J. King Saud Univ., Vol. 15, *Comp. & Info. Sci.*, pp. 33-66 (A.H. 1423/2003)

A New Disk-based Technique for Solving the Largeness Problem of Stochastic Modeling Formalisms

Samir M. Koriem and W. S. El-Kilani*
Department of Systems and Computer Engineering,
Faculty of Engineering, El-Azhar University, Cairo, Egypt
** Department of Information Technology, Faculty of Computer and Information,*
Menoufia University, Shabeen El-Koum, Egypt

(Received 14 October 2001; accepted for publication 11 February 2002)

Abstract. Stochastic modeling formalisms such as stochastic Petri nets, generalized stochastic Petri nets, and stochastic reward nets can be used to model and evaluate the dynamic behavior of realistic computer systems. Once we translate the stochastic system model to the underlying corresponding Markov Chain (MC), the developed MC grows wildly to several hundred thousands states. This problem is known as the largeness problem. To tolerate the largeness problem of Markov models, several iterative and direct methods have been proposed in the literature. Although the iterative methods provide a feasible solution for most realistic systems, a major problem appears when these methods fail to reach a solution. Unfortunately, the direct method represents an undesirable numerical technique for tolerating large matrices due to the fill-in problem. In order to solve such problem, in this paper, we develop a Disk-Based Segmentation (DBS) technique based on modifying the Gauss Elimination (GE) technique. The proposed technique has the capability of solving the consequences of the fill-in problem without making assumptions about the underlying structure of the Markov processes of the developed model. The DBS technique splits the matrix into a number of vertical segments and uses the hard disk to store these segments. Using the DBS technique, we can greatly reduce the memory required as compared to that of the GE technique. To minimize the increase in the solution time due to the disk accessing processes, the DBS utilizes a clever management technique for such processes. The effectiveness of the DBS technique has been demonstrated by applying it to a realistic model for the Kanban manufacturing system.

Keywords: Stochastic reward nets; largeness tolerance methods; direct methods; Markov chain; disk-based segmentation; gauss elimination.

1. Introduction

Rapid advances in technology have resulted in the proliferation of complex computer systems that are used in different applications, ranging from spacecraft flight control to information and financial services. Modelling and evaluating mechanisms provide

a good methodology for examining the behavior of these systems from the design stage to the implementation and final deployment. Recently, the use of Continuous Time Markov Chains (CTMCs) has proven to be a very useful modeling formalism for evaluating these systems [1, 2]. Many problems have been encountered in the use of Markov models such as *stiffness* and *largeness* [1,3].

Stiffness is an undesirable characteristic of many practical Markov models. Stiffness adversely affects the computational efficiency of the numerical solution technique [3]. The stiffness problem arises if the model solution has rates that differ widely. Moreover, since most Markov models of real systems are very large, the actual model of the desired system is usually specified using a high-level description such as Stochastic Reward Nets (SRNs) [4]. The SRN technique has the capability of modeling the dynamic behavior of large and complex systems in a compact way. Once we translate the SRN system model to the underlying corresponding CTMC, the developed CTMC grows wildly to several hundred thousands states. This problem is known as the *largeness problem* [5].

The largeness problem can be overcome by the use of the largeness avoidance or largeness tolerance technique [5]. The *largeness avoidance* approach needs to discover the structure of the CTMC under study to facilitate its solution. Examples of such technique are the stochastic process algebras, matrix-geometric stochastic Petri nets, product-form queuing networks, and product-form stochastic process algebras [6-8]. These techniques have the capability of exploiting the structure of the CTMC under study to obtain an efficient CTMC generation. Subsequently, the solution of the CTMC becomes possible.

On the other hand, the *largeness tolerance* approach employs the sparse storage techniques and computational procedures that do well in combination with sparsely processing solution methods. This approach gives a facility to the modeler to accommodate the desired models that are as large as possible. In practice, the concept of this approach means that the CTMC size is restricted by the size of the main memory of the system on which the model is constructed and solved. CTMCs with several hundred thousand states have been solved using this approach [9]. It should be noted that the model reduction is employed only in the case of the largeness avoidance approach. In this paper, we deal only with the largeness tolerance approach. In the literature, three important numerical solution techniques for the largeness tolerance problem have been considered: Superposed Generalized Stochastic Petri Nets (SGSPNs) [10], on-the-fly technique [11] and disk-based method [12].

The idea of the SGSPN technique [10] is to combine a set of originally independent GSPNs into a single superposed GSPN by synchronization of transitions. At net level, the synchronization takes place by merging the desired transitions. The resulting transitions are called synchronized transitions. This concept is closely related to

the concept of stochastic automata networks [13] and Markovian process algebras [14]. The SGSPN technique has two substantial restrictions: (i) the structure of the developed model should incorporate independent components with limited interaction between them, and (ii) the sub-models of the developed model should have approximately the same size. Therefore, the SGSPN technique has the disadvantage of requiring a special structure in the model in order to work efficiently [15].

On-the-fly technique [11] introduces a new iterative solution method, called a modified adaptive Gauss-Seidel. This technique eliminates the need to explicitly store the matrix at all when generating the rows and columns of the state transition-rate-matrix on-the-fly. Further, this approach permits the caching of portions of the matrix. This process leads to the reduction of the solution time.

A Disk-based technique [12] uses a workstation disk to hold the transition-rate matrix of the CTMC under study, the variant of the Gauss-Seidel block as an iterative solution method, and an innovative implementation that involves the following two parallel processes. The first process retrieves portions of the transition matrix from the workstation disk. The second process repeats the computation on small portions of the matrix. Disk-based methods have the potential to greatly outperform the SGSPN method by using a large disk to allow to the holding of the state-transition-rate matrix of the Markov model being solved. Further, high performance disks are inexpensive, relative to the cost of main memories (RAMs).

It is interesting to note that largeness tolerance methods such as the SGSPN [10], the on-the-fly technique [11] and the disk-based method [12] utilize powerful iterative methods such as the Jacobi method, the power method, and the Gauss-Seidel method. A major problem appears when the largeness tolerance methods fail to reach a solution for the desired model or spend a long time to converge to the prescribed precision [16]. To solve such problem, we can use the *direct method* as an alternative approach for the numerical solution of the Markov models. Unfortunately, the direct method is an undesirable numerical technique for tolerating *large matrices* due to the following problems [17, 18].

- The reduction of the matrix from its regular form to the upper triangular form (reduction phase) often needs to perform subtraction operations among the rows. These operations lead to the creation of many non-zero elements in positions that previously contained zeros and the fading out of the undesired elements. This kind of problem is known as the *fill-in* problem. The appearance of new elements and the fading out of undesired elements needs a continuous process of data insertion and deletion from the memory. This process needs a lot of time. Consequently, the direct method suffers from a time consumption problem.

- The subtraction operations among the rows that occur during the reduction phase also lead to a continuous change of matrix elements. This change in turn leads to a build up of rounding error in the matrix elements.

The main objective of this paper is to develop a suitable largeness tolerance methodology based on modifying a version of the Gauss Elimination (GE) technique. The proposed technique has the capability of solving the consequences of the *fill-in* problem that usually arises in the GE procedure. Our proposed methodology modifies the GE procedure in order to be able to solve the large Markov chain matrices generated from the stochastic reward nets. We call such a technique "Disk-Based Segmentation" (DBS) technique. The basic concept of the DBS technique depends on splitting the whole matrix into a number of vertical segments. Each vertical segment consists of a band of successive columns. Each vertical segment is acted upon by a modified version of the GE procedure in such a way that these segments are transformed and become a part of the triangular form of the original matrix. We make use of the hard disk to store the segments before and after the transformation processes and carry the records of the computation operations applied. It is interesting to note that the dependence of the proposed technique on the hard disk for data retrieval and loading gives it the name DBS method. In order to identify the merits of our proposed technique, we measure its efficiency with respect to time and space under different conditions. We have also implemented a tool to support the proposed methodology. This tool works on PCs and provides us with the following capabilities.

- Generating and solving Markov large matrices resulting from the SRN models.
- Performing both steady and transient analysis for these matrices by utilizing a number of traditional direct and iterative methods proposed in the literature [1-3], [19].

The remaining parts of this paper are organized as follows. Section 2 gives a brief description for our developed EasySPN tool that is used to support the DBS technique. Section 3 illustrates the main steps of the proposed DBS technique. In Section 4, three implementation issues for the DBS technique have been analyzed. Section 5 gives the numerical results for the application processes of the DBS technique on a large matrix generated from a practical SRN system model. Section 6 concludes this paper.

2. The Easy SPN Tool

In this section, we give a brief description of the proposed tool that is used to support the DBS technique. The proposed tool is called the *EasySPN*. This tool has the capability of generating and solving the Markov model that is obtained from the SRN system model. The EasySPN tool is a windows-based application that runs on Microsoft (MS) Windows 2000 (or its later versions). The capability of utilizing the powerful memory

management facilities of MS Windows represents the main advantage of the proposed tool. Further, the proposed tool has been implemented in a modular fashion using the object-oriented facilities of MS Visual C++ package.

This *EasySPN* package consists of five main modules: the graphical user interface (GUI) module, the state space generator (SSG) module, the state probability solver (SPS) module, the performance measures (PM) module, and the memory management (MM) module. A block diagram of the tool is shown in Fig. 1. The GUI module manipulates four menus: net menu, firing menu, solver menu and performance menu. The net menu allows the user to enter the description of the modeled system according to the GSPN, or SRN formalisms. Using the SSG module, the firing menu permits firing of the desired net giving rise to the Reachability Set (RS) and the Reachability Graph (RG). The RS is the set of states resulting from firing the transitions of the SRN model. The description of the connections (in terms of the transition rates of the SRN model) between the various states of the RS is contained in the RG. Both the RS and RG are stored in two separate files in a way allowing easy access by other modules of the program. It is well known that the developed RG of the stochastic nets is isomorphic to an ergodic CTMC [18]. This CTMC is governed by the following Kolmogorov differential equation [17].

