On Achieving Reachability Paths of Petri nets

Hanife Apaydın Özkan

Department of Electrical and Electronics Engineering, Anadolu University, Eskişehir, Turkey
hapaydin1@anadolu.edu.tr

Abstract

Petri net is a powerful graphical and mathematical tool for analysis and design of discrete event systems. This paper focuses on the reachability path problems in Petri nets. Thereby, two algorithms are developed to create the set of minimal paths and the shortest path to lead the system from the given initial state to a desired state. Both of them are enlightened by dynamic programming approach; that is to say, they are backward techniques. Proposed algorithms do not deal with the reachability tree or graph of the net under analysis and use memory only for storing the obtained paths unlike the approaches based on the reachability tree. Moreover, the algorithms can be applied to general Petri nets without any restriction.

1. Introduction

Petri nets are frequently used for modeling and analysis of Discrete Event Systems (DES)s such as communication protocols, manufacturing systems, transport systems and others [1,2,3,4].

In most cases, it is interested in driving a system from a given initial state to a desired state, and obtaining the required operation sequence. In terms of Petri nets this type of problems are called as *reachability path problem*s, such as minimal path problem and shortest path problem. Obtaining the paths that do not go through the same state multiple times is called as *minimal path problem*. For minimizing the cost, it is aimed to reach the desired state by minimum number of operations. For Petri nets, this problem is called as *shortest path problem* which aims to compute the shortest transition sequence, i.e., transition sequence with the minimum number of transitions, firing which from the initial state drives the system to the desired state.

Reachability analysis techniques and reachability path problems are strictly connected, since the reachability analysis techniques may also be utilized to solve reachability path problems. Two main reachability analysis techniques of Petri nets are reachability tree approach which is obtained by enumerating all reachable markings and the matrix equation approach. But, constructing the reachability tree and investigating the paths to a desired state would be rather difficult task. The main drawback in the use of matrix equation approach is that the solution of the fundamental equation does not give any information about the order of firings.

Many works have been presented to contribute the reachability analysis and the reachability path problems. The author in [7] deals shortest path problem directly and the length of the shortest path is obtained for some subclasses of Petri nets in that work. But the transition sequence of the corresponding path is not referred. In [5], Petri nets without transition invariants are studied, while only strictly monotone Petri nets are considered in [6]. In [3], reachability condition of some subclasses of Petri nets is exhibited. Although, these works upgrade the reachabiliy analysis techniques for some subclasses, their complexity and expansions are still open.

In this paper, two algorithms are developed to solve minimal paths and shortest path problems. Both of them are enlightened by dynamic programming approach, thus they are backward techniques. Given an initial state and a desired state, the first algorithm obtains the shortest path while the second one results in all minimal paths driving the system from the initial state to the desired state.

The main contributions of this paper are as follow: The proposed algorithms do not deal with the reachability tree or reachability graph of the net under analysis and use memory only for storing the obtained path as distinct from the approaches based on reachability tree. Moreover, developed algorithms can be applied to general Petri nets without any restriction.

2. Petri nets

This section provides background on Petri nets related to the discussion of this work and presents an introductory example. The readers are recommended to see [2,3] for more detailed information.

Petri net is a tuple $\mathcal{N} = \langle P, T, \boldsymbol{N}, \boldsymbol{O} \rangle$, where P is the set of *places*, T is the set of transitions, $\boldsymbol{N} : P \times T \to \mathcal{N}$ is the *input matrix* that specifies the weights of arcs directed from places to transitions, $\boldsymbol{O} : P \times T \to \mathcal{N}$ is the *output matrix* that specifies the weights of arcs directed from transitions to places. Here, $\mathcal{N}$ is the set of non-negative integer numbers.

$M : P \to \mathcal{N}$ is a *marking vector* (or *marking*), $M(p)$ indicates the number of *tokens* assigned by marking M to place $p \in P$, and the *initial marking* of the system is denoted by m_0. A transition $t \in T$ is *enabled* if and only if

$$M(p) \geq N(p,t) \quad \forall p \in P. \tag{1}$$

which is called as *enability condition*. The set of transitions which are enabled at a marking M is denoted by $\mathcal{E}(G, M)$.

