IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 23 July 2024, accepted 6 August 2024, date of publication 13 August 2024, date of current version 22 August 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3442883

RESEARCH ARTICLE

Improved Local Partitioning Minimal-Siphon Enumeration Method

JUAN PAN AND DAN YOU, (Member, IEEE)
School of Information and Electronic Engineering (Sussex Artificial Intelligence Institute), Zhejiang Gongshang University, Hangzhou 310018, China

Corresponding author: Dan You (youdan000@hotmail.com)

This work was supported in part by the Research Start-Up Funding of Zhejiang Gongshang University under Grant 1120XJ2322020 and in part by the Zhejiang Provincial Key Laboratory of New Network Standards and Technologies under Grant 2013E10012.

ABSTRACT Siphon computation is a basic step for developing siphon-based deadlock control approaches in a Petri net (PN) system. This work studies the enumeration of minimal siphons in a PN. Due to the fact that the number of siphons in a PN theoretically grows exponentially with the net size, the siphon enumeration is basically time-consuming especially in a large-size net. To our best knowledge, the method of *local partitioning minimal-siphon enumeration* (LPMSE) has the best performance among all the methods applicable to any arbitrary PN. In this paper, we show that the improvement of LPMSE is possible and thereby develop an improved LPMSE. It is validated by experimental results that the improved LPMSE consumes less time than LPMSE.

INDEX TERMS Petri nets, minimal siphons, siphon computation, problem partitioning.

I. INTRODUCTION

Petri nets (PN) are a popular mathematical modeling tool for tackling privacy issues [1], [2], [3], diagnosability problems [4], [5], [6], prognosability problems [7], [8], and deadlock problems [9], [10], [11], [12], [13] in discrete event systems. In the formalism of PN, deadlock control approaches are basically classified into two categories, i.e., those based on reachability analysis [14], [15], [16], [17], [18], [19], [20] and those based on structural analysis [21], [22], [23], [24], [25], [26], [27], [28], [29], [30]. Siphon-based methods fall into the latter category. A siphon [31] is a structure of a PN that is essentially a set of places satisfying a certain property. A deadlock may appear when a siphon is not sufficiently marked at a reachable marking. To prevent the occurrence of deadlocks in a PN system, siphon-based deadlock control approaches add monitors to control siphons such that they are all sufficiently marked at every reachable marking. As a result, the computation of minimal siphons is required and the efficiency of siphon computation is an important factor in deciding the performance of such an approach. Consequently, much work focuses on the computation of minimal siphons, see, e.g., [32], [33].

The associate editor coordinating the review of this manuscript and approving it for publication was Zhiwu Li.

We know that the number of siphons in a PN theoretically grows exponentially with the net size. Thus, finding an efficient minimal-siphon computation method becomes more challenging and is also of great importance. Many methods have been provided for minimal-siphon enumeration in an arbitrary PN, such as integrated net analyzer (INA) [34], linear integer programming methods [22], [35], [36], methods based on semi-tensor product of matrices [37], and methods based on problem partitioning [38], [39]. To our best knowledge, the methods of *global partitioning minimal-siphon enumeration* (GPMSE) and *local partitioning minimal-siphon enumeration* (LPMSE) proposed by Cordone et al. [38] have higher computational efficiency than the other approaches. Since the proposal of these two methods, it is hard to find a method better than them. On the other hand, some researchers propose siphon enumeration methods that are applicable to specific classes of PNs only, such as methods based on loop resource subsets [40], methods based on resource circuits [41], [42], methods based on pruning graphs [43], [44], parallel algorithms [45], and genetic-algorithm-based methods [46]. By utilizing specific properties of considered nets, the computational efficiency can be improved.

In this paper, we consider the minimal-siphon enumeration methods applicable to arbitrary PNs. In particular, we focus on the methods of LPMSE and GPMSE since they provide good performance and are widely used to enumerate minimal siphons. In more detail, these two methods are based on the idea that iteratively partitioning a problem into several simper sub-problems by introducing place constraints. The main difference between the two methods is that LPMSE uses a computed siphon to partition the current problem only, while GPMSE uses a computed siphon to partition all the existing problems. In our previous work [47], we show that the improvement of GPMSE is possible and proposed the so-called improved GPMSE. Although the improved GPMSE behaves much better than GPMSE, it can hardly beat LPMSE especially in large-size PNs. Thus, in this work, we consider the improvement of LPMSE. The improvements mainly lie in

1) the size of a PN considered in a problem is further reduced;

2) the set P_{in} of places required to be contained in a searched minimal siphon is further expanded; and

3) a condition that P_{in} strictly contains a siphon is added to terminate the partitioning of a problem.

To validate the superiority of the improved LPMSE over LPMSE, we implement both LPMSE and the improved LPMSE by developing a tool written in C++. Using this tool, we carry out an experiment. The experimental results show that the improved LPMSE consumes less time than LPMSE in computing minimal siphons, which becomes more evident with the increase of the net size.

The remainder of the paper is organized as follows. Section II recalls the basic knowledge of PNs. Section III introduces the improved LPMSE together with an example illustrating the improved method. In Section III-C, we compare the performance of the improved LPMSE with its original version. Section IV concludes this paper.

II. BASIC KNOWLEDGE OF PETRI NETS

A *Petri net* (PN) is a three-tuple $Q = (P, T, F)$ where P is the set of *places*, T is the set of *transitions*, and $F \subseteq (P \times T) \cup (T \times P)$ is the *flow relation*. It is assumed that P and T are nonempty, finite and disjoint sets.

A node (i.e., place or transition) with no input is called a *source* node, and one with no output is called a *sink* node.