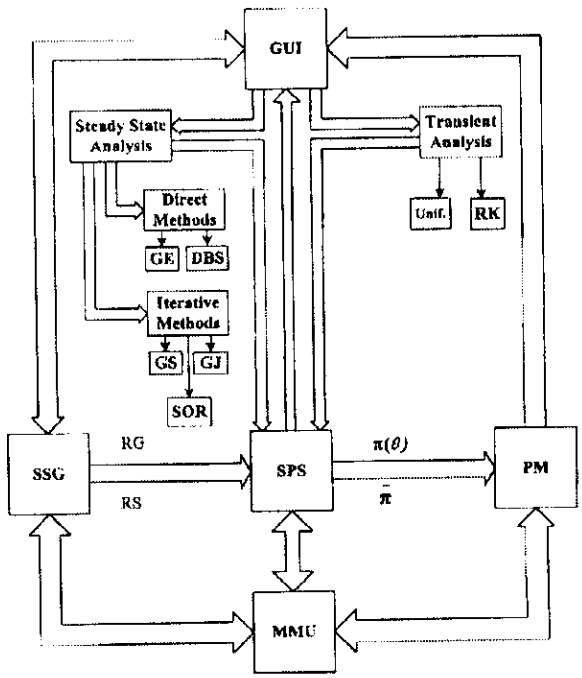


Fig.1. Block diagram of the *Easy SPN* tool.

$$d\pi(\theta) / d\theta = \pi(\theta)\, Q; \qquad \theta \geq 0 \tag{2.1}$$

With

$$\sum_{i=0}^{N-1} \pi_i(\theta) = 1 \qquad \text{for all } \theta \geq 0 \tag{2.2}$$

Where N is the number of the RS states, $\pi \equiv \{\pi_i;\ 0 \leq i \leq N\text{-}1\}$, and $\pi_i(\theta)$ is the probability of being in a state S_i at time θ. The matrix $Q \equiv \{q_{ij}\}_{N \times N}$ is the infinitesimal generator or transition rate matrix of the CTMC under study. Each element of the matrix Q is related to the infinitesimal transition rate λ_t from a state S_i to a state S_j as follows.

$$q_{ij} = \sum_{t \in T: S_i \xrightarrow{t} S_j} \lambda_t ; \quad 0 \leq i \leq N\text{-}1 \tag{2.3}$$

With the constraints:

$$q_{ii} = -\sum_{\substack{j=0 \\ i \neq j}}^{N-1} q_{ij} \tag{2.4}$$

The SPS module computes $\pi(\theta)$ as well as the steady state distribution $\bar{\pi}$ using equations 2.1, 2.2, and the following equation.

$$\bar{\pi} Q = 0 \quad , \quad \sum_{i=0}^{N-1} \bar{\pi}_i = 1 \tag{2.5}$$

We have supported the SPS module with several well-known algorithms for steady state analysis such as Gauss Seidal, Gauss Jordan and successive over-relaxation. Also, several algorithms for transient analysis such as Runge Kuta, and uniformization have supported the SPS module. Depending on the nature of the net (i.e. values of its Q elements), the user may choose any of the previously mentioned techniques using items of the solver menu. Using $\pi(\theta)$ and $\bar{\pi}$, the user can compute various performance, reliability and availability estimates of the modeled system by programming the PM

modules. The execution of PM programs is activated using items of the performance menu. Finally, the MM module is responsible for allocating enough memory blocks for the different modules of the package. The MM module also allows clever retrieval and storage of data through these memory blocks.

Although several packages have been implemented (under Sun OS and Linux) to deal with SPNs such as: SPNP [20], GreatGSPN [9] and DSPNExpress [21], our package has the ability to work with MS Windows. This advantage enables our package to interact with other MS packages. Another advantage of our package is its ability to manage the full memory of the PC. Hence, our package performs on the PC as other packages perform on other platforms (e.g. Sun platforms). We will not go further in the description of our package, since this is not the objective of this paper.

3. The Disk-Based Segmentation Technique

In this section, we propose the DBS technique for solving the fill-in problem. This technique depends on modifying the classical GE method in which the whole matrix is reduced to its upper triangular form by eliminating non-zero elements under the matrix diagonal. The basic concept of the DBS technique depends on splitting the whole matrix into a number of vertical segments K. Each vertical segment consists of a band of successive columns. To obtain the upper triangular form of the matrix, the DBS technique transforms each vertical segment (denoted by VS_l; $1 \le l \le K$) to the *Segmental Upper Triangular* (SUT) form. The path of the original matrix diagonal through the segment VS_l determines the SUT form of this segment. Since the matrix diagonal traverses each $VS_l \in VS \equiv \{VS_l;\ 1 \le l \le K\}$ in a different position, each VS_l will have its own SUT form. The SUT form of VS_l can be defined as follows.

$$VS_l(x,y) = \begin{cases} = 0 & \text{If } (Col_{bl} \le x \le Col_{el}),\ (Col_{bl} \le y \le x-1) \\ = 0 & \text{If } (x > Col_{el}) \\ = \text{any value} & \text{otherwise} \end{cases}$$

Where

$VS_l(x,y)$: represents the value of the VS_l element at row x and column y in the matrix Q^T;

Col_{bl}: represents the first column in the segment VS_l;

Col_{el}: represents the last column in the segment VS_l.

The SUT form of the segment VS_l can be obtained by performing a *reduction analysis* (RA) on this segment. The RA of VS_l means the elimination processes of non-zero elements under the path of the matrix diagonal through VS_l. Applying subtraction operations between the rows of these non-zero elements and the corresponding diagonal rows after scaling the diagonal rows can perform these elimination processes. By diagonal rows, we mean the rows of the vertical segment intersecting the path of the diagonal. Computational operations performed during the RA of VS_l are stored on the *hard disk* (denoted by HD). We call these operations, the *history list* of VS_l (denoted by HL_l).

Moreover, before applying the RA of vertical segment VS_l, we must first apply on VS_l the computational operations (division and subtraction operations) that have been performed during the RA of previous vertical segments, i.e. HL_i; $1 \le i \le l\text{-}1$. We call this step, the *history analysis* of VS_l (denoted by HA_l). Accordingly, the z-*th* history analysis of VS_l means applying the computational operations (division and subtraction operations) that have been performed during the RA of $VS_z \in VS_i$; $1 \le i \le l\text{-}1$ to the current segment VS_l. The reason for this step can be explained as follows. Each vertical segment contains segments of all rows (e.g. row_i) of the original matrix $\mathbf{Q}^T$ (transpose of $\mathbf{Q}$). Hence, the computational operations performed on one segment of row_i in one vertical segment VS_j should be applied also to the other segments of row_i in the vertical segments VS_l; $l > j$.

Figure 2 illustrates the steps of the DBS technique, where the DBS technique is applied on a matrix split into three vertical segments. The DBS scheme can be summarized in the following steps.

Algorithm: The DBS technique steps

Step 1. Develop the full matrix $\mathbf{Q}^T$ ($N \times N$) from the RG of the desired model (see Fig. 2.a).

Step 2. Partition the matrix $\mathbf{Q}^T$ into a number K of vertical segments (Figs. 2.b.1, 2.c.1, and 2.d.1). The set of vertical segments **VS** is defined as $\mathbf{VS} = \{VS_1, VS_2, ..., VS_K\}$. This set gives the whole matrix $\mathbf{Q}^T$. Each vertical segment VS_l consists of N rows $\times$ M columns (Fig. 2.b.1).

Step 3. Store each VS_l in a file on the HD. The storage processes of the vertical segments have been implemented in a suitable manner so that the overhead of retrieving them from the HD is minimized.

Step 4. Perform the following operations on the segments VS_1, VS_2 ,..., VS_K to transform the full matrix Q^T into its triangular form ∇Q^T.
For $l = 1$ *To K do*

a) Retrieve VS_l from the HD to the RAM. This RAM represents the available memory that is used for applying our DBS technique on the desired model. The determination of RAM can be done according to the resources of the platform that is used to run the algorithm.

b) Use the following loop to perform the HA_l.
For $z = 1$ To l-1 do
Perform the *z-th* history analysis of VS_l, or apply the computational operations contained in HL_z to VS_l.
Figure 2.c.2 shows the application of the computational operations contained in HL_1 to VS_2.

c) Use the RA to transform VS_l into its SUT form.
Figure 2.c.3 shows the SUT form of VS_2 after the application of its RA.

d) Save a list of all operations that have been done during the RA of VS_l to HL_l.

e) Store the final SUT form of VS_l on the HD in a separate file.

Step 5. Calculate the steady state probabilities π_0, π_1, ..., π_{N-1} for the states of $RS=\{S_0, S_1, ..., S_{N-1}\}$ by retrieving the rows of ∇Q^T [17, 18]. Notice that the elements of any row of ∇Q^T are now present in the vertical segment files. Hence, to retrieve the elements of a certain row, we must retrieve its corresponding elements from all vertical segment files.

Remark 3.1. The history analysis of VS_1 implies no computational operations applied on it, or no HA is applied on VS_1.

Remark 3.2. We only need to apply the GE method on a few number of columns instead of all columns (i.e. one vertical segment at a time) of the whole matrix. Therefore, we obtain a fewer number of new elements (zero elements are changed to be non-zero ones).

This minimization has the advantage of substantially reducing the effect of the fill-in problem. Thus, our methodology is able to enhance the capability of RAM to accommodate large matrices.