A *transition sequence g* is defined as firing sequence of enabled transitions $t_1 t_2 \dots t_k$, where $t_1, t_2, \dots, t_k \in T$. A marking M' is said to be reachable from M if there exists a transition sequence $g = t_{i_1}, t_{i_2}, \dots \; 1 \leq i_k \leq |T|, k = 1, 2, \dots$ which can be fired starting from M (i.e., the first transition of the sequence fires at M) and yielding M' (i.e., the final transition of the sequence yields M') according to the following so-called *state equation*:

$$M' = M + C\sigma_g \tag{2}$$

This work was supported by Anadolu University under the project 1404F185.

where, $\boldsymbol{C} := \boldsymbol{O} - \boldsymbol{N}$ denotes the incidence matrix, $\sigma_g : T \to \mathcal{Z}$ denotes the firing count vector whose j^{th} element indicates how many times t_j is fired in the transition sequence g. $M \xrightarrow{g} M'$ denotes that the marking M' is reachable from a marking M by firing the transition sequence sequence g. The set denoted by $R(G,M)$ is the set of all markings reachable from M. The set of all reachable markings from m_0 is called reachability set of Petri net and denoted by $R(G,m_0)$.

If $\exists y \geq 0$ such that $y \cdot C = 0$ then, y is called as P-semiflow of the net and every marking M reachable from m_0 satisfies:

$$y \cdot M = y \cdot m_0 \tag{3}$$

This provides *a token balance law* which is the necessary reachability condition.

For a given Petri net system, as many *new* markings as the number of the enabled transitions can be obtained from the initial marking. From each new marking, more new markings can be reached until repeated nodes are encountered (*old*) or no transitions are enabled (*dead-end*). This process yields tree representation of the evaluation of the system which is called as *reachability tree*. In the reachability tree each node represents a marking and the firing of a transition transforming one marking to another is represented by arcs.

A reachability graph of a PN is a directed graph $G = (R(G,m_0),E)$, where $e \in E$ represents a directed arc from a class of markings to the other class of markings. The reachability graph demonstrates a better performance than the reachability tree [8].

Example 1: Let us consider the PN in Fig. 1 with $P = \{p_1, p_2, p_3, p_4, p_5, p_6, p_7\}$, $T = \{t_1, t_2, t_3, t_4, t_5, t_6\}$ and $m_0 = [1\ 0\ 0\ 1\ 1\ 0\ 0]^T$. P-semiflows of this net are $\mathcal{P}_1 = [1\ 1\ 1\ 0\ 0\ 0\ 0]$, $\mathcal{P}_2 = [0\ 0\ 1\ 1\ 0\ 0\ 1]$ and $\mathcal{P}_3 = [0\ 0\ 0\ 0\ 1\ 1\ 1]$.

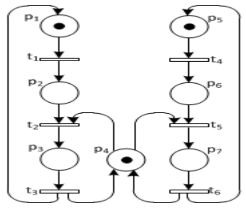

Figure 1. An example Petri net.

The reachability graph for this PN system is given in Fig.2. where reachable markings are denoted by $m_0 = M_1 = [1\ 0\ 0\ 1\ 1\ 0\ 0]^T$, $M_2 = [0\ 1\ 0\ 1\ 1\ 0\ 0]^T$, $M_3 = [1\ 0\ 0\ 1\ 0\ 1\ 0]^T$, $M_4 = [0\ 0\ 1\ 0\ 1\ 0\ 0]^T$, $M_5 = [0\ 1\ 0\ 1\ 0\ 1\ 0]^T$, $M_6 = [1\ 0\ 0\ 0\ 0\ 0\ 1]^T$, $M_7 = [0\ 0\ 1\ 0\ 0\ 1\ 0]^T$, $M_8 = [0\ 1\ 0\ 0\ 0\ 0\ 1]^T$. It is possible to obtain all reachable markings and to get corresponding transition sequences to reach these markings from the reachability tree by enumeration. That is, firing the transition sequence "$g = t_4t_5$"from m_0 drives the system to the state M_6. Unfortunatelly, for Petri nets with great number of reachable states, using reachability graph to find the paths driving the system from the initial marking to desired marking is an exhausting way.