Given a node $x \in P \cup T$, we denote ${}^\bullet x = \{y \in P \cup T \mid (y, x) \in F\}$ the *preset* of x and $x^\bullet = \{y \in P \cup T \mid (x, y) \in F\}$ the *post-set* of x. Given a set of nodes $X \subseteq P \cup T$, we denote ${}^\bullet X = \bigcup_{x \in X} {}^\bullet x$ and $X^\bullet = \bigcup_{x\in X} x^\bullet$. Moreover, ${}^{\bullet\bullet}x = {}^\bullet({}^\bullet x)$, and $x^{\bullet\bullet} = (x^\bullet)^\bullet$, $\forall x \in P \cup T$.

Given a set of places $P' \subseteq P$ and a set of transitions $T' \subseteq T$, $Q' = (P', T', F')$ is a *subnet* of Q generated by P' and T' if $F' = F \cap [(T' \times P') \cup (P' \times T')]$.

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

Given a set $P_{in} \subseteq P$, a siphon is said to be P_{in}*-minimal* if it includes *all* places in P_{in} and does not contain any other siphon including all places in P_{in}. Consider the PN in Fig. 1. We can see that $S = \{p_4, p_6, p_8, p_{13}, p_{14}\}$ is a P_{in}-minimal siphon with $P_{in} = \{p_8, p_{13}\}$. Note that a P_{in}-minimal siphon is a minimal siphon if $P_{in} = \emptyset$ and otherwise it is not necessarily a minimal siphon. Here, $S = \{p_4, p_6, p_8, p_{13}, p_{14}\}$ is not a minimal siphon in the PN in Fig. 1 since we can find a siphon, e.g., $S' = \{p_6, p_8, p_{14}\}$, contained in S.

III. IMPROVED LPMSE

The method of Local Partitioning Minimal-Siphon Enumeration (LPMSE) is proposed by Cordone et al. [38]. Their basic idea is iteratively partitioning a problem to be solved into several simpler sub-problems so that the solution to the original problem can be more easily obtained. LPMSE generally leads to a large quantity of sub-problems, especially when applied to a PN with large size. In this section, we propose an improved version of LPMSE, aiming to get higher computational efficiency by generating a reduced number of sub-problems.

First, we define the concept of a *problem* that is investigated in the paper.

Definition 1: Given a PN $Q = (P, T, F)$ and a set of places P_{in}, $\pi = (Q, P_{in})$ is a *problem* of finding all minimal siphons in Q that includes all places of P_{in}. We denote Σ_π the solution to the problem $\pi = (Q, P_{in})$, i.e., the set of all minimal siphons in Q that includes all places of P_{in}.

Trivially, $\pi = (Q, \emptyset)$ is a problem of finding all minimal siphons in the PN Q. We next introduce the improved LPMSE that solves the problem. In more detail, in the following subsections, we first introduce some preliminary functions, then provide an algorithm of the improved LPMSE, and finally present an example to illustrate how the improved LPMSE works.

A. PRELIMINARY FUNCTIONS

In this subsection, we introduce some functions that will be used in the algorithm of the improved LPMSE. These functions are designed based on some well-known properties of (minimal) siphons.

Function $Q' \leftarrow NetReduce(Q)$
Input: $Q = (P, T, F)$ with no source place.
Output: $Q' = (P', T', F')$.

```
1.  $Q' \leftarrow Q$; /*Equivalently, it is $(P', T', F') \leftarrow (P, T, F)$. */
2.  while $\exists t \in T'$ s.t. ${}^\bullet t = \emptyset$ in $Q'$ do
3.      $T' \leftarrow T' \setminus \{t\}$;
4.      $P' \leftarrow P' \setminus t^\bullet$;
5.      $F' \leftarrow F' \cap ((T' \times P') \cup (P' \times T'))$;
6.  end while
7.  while $\exists x \in P' \cup T'$ s.t. $x^\bullet = \emptyset$ in $Q'$ do
8.      if $x \in P'$ then
9.          $P' \leftarrow P' \setminus \{x\}$;
10.     else
11.         $T' \leftarrow T' \setminus \{x\}$;
12.     end if
13.     $F' \leftarrow F' \cap ((T' \times P') \cup (P' \times T'))$;
14. end while
15. output: $Q'$.
```

Function *NetReduce* is used to reduce the size of a PN while not changing the result of minimal-siphon enumeration. In particular, it consists of two stages in reducing the net size. In the first stage, a source transition and its output places are repeatedly removed; in the second stage, a sink node (transition/place) is repeatedly removed. Typically, by reducing the net size, we may speed up the enumeration of minimal siphons.

```
Function $P'_{in} \leftarrow PinExpand(P_{in}, Q)$
Input: $Q = (P, T, F)$ with no source place and a set of places $P_{in} \subseteq P$;
Output: A set of places $P'_{in} \subseteq P$.
1.  $P'_{in} \leftarrow P_{in}$;
2.  while($\exists t \in {}^\bullet P'_{in} \backslash P'^{\bullet}_{in}$ s.t. ${}^\bullet t = \{p'\} \vee$
            $\exists p \in P'_{in}, \exists p' \in P\backslash P'_{in}$ s.t. $p^{\bullet\bullet} = \{p'\}$) do
3.      $P'_{in} \leftarrow P'_{in} \cup \{p'\}$;
4.  end while
5.  output: $P'_{in}$.
```

Function *PinExpand* is used to expand the set P_{in} that is required to be contained in a searched minimal siphon while not changing the result of minimal-siphon enumeration. This is guaranteed by the fact that when some places are included in a minimal siphon, some other places are included as well. In particular, function *PinExpand* repeatedly expands P_{in} by including a place $p' \in P\backslash P_{in}$ by considering the following two cases. Case 1: there is an input but not output transition of P_{in} who has a unique input place that is p'; Case 2: there is a place p in P_{in} who has a unique output place that is p'. Basically, by expanding the set P_{in}, we may enumerate minimal siphons more quickly.

```
Function $ans = PinContainSiphon(P_{in})$
Input: A set of places $P_{in}$.
Output: $ans \in \{$True, False$\}$. /* $ans$ = True implies $P_{in}$ contains a siphon and $ans$ = False implies not. */
1.  $ans \leftarrow$ False;
2.  obtain the subnet $Q_{Pin}$ generated by $P_{in}$ and ${}^\bullet P_{in} \cup P^\bullet_{in}$;
3.  if $NetReduce(Q_{Pin}) \neq \emptyset$ then
4.      $ans \leftarrow$ True;
5.  end if
6.  output: $ans$.
```

Function *PinContainSiphon* determines whether or not a set of places P_{in} contains a siphon by calling function *NetReduce* on the subnet Q_{Pin} generated by P_{in} and ${}^\bullet P_{in} \cup P^\bullet_{in}$.

