Abstract—Minimal siphons play an important role in the development of deadlock control policies for discrete event system modeled by Petri net. A new algorithm based on depth-first search of problem decomposition process is proposed to compute all minimal siphons in an ordinary Petri net. The algorithm can reduce the number of problems in the problem list. The proposed algorithm can solve the problem of high requirement for computer memory in computing all minimal siphons and decrease the memory consumption because the computer memory size is closely related to the number of problems in the problem list. Some examples are used to illustrate the superiority of the proposed algorithm.

Index Terms—Petri nets, Minimal siphons, Deadlock

I. INTRODUCTION

A Petri net is a graphical and mathematical tool that is widely used to describe and analyze the behavior of discrete event systems, including flexible manufacturing systems (FMS) [1]–[7], workflow management system, and automated guided vehicles. In recent years, more and more researchers adopt Petri net models to handle deadlock control problems, which are closely related to siphons. Siphons are a well-known structural object in a Petri net, which are closely related to some basic behavioral properties of the net, such as deadlock-free and liveness. Briefly, a siphon is a set of places such that their input transition set is included in their output transition set. It remains permanently unmarked once it loses all tokens. As a result, if a siphon is empty, their output transitions become permanently disabled, causing a partial or total system deadlock. Therefore, siphon is crucial in deadlock prevention policy of Petri net.

Deadlock control based on siphon [8] is first to compute minimal siphons whose efficient computation is fundamentally important and is much studied in the literature. As a result, complete or partial minimal siphon computation becomes necessity necessary. In the past two decades, researchers have proposed many methods to compute minimal siphons. In [9], Wang proposes a minimal siphons-extraction algorithm based on loop resource subsets. In his approach, Wang utilizes loop resource subsets to compute all the minimal siphons in S3PR. Compared with the method in [10], [11], the algorithm proposed by Wang has higher computational efficiency via many generated examples [12]. Moreover, it can solve some problems in [10].

The methods mentioned above to compute minimal siphons have higher computational efficiency, but they can only be limited to a subclass of Petri nets called Systems of Simple Sequential Processes with Resources (S3PR). Note that the INA-based method and the sign matrix one can be used to compute all minimal siphons for an ordinary Petri net. But the two methods have lower computation efficiency. An effective method is presented in [13], [14] to compute minimal siphons for ordinary Petri net and it has significant representativeness.

In [13], [14], Roberto Cordone and Luca Ferrarini have presented an interesting approach that is based on breadth-first search of problem decomposition. The approach in [13] includes two algorithms which mainly differ in the application of the partitioning procedure. The first one decomposes only the current problem, called local partitioning. The other one decomposes all problems in the unsolved problem list, called global partitioning. The local partitioning algorithm possibly generates spurious solutions, i.e., non-minimal siphons, whereas the global one finds exactly the complete set of minimal siphons. The algorithm in [13] has higher computation efficiency in computing minimal siphons for an ordinary Petri net. However, with the
expansion of the size of the Petri net, too many problems need to be decomposed, leading to high requirement for computer memory. So, siphon computation is expensive.

In this paper, a new algorithm based on depth-first search of problem decomposition process is proposed to compute all minimal siphons in an ordinary Petri net. We know that the computer memory size occupied by the developed program based on the proposed algorithm depends on the number of problems within the problem list. Similarly, the maximum requirement of memory occupied by the developed program depends on the maximum number of problems within the list. Therefore, the proposed algorithm can reduce the number of problems in the problem list and simplify the minimal siphon computation process, solving the problem of high requirement for computer memory in the course of computing all minimal siphons.

The rest of this paper is organized as follows. Section 2 presents the preliminaries used throughout this paper. Algorithm for finding minimal siphons is introduced in Section 3. Illustrative example and comparison with Cordone’s algorithm is given in Section 4. Section 5 concludes this paper.

II. PRELIMINARIES

A Petri net (PN) 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. In a generic way, elements belonging to \( P \cup T \) are called nodes. \( F \subseteq (P \times T) \cup (T \times P) \) is called the flow relation or the set of directed arcs. The flux relation can be given in the form of matrices, namely the input (PRE), output (PST), and incidence (\( C = PST - PRE \)) matrices. Given a net \( N = (P, T, F) \) and a node \( x \in P \cup T \), \( \Sigma_{x} = \{y \in P \cup T | (x, y) \in F\} \) is called the preset of node \( x \), whereas \( \Sigma_{x} = \{y \in P \cup T | (x, y) \in F\} \) is called the postset of node.

A nonempty set \( S \subseteq P \) is a siphon iff \( S \subseteq S_{\phi} \). A trap is called a trap iff some of its places have token(s). A siphon is minimal iff there is no siphon contained in it as a proper subset.

