2009 Ninth International Conference on Intelligent Systems Design and Applications

# Efficient Computation Methods for the Kleene Star in Max-Plus Linear Systems

Hiroyuki Goto
*Department of Management and Information Systems Science*
*Nagaoka University of Technology*
*Nagaoka, Niigata, 940-2188 Japan*
*e-mail: hgoto@kjs.nagaokaut.ac.jp*

Munenori Kasahara
*Faculty of Management and Information Systems Engineering*
*Nagaoka University of Technology*
*Nagaoka, Niigata, 940-2188 Japan*
*e-mail: s063349@ics.nagaokaut.ac.jp*

***Abstract*—This research proposes efficient calculation methods for the transition matrices in discrete event systems, where the adjacency matrices are represented by directed acyclic graphs. The essence of the research focuses on obtaining the Kleene Star of an adjacency matrix. Previous studies have proposed methods for calculating the longest paths focusing on destination nodes. However, in these methods the chosen algorithm depends on whether the adjacency matrix is sparse or dense. In contrast, this research calculates the longest paths focusing on source nodes. The proposed methods are more efficient than the previous ones, and are attractive in that the efficiency is not affected by the density of the adjacency matrix.**

***Keywords*-max-plus algebra; state-space representation; adjacency matrix; directed acyclic graph; Kleene star;**

## I. Introduction

This research proposes two efficient computation methods for the representation matrix of the state equation, frequently used as an approach in scheduling problems for a certain class of discrete event system. The focused systems have the following features: (1) parallel execution of multiple processes, (2) synchronization of multiple processes, and (3) no concurrency in internal resources. We focus on discrete event systems in which the execution sequences of processes can be represented by Directed Acyclic Graphs (DAG). The state changes in this kind of system can be represented by simple linear equations, referred to as the state equation in max-plus algebra [1], [2]. It is often used in solvers for scheduling problems in manufacturing systems [3], transportation systems [4], project management [5], etc.

The bottleneck in these systems and other relevant research is in computing the representation matrix referred to as the transition matrix in the state equation. The matrix represents the event propagation times in the system, and can be calculated by applying the 'Kleene Star' to the adjacency matrix that regulates the execution sequences and the times of the processes. Let there be $n$ nodes in a graph describing the execution sequences of processes. If we use the naïve definition, the time complexity is $\mathrm{O}(n^4)$. Accordingly, if $n$ is large, a great deal of time is required to compute the transition matrix and state equation. In the light of this, we previously proposed efficient algorithms for computing the transition matrix [6] using the concept of a topological sort [7]. Denoting the number of nodes and arcs by $n$ and $m$, respectively, these methods have time complexities of (a) $\mathrm{O}(n^3)$ and (b) $\mathrm{O}(n \cdot (n+m))$. The former method is based on an adjacency matrix, and is suitable for cases where the adjacency matrix is dense. In contrast, method (b) is based on an adjacency list, and is efficient if the adjacency matrix is sparse. However, the criterion for evaluating the density of the matrix is vague, and it is often difficult to estimate it in advance.

This research, therefore, proposes two new methods for efficiently calculating the Kleene Star regardless of the density of the adjacency matrix. Hereafter, we refer to these as methods (c) and (d), respectively. In method (c), operations akin to the elementary transformation of a matrix are performed from topologically upstream nodes towards downstream nodes. In method (d), the nodes for the adjacency matrix are first sorted in topological order, and then, a calculation similar to that in method (c) is carried out.

Both methods (c) and (d) have the same theoretical time complexity, $\mathrm{O}(n \cdot (n+m))$, which is the same as method (b). However, since the resulting matrix can be obtained only through elementary transformations, the new methods have the following attractive properties in practice:

- Since the algorithms consist only of simple vector operations, faster computation can be achieved by using a computer and language suited to vector and matrix calculations.
- Block splitting according to columns can be done in a straightforward manner. Thus the algorithms can be ported to systems that have parallel computation capabilities.