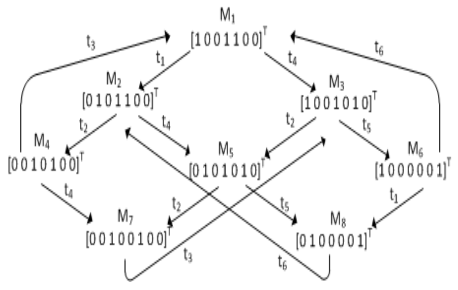

Figure 2. Reachability graph.

3. Reachability Path Problems:Shortest Path&Minimal Paths

In this work, the main goal is to develop efficient techniques to solve reachability path problems of general Petri nets. Firstly, we give more formal definitions for the reachability path problems considered in this work.

Definition 3.1 (Shortest Path). *Let $(\mathcal{N},m_0)$ be a Petri net system and $M_d \in R(G,m_0)$ a marking. The transition-marking sequence leading m_0 to M_d while including minimum number of transition firing is called as shortest path.*

Definition 3.2 (Minimal Path). *Let $(\mathcal{N},m_0)$ be a Petri net and $M_d \in R(G,m_0)$ a marking. The transition-marking sequence leading m_0 to M_d without passing through the same marking multiple times is called as minimal path.*

In order to solve reachability path problems above, an exhausting approach may be composing reachability tree. But this is time consuming method especially for big sized Petri nets. The proposed backward method is introduced in the following part.

3.1. Backward Method

According to the proposed backward method, for a Petri net system $(\mathcal{N},m_0)$ the transition sequence driving the system from m_0 to $M' \in R(G,M)$ will be attained by obtaining the fired transitions one by one starting from M_d and going back to m_0.

If a transition $t_j \in T$ is fired at a marking M yielding M', element-wise representation of the state equation can be written as

$$M'(p_i) = M(p_i) + O(p_i,t_j) - N(p_i,t_j), \ \forall p_i \in P \tag{4}$$

yielding

$$M'(p_i) - O(p_i,t_j) = M(p_i) - N(p_i,t_j), \ \forall p_i \in P \tag{5}$$

Necessary condition for transition t_j to be enabled at a marking M yielding M' can be reformulated in terms of M' by combining equation (5) and enabling condition in (1) together:

$$M'(p_i) - O(p_i,t_j) \geq 0, \ \forall p_i \in \bullet t_j \tag{6}$$

where $\bullet t_j$ denotes the set of all places from which there exists a directed arc to t_j, i.e., $\bullet t_j = \{p | N(p,t_j) \geq 1, \ t_j \in T\}$

From a marking $M \in R(G,m_0)$, firing any transition $t_j \in T$ which satisfies the condition in (1) may yield M'. Corresponding M is obtained by solving the following equation system:

$$\begin{aligned} M &= M' - O(:,t_j) + N(:,t_j) \quad &(a) \\ B^T M &= B^T M' &(b) \end{aligned} \tag{7}$$

where equation (7)(a) is the state equation and equation (7)(b) corresponds to reachability of M from m_0, i.e. $M \in R(G,m_0)$ [3].

Algorithm 1 goes through with the proposed backward method to achieve the shortest path from the initial marking m_0 to the desired marking M_d. It starts from M_d, and constructs the *Path* by going backward until m_0 is reached. *Path* is composed of the ordered set of couples (called node) of state M on the path from m_0 to M_d and the transition $t \in T$ fired from M. Hence, the node of M for the operation $M \xrightarrow{t} M'$ is (M,t). Note that, j^{th} ($j \in \{1,2\}$) component of i^{th} node of *Path* is denoted by $Path_{ij}$ and the number of states in a set $*$ is denoted by $|*|$. In Algorithm 1, the 1^{st} component of the last node of *Path* ($Path_{|Path|_1}$) is taken as the present marking M'. Then for each transition, firing which from a preceding marking may yield M' (thus, for each transition satisfying condition in (6)), corresponding preceding marking, M, is calculated. Hereby, all potential preceding markings of the present marking M' are obtained.