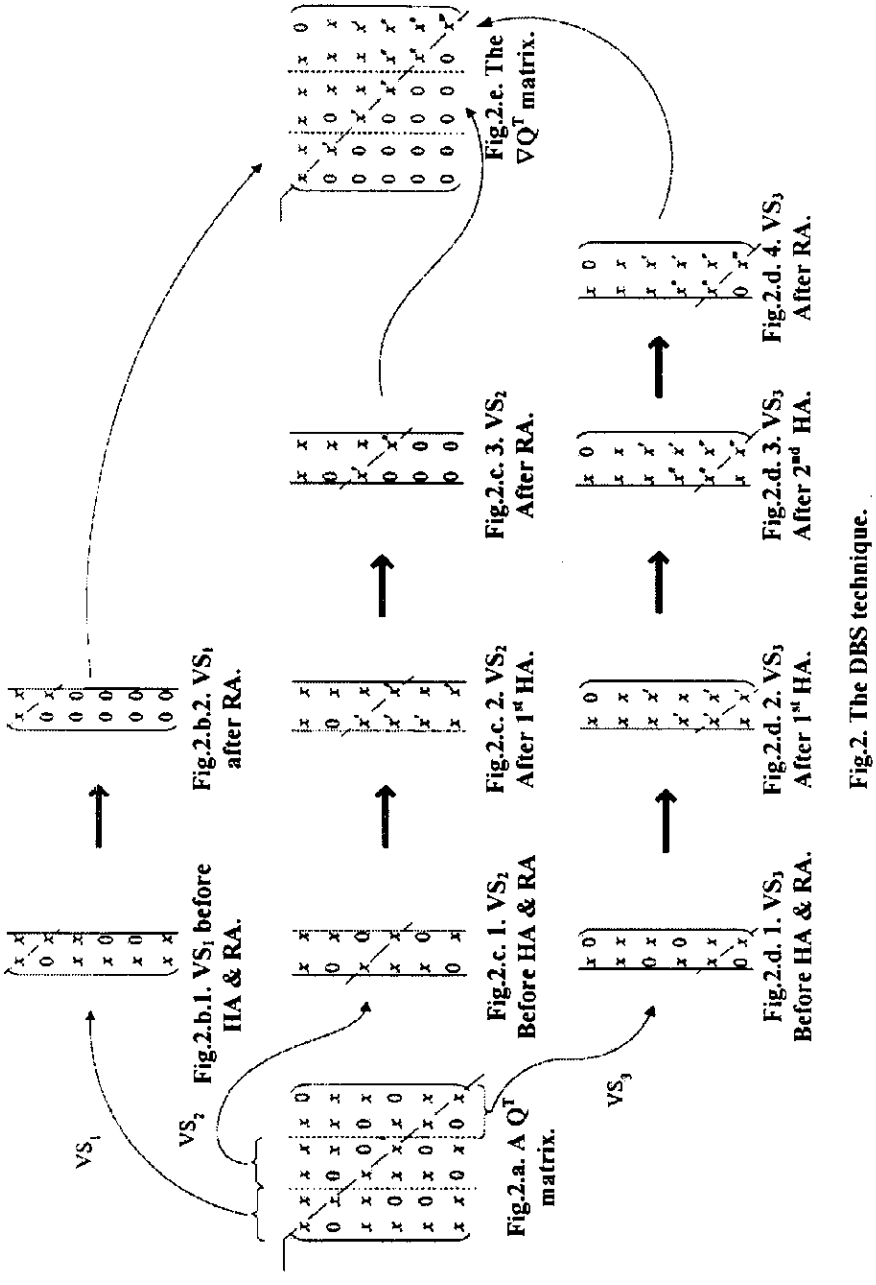


Fig.2. The DBS technique.

From the previous discussion, we remark that the DBS technique manages both the RAM and HD during its run-time. This management may result in many problems affecting the overall performance negatively. We have implemented the DBS technique in an intelligent way to allow minimizing considerably the negative impact of these problems on the desired performance measures. In the following section, we discuss several of the implementation issues for the DBS technique.

4. Implementation of the DBS Technique

In this section, we explain how the DBS technique is implemented to obtain the performance measures of the modeled system. To fulfill this objective, we have encountered several obstacles. The first of these obstacles stems from the enormous time consumed due to the back-and-forth movement of the vertical segment of non-zero elements during the RA and HA computations. In Section 4.1, we give a brief description of this obstacle. In addition, we present the technique that is utilized by the DBS technique to eliminate these movements. Further, since the RAM may suffer from overflow due to the exponential increase of the vertical segment of non-zero elements, we explain in Section 4.2 how the number of vertical segments K can be determined. Finally, in view of the fact that the history list will be accessed excessively during the HA of any vertical segment consuming a lot of time, Section 4.3 introduces an intelligent way for managing the records of the history list.

4.1 Implementation of the reduction and history analysis

The computation operations performed during the RA and HA of any vertical segment (e.g. VS_l) usually incorporate the following two processes. First, the insertion process of new elements among the elements of $VS_l \in VS$, e.g. Fig. 2.c.2. Secondly, the deletion process of the elements situated under the path of the diagonal (i.e. change of non-zero elements into zeros), e.g. Fig. 2.c.3. Insertion or deletion processes of elements in each $VS_l \in VS$ require back-and-forth displacements of these elements in the RAM. These displacement processes consume a lot of time. These processes represent the main drawback of the DBS technique. Therefore, in this section, we illustrate how the RA and HA can be implemented in an intelligent way to allow eliminating these back-and-forth displacements.

To achieve this objective, we have been motivated by the *Successive Row Generation and Reduction* (SRGR) technique [17, 18]. This technique transforms the whole matrix into its upper triangular form by reducing the rows, one by one, to their *Upper Triangular Row* (UTR) form. The zero elements of the UTR form can be defined for row_i as follows.

$$\text{Row}_i(j) = 0 \qquad \text{for } j < i \tag{4.1.1}$$

Where, $\text{row}_i(j)$ is the jth element of row_i. To transform any row_i into its UTR form, the SRGR technique retrieves row_i from the HD to the desired buffer. Then, the SRGR transforms the retrieved row into its UTR form by subtracting this row in a scaled form from the previous rows (row_l; $1 \le l \le i\text{-}1$) stored in the RAM. The reduced form of row_i is then stored in the RAM.

In order to avoid handling whole rows of the matrix, we have used a modified version of the SRGR technique. We call it, the MSRGR technique. Let row portions denote the rows of each vertical segment. Let row_{il}; $0 \le i < N$ denotes the row portion of a vertical segment VS_l. The MSRGR technique helps us to reduce the row portions to its *Segmental Upper Triangular Row* (SUTR) form. The zero elements of row_{il} in its SUTR form are defined as follows.

$$\text{row}_{il}(j) = 0 \quad \text{for} \quad (j < i, \text{Col}_{bl} \le j \le \text{Col}_{el}) \tag{4.1.2}$$

Where $\text{row}_{il}(j)$ is the element at the *jth* column of row_i. The definition of the SUTR form follows from the fact that each segment VS_l is constructed from a set of N row portions (i.e. $\text{VS}_l \equiv \text{row}_{il}$; $0 \le i < N$) which extend from column Col_{bl} to column Col_{el}.

To achieve the transformation of rows of each vertical segment to its SUTR form, we first retrieve row_{il} from the HD to the desired buffer. Next, we perform on row_{il} the computation operations that have been done during the reduction processes of row_{i0}, row_{i1}, and $\text{row}_{i,(l-1)}$ to transform them to their SUTR forms. We call such operations, the history analysis of row_{il}. In this way, row_{il} is ready to be transformed into its SUTR form using the previously transformed rows (row_{0l} to $\text{row}_{i-1,l}$) that have been stored in the RAM. We call such transformation, the *reduction analysis* of row_{il}. After reducing the row_{il} into its SUTR form, we store it in the RAM.

Algorithm: Re-implementation processes of step 4 of the DBS technique (explained in Section 3) according to the MSRGR technique described above.

Step 4. Perform the following operations on the segments VS_1, VS_2, VS_3 ,...,VS_K to transform the full matrix Q^T to its triangular form VQ^T. Note that Fig. 3 shows

the execution processes of the combined RA and HA for VS_2 of the matrix shown in Fig. 2.a.

For l =1 To K do

(a) Retrieve $row_{0l} \in row_0$ of VS_l from the HD to the RAM (see Fig. 3.b for VS_2).

(b) Retrieve $row_{11} \in row_1$ of VS_l from the HD to the reduction buffer (see Fig. 3.b for VS_2).

(c) For i = 2 To N-1, $l \geq 2$ do

(i) Apply the history analysis of row_{il}.

(ii) Transform the row_{il} to its SUTR form by applying reduction analysis.

(iii) Transfer row_{il} from the reduction buffer to the RAM, as shown in Fig. 3.

(iv) Register all computational operations performed during the RA of row_{il} in a *portion of the history list* (denoted by PHL_{il}), see Figs. 3.b and 3.f.

It is interesting to note that not all the regions of the row portions are subjected to RA. To clarify this statement, we define the following terms. Let the *upper diagonal region* be the region existing above the diagonal path through any vertical segment. Let the *diagonal region* be the region intersecting the diagonal path. Let the *lower diagonal region* be the region existing below the diagonal path. From our analysis, we have remarked the following interesting points. If we split the vertical segment VS_l into these three regions, the row portions of the upper diagonal region (row_{il}; $0 \leq i \leq Col_{bl}$) will not be subjected to the reduction analysis. On the other hand, the row portions of the diagonal region (row_{il}; $Col_{bl} < i \leq Col_{el}$) and the lower diagonal region (row_{il}; $i > Col_{el}$) will be subjected to the reduction analysis. In Figs. 3.c, 3.d and 3.e, we describe the application processes of the RA to the row portions: row_{12}, row_{32}, and row_{52}, as examples of these regions. The results of this application can be illustrated as follows. The result of the RA of the row portions of the lower diagonal *region* is the complete elimination of its elements as shown in Fig. 3.e for row_{52} of VS_2. On the other hand, the result of the RA of the row portions of the diagonal region is the partial elimination of its elements as shown in Fig. 3.c for row_{32} of VS_2.