## II. Mathematical Background

### A. Max-Plus Algebra

Denoting the real field by $\boldsymbol{R}$, we define a field $\mathcal{D} = \boldsymbol{R} \cup \{-\infty\}$. Then for $x, y \in \mathcal{D}$, the following operators are defined: $x \oplus y = \max(x, y),\ x \otimes y = x + y,\ x^{\otimes y} = x \cdot y$. Let the unit elements for operators $\oplus$ and $\otimes$ be $\varepsilon\ (= -\infty)$ and $e\ (= 0)$, respectively. If $m \leq n$,

$$\bigoplus_{k=m}^{n} x_k = \max(x_m,\, x_{m+1}, \cdots, x_n).$$

With respect to matrices, $[\boldsymbol{X}]_{ij}$ denotes the $(i,\,j)$th element of matrix $\boldsymbol{X}$. If $\boldsymbol{X}, \boldsymbol{Y} \in \mathcal{D}^{m\times n}$ and $\boldsymbol{Z} \in \mathcal{D}^{n\times l}$,

$$[\boldsymbol{X} \oplus \boldsymbol{Y}]_{ij} = [\boldsymbol{X}]_{ij} \oplus [\boldsymbol{Y}]_{ij},\ [\boldsymbol{X} \otimes \boldsymbol{Z}]_{ij} = \bigoplus_{k=1}^{n} [\boldsymbol{X}]_{ik} \otimes [\boldsymbol{Z}]_{kj}.$$

Let the unit matrices for $\oplus$ and $\otimes$ be $\boldsymbol{\varepsilon}$ and $\boldsymbol{e}$, respectively. $\boldsymbol{\varepsilon}$ is a matrix, all elements of which are $\varepsilon$, while $\boldsymbol{e}$ is a matrix, the diagonal elements of which are $e$ and the off-diagonal elements are $\varepsilon$. The priority of operator $\otimes$ is higher than $\oplus$, and it is often omitted when no confusion is likely to arise.

## B. State Equation and Kleene Star

Here we review briefly the state-equation in a max-plus algebraic system and its transition matrix. In the remainder of the paper, terms applicable to manufacturing systems are used.

Let the number of facilities be $n$, the collection of preceding facilities of the $i$th $(1 \le i \le n)$ facility be $\mathcal{P}(i)$, and the job number be $k$. For the $k$th job, we denote the processing times by $\boldsymbol{d}(k)$, the processing completion times by $\boldsymbol{x}(k)$, and their minimum values by $\boldsymbol{u}(k)$. Then, the earliest processing completion times of the corresponding job $\boldsymbol{x}_E(k)$ are [5]:

$$\boldsymbol{x}_E(k) = \boldsymbol{A}_k \otimes [\boldsymbol{x}(k-1) \oplus \boldsymbol{u}(k)], \tag{1}$$

where

$$\begin{aligned} \boldsymbol{A}_k &= \boldsymbol{P}_k \boldsymbol{X}_k^*,\ \boldsymbol{P}_k = \mathrm{diag}[\boldsymbol{d}(k)],\ \boldsymbol{X}_k = \boldsymbol{F}_0 \boldsymbol{P}_k, \\ \text{if } j &\in \mathcal{P}(i), [\boldsymbol{X}_k]_{ij} = [\boldsymbol{d}(k)]_j \text{ and } [\boldsymbol{F}_0]_{ij} = e, \\ \text{if } j &\notin \mathcal{P}(i), [\boldsymbol{X}_k]_{ij} = [\boldsymbol{F}_0]_{ij} = \varepsilon. \end{aligned}$$

Equation (1) is known as the state-equation, with $\boldsymbol{A}_k$ denoting the transition or system matrix, $\boldsymbol{F}_0$ the adjacency matrix, and $\boldsymbol{X}_k$ the weighted adjacency matrix. Hereafter, we simply denote $\boldsymbol{X}_k$ by $\boldsymbol{X}$. Operator * is called the Kleene Star, which holds the property given below with respect to $\boldsymbol{X}$:

$$\boldsymbol{X}^* = \bigoplus_{l=0}^{s-1} \boldsymbol{X}^l = \boldsymbol{e} \oplus \boldsymbol{X} \oplus \cdots \oplus \boldsymbol{X}^{s-1}, \tag{2}$$

where $\boldsymbol{X}^{s-1} \ne \boldsymbol{\varepsilon},\ \boldsymbol{X}^s = \boldsymbol{\varepsilon}\quad (1 \le s \le n)$. $s$ depends on the precedence relationships of the system. Equation (2) is an expression, the precedence constraints of which can be represented by a DAG, whereas its general form is defined by summation of an infinite series. It should be noted that each element of $\boldsymbol{X}^*$ in (2) satisfies the following properties [2]:

$$[\boldsymbol{X}^*]_{ij} = \begin{cases} \tau_{ij} & : \text{if facility } j \text{ is reachable from facility } i, \\ \varepsilon & : \text{if facility } j \text{ is not reachable from facility } i, \\ e & : \text{if } i = j, \end{cases} \tag{3}$$

where $\tau_{ij}$ is the sum of the processing times in the preceding mid-facilities from facility $j$ to facility $i$, and is equal to the maximum value if there are multiple paths between these facilities.