Finally, we notice that functions *DeletePlace*, *SiphonIsMini* and *GetaPinMiniSiphon* will also be called in the algorithm of the improved LPMSE. We introduce them in brief as follows. The readers are referred to [47] for their detailed pseudocode.

- $Q' \leftarrow DeletePlace(Q, p)$: *DeletePlace* returns a PN Q' by removing the place p and its related arcs from the PN Q.
- $ans \leftarrow SiphonIsMini(S)$: *SiphonIsMini* determines whether the input siphon S is minimal. It returns ans = True if S is minimal and ans = False otherwise.
- $S \leftarrow GetaPinMiniSiphon(Q, P_{in})$: *GetaPinMiniSiphon* returns a P_{in}-minimal siphon S in the PN Q. Note that S is not necessarily a minimal siphon.

B. ALGORITHM OF THE IMPROVED LPMSE

In this subsection, we provide an algorithm of the improved LPMSE.

```
Algorithm 1: Improved LPMSE
Input: $Q = (P, T, F)$.
Output: The set $\Phi$ of all minimal siphons in $Q$.
1.  $\Phi \leftarrow \emptyset$;
2.  for each source place $p$ in $Q$ do
3.      $\Phi \leftarrow \Phi \cup \{\{p\}\}$;
4.  end for
5.  update the net $Q = (P, T, F)$ by deleting all source places and their related arcs from $Q$;
6.  $Q \leftarrow NetReduce(Q)$;
7.  if $Q \neq \emptyset$ then
8.      create a tree with the root node $(Q, \emptyset)$;
9.      $S \leftarrow GetaPinMiniSiphon(Q, \emptyset)$;
10.     $\Phi \leftarrow \Phi \cup \{S\}$;
11.     $\Phi \leftarrow LocalPartition(Q, \emptyset, S, \Phi)$;
12. end if
13. output: $\Phi$.
```

We explain Algorithm 1 as follows. First, every source place is searched since every source place itself constitutes a minimal siphon. Then, we search the remaining minimal siphons in the net, which is equivalent to solving the problem $(Q, \emptyset)$, where Q is the resulting net by removing all source places from the original net and reducing the net size. The problem $(Q, \emptyset)$ is solved based on repeatedly partitioning problems, which is realized by recursively calling function *LocalPartition*, and the procedure is described by a tree.

Without loss of generality, suppose that (Q, P_{in}) is a problem to be partitioned. The procedure of partitioning the problem (Q, P_{in}) is detailed as follows.

We compute a P_{in}-minimal siphon S, by which the problem (Q, P_{in}) is partitioned by calling $LocalPartition(Q, P_{in}, S, \Phi)$. Note that we should determine if S is a minimal siphon. If so, we add S to the solution set Φ. Suppose that $S\backslash P_{in} = \{p_1, p_2, \ldots, p_k\}$. Then, there are k sub-problems, namely,

$\pi_1 = (DeletePlace(Q, p_1), P_{in})$,
$\pi_2 = (DeletePlace(Q, p_2), P_{in} \cup \{p_1\})$,
$\pi_3 = (DeletePlace(Q, p_3), P_{in} \cup \{p_1, p_2\})$,
$\ldots$,
$\pi_k = (DeletePlace(Q, p_k), P_{in} \cup \{p_1, p_2, \ldots, p_{k-1}\})$.

Trivially, the solution to the problem (Q, P_{in}) consists of solutions to all the above k sub-problems together with S if S is a minimal siphon. We observe that not every sub-problem needs to be further partitioned. Consider a sub-problem $\pi_i = (Q', P'_{in})$ as an example. We reduce the size of Q' and expand

```
Function Φ ← LocalPartition(Q, P_in, S, Φ)
Input: Q = (P, T, F), a set P_in of places, a siphon S, and a set Φ of minimal siphons.
Output: An updated set Φ of minimal siphons.
1.  $P'_{in} \leftarrow P_{in}$;
2.  for $p \in S \backslash P_{in}$ do
3.     $Q' \leftarrow DeletePlace(Q, p)$;
4.     $Q'' \leftarrow NetReduce(Q')$;
5.     if $Q'' \neq \emptyset$ then
6.       $P''_{in} \leftarrow PinExpand(P'_{in}, Q)$;
7.       introduce a new node $(Q'', P''_{in})$ and add an arc with the label "$p$" from node $(Q, P_{in})$ to node $(Q'', P''_{in})$;
8.       if $P''_{in} \subseteq P''$ then
9.         if $PinContainSiphon(P''_{in})$ then
10.          if $P''_{in}$ is a siphon $\wedge$ $SiphonIsMini(P''_{in})$ then
11.            $\Phi \leftarrow \Phi \cup \{P''_{in}\}$;
12.          end if
13.        else
14.          $S' \leftarrow GetaPinMiniSiphon(Q'', P''_{in})$;
15.          if$SiphonIsMini(S')$ then
16.            $\Phi \leftarrow \Phi \cup \{S'\}$;
17.          end if
18.          $\Phi \leftarrow LocalPartition(Q'', P''_{in}, S', \Phi)$;
19.        end if
20.      end if
21.    end if
22.    $P'_{in} \leftarrow P'_{in} \cup \{p\}$;
23. end for
24. output:Φ.
```

the set P'_{in}, which leads to a problem denoted as (Q'', P''_{in}). In the case that $P''_{in} \not\subset P''$ or P''_{in} contains a siphon, we certainly know that the solution to (Q'', P''_{in}) is empty or consists of a set P''_{in} and thus the problem (Q'', P''_{in}) does not need to be further partitioned. In the other cases, we again compute a P_{in}-minimal siphon S' to partition the problem (Q'', P''_{in}) by repeating the above procedure. Also, if S' is a minimal siphon, it is added to the set Φ.