4.2 Estimation of the suitable number of vertical segments

The proper operation of the DBS technique depends, largely, on the choice of the suitable K that prevents RAM overflow. This overflow occurs due to the enlargement of the vertical segment sizes during their transformation to their SUT forms. We have supported the DBS technique with a utility that allows the prediction of the final sizes of the vertical segments with the help of which a suitable K can be determined according to

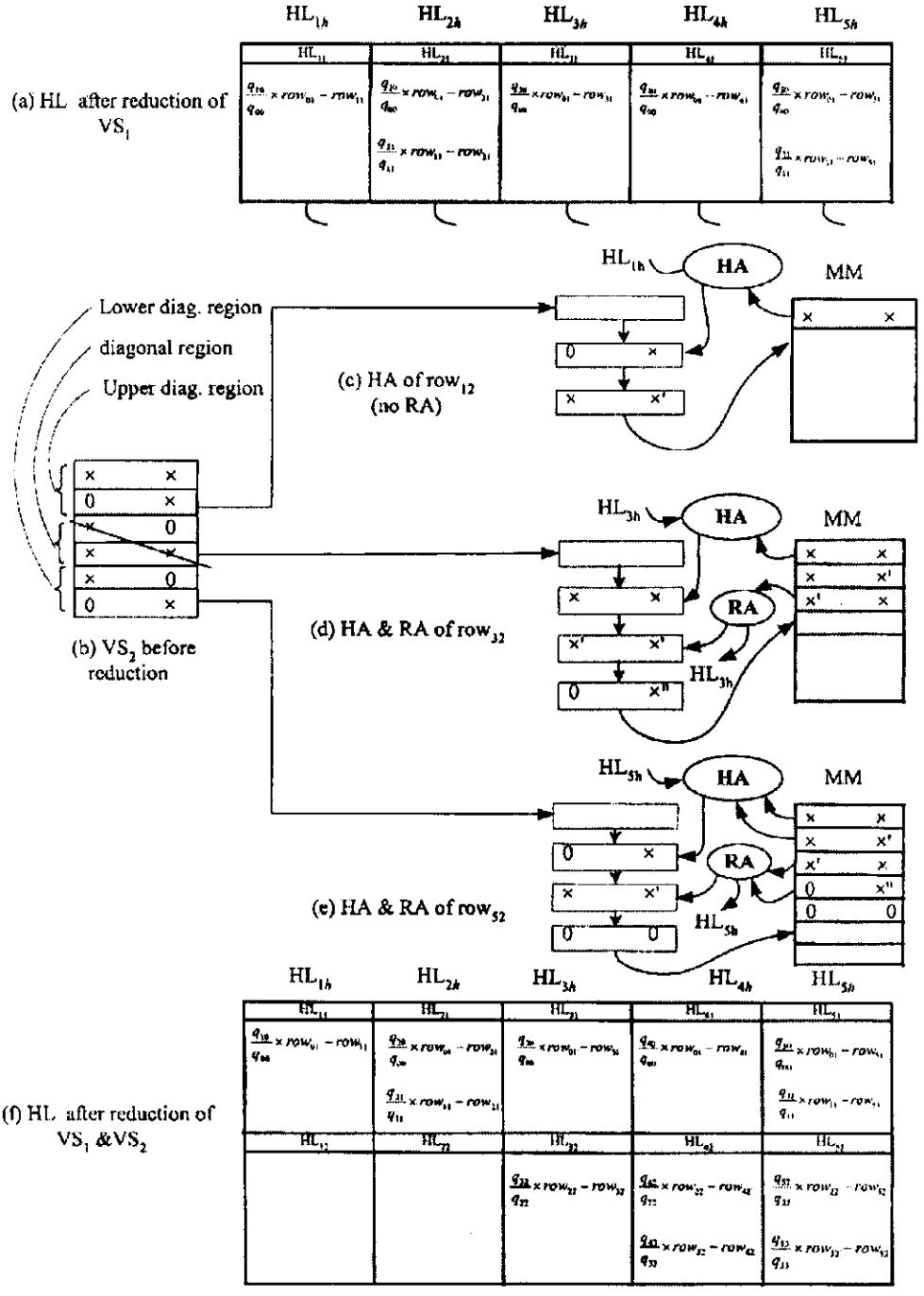


Fig. 3. Reduction of VS_2 using MSRGR technique.

the available RAM. In this section, we describe the implementation of this utility. The problem of choosing K stems from the fact that the transformation of a vertical segment into its SUT form results in an exponential increase of its non-zero elements. These newly generated elements may cause RAM overflow, or the RAM may not be able to accommodate the enlarged size of some vertical segments. To avoid memory overflow, one may choose to increase K. When we increase the parameter K, the final sizes of the vertical segments after transformation decrease. This decrement in turn may prevent memory overflow. Unfortunately, increasing the parameter K leads to the increase of the time needed to complete the solution due to the overhead processing needed to manage the history list file. Therefore the *minimum number* of vertical segments (denoted by K_{min}) that prevents the memory overflow, must be determined carefully. Yet, choosing K requires the knowledge of the final sizes of vertical segments to determine whether the RAM will accommodate these sizes. Since the vertical segment's final forms are successive groups of columns of ∇Q^T, the estimation processes of the final sizes of vertical segments can be achieved by estimating the non-zero elements in all columns of $\nabla \mathbf{Q}^T$. We have made use of the following lemma to estimate the number of non-zero elements in any *column j* (denoted by Col_j).

Lemma:
Let e_{ij}; $0 \le i,j \le N-1$ denotes the non-zero element of Col_j and row_i. Let u_j be the row index of the uppermost non-zero element of Col_j. Define the following formula as the set of non-zero elements of Col_j in the transition matrix Q^T.

$$Col_j \equiv \{e_{ij};\ i \ge u_j\} \tag{4.2.1}$$

After applying the GE method to the matrix Q^T, the number of elements of any Col_j can be calculated by the following formula.

$$NE_j \cong |u_j - j + 1| \tag{4.2.2}$$

Where, u_j is the row index of the uppermost element of Col_j.

Proof:
To prove this lemma, we use the simple SRN model shown in Fig. 4.a. The matrix Q^T (14×14) generated from this model is shown in Fig. 4.b. Simulate the effect of the GE method on the elements of the matrix Q^T shown in Fig. 4.b. The GE method transforms the matrix Q^T ($N \times N$) into its upper triangular form in N steps. In the step i, the GE method uses row_i to eliminate the non-zero elements existing in Col_i under the diagonal element e_{ii}. Accordingly, to simulate the effect of the GE method on the elements of the

matrix Q^T shown in Fig. 4.b, we need to monitor the elimination operations that the GE method performs on the elements of the matrix Q^T in each of its N steps. Assume we first monitor the second step performed by the GE method on the elements of the matrix Q^T (i.e. eliminating the non-zero elements existing in the Col_1 under the diagonal element e_{11}). Actually, this step can be performed using row_1 to act upon the row_2, row_3, and row_6 in order to eliminate the set of elements $\{e_{21}, e_{31}, e_{61}\}$ existing under e_{11}. However, eliminating this set of elements induces the following elements: $\{e_{23}, e_{28}, e_{29}\}$ in row_2, $\{e_{38}, e_{39}\}$ in row_3, and $\{e_{68}, e_{69}\}$ in row_6. A considerable part of these induced elements will be situated in the gap between the uppermost elements of Col_3, Col_8 and Col_9 (i.e. e_{13}, e_{08}, e_{19}), and the corresponding diagonal elements (i.e. e_{33}, e_{88}, e_{99}). Figure 4.c shows the execution of the second step performed by the GE method on the elements of the matrix Q^T, where e_{23} is generated in the gap between e_{13} and e_{33}. Also, $\{e_{28}, e_{38}\}$ will be induced in the position of zero elements present between e_{08} and e_{88}. Moreover, $\{e_{29}, e_{39}, e_{69}\}$ will be situated in the position of zero elements in the gap between e_{19} and e_{99}.

Finally, there are several non-zero elements that will be induced in the row_2 and row_3 during the elimination step $\{e_{21}, e_{31}\}$ due to the following reasons. The elements of the matrices $Q \equiv \{q_{ij}\}_{N \times N}$ (or Q^T) resulting from systems modeled by the Petri net (PN) theory exist near the diagonal [9], [20]. Thus, most elements of any row of the matrix Q (or Q^T) will exist near the matrix diagonal. Consequently, the non-zero elements of any column will exist in several neighboring rows. Subsequently, applying any step i of the GE method on the matrix Q^T will induce non-zero elements in several neighboring rows to row_i. A part of these induced elements will reduce the gap between the uppermost elements and the corresponding diagonal elements, while the other part will be situated under the matrix diagonal to be eliminated through subsequent elimination steps. Hence, we deduce that the third step performed by the GE method on the matrix Q^T will induce new non-zero elements in neighboring rows to row_3. Moreover, the elements induced in row_3, during the second step of the GE method, will produce more non-zero elements in the gap between the uppermost elements of row_3, row_8, and row_9 as well as the corresponding diagonal elements. Figure 4.d shows the execution of step 3 of the GE method on the matrix Q^T, where e_{28} (induced in step 2 of the GE method) produces e_{48} and e_{78}. Also, e_{29} (induced in step 2 of the GE method) produces e_{49} and e_{79}. This process further tightens the gap between e_{08} and e_{88} as well as e_{19} and e_{99}.

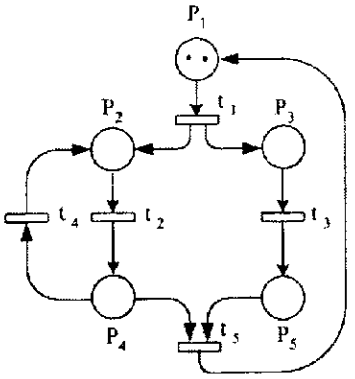


Fig.4.a. Simple SRN net.