## C. Representations in Terms of Graph Theory

In terms of graph theory, a facility, precedence constraint, and processing time can be interpreted as a node, arc, and weight of arc, respectively. Moreover, solving $\boldsymbol{X}^*$ is equivalent to solving the longest path problem for all pairs of nodes. However, using (2) directly, results in a highly inefficient computation with time complexity $\mathrm{O}(n^4)$. This has led to efficient computation methods based on the idea of a topological sort in graph theory.

If node $i$ is physically located upstream of node $j$, we denote this by $i \prec j$. Moreover, the index number of node $i$ is denoted by $\hat{i}$, is all nodes have been topologically sorted. The topological sort is a method of aligning nodes for which the relationship below holds for all pairs of nodes $(i,\,j)$:

$$\text{If } i \prec j,\ \text{ then } \hat{i} < \hat{j}. \tag{4}$$

Note that the inverse of (4) does not hold. This implies that even if node $i$ is located upstream of node $j$ topologically, node $i$ is not always located upstream physically. Moreover, there could be cases where node $j$ is not reachable from node $i$. Hereafter, the terms 'upstream' and 'downstream' are used in the context of topological positional relationships. There is a well known efficient computation method called the Depth First Search (DFS). Letting the number of nodes and arcs be $n$ and $m$, respectively, the time complexity based on this method is $\mathrm{O}(n+m)$. It should be noted here that the result is not unique and depends on its implementation.

Once all nodes have been topologically sorted, the original node number of the node with index $l$, is given by $\tilde{l}$. This implies $\hat{\tilde{l}} = l$ and $\tilde{\hat{l}} = l$. If node $j$ physically precedes node $i$, we represent this relationship by $j \rightarrow i$. In this case, both $[\boldsymbol{X}]_{ij} \ne \varepsilon$ and $j \prec i$ hold.

## D. Existing methods

Efficient computation methods for the Kleene Star previously proposed in [6] are now reviewed briefly. Suppose that the weighted adjacency matrix of a system with DAG type precedence constraints is given by $\boldsymbol{X}$, and its precedence constraints are identified by a topological sort based on DFS.

First, initialize a work matrix $\boldsymbol{Z}$ as $\boldsymbol{Z} \Leftarrow \boldsymbol{e}$. Then, in method (a), $\boldsymbol{Z}$ is iteratively updated according to the following procedure:

$$\begin{aligned} [\boldsymbol{Z}]_{\tilde{i}\tilde{j}} \quad \Leftarrow \quad & [\boldsymbol{Z}]_{\tilde{i}\tilde{j}} \oplus \bigoplus_{l=1}^{n} [\boldsymbol{X}]_{\tilde{i}l} \otimes [\boldsymbol{Z}]_{l\tilde{j}}, \\ & \text{for all } \tilde{i} \text{ and } \tilde{j}\ (1 \le i \le n,\ 1 \le j \le i-1) \end{aligned} \tag{5}$$

For node $\tilde{i}$, working upstream to downstream, this repetition is carried out for all $i\,(1 \le i \le n)$. For node $\tilde{j}$, it is carried out from the uppermost upstream node to the node preceding node $i$.

Method (b) on the other hand, uses an adjacency list. The update is carried out selectively for elements that satisfy $[\boldsymbol{X}]_{\tilde{i}l} \neq \varepsilon$ in the following way:

$$[\boldsymbol{Z}]_{\tilde{i}\tilde{j}} \Leftarrow [\boldsymbol{Z}]_{\tilde{i}\tilde{j}} \oplus \bigoplus_{l \in \mathcal{P}(\tilde{i})} [\boldsymbol{X}]_{\tilde{i}l} \otimes [\boldsymbol{Z}]_{l\tilde{j}}, \quad \text{for all } \tilde{i} \text{ and } \tilde{j}\ (1 \leq i \leq n,\ 1 \leq j \leq i-1), \tag{6}$$

where $\mathcal{P}(\tilde{i})$ represents the collection of nodes $l$ that satisfy $l \to \tilde{i}$. The iterative procedure for nodes $i$ and $j$ is the same as in method (a).