We notice that the depth-first search is adopted in Algorithm 1 to repeatedly generate sub-problems. In the case that no more sub-problem needs to be generated, the final set Φ is clearly the set of all minimal siphons of the input PN. Hence, we have the following conclusion.

Theorem 1: Given a PN as the input, Algorithm 1 (i.e., the improved LPMSE) enumerates all minimal siphons.

Proof: Straightforward from the above explanation.

Compared with LPMSE in [38], the proposed method mainly includes the following three improvements:

1) The size of a PN is further reduced;

LPMSE only considers repeatedly removing a source transition and its output places to reduce the net size, while the proposed method considers repeatedly removing a sink node as well. Such an improvement is realized by function *NetReduce*.

2) The set P_{in} is further expanded;

LPMSE repeatedly expands P_{in} by including a place $p' \in P \backslash P_{in}$ only in the case that there is an input but not output transition of P_{in} who has a unique input place that is p', while the proposed method repeatedly expands P_{in} by including a place $p' \in P \backslash P_{in}$ also in the case that there is a place p in P_{in} who has a unique output place that is p'. Such an improvement is realized by function *PinExpand*.

3) A condition that P_{in} strictly contains a siphon is added to terminate the partitioning of a problem.

Consider a problem $\pi = (Q, P_{in})$, where $Q = (P, T, F)$ is a PN output by function *NetReduce*. LPMSE terminates its further partitioning in the case that $P_{in} \not\subset P$ or P_{in} itself is a siphon, while the proposed method terminates its further partitioning in the case that $P_{in} \not\subset P$ or P_{in} contains a siphon. Clearly, P_{in} strictly contains a siphon is an additional termination condition in our proposed method.

C. EXAMPLE

In this subsection, we present an example to illustrate the improved LPMSE.

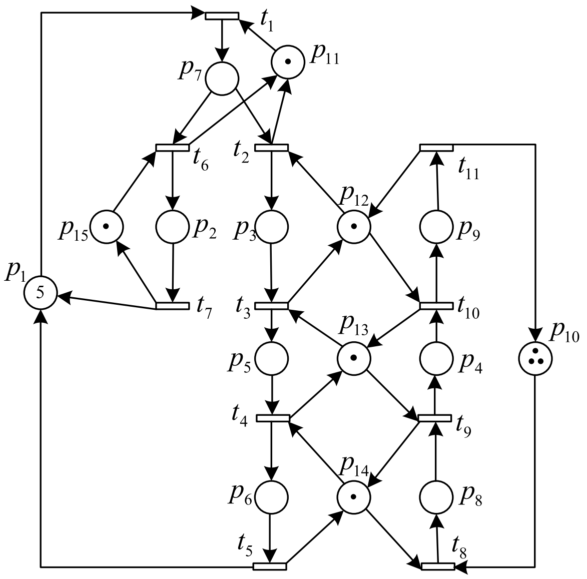


FIGURE 1. PN $Q = (P, T, F)$.

TABLE 1. Numerical results regarding trees generated by LPMSE and the improved LPMSE when they are applied to the net in Fig. 1.

	Number of nodes in the tree	Number of nodes that need to be partitioned	Maximum number of nodes that need to be saved in memory
LPMSE	47	19	6
Improved LPMSE	29	10	6

Consider a PN $Q = (P, T, F)$ in Fig. 1. The improved LPMSE works as follows.

TABLE 2. Experimental results ($d_i = d_o = 0.05$).

PN Size $n=\|P\|=\|T\|$	Average number of siphons	LPMSE		Improved LPMSE		Percentage of CPU time reduced
		Average number of nodes	Average CPU time (s)	Average number of nodes	Average CPU time (s)	
10	5.82	0	0.000024054	0	0.00000757	68.53%
15	6.4	0.06	0.000032728	0.06	0.00001382	57.77%
20	6.62	0.14	0.00004438	0.14	0.000021532	51.48%
25	5.88	0.42	0.00007725	0.42	0.00005718	25.98%
30	5.88	1.18	0.000270848	1.18	0.000169866	37.28%
35	15.66	26.32	0.00596966	24.88	0.00425216	28.77%
40	27.24	126.56	0.0283076	109.1	0.0191448	32.37%
45	240.38	2003.98	0.69751	1735.46	0.498841	28.48%
46	363.58	3714.18	1.17566	2995.9	0.763581	35.05%
47	474.72	3607.82	1.37158	3109.82	1.06459	22.38%
48	-	-	-	-	-	-

TABLE 3. Experimental results ($d_i = d_o = 0.08$).

PN Size $n=\|P\|=\|T\|$	Average number of siphons	LPMSE		Improved LPMSE		Percentage of CPU time reduced
		Average number of nodes	Average CPU time (s)	Average number of nodes	Average CPU time (s)	
10	4.18	0	0.000027626	0	0.000011142	59.67%
15	4.9	0.16	0.000043125	0.16	0.000022258	48.39%
20	4.68	0.26	0.000032983	0.26	0.000030819	6.56%
25	4.64	1.66	0.000176798	1.42	0.00012896	27.06%
30	13.56	47.08	0.00636905	34.72	0.0036855	42.13%
35	74.66	315.88	0.0592692	280.68	0.0426478	28.04%
40	1017.12	10623.7	2.95294	9667.12	2.8372	3.92%
45	11352	155279	215.175	132425	138.738	35.52%
46	-	-	-	-	-	-

TABLE 4. Experimental results ($d_i = d_o = 0.2$).