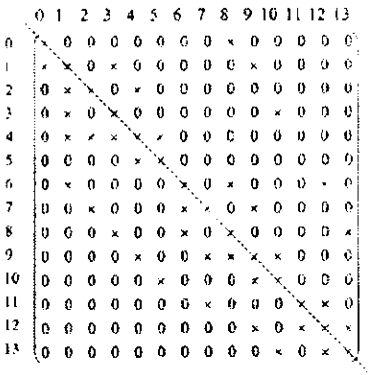


Fig.4.b. The Q^T matrix.

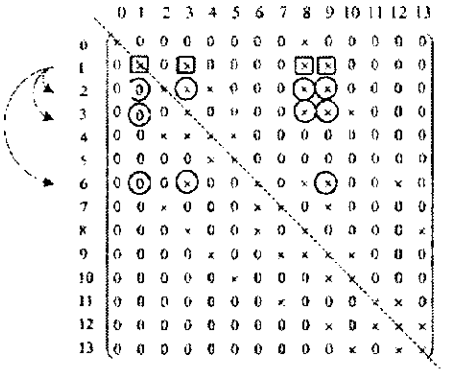


Fig.4.c. The effect of step 2 of GE.

□ : original element of Q^T.
○ : element induced due to step 1
● : element induced due to step 2

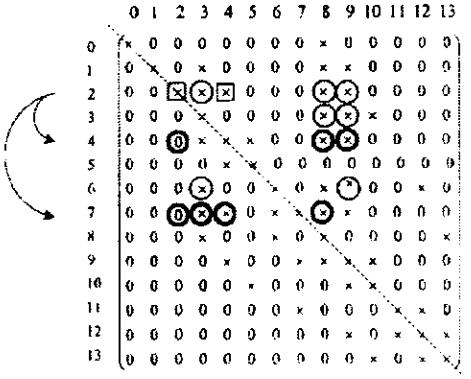


Fig.4.d. The effect of step 3 of GE.

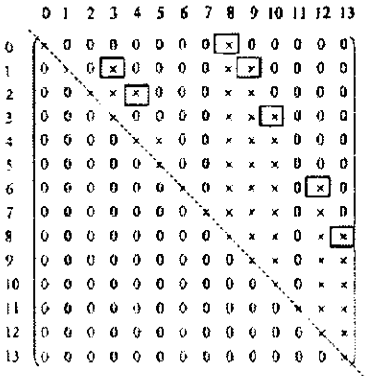


Fig.4.e. The ∇Q^T matrix.

Fig.4. Simulation of the effect of GE on Q^T.

Since the execution of GE steps gradually reduces the gaps between the uppermost elements and the corresponding diagonal elements, the number of non-zero elements in Col_i of ∇Q^T can be calculated by the following formula.

$$NE_i \cong | u_i - i+1 |$$

Where, u_i is the row index of the uppermost element of Col_i.

From our experience with the SRN models and their generated Q^T matrices, we have found that the estimating processes of the 14×14 matrix are still valid for the Q^T matrices generated from most SRN system models. This result is due to the previously mentioned property: the elements of the Q^T matrices resulting from the PN models exist near the diagonal. In Section 5, we illustrate the validity of our estimating processes of large Q^T matrices.

From our analysis, we have found that the calculation of NE_i for any Col_i of ∇Q^T depends only on its uppermost elements. Thus, we can easily get a set NE=$\{NE_0, NE_1,..., NE_{N-1}\}$, where NE_i carries the number of non-zero elements in the corresponding column of ∇Q^T. Having the set NE, one can calculate the minimum number of vertical segments (K_{min}) that can be accommodated by a given RAM. In this algorithm, we take into consideration that any non-zero element of a vertical segment needs 12 bytes of memory (4 bytes for the column index, and 8 bytes for the element storage). The following algorithm illustrates how the set NE supports the calculation of the parameter K_{min}.

Algorithm: Calculation of K_{min} for a given RAM

Step 1. Initialize the minimum number of vertical segments such that K_{min} =1

Step 2. Calculate the number of non-zero elements in vertical segments ($no\_elem\_VS_i$) through the following loop.

For i =1 To K_{min}

(i) Calculate Col_{bi} and Col_{ei} of VS_i according to

$$no\_col\_VS = \lfloor N / K_{min} \rfloor$$

Where no_col_VS is the number of columns of a vertical segment VS_i.

$\lfloor x \rfloor$ is the maximum integer less than x.

Hence

$$Col_{bi} = (i\text{-}1) \times no\_col\_VS$$
$$Col_{ei} = i \times no\_col\_VS - 1, \quad \text{for } i < K_{min}$$
$$Col_{ei} = N\text{-}1, \text{ for } i = K_{min}$$

(ii) Calculate $no\_elem\_VS_i$ according to the following equation:

$$no\_elem\_VS_i = \sum_{i=Col_{bi}}^{Col_{ei}} NE_i$$

If $(no\_elem\_VS_i \times 12) \geq$ RAM size,

Then Go To Step 3

Else
- $K_{min} = K_{min} + 1$
- $i = 1$
- Go To Step 2

Step 3. Display K_{min}

4.3. Management of history analysis

As we have explained in Section 4.1, the HA must be performed on each row portion. Consequently, during the HA of row portion row_{il}, we must perform on row_{il} all the computational operations done during the RA of the previous row portions $S \equiv \{row_{ij};\ 1 \leq j \leq (l\text{-}1)\}$. On the other hand, many of these computational operations may not to be needed during the HA of a row portion. Since the records of these computational operations are present in the form of the history list (Section 3) on the HD, the determination processes of their applicability need to retrieve them from the HD.

In this section, we present the *management technique* utilized by the HA to guess the applicable operations without the need to access the HD. The main advantage of the proposed management technique is the elimination of time consumed during the accessing processes of the HD for inapplicable operations. In addition, we get rid of the burden of performing these inapplicable operations. To clarify how our proposed management technique performs the HA of row portions of a vertical segment, let row_{im} be one of the row portions of VS_m. Also, we assume that the HA for the row_{im} is required. It is interesting to note that the row_{im} may be situated in the upper diagonal region, or the diagonal region, or the under diagonal region with respect to VS_m. The

position of row_{im} greatly affects the way of handling its HA by the proposed management technique. In the following, we illustrate how the proposed management technique handles the history list records corresponding to the row portions of the previously mentioned regions. For this purpose, let us define the following. Let the *non-zero row portion* be the row portion that some of its elements are non-zero. Let the *zero row portion* be the row portion that some of its elements are zero.

To illustrate how the HA of the zero row portion and the non-zero row portion can be managed, suppose that the row_{im} is situated in the upper diagonal region of VS_m (i.e. $0 \leq i < Col_{bm}$). The HA of the non-zero row_{im} can be illustrated throughout applying on row_{im} all the computational operations registered in the history list corresponding to row_{ij}; $0 \leq j \leq m-1$. From our experience with the matrix Q generated from the PN system models, we have found that the non-zero elements of any row in this Q are confined to the area near the matrix diagonal. Also, each row_i has a set of leftmost zero row portions given by the set $A \equiv \{row_{ij} ; 0 \leq j < l_i\}$, where row_{il_i} is the leftmost non-zero row portion of row_i.

Since the row portions of A are zero row portions, the members of A will not be subjected to any RA. In turn, no computational operations will be registered in the history list corresponding to the set A. On the contrary, we expect that the history list will include records of computational operations corresponding to the non-zero row portions of the set $B \equiv \{row_{ij} ; l_i \leq j < d_i\}$, where row_{id_i} is the non-zero row portion situated in the diagonal region of VS_{d_i} (i.e. $Col_{bd_i} \leq i \leq Col_{ed_i}$). This expectation is due to the fact that all the members of B are situated under the matrix diagonal. Hence, B will be subjected to RA.

From the above discussion, we conclude the following. During the HA of row_{im}, we only need to apply on row_{im} the set of operations stored in $S \equiv \{PHL_{ij} ; l_i \leq j < d_i\}$, where PHL_{ij} is a part of the history list carrying the group of computational operations performed on row_{ij} during its RA (as shown in Section 4.2). Having the set S, we can define the *history domain* of row_{im} (denoted by Dom_{im}) as follows.

$$Dom_{im} \equiv \{j ; l_i \leq j \leq d_i\} \tag{4.3.1}$$

We can then designate l_i and d_i as the left and right history domain sides of row_{im} and denote them as $LDom_{im}$ and $RDom_{im}$ respectively. The HA of

$row_{im} \in \{row_{jm}, 0 \le j < u_m\}$ implies all the computational operations that have been performed on row_{im}. These operations have been stored in the history list corresponding to row_{im}. As shown in Section 4.1, these computational operations have been done between row_{im} and the rows higher than row_{im} (i.e. between row_{im} and row_{jm}; $1 \le j < i$). Since row_{im} and the rows higher than row_{im} are all zero row portions, the history domain of row_{im}; $0 \le i < u_m$ can be assumed to be equal to ϕ, where ϕ is the empty set. Thus, the history domain of the zero row portions extending higher than the uppermost row portion of VS_m is equal to ϕ.

In the following, we illustrate the processes of the HA of zero row_{im}. To perform these processes, we use the same processes that have been applied on the non-zero row_{im}. In this case, l_i and d_i will be also the history domain sides of row_{im}. Throughout the HA of zero row_{im}, we need to apply on row_{im} the computational operations stored in S. Yet, the state of row_{im} will be only changed if the computational operations stored in PHI_{ij}; $l_i \le j \le d_i$ are performed using the non-zero row portions. This constraint can be accomplished by redefining the history domain of the zero row_{im} as follows.

$$Dom_{im} \equiv \{ j ; l_i \le j \le d_i , Col_{ej} \ge u_m \} \qquad (4.3.2)$$

The PHL_{ij}; $l_i \le j < d_i$ contains the records of computational operations performed between the members of row_{km}; $Col_{bj} \le k \le Col_{ej}$ and the row_{im}. Consequently, the condition $Col_{ej} \ge u_m$ guarantees that the set row_{km}; $Col_{bj} \le k \le Col_{ej}$ contains the non-zero row portions. This condition acquires a great validity due to the following reason. From our experience with the matrix Q generated from the PN system models, most of the non-zero elements of a vertical segment are concentrated in the diagonal region rows. The presence of these elements decreases significantly as we move up and down through the rows of the upper and lower diagonal regions, respectively. Moreover, in large matrices of practical systems, the non-zero elements vanish completely in most rows of the upper and lower diagonal regions.