The set which is called as *Candidates* is constructed by the nodes of M''s potential preceding markings, which are reachable from m_0 and not added to the sets *Path* or *Forbidden* previously. If *Candidates* is nonempty, index function (which returns the index of the node whose distance to m_0 is minimum through the set *Candidates*) is called to place most promising one into *Path*. Otherwise, the algorithm removes the node of the present marking M' from *Path* and adds M' to the set named *Forbidden*. Then it repeats the previous step again to choose another present marking M'. This procedure proceeds until the node of m_0 is put in *Path*.

Algorithm 1 Shortest Path

```
Input: $(\mathcal{N}, m_0), M_d$
Path $= \{(M_d, 0)\}$
$M' = M_d$
do-while $Path_{|Path|_1} \neq m_0$
  $Candidates = \emptyset$
  for $j = 1 : |T|$
    if $M'(p_i) - O(p_i,t_j) \geq 0, \ \forall p_i \in \bullet t_j$
      $M = M' - O(:,t_j) + N(:,t_j)$
      if $B^T M' = B^T M$ & $M' \notin Path$ & $M' \notin Forbidden$
        $Candidates = Candidates \cup (M,t_j)$
      end-if
    end-if
  end-for
  if $Candidates \neq \emptyset$
    $i$=Index($Candidates$)
    $Path = Path \cup Candidates_i$
    $M' = Path_{|Path|_1}$
  else
    $Forbidden = Forbidden \cup M'$
    $Path = Path \backslash Path_{|Path|}$
    $M' = Path_{|Path|_1}$
end-while
Output: $Path$
```

Backward approach can also be used to find all minimal paths from m_0 to M_d. Algorithm 2 carries out the corresponding procedure. In the algorithm, *Pathset* is composed of the set

Index function

```
Input: $Candidates$=$\{(M^1,t^1),...,(M^{|Path|},t^{|Path|})\}$
index=1
for $i = 1 : |Candidates|$
  if $|M^i - m_0| < |M^{index} - m_0|$
    index=i
  end-if
end-for
Output: index
```

of uncompleted paths starting from M_d and being constructed by going backward to m_0. As in Algorithm 1, each path is represented by the ordered set of couples of state $M \in R(G,m_0)$ and the fired transition $t \in T$ from M. Recall that, the node of M for this operation (i.e. $M \xrightarrow{t} M'$) is (M,t)

$Pathset_{ij_k}$ is the k^{th} ($k \in \{1,2\}$) component of the j^{th} node of the i^{th} path in *Pathset*. In Algorithm 2, an arbitrarily chosen path is removed from *Pathset* and assigned to a temporary path *TPath*. For the 1^{st} component of the last node of *TPath*, i.e. $M' = TPath_{|TPath|_1}$, all enabled transitions to reach M' are obtained. Then for each of these transitions the preceding marking, M, from which the transition is fired and M' is reached, is calculated. In the case that M is reachable from m_0 and it is not previously added to *Tpath*, then the node of M is added to *TPath*. If *TPath* is completed, i.e. $M = m_0$, *TPath* is added to the set of completed paths, *FPathSet*, otherwise it is added to the set of uncompleted paths, *PathSet*.

This process is continued until all candidate paths is completed by obtaining all nodes on the considered path from m_0 to M_d, that is *Pathset* is emptied.

Algorithm 2 Minimal Paths

```
Input: $(\mathcal{N}, m_0), M_d$
$PathSet = \{\{(M_d, 0)\}\}$
$FPathSet = \emptyset$
do-while $PathSet \neq \emptyset$
  choose $i \in \{1,2,...,|PathSet|\}$
  $TPath = PathSet_i$
  $PathSet = PathSet \backslash PathSet_i$
  $M = TPath_{|TPath|_1}$
  for $j = 1 : |T|$
    if $M(p_i) - O(p_i,t_j) \geq 0, \ \forall p_i \in \bullet t_j$
      $M' = M - O(:,t_j) + N(:,t_j)$
      if $B^T M' = B^T M$ & $M' \notin TPath$
        if $M' = m_0$
          $FPathSet = FPathSet \cup \{TPath \cup (M',t_j)\}$
        end-if
        $PathSet = PathSet \cup \{TPath \cup (M',t_j)\}$
      end-if
    end-if
  end-for
end-while
Output: $FPathSet$
```