After the update process according to (5) and (6), the values of $\boldsymbol{X}^*$ are stored in $\boldsymbol{Z}$. In the existing methods, (a) and (b), we first fix the destination node $\tilde{i}$, and then the update procedure is performed for the corresponding source nodes. This procedure coincides intuitively with the mathematical interpretation in which obtaining $\boldsymbol{X}^*$ is equivalent to solving the longest path problem for all pairs of nodes. However, the update of $\boldsymbol{Z}$ focuses on element $(\tilde{i},\ \tilde{j})$, and not on element $(i,\ j)$, and consequently, it is not performed on successive elements. As such, it is difficult to update multiple elements at one time, which means that it is not easy to apply efficient computational techniques such as vectorization or parallelization, which are frequently used in matrix and vector operations, to the algorithms. Thus this paper proposes efficient new algorithms for calculating $\boldsymbol{X}^*$, focusing on source nodes.

## III. Proposed Algorithms

Two efficient computation methods for the Kleene star of the weighted adjacency matrix $\boldsymbol{X}$ are proposed.

### *A. Method based on an elementary transformation of the adjacency matrix*

Hereafter, this method is referred to as method (c). Assume the weighted adjacency matrix is given by $\boldsymbol{X}$ and the precedence constraints of nodes are identified by preprocessing including a topological sort. In other words, assume that $\{\tilde{1},\ \tilde{2},\ \tilde{3},\ \cdots\ \tilde{n}\}$ have already been obtained. Then, $\boldsymbol{X}^*$ can be computed as follows.

First, initialize the work matrix $\boldsymbol{Z}$, as

$$\boldsymbol{Z} \Leftarrow \boldsymbol{e}. \tag{7}$$

Then, for source node $\tilde{l}$, obtain a collection of destination nodes $i$, that is, a collection of $i$, denoted by $\mathcal{S}(\tilde{l})$, that satisfies $\tilde{l} \to i$. Then, we update the work matrix $\boldsymbol{Z}$ according to Eq. (8):

$$[\boldsymbol{Z}]_{ij} \Leftarrow [\boldsymbol{Z}]_{ij} \oplus [\boldsymbol{X}]_{i\tilde{l}} \otimes [\boldsymbol{Z}]_{\tilde{l}j}, \quad \text{for all } i \in \mathcal{S}(\tilde{l}) \text{ and } j\ (1 \leq j \leq n). \tag{8}$$

This is repeated for all $\tilde{l}$, from the uppermost upstream node working downstream. This means that the update is carried out for $(1 \leq l \leq n)$ by increasing $l$.

Equation (8) can be interpreted as the operation of adding to the $i$th row of $\boldsymbol{Z}$, the values of the $\tilde{l}$th row of matrix $\boldsymbol{Z}$ multiplied by a constant $[\boldsymbol{X}]_{i\tilde{l}}$. This procedure is equivalent to the elementary transformation of a matrix, frequently used in conventional linear algebra.

Next, we consider the time complexity. To execute (7), the complexity is $\mathrm{O}(n^2)$. The time complexity for all update procedures of (8), noting that there are $n$ repetitions of $j$, and that the number of combinations of $(i,\ \tilde{l})$ that satisfy $[\boldsymbol{X}]_{i\tilde{l}} \neq \varepsilon$ is $m$, is $\mathrm{O}(n \cdot m)$. Consequently, the time complexity of the proposed method is $\mathrm{O}(n \cdot (n+m))$, which is the same as method (b).

To confirm that the above algorithm gives $\boldsymbol{X}^*$, we provide a proof for Theorem 1 below.

*Thorem 1:* After the update process given by (7) and (8), the values of $\boldsymbol{X}^*$ are stored in $\boldsymbol{Z}$.

*Proof.* Since (8) is an expression that focuses on the source node $\tilde{l}$, the number of updates of the $i$th row of $\boldsymbol{Z}$ is unknown. This means that the row may never be updated or may be updated multiple times. Hence, we derive another expression in which the update operations for the $i$th row are combined. It can be rewritten thus:

$$[\boldsymbol{Z}]_{ij} \Leftarrow [\boldsymbol{Z}]_{ij} \oplus ([\boldsymbol{X}]_{il_1} \otimes [\boldsymbol{Z}]_{l_1 j}) \oplus ([\boldsymbol{X}]_{il_2} \otimes [\boldsymbol{Z}]_{l_2 j}) \oplus \cdots \quad \text{for all } j\ (1 \leq j \leq n), \tag{9}$$

where $l_1, l_2, \cdots$ represent the source nodes for the destination node $i$, which is henceforth denoted by $\mathcal{P}(i)$. Here, since we do not assume that the repetition involving $l$ is topological, the tilde suffixes are not attached. According to (9), there is only one update for the $i$th row of $\boldsymbol{Z}$, and the equation can be restated as:

$$[\boldsymbol{Z}]_{ij} \Leftarrow [\boldsymbol{Z}]_{ij} \oplus \bigoplus_{l \in \mathcal{P}(i)} [\boldsymbol{X}]_{il} \otimes [\boldsymbol{Z}]_{lj}, \quad \text{for all } j\ (1 \leq j \leq n). \tag{10}$$