Having finished from the analysis of the upper diagonal region, suppose the row_{im} is situated in the diagonal region of VS_m (i.e. $Col_{bm} \le i \le Col_{em}$). According to equations 4.3.1 and 4.3.2, we can easily derive the history domain of row_{im} as follows.

$$Dom_{im} \equiv \{j; l_i \le j \le m-1\} \qquad (4.3.3)$$

It must be noted that all the row portions of the diagonal region are of the non-zero kind. Hence, there is no need to discuss the zero row portions.

Finally, suppose row_{im} is situated in the lower diagonal region of VS_m (i.e. $\text{Col}_{em} < i < N$). Row portions of these regions can be non-zero or zero type. The history domain of non-zero row_{im} is the same as that of equation 4.3.3. The history domain of zero row_{im} is defined as follows.

$$Dom_{im} \equiv \{j \; ; \; l_i \le j \le m\text{-}1 \; , \; \text{Col}_{ej} \ge u_m\} \tag{4.3.4}$$

In the following algorithm, we explain how our management technique uses equations 4.3.1- 4.3.4 to perform the HA of row_{im}.

Algorithm: History Analysis of row_{im} of $\text{VS}_m(Dom_{im})$

Step 1. Determine the Dom_{im} of row_{im} through the following steps.

(a) If $(0 \le i < u_m)$ (the upper *diagonal* region), then

$Dom_{im} = \phi$

(b) If $(u_m \le i < \text{Col}_{bm})$ (the upper *diagonal* region), then

(i) Dom_{im} of zero $\text{row}_{im} \equiv \{j; \; l_i \le j \le d_i, \; \text{Col}_{ej} \ge u_m \}$

(ii) Dom_{im} of non-zero $\text{row}_{im} \equiv \{j; \; l_i \le j \le d_i \}$.

(c) If $(\text{Col}_{bm} \le i)$ (the *diagonal* region and the *lower diagonal* region), then

(i) Dom_{im} of zero $\text{row}_{im} \equiv \{j; \; l_i \le j \le m\text{-}1, \; \text{Col}_{ej} \ge u_m \}$

(ii) Dom_{im} of non-zero $\text{row}_{im} \equiv \{j; \; l_i \le j \le m\text{-}1\}$.

Step 2. Use the Dom_{im} of row_{im} to apply on row_{im} the set of operations $\mathbf{A} \equiv \{\text{PHL}_{ij} \; ; j \in Dom_{im}\}$ where, PHL_{ij} is a part of the history list carrying the computational operations performed during the RA of row_{ij}.

5. Numerical Results

To illustrate the merits of using the DBS technique, we have applied the DBS on the SRN model of the Kanban manufacturing system shown in Fig. 5. This model has been used previously in the literature to show the effectiveness of several largeness tolerance techniques [11, 12]. The SRN Kanban model is composed of four subnets [22].

At each subnet, a token enters, spends some time, and exits or restarts with certain probabilities. Once the token leaves the first net, it may enter the second or third subnet. The token must go through the fourth subnet to leave the system.

To solve the SRN Kanban model of Fig. 5, we have represented the synchronization transitions by the timed transitions. Assuming that the places p_{kanban_1}, p_{kanban_2}, p_{kanban_3} and p_{kanban_4} contain 3, 2, 2 and 2 tokens respectively, we get the matrix Q^T of dimension N×N, where N=18400 as shown in Fig. 6. In this figure, the presence of the matrix elements near the diagonal can be observed. Moreover, suppose that the matrix Q^T cannot be solved using the traditional iterative methods due to the stiffness of its rates. Consequently, the matrix Q^T can be managed only by using the direct methods such as the GE technique. Unfortunately, applying the GE technique on the matrix Q^T causes memory overflow even for 256 Mbytes of RAM (which is a common resource for many platforms). If the algorithm of Section 4.2 supports the DBS technique, the matrix Q^T can be solved using the DBS technique for $K \geq 5$ due to the following. The estimated memory used by the DBS technique is more than 256 Mbytes, which is beyond the available memory. Consequently, we have applied the DBS technique on the matrix Q^T for the values of K ranging from 20 to 200. For these variations of K, we have measured the time needed by the DBS technique to solve the matrix Q^T.

All the experiments done using the DBS technique are performed on an Intel Pentium III-550 MHz with 256 Mbytes of RAM and 20 Gbytes of HD. Moreover, for a certain number of vertical segments K, the size of the vertical segment SUT form varies from one vertical segment to another as shown in Fig. 7 for $K = 40$. Therefore, we are interested in monitoring the maximum size of these SUT forms. By allocating enough memory for this maximum size (denoted by $M_{max,K}$), the DBS technique has the capability to completely convert all the vertical segments to their SUT forms. Table-I shows the time (denoted by T_K) and the $M_{max,K}$ needed by the DBS technique to completely convert the vertical segments into their corresponding SUT forms for $20 \leq K \leq 200$.

It is interesting to note that the proposed DBS technique has been developed based on the modification of execution sequences of the GE procedure. Therefore for a given K, the size of the vertical segment SUT form (denoted by M_i; $1 \leq i \leq K$) is related to the size of the matrix Q^T (denoted by $M_{\nabla Q}$) by the following relation.

$$M_{VQ} = \sum_{i=1}^{K} M_i \qquad (5.1)$$

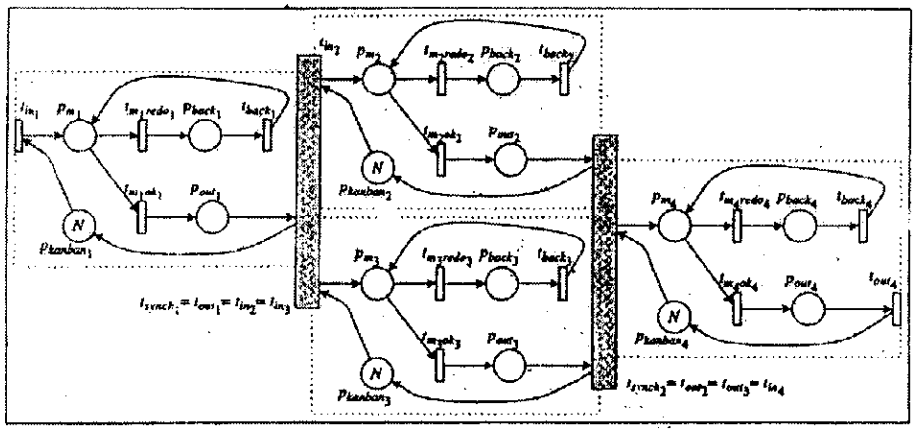


Fig. 5. The model of the kanban manufacturing system.

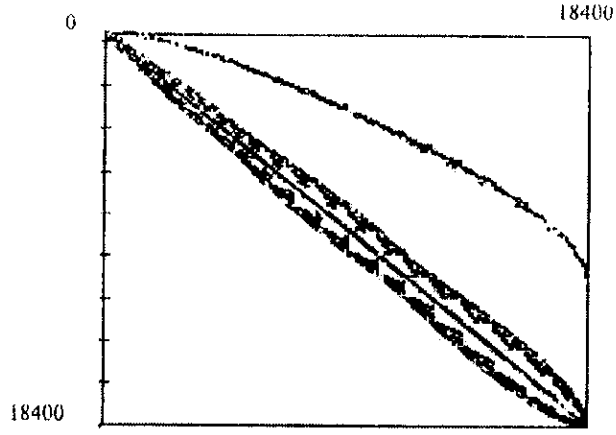


Fig. 6. The 18400x18400 Q^T of the kanban model.

Table 1. Time (T_k), maximum memory (M_k), percentage increase in time ($\%T_k$) and percentage reduction in memory ($\%M_k$) consumed by the DBS for $10 \leq k \leq 200$

K	0	10	20	40	60	80	100	120	140	160	180	200
T_k (sec)	13683	14863	15221	16302	17676	20209	20916	21135	22957	23266	24711	27480
$\%T_k$	0	8.62	11.24	19.14	29.18	36.92	52.86	54.46	67.78	70.04	80.60	100.83
$T_{max,k}$ (MB)	675.5	119.42	61.04	30.61	20.64	15.57	12.50	10.50	8.99	7.82	8.99	6.26
$\%M_k$	0	82.32	90.96	95.47	96.94	97.70	98.15	98.44	98.67	98.84	98.67	99.07

Using equation 5.1, the $M_{\nabla Q}$ has been found to be 675.5 Mbytes (i.e. 59,025,755 non-zero elements). The enormous size of this matrix $\mathbf{Q}^T$ indicates why the GE technique is not able to solve such a matrix. To illustrate the advantage of using the proposed DBS technique, we have calculated the reduction percentage in RAM needed by the DBS (denoted by $\%M_K$) for all variations of K. The percentage $\%M_K$ can be obtained using the following formula.

$$\%M_K \equiv \frac{(M_{\nabla Q} - M_{max,K})}{M_{\nabla Q}} \times 100 \qquad (5.2)$$

In Table 1, we illustrate the values of $\%M_K$ when $20 \leq K \leq 200$. From Table 1, we have remarked the following interesting points.