Example 2: Let us consider the PN system considered in Example 1 again with desired target marking $M_d = M_5$. In order to obtain the shortest path from m_0 to M_d Algorithm 1 is executed, which processes as follow: At M_d the transitions satisfying the condition in (6) are t_2 and t_4. For transition t_4, the preceding marking is obtained as M_2 (i.e. $M_2 \xrightarrow{t_4} M_d$);

for transition t_2, the preceding marking is obtained as M_3 (i.e. $M_3 \xrightarrow{t_2} M_d$). Since both M_2 and M_3 are reachable from m_0 (they satisfy the condition in (7)b) and they have not already been added to the *Path*, $Candidates = \{(M_2,t_4),(M_3,t_2)\}$. Index function is called to choose the closest one to m_0 through the set *Candidates*. Since their distance from m_0 are same, either of them is added to *Path*, i.e. *Path*=$\{(M_d,0),(M_2,t_4)\}$. Then the procedure is repeated for the marking of the last node of *Path*, i.e. M_2. The previously firable transitions and corresponding preceding markings forms new candidate nodes. Hence, this once *Candidates*=$\{(M_1,t_4),(M_8,t_6)\}$. The minimum distance node through the candidate nodes is (M_1,t_4) and added to *Path*. Since $m_0 = M_1$ the *Path* is completed as *Path*=$\{(M_d,0),(M_2,t_4),(m_0,t_1)\}$ and the algorithm terminates. That is, the shortest path driving the system from m_0 to M_d is obtained as $m_0 \xrightarrow{t_1} M_2 \xrightarrow{t_4} M_d$

For the same system when Algorithm 2 is executed, the set of minimal paths, that is *Pathset* is obtained as follow:

$$\begin{aligned} Pathset = \{\ & \{(M_d,0),(M_2,t_4),(M_1,t_1)\}, \\ & \{(M_d,0),(M_2,t_4),(M_8,t_6),(M_6,t_1), \\ & \quad (M_3,t_5),(M_1,t_4)\}, \\ & \{(M_d,0),(M_2,t_4),(M_8,t_6),(M_6,t_1), \\ & \quad (M_3,t_5),(M_7,t_3),(M_4,t_4),(M_2,t_2), \\ & \quad (m_0,t_1)\}, \\ & \{(M_d,0),(M_3,t_2),(M_1,t_4)\}, \\ & \{(M_d,0),(M_3,t_2),(M_7,t_3),(M_4,t_4), \\ & \quad (M_2,t_2),(m_0,t_1)\}, \\ & \{(M_d,0),(M_3,t_2),(M_7,t_3),(M_4,t_4), \\ & \quad (M_2,t_2),(M_8,t_6),(M_6,t_1),(M_3,t_5), \\ & \quad (M_1,t_4)\}\ \} \end{aligned} \quad (8)$$

Example 3: Let us consider the Petri net system in Fig.3 taken from [3], which models a multiprocessor system. In the PN model, place p_1 contains tokens which represent the active processors whereas tokens inside p_2 represent the available buses; transition t_1 represent the sending of a request of access and place p_3 contains the requests which have not been still served. Tokens inside p_4 represent processors which are accessing to a memory whereas tokens inside p_5 represent processors which are waiting for the memory occupied by tokens inside p_4. The firing of transition t_5 models the end of the access to the memory which is requested by tokens inside p_5; the firing of t_4, instead, represents the end of the access to a memory there are no more requests. Transitions t_2 and t_3 model the two possible choiches of memory: firing t_3 means to choose the memory which is at the moment utilized by processor in p_4 whereas firing t_2 means the choice of any other memory.

Suppose that at the initial state there are two active processors and two available buses (i.e. $m_0 = [2\ 2\ 0\ 0\ 0]^T$). It is desired to reach the state where there are one active processor, one bus while one processor is accessing to the memory, i.e. $M_d = [1\ 1\ 0\ 1\ 0]^T$. By using Algorithm 1, the shortest Path to reach $M_d = [1\ 1\ 0\ 1\ 0]^T$ from $m_0 = [2\ 2\ 0\ 0\ 0]^T$ is obtained as $m_0 \xrightarrow{t_1} [1\ 2\ 1\ 0\ 0]^T \xrightarrow{t_2} [1\ 1\ 0\ 1\ 0]$.

