Complete enumeration of minimal siphons in ordinary Petri nets based on problem partitioning

ShouGuang Wang*, Dan You, Carla Seatzu, and Alessandro Giua

Abstract

A siphon is a structural object in Petri nets that is important both from a theoretical and a practical point of view. Particularly, the performance of siphon-based deadlock control policies largely depends on siphon enumeration. This work studies complete minimal-siphon enumeration in ordinary Petri nets. A recent approach, called *global partitioning minimal-siphon enumeration* (GPMSE) has been recently proposed by Cordone *et al.* [1] and provides good performance compared with other methods. In this paper we show that further improvements are possible and we propose a novel approach, called *improved GPMSE*, which requires lower computational complexity and memory consumption than the original method, especially for nets with large size. Experimental results are presented to validate the above claim.

Accepted as:
S.G. Wang, D. You, C. Seatzu, A. Giua, "Complete Enumeration of Minimal Siphons in General Petri Nets Based on Problem Partitioning," 54nd IEEE Conf. on Decision and Control (Osaka, Japan), Dec. 15-18, 2015.

*Research supported by National Natural Science Foundation of China under Grant 61472361, Zhejiang Provincial Natural Science Foundation for Distinguished Young Scholars (No. LR14F020001), Zhejiang Sci. & Tech. Project under Grant 2013C31111, Zhejiang NNST Key Laboratory under Grant 2013E10012, and Zhejiang Gongshang University Innovation Project under Grant CX201411010.

S. G. Wang is with School of Information & Electronic Engineering, Zhejiang Gongshang University, Hangzhou, 310018, China (corresponding author: 0571-28877734; e-mail: wsg5000@hotmail.com).

D. You is with School of Information & Electronic Engineering, Zhejiang Gongshang University, Hangzhou, 310018, China (e-mail: youdan000@hotmail.com).

C. Seatzu is with the Dep. of Electrical and Electronic Engineering, University of Cagliari, Cagliari, 09124, Italy (e-mail: seatzu@diee.unica.it).

A. Giua is with Aix Marseille Université, CNRS, ENSAM, Université de Toulon, LSIS UMR 7296, Marseille 13397, France, and also with the Dep. of Electrical and Electronic Engineering, University of Cagliari, Cagliari, 09124, Italy (e-mail: alessandro.giua@lsis.org; giua@diee.unica.it).

1 Introduction

A siphon is a set of places in a Petri net (PN) with the following property: once a siphon loses all its tokens, it remains permanently unmarked and some transitions thereby are disabled forever. As a result, deadlock arises. For this reason, most deadlock control policies are based on siphon control [5]. Unfortunately, the number of siphons in a PN grows exponentially with the size of the net, and this may hinder the use of siphon-based deadlock control policies. Hence, the efficient computation of siphons is of major importance to guarantee good performance of siphon-based deadlock control policies, and much work has been done to decrease the computational complexity of siphon enumeration [1]-[4], [6]-[14].

Siphon enumeration methods can be divided into two broad classes. The first class includes approaches that apply to PNs with specific structures, such as methods based on resource circuits [4], [11], methods based on loop resource subsets [14], methods based on the pruning graph [2], parallel algorithms [10], and genetic-algorithm-based methods [8], [9]. The second class includes approaches applicable to ordinary PNs, such as the INA-based method [7], linear integer programming methods [3], [6], and methods based on problem decomposition [1], [13]. This work proposes an approach that applies to ordinary PNs, and thus belongs to the second class.

Problem decomposition is essentially based on the idea that a problem can be decomposed into several simper sub-problems and once the solution to each sub-problem is obtained, the solution to the original problem immediately follows. There are a variety of methods based on problem decomposition for minimal siphon enumeration that greatly differ in computational complexity and memory consumption. *global partitioning minimal-siphon enumeration* (GPMSE) and *local partitioning minimal-siphon enumeration* (LPMSE), proposed by Cordone *et al.* [1], have a very low computational complexity compared with other methods for ordinary PNs and are thereby extensively used. In more detail, the algorithms in [1] exploit the information conveyed by already-found solutions to progressively narrow the search process: when a minimal siphon is obtained, the solution space can be efficiently partitioned, to exclude the already-found siphon and all siphons containing it. The current search problem is transformed into a list of suitable simpler sub-problems, by adding specific place constraints, either in the form of set of places forced to belong to the solutions of the problem or in the form of set of places required not to belong to the same solutions. Two versions of the search technique are proposed in [1], which mainly differ in the application of the partitioning procedure.