Next, consider the operational sequence with respect to $i$ and $j$. Since the $i$th row of $\boldsymbol{Z}$ is updated only once by (10), the final value of $[\boldsymbol{Z}]_{lj}$ $(l \in \mathcal{P}(i))$ must already be stored in the second term on the right hand-side. Moreover, (10) indicates that elements of the $j$th $(1 \leq j \leq n)$ column must be updated consecutively for an instance $i$. As this implies, when updating the values of nodes $j$ and $i$, the values of nodes $j$ and $l$ must already have been obtained. In addition, this update is carried out strictly by maintaining relationship $l \to i$. Thus it is necessary to update the value of node $j$ beforehand when it is located upstream of node $i$. On the other hand, if $j = i$ or node $j$ is located downstream of node $i$, the value of $[\boldsymbol{Z}]_{ij}$ is never updated and remains the initial value $[\boldsymbol{e}]_{ij}$. In the former case, the value is $e$, whereas in the latter case it is $\varepsilon$. This satisfies the properties of $\boldsymbol{X}^*$ in (3).

The repetition of $i$ must be carried out for all $i$ $(1 \leq i \leq n)$. However, as mentioned earlier, if there is a node $l$ that satisfies $l \rightarrow i$ downstream of node $j$, the value with respect to nodes $j$ and $l$ must have been obtained already. As such, $i$ and $j$ should be iterated according to the following sequence:

$$\text{For all } \tilde{i} \text{ and } \tilde{j} \ (1 \leq i \leq n,\ j \prec i).$$

This means that the operation in (10) is equivalent to (6), that is, method (b). The fact that the values of $\boldsymbol{X}^*$ are stored in $\boldsymbol{Z}$ through this procedure has already been proved in [6]. ■

In the method based on (6), the update of the $\tilde{i}$th row of $\boldsymbol{Z}$ is carried out for discontinuous elements. On the other hand, (8) can update the $j$th column of $\boldsymbol{Z}$ in an arbitrary order. That is, the update need not necessarily be performed in topological order, and can be carried out according to the original sequence of columns. Thus this algorithm could increase the computation efficiency in systems suitable for vector and matrix calculations. Moreover, the algorithm can easily be implemented on processors with vector instructions such as SIMD [8]. Furthermore, recall the elementary transformation of a matrix that includes the addition of values multiplied by a constant, which is independent of column direction. Thus the transformation can be broken up into arbitrary sets of columns, each of which can be computed independently. This indicates that it is easy to parallelize the procedure [9].

### B. Another Efficient Algorithm

Consider systems that have precedence constraints of the DAG type. If $i \prec j$, node $i$ is not reachable from node $j$, and $[\boldsymbol{X}^*]_{ij} = \varepsilon$ holds true. Accordingly, with respect to node $i$, it would not be necessary to calculate the longest path from the node to the downstream node $j$. Utilizing this feature, in this subsection we propose another method for calculating $\boldsymbol{X}^*$ based on (8), after sorting the columns and rows of the weighted adjacency matrix $\boldsymbol{X}$ using a topological sort. Henceforth, this method is referred to as method (d). The time complexity of method (d) is $\mathrm{O}(n \cdot (n+m))$, which is equal to method (c) described above. However, by utilizing desirable properties of the DAG, the computation time can be reduced several fold, the detail of which is discussed below.

*Thorem 2:* The weighted adjacency matrix $\boldsymbol{X}$ is a DAG type whose rows and columns are topologically sorted in the lower triangular matrix. In addition, the work matrix $\boldsymbol{Z}$ in the update process specified by (7) and (8) is also a lower triangular matrix, whose diagonal elements are $e$ at any time during the process.

*Proof.* Since the rows and columns of $\boldsymbol{X}$ are topologically sorted, if precedent constraint $j \rightarrow i$ is true, $j < i$ holds. In other words, if there are instances that satisfy $[\boldsymbol{X}]_{ij} \neq \varepsilon$, then $j < i$. This clearly indicates that $\boldsymbol{X}$ is a lower triangular matrix.