All possible paths from $m_0 = [2\ 2\ 0\ 0\ 0]^T$ to $M_d = [0\ 1\ 1\ 1\ 0\ ]^T$ can be obtained by Algorithm 2 which yields 4 minimal paths. One of these paths is:

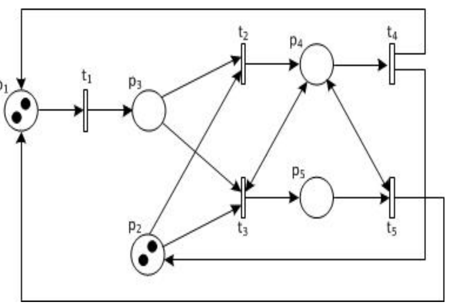

Figure 3. An example Petri net [3].

$[2\ 2\ 0\ 0\ 0]^T \xrightarrow{t_1} [1\ 2\ 1\ 0\ 0]^T \xrightarrow{t_1} [0\ 2\ 2\ 0\ 0] \xrightarrow{t_2} [0\ 1\ 1\ 1\ 0] \xrightarrow{t_3} [0\ 1\ 0\ 1\ 1] \xrightarrow{t_5} [1\ 1\ 0\ 1\ 0]$.

Note that both of the proposed algorithms can be used to obtain paths leading the systems back to m_0 from m_0, if it is possible. For example, in order to find the shortest path from m_0 to m_0, Algorithm 1 is executed with $M_d = m_0$, and it turns out the shortest path as $m_0 \xrightarrow{t_1} [1\ 2\ 1\ 0\ 0]^T \xrightarrow{t_2} [1\ 1\ 0\ 1\ 0] \xrightarrow{t_4} m_0 = [2\ 2\ 0\ 0\ 0]^T$.

4. Conclusion

In this work, two algorithms are developed to contribute reachability problems of Petri nets. Both of them deal with the paths from m_0 to mf while Minimal Paths Algorithm obtains all the paths which do not pass through the same marking and Shortest Path Algorithm obtains the shortest path which includes minimum number of transition to reach m_f. The proposed algorithms are backwars techniques and do not deal with the reachability tree or graph of the net under analysis and use memory only for storing the obtained path as distinct from the approaches based on reachability tree. Moreover, the algorithms can be applied to general Petri nets without any restriction.

Future research directions include using integer and mixed-integer programming techniques for reachability path problems and comparing complexity and computational time with the present methods.

5. References

[1] Desrochers, A. A. and Al-Jaar, R. Y., “Applications of Petri Nets in Manufacturing Systems”, *The Institute of Electrical and Electronics Engineers Inc.,* New York, 1995.

[2] Proth, J. and Xie, X., “Petri Nets: A Tool for Design and Management of Manufacturing Systems”, *John Wiley & Sons, West Sussex,* New York, 1996.

[3] Murata, T., “Petri nets: properties, analysis and applications”, *Proceedings of the IEEE*, vol.77, no.4, pp:541-580, 1989.

[4] Zhou, M.C., Dicesare, F. and Guo, D.L., “Modeling and performance analysis of a resource-sharing manufacturing system using stochastic Petri nets”, in *IEEE Symp. on Intelligent Control*, Phildephia, PA, 1990, pp. 1005-1010.

[5] Kostin, A., “Reachability analysis in t-invariant-less Petri nets”, *IEEE Transactions on Automatic Control*, vol.48, no.6, pp: 1019-1024, 2003.

[6] Ru, Y. and C.N.Hadjicostis, "Reachability analysis for a class of Petri nets", in *Proceedings of IEEE Conference on Decision and Control*, Shanghai, 2009, pp. 1261-1266.

[7] Desel, J., "Shortest paths in reachability graphs", *Journal of Computer and System Sciences*, vol.51, pp: 314-323, 1995.

[8] X. Ye, J. Z. and Song, X., "On reachability graphs of Petri nets", *Computers and Electrical Engineering*, vol.29, no.2, pp:263-272, 2003.