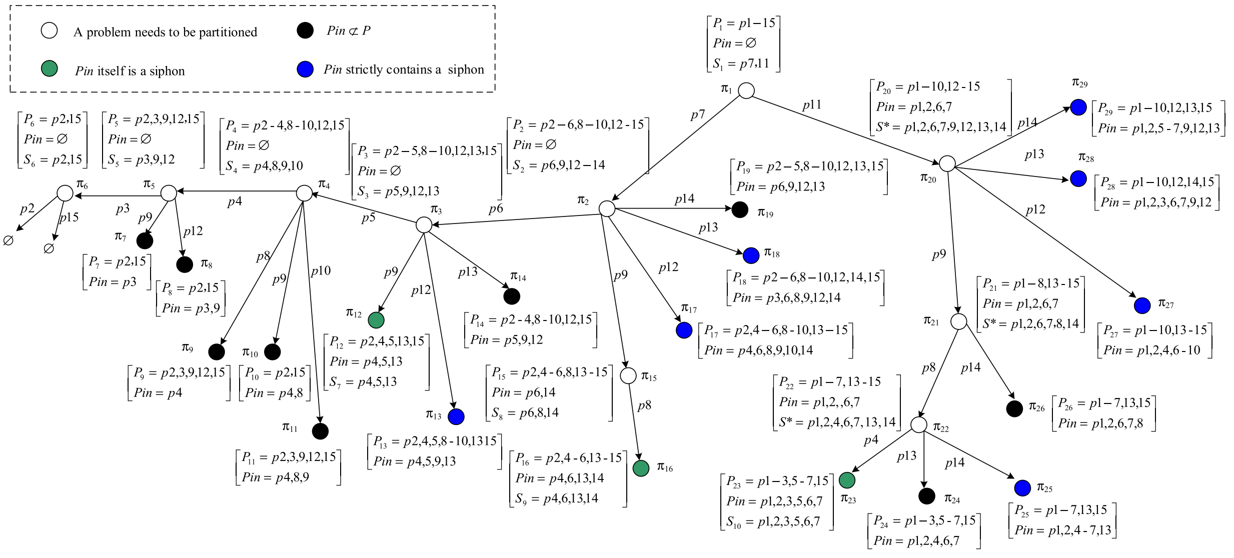
PN Size $n=\|P\|=\|T\|$	Average number of siphons	LPMSE		Improved LPMSE		Percentage of CPU time reduced
		Average number of nodes	Average CPU time (s)	Average number of nodes	Average CPU time (s)	
10	2.98	1.2	0.000052502	1.2	0.000048818	7.02%
15	13.6	26	0.00249879	24.5	0.00188927	24.39%
20	124.84	418.64	0.0448279	385.86	0.0325492	27.39%
25	1111.3	6623.52	0.777903	6167.34	0.600251	22.84%
30	9318.48	77365.4	12.2428	71938.5	12.2375	0.04%
31	-	-	-	-	-	-

First, Φ is initialized at $\emptyset$, representing the set of all minimal siphons in the PN Q. Since no source and sink node can be found in Q, after handling source places and calling *NetReduce*(Q), Φ is still empty and the resulting net is still Q. Then, we solve the problem $\pi_1 = (Q_1, \emptyset)$, where $Q_1 = Q$, by problem partitioning. The procedure of problem partitioning is detailed as follows and described by a tree as shown in Fig. 2(a).

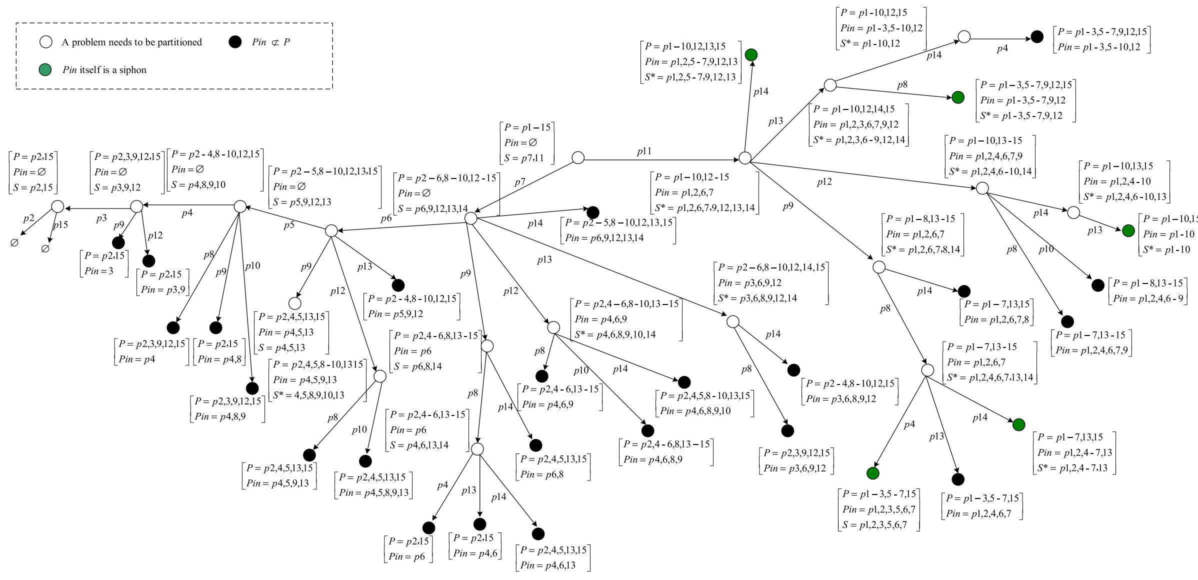
- We create the root node $\pi_1 = (Q_1, \emptyset)$. Note that, in Fig. 2 (a), we simply denote a PN by its set of places. By calling *GetaPinMiniSiphon* ($Q_1, \emptyset$), we then get a minimal siphon $S_1 = \{p_7, p_{11}\}$ since $P_{in} = \emptyset$. Thus, Φ is updated as $\{S_1\}$. Using the siphon S_1, we partition problem π_1 by calling *LocalPartition*(Q_1, $\emptyset$, S_1, Φ).