Subsequently, we prove that $\boldsymbol{Z}$ is lower triangular and its diagonal elements are all $e$. First, immediately after the initialization process by (7), the proposition clearly holds true. Next, suppose $\boldsymbol{Z}$ is lower triangular and its diagonal elements are $e$ during a certain update process based on (8). Under this assumption, and by enforcing $[\boldsymbol{Z}]_{\tilde{l}j} \neq \varepsilon$, this can be accomplished only when $j \prec \tilde{l}$, that is $\hat{j} < \hat{\tilde{l}} = l$. On the other hand, $i \in \mathcal{S}(\tilde{l})$ indicates that a relationship $\tilde{l} \rightarrow i$ holds true and yields $\hat{\tilde{l}} = l < \hat{i}$. Thus the update is carried out only for elements $[\boldsymbol{Z}]_{ij}$ that satisfy $\hat{j} < \hat{i}$. This also indicates that elements $[\boldsymbol{Z}]_{ij}$ which satisfy $\hat{j} \geq \hat{i}$ are not updated. Accordingly, the diagonal elements of $\boldsymbol{Z}$ remain $e$. Moreover, if $\hat{i} < \hat{j}$, node $j$ is not a preceding node of node $i$, which yields $[\boldsymbol{Z}]_{ij} = \varepsilon$. This proves the proposition. ■

As the above theorem implies, it is not required to update the $l$ $(i \leq l \leq n)$th column of $\boldsymbol{Z}$ when updating the $i$th row. This indicates that the number of elements that must be updated can be reduced to less than half. However, it should be noted that a topological sort of $\boldsymbol{X}$ must be carried out twice, in both pre- and post-processing, thereby incurring additional overhead. The time complexity for these sorts is $\mathrm{O}(n^2)$, which may not be ignored if $n$ is small. Hence, it can be said that method (d) is suitable only if $n$ is sufficiently large.

## IV. Efficiency of the Proposed Methods

The effectiveness of the proposed methods is validated through an illustrative example and numerical experiments.

### A. Illusrative example

Let us demonstrate the algorithm in method (c) for a simple illustrative manufacturing system of the DAG type. Figure 1 depicts a simple manufacturing process with one input, one output, and five facilities. Each value in the box represents a facility number, and (*) gives the processing time. In this case, the weighted adjacency matrix is:

$$\boldsymbol{X} = \begin{bmatrix} \varepsilon & \varepsilon & \varepsilon & \varepsilon & 1 \\ 3 & \varepsilon & \varepsilon & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & \varepsilon & 1 \\ \varepsilon & 5 & 4 & \varepsilon & \varepsilon \\ \varepsilon & \varepsilon & \varepsilon & \varepsilon & \varepsilon \end{bmatrix}.$$

The results of the topological sort for this system are illustrated in Fig. 2, which yields: $(\tilde{1},\ \tilde{2},\ \tilde{3},\ \tilde{4},\ \tilde{5}) = (5,\ 3,\ 1,\ 2,\ 4)$. Note that this is not unique as previously mentioned. First, let $l = 1$, giving $\tilde{l} = 5$. The set that obeys $[\boldsymbol{X}]_{i\tilde{l}} \neq \varepsilon$ is $\mathcal{S}(\tilde{l}) = \{1,\ 3\}$. Then, we set $i = 1$ $(\in \mathcal{S}(\tilde{l}))$. Since we enforce $[\boldsymbol{X}]_{15} = 1$, values of the fifth row of $\boldsymbol{Z}$ multiplied by one are added to the first row. Next, we set $i = 3$. Since $[\boldsymbol{X}]_{35} = 1$ is true, we add values of the fifth row of $\boldsymbol{Z}$ multiplied by one to the third row.

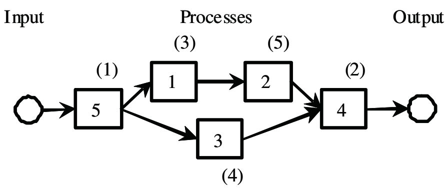

Figure 1. A simple manufacturing system.

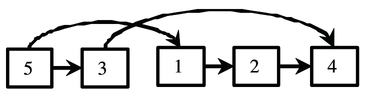

Figure 2. Result of the topological sort.

