implemented in C and employs a commercial MIP solver, whereas the latter is realized with a MATLAB routine. The results are not reported when the average computation time exceeds 30 minutes.

Table 1 Total CPU time required for the minimal siphon enumeration

| PN size | number of min. siphons | Total CPU time (*s*) proposed algorithm | MIP approach | constructive algorithm |
|---|---|---|---|---|
| 5 | 2.53 | 0.01 | 0.02 | 0.01 |
| 10 | 10.98 | 0.01 | 0.14 | 0.89 |
| 15 | 60.04 | 0.02 | 2.73 | 300.85 |
| 20 | 302.44 | 0.16 | 151.73 | - |
| 25 | 1591.33 | 4.91 | - | - |
| 30 | 8544.67 | 266.79 | - | - |

Notice that the increase in computational time with respect to PN size depends on several factors:

1. the number of minimal siphons increases exponentially,
2. the total number of nodes generated and examined increases even more rapidly,
3. the computational time to examine a single node

increases polinomially with size.

Table 2 reports the average number of nodes generated by the algorithm, classified according to the results of the siphon enumeration procedure:

1. *minimal siphon nodes* – nodes corresponding to actual minimal siphons,
2. *redundant nodes* – nodes which result in locally minimal siphons (i.e. siphons which are minimal with respect to the place constraints, but not with respect to the original PN),
3. *empty nodes* – nodes corresponding to siphon computation problems with no admissible result.

Processing of empty nodes is fast, since the function *FindMinimalSiphon* is not executed. On the other hand, the reduction of redundant nodes could significantly improve the algorithm speed.

Table 2 Nodes examined by the branching procedure

| PN size | min. siphon nodes | redundant nodes | empty nodes | total nodes |
|---|---|---|---|---|
| 5 | 2.53 | 0.02 | 1.27 | 3.82 |
| 10 | 10.98 | 1.53 | 9.13 | 21.64 |
| 15 | 60.04 | 20.20 | 59.09 | 139.33 |
| 20 | 302.44 | 186.73 | 413.87 | 903.04 |
| 25 | 1591.33 | 1679.51 | 2992.09 | 6262.93 |
| 30 | 8544.67 | 15373.13 | 25052.29 | 48970.09 |

## 5. CONCLUSIONS

In the paper, the problem of the enumeration of all minimal siphons in a ordinary PN has been addressed. Some theoretical results have been developed which lay out the basis for a decomposition technique of the search problem. The sub-problems generated are progressively more constrained, and globally reduce the solution space to the set of minimal siphons of the PN. This results in an efficient recursive search algorithm which has been shown to perform well compared to other known approaches.

Future work will include the development of a full software package for the analysis of large-size PNs. A computational comparison with other siphon enumeration algorithms among the many different ones available in the literature will be also considered.

## REFERENCES


Barkaoui, K. and M. Minoux (1992). A Polynomial-Time Graph Algorithm to Decide Liveness of Some Basic Classes of Bounded Petri Nets. *Application and Theory of Petri Nets*, pp. 62-75.

Boer, E.R. and T. Murata (1994). 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), pp. 266-271.

Cordone, R., L. Ferrarini and L. Piroddi (2002). Characterization of Minimal and Basis Siphons with Predicate Logic and Binary Programming. *IEEE Int. Conf. on Computer Aided Control System Design*, Glasgow, Scotland, pp. 193-198.

Ezpeleta, J. and J.M. Couvreur (1991). A new technique for finding a generating family of siphons, traps and st-components. Application to colored Petri nets. *Proc. of 12th Conf. on Applications and Theory of Petri Nets*, pp. 126-147.

Jeng, M.D. and M.Y. Peng (1996). Generating minimal siphons and traps for Petri nets. *Proc. of IEEE Int. Conf. on Systems, Man and Cybernetics*, Beijing, China, pp. 2996-2999.

Jeng, M.D. and M.Y. Peng (1997). Petri nets liveness analysis by minimal siphons. *Proc. of 6th Int. Conf. on Emerging Technologies and Factory Automation*, UCLA, Los Angeles, pp. 315-320.

Horn, A. (1956). On Sentences which are True of Direct Unions of Algebras. *Journal of Symbolic Logic*, **16**, pp. 14-21.

Kinuyama, M. and T. Murata (1986). Generating siphons and traps by Petri net representation of logic equations. *Proc. of 2th IECE Conf. on Net Theory*, pp. 93-100.

Lautenbach, K. (1987). Linear algebraic calculation of deadlocks and traps. In: *Concurrency and Nets – Advances in Petri Nets* (Voss, Genrich and Rozenberg., Eds.), pp. 315-336, Springer-Verlag, New York.

Minoux, M. (1988). LTUR: A Simplified Linear-Time Unit Resolution Algorithm for Horn Formulae and Computer Implementation. *Information Processing Letters*, **29**, pp. 1-12.

Minoux, M. and K. Barkaoui (1990). Deadlocks and traps in Petri nets as Horn-satisfiability solutions and some related polynomially solvable problems. *Discrete Applied Mathematics*, **29**, pp. 195-210.

Moody, J.O. and P.J. Antsaklis (1998). *Supervisory Control of Discrete Event Systems using Petri Nets*. Kluwer Academic Publishers, Norwell, MA.

Murata, T. (1989). Petri nets: properties, analysis and application", *Proc. of the IEEE*, **77** (4), pp. 541-580.

Yamauchi, M., S. Tanimoto and T. Watanabe (1998). Extracting siphons containing a specified set of places in a Petri net. *IEEE Int. Conf. on Systems, Man and Cybernetics*, pp. 142-147.

Yamauchi, M. and T. Watanabe (1999). Algorithms for extracting minimal siphons containing specified places in a general Petri net. *IEICE Trans. Fundamentals*, **E82-A** (11), pp. 2566-2575.