In this paper we focus on the GPMSE approach that uses a search technique that partitions all the problems in the unsolved problem list and finds exactly the complete set of minimal siphons. We provide a way to further decrease its computational complexity and reduce memory requirements. We call the resulting approach *improved GPMSE*. The basic idea behind improved GPMSE can be summarized as follows. The set of places forced to belong to minimal siphons is expanded and two conditions are added to restrict the further decomposition of the problem. This allows to reduce the number of sub-problems to be considered. Moreover, in improved GPMSE we use depth-first search instead of width-first search as in [1], since experimental results show that this leads to lower memory requirements.

To evaluate the effectiveness of the proposed approach we developed a tool [12], written in C++, that implements three different methods: the GPMSE approach as proposed in [1], the GPMSE approach where depth-first search is used instead of width-first search, and the improved GPMSE approach. A series of simulations on nets of different size and Pre and Post matrices randomly generated have been carried out. Results are discussed in Section V.

2 Basic Background on Petri Nets and Siphons

An *ordinary Petri net* is a 3-tuple $N=(P, T, F)$ where P and T are finite, nonempty, and disjoint sets. P is the set of *places*, and T is the set of *transitions*. The set $F \subseteq (P \times T) \cup (T \times P)$ is the *flow relation*. Given a net $N=(P, T, F)$ and a node $x \in P \cup T$, ${}^\bullet x = \{y \in P \cup T \mid (y, x) \in F\}$ is the *preset* of x, while $x^\bullet = \{y \in P \cup T \mid (x, y) \in F\}$ is the *post-set* of x. $\forall X \subseteq P \cup T$, ${}^\bullet X = \bigcup_{x \in X} {}^\bullet x$, and $X^\bullet = \bigcup_{x \in X} x^\bullet$. $\forall x \in P \cup T$, ${}^{\bullet\bullet} x = {}^\bullet({}^\bullet x)$, and $x^{\bullet\bullet} = (x^\bullet)^\bullet$. Note that all the paper deals with ordinary Petri nets. Hence, we call them Petri nets for simplification.

Let $N=(P, T, F)$ be a Petri net with $P_X \subseteq P$ and $T_X \subseteq T$. $N_X=(P_X, T_X, F_X)$ is called a subnet generated by P_X and T_X if $F_X = F \cap [(P_X \times T_X) \cup (T_X \times P_X)]$.

A transition (resp., place) without any input place (resp., transition) is called a *source transition* (resp., place), and one without any output place (resp., transition) is called a *sink transition* (resp., place).

A nonempty set $S \subseteq P$ is a *siphon* if ${}^\bullet S \subseteq S^\bullet$. A siphon is called *minimal* if it does not contain any other siphon.

A siphon is called P_{in}-*minimal* if it includes *all* places in a set $P_{in} \subseteq P$ and does not strictly contain any other siphon including all places in P_{in}. Particularly, a P_{in}-minimal siphon is a minimal siphon if $P_{in}=\varnothing$. Otherwise, a P_{in}-minimal siphon is not necessarily a minimal siphon.

Definition 1: Let $N=(P, T, F)$ be a Petri net and P_{in} be a set of places. $G=(N, P_{in})$ is defined as the problem of finding the set Σ_G of all minimal siphons containing P_{in} of N.

In particular, $G=(N, \varnothing)$ is the problem of finding the set Σ_G of all minimal siphons of N.

3 Preliminary Definitions and Results

In this section we introduce some functions that will be useful in the following, and also provide (without a formal proof) some theoretical results related to them. Note that most of these functions are based on well-known siphon properties.

Function (N', Φ)= *DeleteSourcePlace*(N)
Input: A Petri net $N=(P, T, F)$.
Output: A Petri net $N'=(P', T', F')$ and a set of minimal siphons Φ.
1) $\Phi=\varnothing$;
2) $P'= P-\{p\in P \mid {}^\bullet p=\varnothing\}$;
3) $T'= T$;
4) $F'= F \cap ((P'\times T') \cup (T'\times P'))$;
5) **for** $p'\in\{p\in P \mid {}^\bullet p=\varnothing\}$ **do**
6) $\Phi=\Phi\cup\{\{p'\}\}$;
7) **end for**
8) **Output:** N' and Φ.