- Consider deleting place p_7 from Q_1. After applying function *NetReduce*, the resulting net is Q_2 with $P_2 = \{p_2$-p_6, p_8-p_{10}, p_{12}-$p_{15}\}$. After calling function *PinExpand*, we create node $\pi_2 = (Q_2, \emptyset)$. Since $P_{in} = \emptyset$, we get a minimal siphon $S_2 = \{p_6, p_9, p_{12}$-$p_{14}\}$. Then, Φ is updated as $\{S_1, S_2\}$ and S_2 is used to partition problem π_2 by calling *LocalPartition*(Q_2, $\emptyset$, S_2, Φ).

- Similarly, nodes π_3-π_6 are created one after another and Φ is updated as $\{S_1$-$S_6\}$. Since π_6 leads to empty nets by problem partitioning, we continue to consider the partitioning of problem π_5 by siphon $S_5 = \{p_3, p_9, p_{12}\}$. Now, we delete p_9 from Q_5 and require P_{in} to be $\{p_3\}$. By applying functions *NetReduce* and *PinExpand*, we create node $\pi_7 = \{Q_7, \{p_3\}\}$ with $P_7 = \{p_2, p_{15}\}$. Since $\{p_3\} \not\subset P_7$, we do not need



(a) A tree generated by improved LPMSE



(b) A tree generated by LPMSE [38]

FIGURE 2. Trees generated for the PN in Fig. 1.

to further partition π_7. Then, we consider delete p_{12} from Q_5 and require P_{in} to be $\{p_3, p_9\}$. Similarly, node π_8 is created and it does not need to be partitioned. By repeating the above operations, nodes π_9-π_{29} are created one after another, leading to the complete tree in Fig. 2(a), and we obtain the final set $\Phi = \{S_1\text{-}S_{10}\}$.

Recall that we terminate the further partitioning of a problem $\pi = \{Q, P_{in}\}$ in the case that $P_{in} \not\subset P$ or P_{in} contains a siphon. Here, to be intuitive, colored nodes denote problems that do not need to be partitioned. In more detail, node ● indicates that $P_{in} \not\subset P$, node ● indicates that P_{in} strictly contains a siphon, and node ● indicates that P_{in} itself is a siphon. Besides, note that not all the computed siphons are minimal siphons. Specifically, siphons found with respect to problems π_{20}-π_{22} are only P_{in}-minimal-siphons but not minimal siphons. Hence, they are only used for decomposing problems but are not added to Φ.

IV. COMPARISON

In this section, we show the performance of the improved LPMSE by comparing it with the original LPMSE [38].

A. COMPARISON VIA THE PN IN FIG. 1

In the last section, the improved LPMSE has been applied to the PN in Fig. 1 to enumeration minimal siphons, which leads to the tree in Fig. 2(a). Now, we apply the LPMSE to the PN in Fig. 1 as well, which results in the tree in Fig. 2(b). Clearly, the tree generated by the LPMSE is more complex than the one generated by the improved LPMSE. The detailed numerical results regarding these two trees are provided in Table 1. Note that the number of nodes in the tree and the number of nodes to be partitioned reflect the computational time of the method, while the maximum number of nodes to be saved in memory reflects the memory requirement of the method. We thus may conclude that the improved LPMSE consumes less time than LPMSE in this example but requires the same memory as LPMSE.

B. EXPERIMENT

To compare the improved LPMSE with LPMSE more evidently, we implement them by developing a tool in C++ and perform an experiment using the tool. The experiment is carried out on a 2.50GHz Intel(R) Core(TM) i5 computer with 4 GB of RAM and Windows 10 operating system.

In the experiment, we enumerate minimal siphons in a large quantity of PNs. Specifically, we consider PNs with different net size and randomly generate 50 nets for each size by setting d_i and d_o, which are the probabilities of having an arc from a place to a transition and having an arc from a transition to a place, respectively. The experimental results are presented in Tables 2-4, where PNs are randomly generated by setting $d_i = d_o = 0.05$, $d_i = d_o = 0.08$, and $d_i = d_o = 0.2$, respectively.

We can see that the improved LPMSE consumes less time than LPMSE. The percentage of CPU time reduced is around 20%-30% in many cases. Indeed, the time reduction percentage is not stable. This is because the computational time of the two methods is related to the structure of a PN to be computed. There exists randomness on the result even although we have 50 nets generated for each net size. On the other hand, due to the fact that the number of minimal siphons grows exponentially with respect to the net size for arbitrary PNs, both of LPMSE and the improved one are of exponential complexity with respect to the net size. Thus, they can hardly get a result within a reasonable time when the size of a PN is big. In Tables 2- 4, we do not present results for those cases for which the method consumes more than one hour but without a result gotten.

V. CONCLUSION

The computation of all minimal siphons in a PN is a difficult task since the number of siphons grows exponentially with the net size. To our best knowledge, the method of local partitioning minimal-siphon enumeration (LPMSE) proposed by Cordone et al. has so far been the one with the highest computational efficiency among all the existing methods that enumerate minimal siphons in an arbitrary PN. In this paper, we propose the improved LPMSE. As validated by the experimental results, the improved LPMSE performs better than LPMSE in terms of computational time.

REFERENCES

[1] T. Qin, L. Yin, N. Wu, and Z. Li, "Verification of current-state opacity in time labeled Petri nets with its application to smart houses," *IEEE Trans. Autom. Sci. Eng.*, pp. 1–13, Dec. 2023, doi: 10.1109/TASE.2023.3346523.