Definition 1 [13]-Reduction Function: Let \( G = (P, T, F) \) be a PN and \( \tilde{P} \subset P \). Then \( \tilde{G} = \text{red}(G, \tilde{P}) \) is a PN \( (\tilde{P}, \tilde{T}, \tilde{F}) \) iff \( \tilde{T} = \{ t \in T | \Sigma_{p} \cap \tilde{P} \neq \phi \} \) and \( \tilde{F}(p, t) = F(p, t) \cap \tilde{P} \neq \phi \) for \( p \in \tilde{P}, \forall t \in \tilde{T} \).

Definition 2 [13]: Let \( G = (P, T, F) \) be a PN, \( P_{in} \subseteq P \), \( P_{out} \subseteq P \). \( \Pi = (G, P_{in}, P_{out}) \) indicates the problem of finding and \( \Sigma_{i} \) indicates the set of all siphons of \( G \) subject to the constraints: i) \( \forall S \subseteq \Sigma_{i}, S \) is a \( P_{in} \)-minimal siphon; ii) \( \forall S \in \Sigma_{i}, S \cap P_{out} = \phi \).

III. ALGORITHM FOR FINDING MINIMAL SIPHONS

In this section, the global partitioning algorithm proposed by Cordone [13] is presented firstly. Then, a new algorithm based on depth-first search is proposed to compute all the minimal siphons. Subsequently, an example is shown to illustrate the proposed algorithm.

A. Cordone’s Algorithm

According to the lemmas introduced in [13], an iterative search algorithm can be devised to find all siphons solving the generic problem \( \Pi = (G, P_{in}, P_{out}) \). This algorithm is based on suitable problem-reduction techniques (Lemmas 9 and 13) [13] and problem decomposition (Lemmas 2 and 12) [13] to explore the solution space.

In the following, lists will be employed, with the following notation. A list \( \Lambda \) is an ordered set of elements \( \Lambda = (\lambda_{1}, \lambda_{2}, \ldots, \lambda_{k}) \). The pop function extracts the first element of a list: \( \text{pop}(\Lambda) \) returns \( \lambda_{1} \) and the list is modified to \( \Lambda = (\lambda_{2}, \ldots, \lambda_{k}) \). An empty list is denoted as \( \Lambda = () \).

Global-partitioning algorithm in [13] can be summarized as follows:

function \( \Sigma_{i} = \text{FindAllMinimalSiphons}(G) \)
\( (\Sigma_{i}, P_{out}) = \text{SinglePlaceSiphons}(G) \)
\( \Pi = (G, \phi, P_{out}) \)
\( \Lambda = (\Pi) \)
\( \Sigma_{i} = \Sigma_{i} \cup \text{SolveList}(\Lambda) \)

function \( \Sigma_{i}, P_{out} = \text{SinglePlaceSiphons}(G) \)
\( \Sigma_{i} = \phi, \tilde{P} = P, P_{out} = \phi \)
while \( \tilde{P} \neq \phi \)
\( p = \text{Get}(\tilde{P}) \)
if \( p = \phi \)
then \( \Sigma_{i} = \Sigma_{i} \cup \{p\}, P_{out} = P_{out} \cup \{p\} \)
endif
\( \tilde{P} = \tilde{P} \setminus \{p\} \)
endwhile

Function SinglePlaceSiphons is used to rule out siphons that are constituted by a single place, which is a minimal siphon.

function \( \Sigma_{i} = \text{SolveList}(\Lambda) \)
\( \Sigma_{i} = \phi \)
while \( \Lambda \neq () \)
\( \Pi = \text{pop}(\Lambda) \)
if \( \Sigma_{i} = \phi \), then \( (S, \Pi) = \text{FindSiphon}(\Pi) \)
else \( S = P \)
endif
if \( S \neq \phi \), then
if \( S \neq P \) then \( S = \text{FindMinimalSiphon}(\Pi) \)
endif
\( \Sigma_{i} = \Sigma_{i} \cup \{S\} \)
\( \Lambda = ((\Pi), \Lambda) \)
\( \Lambda = \text{Partition}(\Lambda, S) \)
endif
endwhile

Function SolveList extracts the first problem in the list and searches for one generic siphon, subject to
the problem’s place constraints, by means of the FindSiphon function.

function (S, Π') = FindSiphon(Π)  
Π' = Π  
isReducible = true  
while isReducible  
if P_in ∩ P_out ≠ ø , then S = ø , return  
elseif P_in ∪ P_out = P then  
if *P_in ⊆ P_in* then S = P_in else S = ø , endif  
return  
endif  
if P_out ≠ ø , then G = red(G, P = P_out), Π' = (G, P_in, ø )  
endif  
(isReducible, Π') = Reduce(Π')  
endwhile  
S = P