Fact 1: Let N be a Petri net and (N', Φ)= *DeleteSourcePlace*(N). We have $\Sigma_{G1}=\Sigma_{G2}\cup\Phi$, where $G_1=(N, \varnothing)$ and $G_2=(N', \varnothing)$.

Next function removes source transitions, as well as their output places, sink transitions and sink places.

Function N'= *PreHandle*(N)
Input: A Petri net $N=(P, T, F)$.
Output: A Petri net $N'=(P', T', F')$;
1) $N'= N$;
2) **while** there exists a source transition t in N' **do**
3) $T'= T'-\{t\}$;
4) $P'= P'- t^\bullet$;
5) $F'= F'\cap ((P'\times T') \cup (T'\times P'))$;
6) **end while**
7) **while** there exists a sink transition t or a sink place p in N' **do**
8) $T'= T'-\{t\}$; or
9) $P'= P'-\{p\}$;
10) $F'= F'\cap ((P'\times T') \cup (T'\times P'))$;
11) **end while**
12) **Output:** N'.

Fact 2: Let $G_1=(N, P_{in})$ and $G_2=(N', P_{in})$ be two problems, where N'=*PreHandle*(N). We have $\Sigma_{G1}=\Sigma_{G2}$.

Facts 1 and 2 imply that functions *DeleteSourcePlace* and *PreHandle* can be used to reduce the size of a net where minimal siphon enumeration is performed.

The following function describes our improved algorithm to expand a given set P_{in} that must be contained in a minimal siphon.

Function P_{in}'=*ExpandPin* (P_{in}, N)
Input: A set of places P_{in} and a Petri net $N=(P, T, F)$ without source places.
Output: An expanded set of places P_{in}'.
1) $P_{in}'= P_{in}$;

2) **while** there exists $t \in {}^\bullet P_{in}' - P_{in}'^\bullet$ such that ${}^\bullet t = \{p'\}$ or there exists $p \in P_{in}'$ and $p' \in P - P_{in}'$ such that $p'^{\bullet\bullet} = \{p'\}$ **do**
3) $P_{in}' = P_{in}' \cup \{p'\}$;
4) **end while**
5) **Output:** P_{in}'.

Fact 3: Let $G_1=(N, P_{in})$ and $G_2=(N, P_{in}')$ be two problems, where N is a Petri net without source places and $P_{in}'=ExpandPin(P_{in}, N)$. We have $\Sigma_{G1}=\Sigma_{G2}$.

Fact 3 indicates that function *ExpandPin* can be used to expand P_{in} when we search for minimal siphons. Note that the bigger P_{in} is, the faster a solution to the corresponding problem can be obtained.

The following Function *PinNotContainSiphon* can determine whether P_{in} contains a siphon.

Function *flag=PinNotContainSiphon* (P_{in})
Input: A set of places P_{in}.
Output: *flag*. /* *flag*=False implies P_{in} contains a siphon and *flag*=True implies not.*/
1) *flag*= True;
2) Obtain the subnet N_{Pin} generated by P_{in} and ${}^\bullet P_{in} \cup P_{in}^\bullet$;
3) **if** $PreHandle(N_{Pin}) \neq \emptyset$ **then**
4) *flag*= False;
5) **end if**
6) **Output:** *flag*.

Fact 4: Let $G=(N, P_{in})$ be a problem, where P_{in} is not a minimal siphon. If *flag=PinNotContainSiphon* (P_{in})=False, we have $\Sigma_G=\emptyset$.

Fact 4 implies that a problem $G=(N, P_{in})$ has no solution if P_{in} strictly contains a siphon.

The following function *FindAPinMiniSiphon* allows to compute a P_{in}-minimal siphon in N given a set of places P_{in}. Clearly, it allows to compute a minimal siphon if we consider $P_{in}=\emptyset$.