In a similar way, for $l = 2$, after $\tilde{l} = 3$ and $S(\tilde{l}) = \{4\}$ have been obtained, let $i = 4\ (\in \mathcal{S}(\tilde{l}))$ and then we obtain $[\boldsymbol{X}]_{i\tilde{l}}\ (= 4)$. Subsequently, add the $\tilde{l}\ (= 3)$th row of $\boldsymbol{Z}$ multiplied by this value to the $i\ (= 4)$th row. Using an analogous procedure to that shown above, the transformation is repeated until $j = 5$. This finally produces:

$$\boldsymbol{Z} = \begin{bmatrix} e & \varepsilon & \varepsilon & \varepsilon & 1 \\ 3 & e & \varepsilon & \varepsilon & 4 \\ \varepsilon & \varepsilon & e & \varepsilon & 1 \\ 8 & 5 & 4 & e & 9 \\ \varepsilon & \varepsilon & \varepsilon & \varepsilon & e \end{bmatrix}. \tag{11}$$

The above procedure is depicted in Fig. 3.

As an application example, we calculate the earliest processing completion times for job $k = 1$. We set the processing times to $\boldsymbol{d}(1) = [3,\ 5,\ 4,\ 2,\ 1]^T$, feeding times of material to $\boldsymbol{u}(1) = [\varepsilon,\ \varepsilon,\ \varepsilon,\ \varepsilon,\ 0]^T$, and the state variables of job $k = 0$ to $\boldsymbol{x}(k-1) = \boldsymbol{\varepsilon}$. Letting $\boldsymbol{Z}$ in (11) be $\boldsymbol{X}^*$, $\boldsymbol{x}_E(1)$ in (1) is calculated as: $\boldsymbol{x}_E(k) = [4,\ 9,\ 5,\ 11,\ 1]^T$. The fact that this is equivalent to the earliest processing completion times in the respective facilities can easily be confirmed by considering Fig. 1.

## B. Performance evaluation

We have implemented methods (a)–(d) to examine their computational efficiency. With respect to the weighted ad-

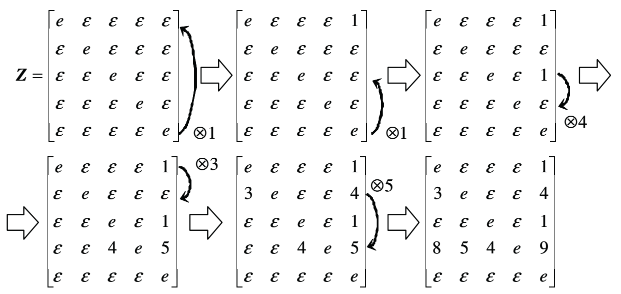

Figure 3. Update processes based on method (c).

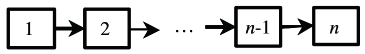

Figure 4. A tandem structured system.

jacency matrix $\boldsymbol{X}$, both dense and sparse matrices are considered. For dense matrices, we apply the precedence constraint $i \to j$ with a probability of 1/2 to all pairs of $(i,\ j)$ that satisfy $i \prec j$. For sparse matrices, we apply precedence constraints with all nodes connected in tandem, as shown in Fig. 4. We assume that the weights, $\boldsymbol{d}(k)$, obey a uniform $[0,\ 1]$ distribution. For both cases, after the adjacency matrix has been generated, the indexes of nodes are sorted randomly. Experiments have been carried out with $n = \{10, 50, 100, 500, 1000\}$.

The execution environment is an IBM PC/AT, Pentium IV 3.0GHz, running Windows XP. The calculation modules are coded using Octave-3.0.3 [10]. Computation times are measured ten times and the average of the eight times, excluding the maximum and minimum times, is calculated. The computation times are measured using 'tic' and 'toc' functions. Table 1 gives the results where the adjacency matrix $\boldsymbol{X}$ is dense, while Table 2 gives the results for sparse matrices. All time units are given as seconds.

In Table 1, the proposed methods (c) and (d) are faster than the existing methods (a) and (b), except where $n = 10$. The difference in performance is significant as $n$ increases. Compared with methods (c) and (d), the results show no significant difference when $n = 50$ or smaller. On the contrary, if $n = 100$ or larger, method (d) is faster and its effectiveness becomes more noteworthy as $n$ increases.

In Table 2, methods (c) and (d) are faster than methods (a) and (b), and their performance is significantly better as $n$ increases. This is the same as in Table 1. On the other hand, unlike Table 1, method (d) is not always faster than method (c) even as $n$ increases. This may be because the reduction effect of handling the lower triangular matrix is comparable with the overhead for sorting indexes of the adjacency matrix $\boldsymbol{X}$.