[2] N. Ran, J. Nie, A. Meng, and C. Seatzu, "Non-interference analysis of bounded Petri nets using basis reachability graph," *IEEE Trans. Autom. Control*, pp. 1–7, May 2024, doi: 10.1109/TAC.2024.3397695.

[3] Y. Tong, Z. Li, C. Seatzu, and A. Giua, "Verification of state-based opacity using Petri nets," *IEEE Trans. Autom. Control*, vol. 62, no. 6, pp. 2823–2837, Jun. 2017, doi: 10.1109/TAC.2016.2620429.

[4] S. Hu and Z. Li, "A digital twin approach for enforcing diagnosability in Petri nets," *IEEE Trans. Autom. Sci. Eng.*, pp. 1–13, Oct. 2023, doi: 10.1109/TASE.2023.3321781.

[5] N. Ran, T. Li, Z. He, and C. Seatzu, "Codiagnosability enforcement in labeled Petri nets," *IEEE Trans. Autom. Control*, vol. 68, no. 4, pp. 2436–2443, Apr. 2023.

[6] F. Basile, M. P. Cabasino, and C. Seatzu, "Diagnosability analysis of labeled time Petri net systems," *IEEE Trans. Autom. Control*, vol. 62, no. 3, pp. 1384–1396, Mar. 2017.

[7] D. You, S. Wang, and C. Seatzu, "Verification of fault-predictability in labeled Petri nets using predictor graphs," *IEEE Trans. Autom. Control*, vol. 64, no. 10, pp. 4353–4360, Oct. 2019, doi: 10.1109/TAC.2019.2897272.

[8] N. Ran, J. Hao, and C. Seatzu, "Prognosability analysis and enforcement of bounded labeled Petri nets," *IEEE Trans. Autom. Control*, vol. 67, no. 10, pp. 5541–5547, Oct. 2022.

[9] M. Bashir, J. Zhou, and B. B. Muhammad, "Optimal supervisory control for flexible manufacturing systems model with Petri nets: A place-transition control," *IEEE Access*, vol. 9, pp. 58566–58578, 2021.

[10] Z. Li, N. Wu, and M. Zhou, "Deadlock control of automated manufacturing systems based on Petri nets—A literature review," *IEEE Trans. Syst., Man, Cybern. C, Appl. Rev.*, vol. 42, no. 4, pp. 437–462, Jul. 2012.

[11] C. Chen, A. Raman, H. Hu, and R. S. Sreenivas, "On liveness enforcing supervisory policies for arbitrary Petri nets," *IEEE Trans. Autom. Control*, vol. 65, no. 12, pp. 5236–5247, Dec. 2020.

[12] K. Xing, F. Wang, M. C. Zhou, H. Lei, and J. Luo, "Deadlock characterization and control of flexible assembly systems with Petri nets," *Automatica*, vol. 87, pp. 358–364, Jan. 2018.

[13] N. Ran, T. Li, S. Wang, and Z. He, "Supervisor synthesis for Petri nets with uncontrollable and unobservable transitions," *IEEE Trans. Autom. Sci. Eng.*, vol. 21, no. 2, pp. 1517–1525, Feb. 2024.

[14] B. Yang and H. Hu, "Maximally permissive deadlock and livelock avoidance for automated manufacturing systems via critical distance," *IEEE Trans. Autom. Sci. Eng.*, vol. 19, no. 4, pp. 3838–3852, Oct. 2022.

[15] H. Dou, M. Zhou, S. Wang, and A. Albeshri, "An efficient liveness analysis method for Petri nets via maximally good-step graphs," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 54, no. 7, pp. 3908–3919, Jul. 2024.

[16] Y. Liu, X. Li, and Z. Li, "Control strategy of discrete event systems modeled by labeled Petri nets based on transition priority," *IEEE Access*, vol. 11, pp. 45442–45455, 2023.

[17] C. Gu, Z. Ma, Z. Li, and A. Giua, "Verification of nonblockingness in bounded Petri nets with min-max basis reachability graphs," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 52, no. 10, pp. 6162–6173, Oct. 2022.

[18] Y. Chen, Z. Li, and K. Barkaoui, "Maximally permissive liveness-enforcing supervisor with lowest implementation cost for flexible manufacturing systems," *Inf. Sci.*, vol. 256, pp. 74–90, Jan. 2014.

[19] L. Qi, Y. Su, M. Zhou, and A. Abusorrah, "A state-equation-based backward approach to a legal firing sequence existence problem in Petri nets," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 53, no. 8, pp. 4968–4979, Aug. 2023, doi: 10.1109/TSMC.2023.3241101.

[20] Y. Su, L. Qi, and M. Zhou, "A backward algorithm to determine the existence of legal firing sequences in ordinary Petri nets," *IEEE Robot. Autom. Lett.*, vol. 8, no. 6, pp. 3190–3197, Jun. 2023.

[21] Y. Hou and K. Barkaoui, "Deadlock analysis and control based on Petri nets: A siphon approach review," *Adv. Mech. Eng.*, vol. 9, no. 5, May 2017, Art. no. 168781401769354.

[22] S. Wang, X. Guo, O. Karoui, M. Zhou, D. You, and A. Abusorrah, "A refined siphon-based deadlock prevention policy for a class of Petri nets," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 53, no. 1, pp. 191–203, Jan. 2023.

[23] A. Al-Ahmari, H. Kaid, Z. Li, and R. Davidrajuh, "Strict minimal siphon-based colored Petri net supervisor synthesis for automated manufacturing systems with unreliable resources," *IEEE Access*, vol. 8, pp. 22411–22424, 2020.

[24] X. Fan, B. Yang, and H. Hu, "Event circular waits and their analysis via Petri nets," *IEEE Access*, vol. 9, pp. 92586–92599, 2021.

[25] C. Chen and H. Hu, "Extended place-invariant control in automated manufacturing systems using Petri nets," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 52, no. 3, pp. 1807–1822, Mar. 2022.