(i). $\%M_K$ increases as K increases;
(ii). For $K>60$, the change in $\%M_K$ is less than 1%; and
(iii). When $K=60$, $\%M_K$ reaches 96%.

Hence it will be satisfactory to use the DBS technique for $K \leq 60$. Regrettably, the increase in the value of $\%M_K$ is accompanied by an increase in the corresponding solution time (denoted by T_K) consumed by the DBS technique to solve the matrix Q^T. In the following, we calculate the percentage of increasing T_K (denoted by $\%T_K$).

$$\%T_K \equiv \frac{(T_K - T_0)}{T_0} \times 100 \qquad (5.3)$$

Where
T_K ($20 \leq K \leq 200$): the time consumed by the DBS technique to solve the matrix Q^T;
T_0 ($K = 0$): the time consumed by the GE technique to solve the matrix Q^T. The behavior of the DBS technique is the same as that of the GE technique when $K = 0$.

The advantages gained from our DBS technique can diminish greatly if $\%T_{60}$ has a comparable value to 96% of $\%M_{60}$. Yet, we have not been able to get T_0 because of the enormous size of the matrix Q^T preventing its solution, as previously stated. As an approximate solution for this problem, we have calculated T_0 by *extrapolating* the set $\{T_K;\ 20 \leq K \leq 200\}$, for $K=0$. To make our extrapolation more accurate, we have also measured T_K for $K=10$. Having T_0, we can compute $\%T_K$ for $20 \leq K \leq 200$ (see also Table 1). We have plotted in Figure 8, the change of $\%T_K$ and $\%M_K$ for $20 \leq K \leq 200$. From Figure 8, we can remark the following points. For $K=60$, while $\%M_K$ reaches 96%, $\%T_K$ reaches only 20%. When $K>60$, $\%T_K$ increases excessively and $\%M_K$ increases slightly (i.e. less than 1%). When $K \leq 60$, there is a remarkable difference

between $\%T_K$ and $\%M_K$ due to the effect of the proposed history management technique (discussed in Section 4.3).

To understand the effect of the proposed history management technique on the performance of the DBS, we must notice that the main difference between the GE and DBS techniques is the time consumed during the HD accessing. A part of the HD accessing occurs during the HA (denoted by ACC_{HA}). Another part occurs during the retrieval and storage of the vertical segment elements (denoted by ACC_{RS}). From our analysis, the value of ACC_{RS} is nearly constant during the variation of K due to the following reasons. The total number of the vertical segment elements is constant. Also, this number does not vary with K (see equation 5.1). The operations (stored in the history list) performed during the HA are the main cause of these elements. Thus, the total number of history list records accessed will be more or less constant. Subsequently, minimizing ACC_{HA} will greatly enhance the performance of the DBS technique. Indeed, we can perform the HA by accessing all the history list records. Yet, we need to access the HD several times.

Figure 7 shows the number of times the history list is accessed by the proposed history management technique for K=40. From this figure, we can remark the difference between the number of history list records and that of the history list records accessed for the vertical segments of K=40. To access the entire history list records, we need to access the HD 258,294,947 times (i.e. ACC_{HA} = 258,294,947) as shown in Fig. 7. On the other hand, the proposed history management technique (explained in Section 4.3) needs to access the HD 180,532,385 times (i.e. ACC_{HA}=180,532,385). Thus, the proposed history management technique achieves 30% reduction in ACC_{HA}. Moreover, ACC_{HA} is usually much greater than ACC_{RS}. For example, from Fig. 7, we found that while ACC_{HA} is equal to 180,532,385, ACC_{RS} equals 59,150,755. We can then state that ACC_{HA} constitutes nearly 75% of the time consumed during accessing the HD. Hence, we can conclude that ACC_{HA} affects greatly the performance of the DBS technique (i.e. time of solution). For each vertical segment $\{VS_i;\ 1 \le i \le K\}$, ACC_{HA} is controlled by the history domain range of the vertical segment VS_i (denoted by $DomR_i$).

$$DomR_i = max\{LDom_{ij};\ 0 \le j \le N\text{-}1\} - min\{RDom_{ij};\ 0 \le j \le N\text{-}1\} \tag{5.4}$$

Figure 9 illustrates that there is a proportional effect for the $DomR_i$ on the solution time of the vertical segments $\{VS_i;\ 1 \le i \le K\}$ when K=40. The effect of $DomR_i$ on the solution time of VS_i ensures the previous conclusion concerning the effect of

ACC_{HA} on the performance of the DBS technique due to the following reason. The increase of $DomR_i$ means the increase of the accessed history list records for any vertical segment VS_i.

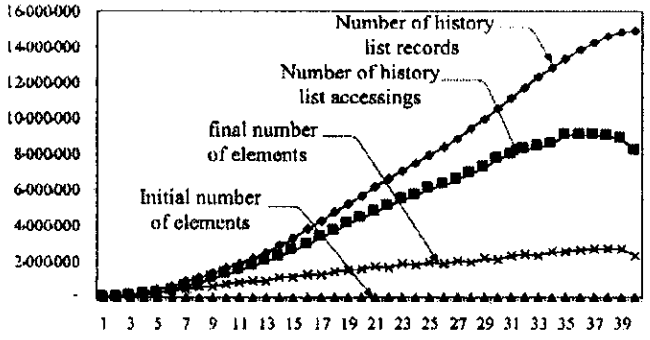


Fig. 7. Number of history list records and history list accessings, initial and final number of elements for vertical segments resulting from applying DBS for K=40.

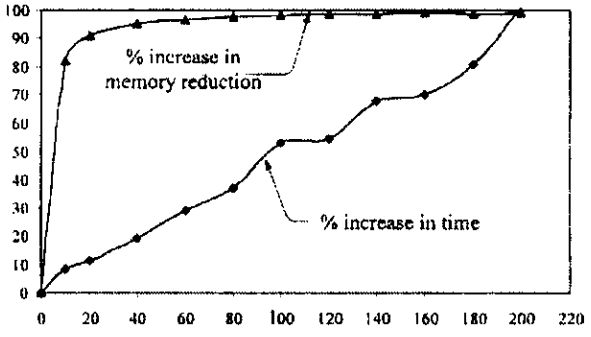


Fig. 8. % increase in memory reduction, % increase in time for k of the DBS varying from 20 to 200.

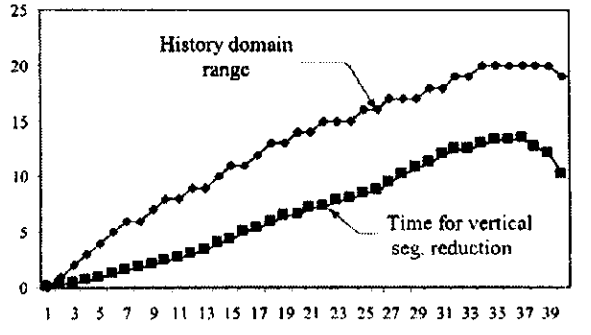


Fig. 9. Time for reduction and history domain range of vertical segments for K=40.

To gain more insight about the operation of the proposed management technique, we use the following example. Let us monitor the HA of VS_{10} and that of VS_{30} during the application of the DBS technique when K=40. As shown in Fig. 9, $DomR_{10}$ =7 and $DomR_{30}$ =17. The number of times the HD is accessed during the execution process of the HA of VS_{10} and that of VS_{30} on the set PHL_i; $1 \le i \le 40$ when K=40 is shown in Fig. 10. From our analysis of Figs. 9 and 10, we have observed the following. These accessing processes are performed on history list portions situated within the domain range of VS_{10} and that of VS_{30} (i.e. $\{PHL_i; 10\text{-}DomR_{10} \le i < 10\}$, $\{PHL_i; 30\text{-}DomR_{30} \le i < 30\}$). The number of times the history list records is accessed corresponding to nearby vertical segments is greater than that of farther vertical segments. For example, during the HA of VS_{30}, the accessing process to PHL_{29} is greater than that to PHL_{28} or PHL_{27}. This behavior is due to the fact that the non-zero elements of a certain row extend through several neighboring vertical segments. Subsequently, during the HA of VS_i, the history list of neighboring vertical segments will contain more applicable operations to row portions of VS_i than that of farther VS_i.

Furthermore, it is interesting to illustrate how the proposed management technique performs the HA of a row portion, as a basic operation of this technique. Fig. 11 shows the change of the left and right history domain sides for the row portions of VS_{10} and those of VS_{30} for K =40, (i.e. $LDom_{ij}$, $RDom_{ij}$; $0 \le i \le$N-1, $j \in (10,30)$). The effect of the matrix diagonal on these history domain sides can be noticed by the descending zigzag appearance of the diagrams shown in Fig. 11. Moreover, as shown in Fig. 12, the history domain of the row portions VS_{10} and that of VS_{30} has not exceeded 2 for VS_{10} and 3 for VS_{30}. Also, we can notice the close relation between Figures 11 and 12 according to equations 4.3.1- 4.3.4. This close relation is due to the fact that the elements of the matrix Q^T are confined to the area near the matrix diagonal. Subsequently, the leftmost row portion row_{il_i} of row_{ij} is situated nearby the vertical segment VS_{l_i} (i.e. $(j\text{-}l_i) << K$). Furthermore, the number of times the HD is accessed by each row portion on the history list is presented in Fig. 13. The effect of $Dom_{10,i}$ and $Dom_{30,i}$ on the number of times the history list is accessed can be noticed by comparing Figs. 12 and 13. From this comparison, the row portions that have an equal history domain access the history list in nearly equal number of times. Also, the row portions that have a greater history domain access the history list more times.