Function *S=FindAPinMiniSiphon* (N, P_{in})
Input: A Petri net $N=(P, T, F)$ and a set of places $P_{in} \subseteq P$.
Output: A P_{in}-minimal-siphon S.
1) **while** $P_{in} \neq P$ **do**
2) $p=Get(P-P_{in})$; /*Function *Get* returns an element of a set.*/
3) $N'=N$;
4) $N=RemovePlace(N, p)$;
/*Function *RemovePlace* returns a net after deleting a place*/
5) **while** there exists a source transition t in N **do**
6) $T=T-\{t\}$;
7) $P=P-t^\bullet$;
8) $F=F \cap ((P \times T) \cup (T \times P))$;
9) **end while**
10) **if** $P_{in} \not\subset P$ **then**
11) $P_{in}=P_{in} \cup \{p\}$;
12) $N=N'$;
13) **end if**
14) $S=P_{in}$;
15) **end while**

Fact 5: Let $N=(P, T, F)$ be a Petri net and $P_{in} \subseteq P$. *S=FindAPinMiniSiphon* (N, P_{in}) is a P_{in}-minimal-siphon.

We want to point out that function *FindAPinMiniSiphon* has already been introduced by Cordone *et al.* in [1]. In particular, in the GPMSE approach [1] it provides a minimal siphon, not only when $P_{in}=\emptyset$, but also in the nontrivial case of $P_{in} \neq \emptyset$. The same result holds in the approach proposed in the following section, namely, the *improved GPMSE* approach.

We conclude this section, introducing a function that allows to determine whether a given siphon S ($|S| \geq 2$) is a minimal siphon.

Function *flag*=*CheckofMiniSiphon* (*S*)
Input: A siphon *S*.
Output: *flag*. /* *flag*=True implies *S* is a minimal siphon otherwise is not.*/
1) *flag*=True;
2) Obtain the subnet N_s generated by *S* and ${}^{\bullet}S \cap S^{\bullet}$;
3) **for** $p \in S$ **do**
4) N'=*RemovePlace*(N_s, *p*);
/*Function *RemovePlace* returns a net after deleting a place*/
5) N'=*PreHandle*(N');
6) **if** $N' \neq \varnothing$ **then**
7) *flag*= False;
8) **end if**
9) **end for**
10) **Output:** *flag*.

Fact 6: Let *S* be a siphon such that $|S| \geq 2$. *S* is a minimal siphon if and only if *flag*=*CheckofMiniSiphon* (*S*)=True.

4 Improved GPMSE Approach

In this section the improved GPMSE approach is presented. This algorithm is coded in function *FindAllMiniSiphon_GP*, which in turns calls function *SonofNode_GP* and some other functions defined in Section III. The main idea behind the improved GPMSE approach is that we iteratively expand a set P_{in} which is forced to be contained in minimal siphons. Moreover, a condition that P_{in} is a siphon is added to restrict further decomposing the problem. The expansion of P_{in} and the added condition can both effectively decrease the total number of sub-problems to be solved. Besides, depth-first search is adopted in improved GPMSE instead of width-first search.

Function (Π)= *FindAllMiniSiphon_GP* (*N*)
Input: A Petri net *N*=(*P*, *T*, *F*).
Output: The set of all minimal siphons Π.
1) $P_{in}=\varnothing$;
2) *Level*=1;
3) (*N*, Π)=*DeleteSourcePlace*(*N*);
4) *N*=*PreHandle*(*N*);
5) **if** $N \neq \varnothing$ **then**
6) Let (N, P_{in}) be the root node of a tree;
7) *S*=*FindAPinMiniSiphon* (*N*, P_{in}); /*Here *S* is a minimal siphon since $P_{in}=\varnothing$ */
8) $\Pi=\Pi \cup \{S\}$;
9) Ψ[*Level*]=*S*; /* Ψ is a linked list for saving minimal siphons that are used for decomposing problems*/
10) Ψ[*Level*+1]=$\varnothing$;
11) *SonofNode_GP* (*N*, P_{in}, *Level*);
12) **end if**
13) **Output:** Π;
14) **End**.