Function FindSiphon iteratively tests the conditions of Lemma 8 [13] for trivial solutions of the siphon search problem and if none is found, reduces the problem according to Lemma 9 by means of the function Reduce.

function (isReducible, Π') = Reduce(Π)  
Π' = Π  
isReducible = true  
T = { t ∈ T , such that • t = ø }, P = T•  
~T = { t ∈ P ∩ P• , such that | • t | = 1 }, ~P = •T ∩ (P − P_in)  
If Π = ø and ~P = ø then  
isReducible = false  
else Π' = (G, P_in ∪ ~P , P_out ∪ Π )  
endif

Function Reduce reduces the problem according to Lemma 9 [13] and returns a nonreducible siphon search problem and a siphon.

function S = FindMinimalSiphon(Π)  
S = P , ~P = S − P_in  
while ~P ≠ ø  
p = Get ( ~P )  
if ( • t ∩ S ) ⊊ { p } or t ∩ S = ø , ∀ t ∈ P• then S = S − { p }  
endif  
~P = ~P − { p }  
endwhile  
~P = S − P_in, ~P_in = P_in  
while ~P ≠ ø  
p = Get ( ~P ) , ~G = red(G, S − { p }) , ~Π = (~G , ~P , ø )  
(~S , ~Π ) = FindSiphon (~Π )  
if ~S ≠ ø then ~S , ~P = S − ~P_in, G = ~G

else ~P = ~P − { p } , ~P_in = P_in ∪ { p } endif  
endwhile

Function FindMinimalSiphon operates on a nonreducible siphon search problem, which admits at least one siphon, equal to the whole set of net places. Function Get returns an element of a set.

function Λ = Partition(Λ , S)  
Λ = ()  
while Λ = ()  
Π = pop ( Λ )  
~P = S − P_in  
while ~P ≠ ø and P_out ∩ P_in = ø  
p = Get ( ~P )  
( S , Π ) = FindSiphon(Π)  
if S≠ ø then  
Λ = ( Λ , ( Π , S ))  
endif  
~P = ~P − { p }, ~P_in = P_in ∪ { p }  
endwhile

Function Partition applies Lemma 12[13] to decompose the current problem in order to exclude all the siphons that contain S from the solution sets of the generated subproblems.

B. The Proposed Algorithm

The proposed algorithm is listed below:

function Σfl = FindMinimalSiphon(Π_0)  
Σfl = ø  
Λ = ø  
( S , Π ) = FindSiphon(Π)  
if S≠ ø then  
Σfl = Σfl ∪ { S }  
Λ = ( Λ , ( Π , S ))  
endwhile

while(Λ≠ ø )  
( Π , S )= pop ( Λ )  
p = Get(S)  
S = S − { p }  
if S≠ ø then  
P_in = P_in ∪ { p }  
Λ = ( Λ , ( Π , S ))  
endwhile

( S , Π ) = FindSiphon(Π)  
if S≠ ø then  
if S≠ P_in then S = FindMinimalSiphon(Π)  
endif  
Σfl = Σfl ∪ { S }  
Λ = ( Λ , ( Π , S ))  
endwhile

© 2014 ACADEMY PUBLISHER
We take a simple Petri net $G$ as an example to illustrate the proposed algorithm. Consider a simple Petri net $G$ in Fig.1, where $P = \{p_1, p_6\}$ and $T = \{t_1, t_4\}$. $G$ has three minimal siphons: $S_1 = \{p_2, p_4, p_5, p_6\}$, $S_2 = \{p_5, p_3\}$, $S_3 = \{p_1, p_2\}$. The tree obtained by the proposed algorithm is shown as in Fig.2, which is the same as the tree obtained by Cordone’s algorithm [13].

The problem and the number of problems can be obtained based on the proposed algorithm in the paper, which are shown in Table 1.

![Figure 1: A Petri net $G$](image1)

![Figure 2: A tree obtained by the proposed algorithm](image2)

**Table 1.** PROBLEM LIST AND NUMBER OF PROBLEMS IN THE LIST

<table>
<thead>
<tr>
<th>problem list</th>
<th># of problem</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\Lambda = (I_0)$</td>
<td>1</td>
</tr>
<tr>
<td>$\Lambda = (I_0, I_1)$</td>
<td>2</td>
</tr>
<tr>
<td>$\Lambda = (I_0, I_2)$</td>
<td>1</td>
</tr>
<tr>
<td>$\Lambda = (I_0, I_1, I_2)$</td>
<td>3</td>
</tr>
<tr>
<td>$\Lambda = (I_0, I_1, I_2)$</td>
<td>3</td>
</tr>
<tr>
<td>$\Lambda = (I_0, I_1)$</td>
<td>2</td>
</tr>
<tr>
<td>$\Lambda = (I_0)$</td>
<td>2</td>
</tr>
</tbody>
</table>

According to the proposed algorithm, we can easily have the following results.