Comparing Tables 1 and 2, the computation times for method (a) are not that different because the time complexity is $\mathrm{O}(n^3)$, which is independent of the number of arcs. In contrast, methods (b)—(d) have $\mathrm{O}(n \cdot (n + m))$ time complexity. Thus for the same $n$, the computation time is shorter if $\boldsymbol{X}$ is sparse. Moreover, for methods (c) and (d), if $n = 50$ or smaller, the computation times are not that different. On the other hand, it might be said that method (d) has the advantage if $n = 100$ or larger. However, if the adjacency matrix is extremely sparse such as shown in Fig. 4, the computation time based on method (c) is sometimes shorter. This should always be taken into account. Also, in practice, method (d) may not be a 'bad selection' regardless of the size of $n$ and the density of $\boldsymbol{X}$.

Table I
CALCULATION TIMES FOR DENSE MATRICES.

| $n$ | 10 | 50 | 100 | 500 | 1,000 |
|---|---|---|---|---|---|
| (a) | 0.01889 | 0.4074 | 1.646 | 65.92 | 395.2 |
| (b) | 0.02732 | 0.6267 | 2.527 | 77.84 | 409.0 |
| (c) | 0.02278 | 0.2964 | 1.183 | 41.17 | 233.6 |
| (d) | 0.02159 | 0.2935 | 1.135 | 32.78 | 159.5 |

Table II
CALCULATION TIMES FOR SPARSE MATRICES.

| $n$ | 10 | 50 | 100 | 500 | 1,000 |
|---|---|---|---|---|---|
| (a) | 0.01901 | 0.4063 | 1.691 | 69.34 | 420.8 |
| (b) | 0.02537 | 0.5308 | 2.067 | 50.94 | 204.8 |
| (c) | 0.01408 | 0.2322 | 0.862 | 20.29 | 81.3 |
| (d) | 0.01585 | 0.2198 | 0.810 | 22.43 | 108.8 |

## V. CONCLUSION

This paper has proposed efficient computation methods for the Kleene Star focusing on discrete event systems in which the adjacency matrix can be represented by a DAG. The time complexity of the new methods is the same as that of the previously proposed methods. However, we have confirmed that the practical computation times are shorter than those in the previous methods. Moreover, the proposed methods do not require estimating, in advance, whether or not the adjacency matrix is dense. This would be advantageous in a practical implementation.

In addition, the proposed methods have the following attractive properties for further speedup in future research; the algorithms are suited to collective calculation such as using vector or matrix operations, and can be divided into several column blocks to be handled in parallel. This could be accomplished using computers with instructions for vector calculations and/or parallel processing. Porting and implementing the proposed system is our future work.

## REFERENCES

[1] B. Heidergott, G. J. Olsder, and L. Woude, *Max Plus at Work: Modeling and Analysis of Synchronized Systems*. New Jersey: Princeton Univ. Pr., 2006.

[2] F. Baccelli, G. Cohen, G. J. Olsder, and J. P. Quadrat, *Synchronization and Linearity*. New York: John Wiley & Sons, 1992. [Online]. Available: http://maxplus.org

[3] G. Schullerus, V. Krebs, B. Schutter, and T. Boom, "Input signal design for identification of max-plus-linear-systems," *Automatica*, vol. 42, pp. 937–943, 2006.

[4] R. Goverde, "Railway timetable stability analysis using max-plus system theory," *Transportation Research Part B*, vol. 41, no. 2, pp. 179–201, 2007.

[5] H. Goto and S. Masuda, "Monitoring and scheduling methods for mimo-fifo systems utilizing max-plus linear representation," *Industrial Engineering and Management Systems*, vol. 7, no. 1, pp. 23–33, 2008.

[6] H. Goto, "Efficient calculation of the transition matrix in a max-plus linear state-space representation," *IEICE Transactions on Fundamentals*, vol. E91-A, no. 5, pp. 1278–1282, 2008.

[7] T. Cormen and C. Leiserson, *Introduction to Algorithms*. Massachusetts: MIT Press, 2001.

[8] P. Cockshott and K. Renfrew, *SIMD Programming Manual for Linux and Windows*. Heidelberg: Springer, 2004.

[9] R. Shonkwiler and L. Lefton, *An Introduction to Parallel and Vector Scientific Computing (Cambridge Text in Applied Mathematics)*. Cambridge Univ. Pr., 2006.

[10] J. Eaton, D. Bateman, and S. Hauberg, *GNU Octave Manual Version 3*. Network Theory Ltd., 2008.