Function *SonofNode_GP* (*N*, P_{in}, *Level*)
Input: A Petri net *N*=(*P*, *T*, *F*), a set of places P_{in}, and *Level* $\in \{1, 2, \ldots\}$.
1) *S*=Ψ[*Level*]; /* a minimal siphon for decomposing*/
2) **if** *S*=$\varnothing$ **then**
3) *S*=*FindAPinMiniSiphon* (*N*, P_{in});
4) Ψ[*Level*]=*S*;
5) Ψ[*Level*+1]= $\varnothing$;
6) $\Pi=\Pi \cup \{S\}$;
7) **end if**
8) $P_{in}'=P_{in}$;
9) **for** $p \in S \backslash P_{in}$ **do**

```
10)     N'=RemovePlace(N, p);
11)   N'=PreHandle(N');
12)   if N'≠∅ then
13)     P_in"=ExpandPin (P_in', N');
14)     Create a node (N', P_in");
15)     Add an arc labeled by "p" from node (N, P_in) to node (N', P_in");
16)     if P_in" ⊆ P' then
17)       if P_in" is a siphon then
18)         if CheckofMiniSiphon (P_in") then
19)             Π=Π∪{P_in"};
20)         end if
21)       else
22)         SonofNode_GP (N', P_in", Level+1);
23)       end if
24)     end if
25)   end if
26)   P_in'= P_in'∪{p};
27)   end for
```

Based on Facts 1-6, we have the following proposition whose proof is omitted for sake of brevity.

Proposition 1: Given a Petri net N, Π= *FindAllMiniSiphon_GP* (N) consists of all minimal siphons in N.

To make the presentation more clear, we illustrate the proposed procedure via a numerical example.

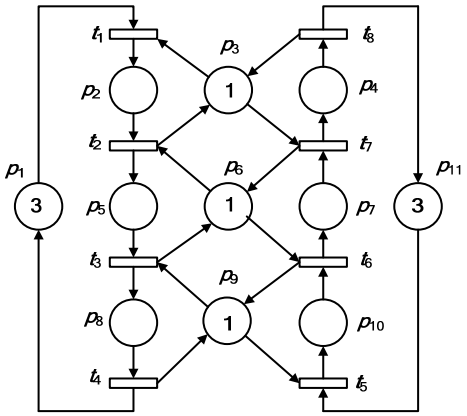


Figure 1. A Petri net N for improved GPMSE

4.1 A Numerical Example to Illustrate the Improved GPMSE

Consider the net N in Figure 1.

1) Since the net has no source and sink elements, Π=∅ and N_1=N after applying functions *DeleteSourcePlace* and *PreHandle*.

2) Let G_1=(N_1, ∅) be the root node of the tree, as shown in Figure 3(a). Note that to make the figure more compact, nets are denoted simply pointing out the indices of its places. An analogous notation is used for siphons. As an example, in Figure 3, P_1=p1-11 is used to denote N_1 with set of places P_1={p_1- p_{11}}.

3) *FindAPinMiniSiphon* (N_1, ∅) is applied, resulting in a minimal siphon S_1={p_8- p_{10}}. Accordingly, we have Π={S_1} and Ψ[1]=S_1. Function *SonofNode_GP* is applied to G_1 with *Level*=1. Details are as follows.

S_1 in Ψ is searched for decomposing problem G_1. First, we delete p_8 from N_1 and then function *PreHandle* is applied. N_2: P_2={p_2-$_7$, p_{10}, p_{11}} is obtained as the output, thus a new node G_2=(N_2, ∅) is created. Now, function *SonofNode_GP* is applied to G_2 with *Level*=2. Since no minimal siphon can be found in Ψ while *Level*=2, a minimal siphon S_2={p_5- p_7} is computed to

decomposing problems in the second level of the tree. Accordingly, we have $\Pi=\{S_1, S_2\}$ and $\Psi[2]=S_2$. After deleting p_5 from N_2, we create a new node $G_3=(N_3, \varnothing)$. Next, Function *SonofNode_GP* is applied to G_3 with *Level*=3. $S_3=\{p_4, p_7, p_{10}, p_{11}\}$ is computed to decomposing problems in the third level of the tree and we have $\Pi=\{S_1\text{-}S_3\}$ and $\Psi[3]=S_3$. After deleting p_4 from N_3, we get a null net. Hence, we delete p_7 from N_3, resulting in a new node G_4. Note that we have $P_{in}=\{p_2\text{-}p_4\}$ in G_4 after Function *ExpandPin* is applied. Here, G_4 is not further decomposed since P_{in} is a siphon.