[26] N. Du and H. Hu, "Robust deadlock detection and control of automated manufacturing systems with multiple unreliable resources using Petri nets," *IEEE Trans. Autom. Sci. Eng.*, vol. 18, no. 4, pp. 1790–1802, Oct. 2021.

[27] G. Liu, L. Zhang, L. Chang, A. Al-Ahmari, and N. Wu, "Robust deadlock control for automated manufacturing systems based on elementary siphon theory," *Inf. Sci.*, vol. 510, pp. 165–182, Feb. 2020.

[28] Y. Feng, K. Xing, M. Zhou, and H. Liu, "Liveness analysis and deadlock control for automated manufacturing systems with multiple resource requirements," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 50, no. 2, pp. 525–538, Feb. 2020.

[29] Y. Feng, M. Zhou, F. Tian, C.-B. Yan, and K. Xing, "Deadlock prevention controller for automated manufacturing systems modeled by S4PR," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 51, no. 12, pp. 7403–7412, Dec. 2021.

[30] H. Liu, Y. Feng, J. Li, and J. Luo, "Robust Petri net controllers for flexible manufacturing systems with multitype and multiunit unreliable resources," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 53, no. 3, pp. 1431–1444, Mar. 2023.

[31] G. Liu and K. Barkaoui, "A survey of siphons in Petri nets," *Inf. Sci.*, vol. 363, pp. 198–220, Oct. 2016.

[32] S. Wang, W. Duo, X. Guo, X. Jiang, D. You, K. Barkaoui, and M. Zhou, "Computation of an emptiable minimal siphon in a subclass of Petri nets using mixed-integer programming," *IEEE/CAA J. Autom. Sinica*, vol. 8, no. 1, pp. 219–226, Jan. 2021.

[33] Y. Chen and G. Liu, "Computation of minimal siphons in Petri nets by using binary decision diagrams," *ACM Trans. Embedded Comput. Syst.*, vol. 12, no. 1, pp. 1–15, Jan. 2013.

[34] P. H. Starke. (2003). *INA: Integrated Net Analyzer*. [Online]. Available: http://www2.informatik.hu-berlin.de/ starke/ina.html

[35] L. Piroddi, R. Cordone, and I. Fumagalli, "Combined siphon and marking generation for deadlock prevention in Petri nets," *IEEE Trans. Syst., Man, Cybern. A, Syst. Humans*, vol. 39, no. 3, pp. 650–661, May 2009.

[36] 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. 431–445, Jul. 2008.

[37] X. Han, Z. Chen, Z. Liu, and Q. Zhang, "Calculation of siphons and minimal siphons in Petri nets based on semi-tensor product of matrices," *IEEE Trans. Syst., Man, Cybern., Syst.*, vol. 47, no. 3, pp. 531–536, Mar. 2017.

[38] 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.

[39] S. G. Wang, Y. Li, C. Y. Wang, and M. C. Zhou, "Computation of all minimal siphons in Petri nets," in *Proc. 9th IEEE Int. Conf. Netw., Sens. Control*, Apr. 2012, pp. 46–51.

[40] S. Wang, C. Wang, M. Zhou, and Z. Li, "A method to compute strict minimal siphons in a class of Petri nets based on loop resource subsets," *IEEE Trans. Syst., Man, Cybern. A, Syst. Humans*, vol. 42, no. 1, pp. 226–237, Jan. 2012.

[41] Z. Li and M. 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, May 2008.

[42] A. Wang, Z. Li, J. Jia, and M. Zhou, "An effective algorithm to find elementary siphons in a class of Petri nets," *IEEE Trans. Syst., Man, Cybern. A, Syst. Humans*, vol. 39, no. 4, pp. 912–923, Jul. 2009.

[43] E. E. Cano, C. A. Rovetto, and J.-M. Colom, "An algorithm to compute the minimal siphons in s 4 PR nets," *Discrete Event Dyn. Syst.*, vol. 22, no. 4, pp. 403–428, Dec. 2012.

[44] S. Wang, D. You, and M. Zhou, "A necessary and sufficient condition for a resource subset to generate a strict minimal siphon in s 4PR," *IEEE Trans. Autom. Control*, vol. 62, no. 8, pp. 4173–4179, Aug. 2017.

[45] F. Tricas and J. Ezpeleta, "Computing minimal siphons in Petri net models of resource allocation systems: A parallel solution," *IEEE Trans. Syst., Man, Cybern. A, Syst. Humans*, vol. 36, no. 3, pp. 532–539, May 2006.

[46] 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," in *Proc. 18th Int. Conf. Syst. Theory, Control Comput. (ICSTCC)*, Oct. 2014, pp. 645–651.

[47] D. You, O. Karoui, and S. Wang, "Computation of minimal siphons in Petri nets using problem partitioning approaches," *IEEE/CAA J. Autom. Sinica*, vol. 9, no. 2, pp. 329–338, Feb. 2022.

JUAN PAN received the bachelor's degree in electronic information science and technology from the School of Information Science and Engineering, Ningbo University, in 2013, and the master's degree in information and communication engineering from the School of Information Engineering, Zhejiang University of Technology, in 2018.

She is currently a member of the Discrete-Event System Group, School of Information and Electronic Engineering, Zhejiang Gongshang University. Her research interests include deadlock control and siphon computation in Petri nets.

DAN YOU (Member, IEEE) received the B.S. degree in electronic and information engineering and the M.S. degree in information and communication engineering from the School of Information and Electronic Engineering, Zhejiang Gongshang University, Hangzhou, China, in 2014 and 2017, respectively, and the Ph.D. degree in electronic and computer engineering from the Department of Electrical and Electronic Engineering, University of Cagliari, Italy, in 2021.

She is currently an Associate Research Professor with the School of Information and Electronic Engineering, Zhejiang Gongshang University. Her research interests include supervisory control of discrete event systems, fault prediction, and deadlock control and siphon computation in Petri nets.