**Theorem 1:** Let $G = (P, T, F)$ be a PN and $\Pi_0 = (G, \phi, \phi)$ be the associated problem of finding all minimal siphons. Then, if either the proposed method or global partitioning algorithm is applied to solve $\Pi_0$, all minimal siphons of $G$ are returned by the algorithm.

**Proof:** Similar to the proof in [13].

**Theorem 2:** Let $G = (P, T, F)$ be a PN and $\Pi_0 = (G, \phi, \phi)$ be the associated problem of finding all minimal siphons. Then, if either the proposed method or global partitioning algorithm is applied to solve $\Pi_0$, the algorithm will not return any non-minimal siphons of $G$.

**Proof:** Similar to the proof in [13].

**Theorem 3:** The maximum number of problems in the list does not exceed the number of minimal siphons.

**Theorem 4:** The maximum number of problems in the list is not more than the number of problem decomposition layer minus 1.

**Theorem 5:** The number of minimal siphons is equal to the number of problem decomposition layer minus 1.

**Proof:** The proof of Theorem 3 to Theorem 5 is obvious according to the proposed algorithm.

The global partitioning algorithm is based on breadth-first search. According to Theorem 1 and Theorem 2, if the global partitioning algorithm is applied to solve $\Pi_0$, all minimal siphons of $G$ are returned by the algorithm. The proposed algorithm is based on depth-first search. According to Theorem 1 and Theorem 2, if the proposed algorithm is applied to solve $\Pi_0$, all minimal siphons of $G$ are returned by the algorithm.

**IV. ILLUSTRATIVE EXAMPLE AND COMPARISON WITH CORDONE’S ALGORITHM**

First, a Petri net that has 36 places and 30 transitions is taken as an example[12] to be used to illustrate the difference between the two algorithms about requirement of computer memory in the course of computing minimal siphons. In Fig.3 (a), the requirement of computer memory is from 1.37 GB to 1.58 GB. In Fig.3 (b), the requirement of computer memory remains always 1.37 GB. The result indicates that the developed program based on the proposed algorithm takes up a relatively small computer memory.

![Figure 3. (a) Requirement of computer memory based on Cordone’s algorithm](image3)
Within the problem list. Similarly, the maximum of problems that need to continue to be decomposed developed program occupied depends on the number under Windows 7 operating system.

2.9-GHz Pentium-III computer with 4-GB memory 2.88 GB. The computer memory is nearly full at 107th requirement of computer memory is from 1.15 GB to 86 places and 70 transitions. In Fig.4 (a), the requirement of computer memory remains always 1.15 GB and the developed program based on Cordone’s algorithm stops running automatically. In Fig.4 (b), the requirements of computer memory Cordone’s algorithm stops running automatically. In Fig.4 (b), the requirements of computer memory remains always 1.15 GB and the developed program based on the proposed algorithm has been running.

Note that the computation above is carried out on a 2.9-GHz Pentium-III computer with 4-GB memory under Windows 7 operating system.

We know that the computer memory size which the developed program occupied depends on the number of problems that need to continue to be decomposed within the problem list. Similarly, the maximum requirement of computer memory depends on the maximum number of problems within the list.

Clearly, in Fig.4 (a), with the expansion of the size of the Petri net, the number of problems will increase exponentially and be far greater than the number of minimal siphons. Then, the more requirement of computer memory will be needed. Eventually, computer memory will be nearly full and the developed program will stop running. But, in Fig.4 (b), with the expansion of the size of the Petri net, the number of problems will be mutative within a certain range and does not exceed the number of minimal siphons. Moreover, the requirement of computer memory remains always constant. We can see that the superiority of the proposed algorithm that can greatly reduce the number of problems within the problem list is obvious. Moreover, with the expansion of the size of the Petri net, this superiority is more and more obvious.

V. Conclusion

It is well know that deadlock problem [15]-[24] is related to minimal siphons. In the paper, a new algorithm based on depth-first search of problem decomposition process is proposed to compute all minimal siphons for an ordinary Petri net. Comparison with Cordone’s algorithm, the proposed algorithm can solve the problem of excessive requirement for computer memory in the course of computing all minimal siphons and decrease requirement of computer memory. Future work includes extending this algorithm to improve computation minimal siphon efficiency for ordinary Petri net.

REFERENCES