Similarly, nodes $G_5, G_6, \ldots, G_{17}$ are created one after another with the recursive call of Function *SonofNode_GP*. Finally, the tree in Figure 3(a) is constructed and we can obtain the set of all minimal siphons $\Pi=\{S_1\text{-}S_7\}$ as shown in Figure 3(a).

Note that node ● implies P_{in} is a siphon but not minimal siphon, node ○ implies P_{in} is a minimal siphon, and node ● implies $P_{in} \not\subset P$. Clearly, problems denoted by these nodes do not need to be further decomposed.

5 Numerical Comparisons with the Approach in [1]

In this section we compare the improved versions of GPMSE with the original versions proposed by Cordone *et al.* [1].

5.1 A Comparison with the PN in Figure 1

Let us first consider the PN in Figure 1. The tree resulting from the application of the improved GPMSE is reported in Figure 3(a). Figure 3(b) shows the tree resulting from the application of the original version of the approach, which is clearly much more complex. More details on the comparison between the two approaches are summarized in Table I where: the second column shows the number of nodes of the trees; the third column shows the number of nodes that need to be decomposed; the fourth column shows the maximum number of nodes that need to be saved in memory.

Looking at Table I we may conclude that the improved approach provides better performance with respect to the original one, both in terms of computational time and memory requirement. In particular, the number of nodes of the tree and the number of nodes that need to be decomposed have an impact on the computational time, while the maximum number of nodes that need to be saved during computation have an impact on the memory requirement.

TABLE I. COMPARISON BETWEEN GPMSE AND IMPROVED GPMSE WHEN THE APPROACHES ARE APPLIED TO THE NET IN FIGURE 1

Method	Number of nodes in the tree	Number of nodes that need to be decomposed	Maximum number of nodes that need to be saved in memory
GPMSE	56	26	6
Improved GPMSE	17	5	3

5.2 A Comparison with PNs Randomly Generated

To provide a significant validation of the effectiveness of the proposed approach, we considered a series of PNs generated at random with an increasing number n of places and transitions, with n =31, 32,…,42. For each value of n we randomly generated 100 nets, assuming that d_i is the probability of having an arc going from any place to any transition, and d_o is the probability of having an arc going from any transition to any place. In all the considered cases it was $d_i=d_o=0.05$.

All the above nets have been analyzed using a tool, written in C++, that we developed to implement the GPMSE approach and the improved GPMSE approach. The tool can be downloaded from [12] and also allows the implementation of the GPMSE with depth-first search.

We first observe that for such values of n no results have been obtained using the GPMSE approach in [1] due to memory limitations. On the contrary, both the improved GPMSE method and the GPMSE method using depth search provided results for all the considered valued of n. Such results are summarized in Figure 2, where the average (over the 100 simulations) CPU time versus n is reported. In more detail, Figure 2(a) shows the results of the comparison for all the considered values of n. Figure 2(b) provides a zoom of such results for n that goes from 34 to 37. From such figures we argue that advantages of the improved method become more evident when the dimension of the net grows.

All experimental results have been carried out on a 2.53 GHz Intel Core i3 computer with 3 GB of RAM and Windows 7 operating system.

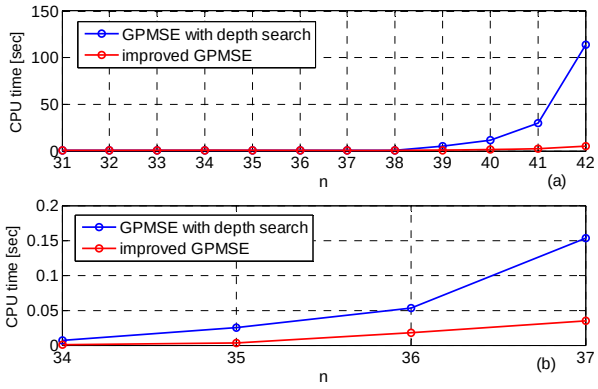


Figure 2. Results of the comparison in Subsection V.B

6 Conclusion and Future Work

The complete siphon enumeration is a difficult problem in Petri nets. This is because the number of siphons grows exponentially with respect to the net size. In this paper we provide an approach that is based on an efficient method proposed by Cordone *et al.* [1]. The novel method guarantees a reduced computational complexity with respect to the original one, even if it is still exponential with respect to the net size, and guarantees a reduced memory requirement especially for large nets.

As a future work, we plan to provide further improvements to the GPMSE approach further expanding the set of places in the minimal siphon and adding more constraints to reduce the number of sub-problems to be solved.

References

[1] R. Cordone, L. Ferrarini, and L. Piroddi, "Enumeration algorithms for minimal siphons in Petri nets based on place constraints," *IEEE Trans. Syst., Man, Cybern., A, Syst., Humans*, vol. 35, no. 6, pp. 844-854, Nov. 2005.

[2] E. E. Cano, C. A. Rovetto, J. M. Colom, "An algorithm to compute the minimal siphons in S4PR nets," *Discrete Event Dynamic Systems*, vol. 22, no. 4, pp, 403–428, 2012.

[3] A. Giua and C. Seatzu, "Modeling and supervisory control of railway networks using Petri nets," *IEEE Trans. Autom. Sci. Eng.*, vol. 5, no. 3, pp. 431-476, 2008.

[4] Z. W. Li and M. C. Zhou, "On siphon computation for deadlock control in a class of Petri nets," *IEEE Trans. Syst., Man, Cybern., A, Syst., Humans*, vol. 38, no. 3, pp. 667-679, 2008.

[5] Z. Li, N. Wu, and M. Zhou, "Deadlock control of automated manufacturing systems based on Petri nets—A literature review," *IEEE Transactions on Systems, Man, and Cybernetics, Part C: Applications and Reviews*, vol. 42, no. 4, pp. 437–462, 2012.

[6] L. Piroddi, R. Cordone, and I. Fumagalli, "Combined siphon and marking generation for deadlock prevention in Petri nets," *IEEE Transactions on Systems, Man, and Cybernetics, Part A: Systems and Humans*, vol.39, no.3, pp. 650-661, 2009.

[7] P. H. Starke, INA: Integrated Net Analyzer, 1992. [Online]. Available: http://www2.info-rmatik.huberlin.de/~starke/ina.html.

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

[9] F. Tricas, J. M. Colom, and J. J. Merelo, "Using the incidence matrix in an evolutionary algorithm for computing minimal siphons in Petri net models," *Proceedings of the 18th International Conference on System Theory, Control and Computing, Sinaia, Romania*, 2014.

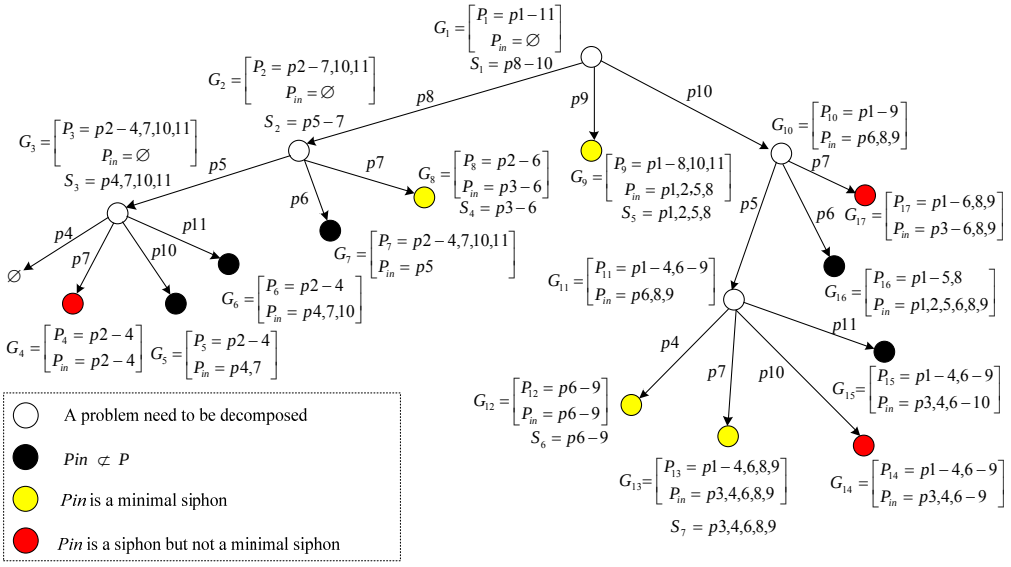
[10] F. Tricas and J. Ezpeleta, "Computing minimal siphons in Petri net models of resource allocation systems: A parallel solution," *IEEE Transactions on Systems, Man, and Cybernetics, Part A: Systems and Humans*, vol.36, no.3, pp. 532-539, 2006.

[11] A. R. Wang, Z. W. Li, J. Y. Jia, and M. C. Zhou, "An effective algorithm to find elementary siphons in a class of Petri nets," *IEEE Transactions on Systems, Man and Cybernetics, Part A*, vol. 39, no. 4, pp. 912-923, Jul. 2009.

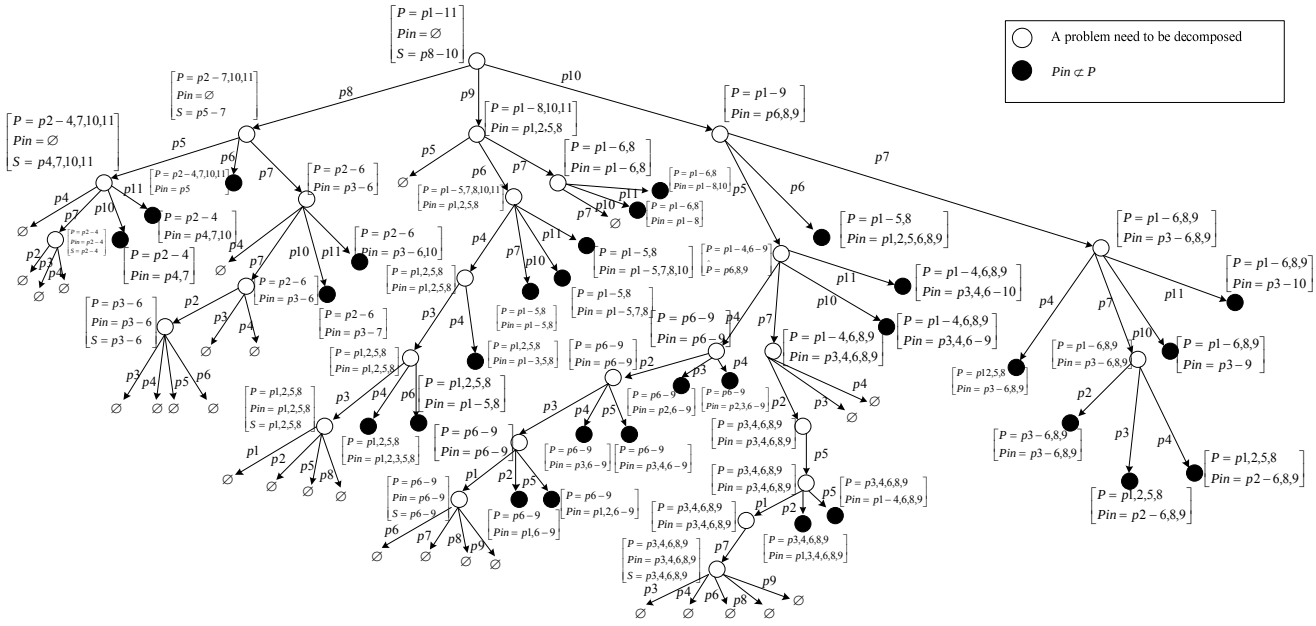
[12] S. G. Wang, the tools for GPMSE, improved GPMSE, and GPMSE with depth-first search, 2015. [Online]. Available: https://www.dropbox.com/sh/4buj8gw1n1s8doy/AADa_xaBnB738s1LzXvYC0ZTa?dl=0.

[13] S. G. Wang, Y. Li, C. Y. Wang, and M. C. Zhou, "Computation of all minimal siphons in Petri nets," in *2012 Proceedings of 9th IEEE International Conference on Networking, Sensing and Control*, pp. 46-51.

[14] S. G. Wang, C. Y. Wang, M. C. Zhou, and Z. W. Li, "A method to compute strict minimal siphons in S3PR based on loop resource subsets," *IEEE Trans. Syst., Man, Cybern., A, Syst., Humans*, vol. 42, no. 1, pp. 226-237, 2012.



(a) A tree generated by improved GPMSE for Figure 1



(b) A tree generated by GPMSE [1] for Figure 1

Figure 3. Two trees generated for Figure 1