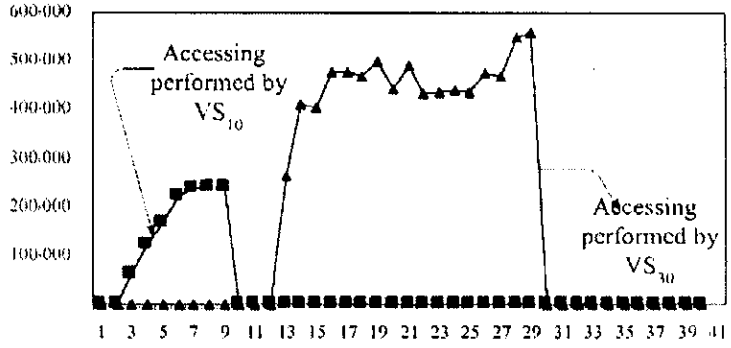


Fig. 10. Number of accessings performed by VS_{10} and VS_{30} on$\{HL_i;1\leq 40i \leq 40\}$ for K=40.

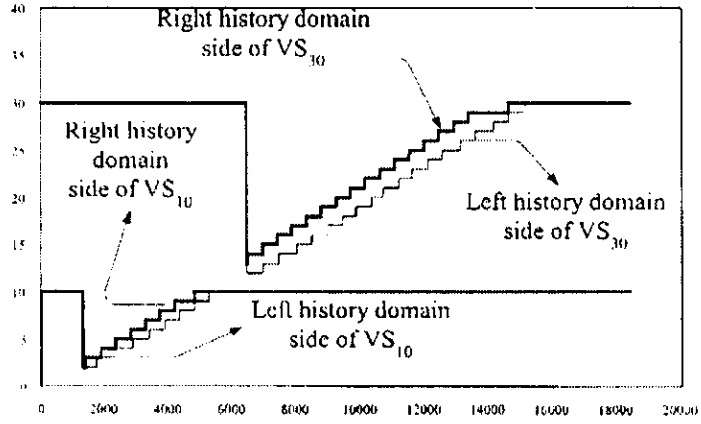


Fig. 11. Left and right history domain sides for row portions of VS_{10} and VS_{30} for K=40.

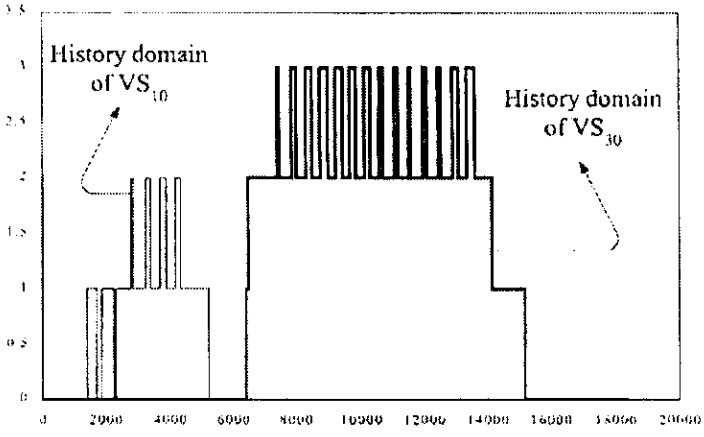


Fig. 12. History domain for row portions of VS_{10} and VS_{30} for K=40.

Using the proposed history management technique, the HA greatly affects the number of non-zero elements enclosed in a vertical segment VS_i. From Fig. 14, we can observe the exponential increase in the number of elements enclosed in VS_{10} and that of VS_{30}. Figure 14 illustrates the effect of the application of the DBS technique on the final number of elements that have been enclosed in the row portions of VS_{10} and that of VS_{30}. As shown in Fig. 14, the number of non-zero elements has reached an average value of 200 and 280 for row portions of VS_{10} and VS_{30}, respectively. On the contrary, the initial number of elements enclosed in the row portions of VS_{10} and that of VS_{30} (before application of the DBS technique) has not exceeded 6 for the row portions of VS_{10} and those of VS_{30}. The curve shown in Fig. 17 increases exponentially due to the application of the HA on the three previously-mentioned regions of VS_{10} and those of VS_{30}. Moreover, we can observe from Fig. 14 that the non-zero elements are only distributed throughout the row portions of the upper diagonal and the diagonal regions, while the non-zero elements present in the row portions of the under diagonal region are eliminated through the RA. Also, these non-zero elements are only present in a small subset of the row portions due to the property of the matrices that can be obtained from the PN models as previously mentioned.

From our analysis of Fig. 14, we can also observe that the final number of elements has a peak value for the set of row portions existing in the border between the upper diagonal and the diagonal regions. This peak in the number of elements is attributed to the peak present in the number of times the history list is accessed as shown in Fig. 13. The presence of this peak is due the distribution of the elements around the diagonal in the border between the upper diagonal and the diagonal regions. This distribution means the existence of many applicable records in the history list for row portions existing in the border between the upper diagonal and the diagonal regions. Consequently, applying the HA of row portions of the border will induce many elements for these row portions. The effect of RA can be shown in the sharp inclination in the number of elements for row portions of the diagonal region.

Finally, the effectiveness of the estimation criteria proposed in Section 4.2 can be observed by comparing Figs. 15.a and 15.b. In Fig. 15.a, the number of elements in each column of ∇Q^T is estimated according to the lemma of Section 4.2. In Fig. 15.b, the actual number of elements in each column of ∇Q^T is monitored from the columns of the vertical segments in their SUT forms. The resemblance between the two figures can be easily observed, where the error in estimation has not exceeded 0.001% for the 59,025,755 elements of ∇Q^T.

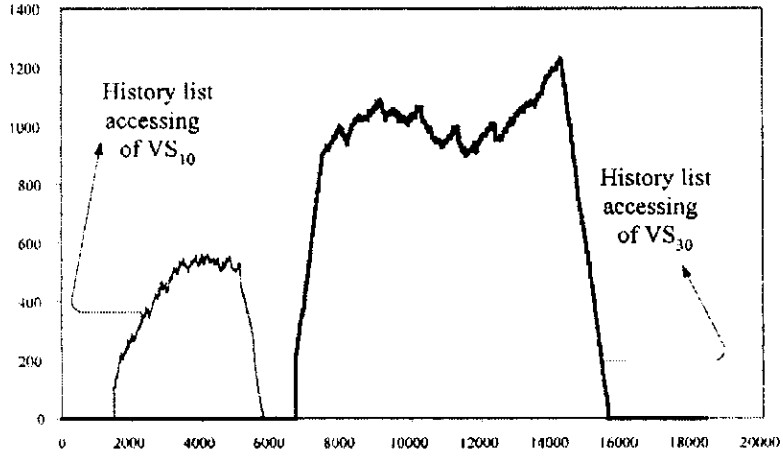


Fig. 13. History list accessings performed during HA of row portions of VS_{10} and VS_{30} for K=40.

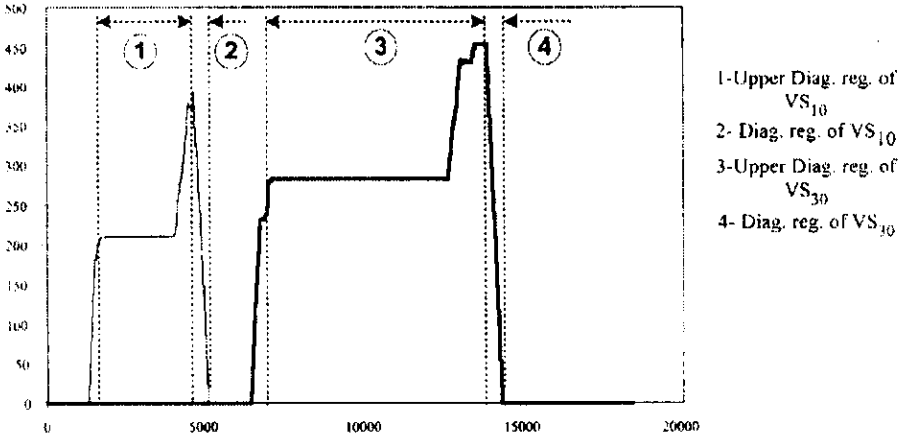
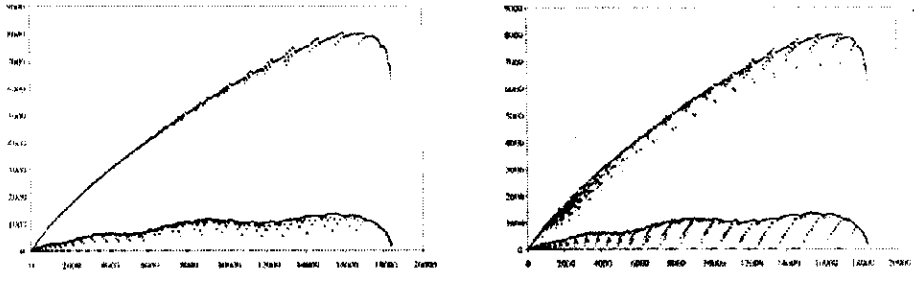


Fig. 14. Final number of elements in row portions of VS_{10} and VS_{30} for K=40.



(a) Estimated (b) Actual

Fig. 15. Estimated and actual number of elements in columns of $V\,Q^T$.

6. Conclusion

In this paper, we have presented the DBS technique as a largeness tolerance technique based on a modified version of the GE procedure. The basic concept of the DBS technique depends on splitting the whole matrix into a number of vertical segments. Each vertical segment consists of a band of successive columns. Each vertical segment is acted upon by a modified version of the GE procedure in such a way that these segments are transformed and become a part of the triangular form of the original matrix. Further, we store these segments in the hard disk to be used in calculating the steady state probabilities. Moreover, a list of the computational operations (performed during the transformation of these vertical segments) is stored on the hard disk. By varying the number of vertical segments, the memory needed by the DBS technique is greatly minimized.

To minimize the DBS solution time, we have supported the DBS by two novel techniques. The first technique enables the DBS to get rid of the time consumed due to the back and forth movements of the non-zero elements of vertical segments during the transformation processes of each of these vertical segments into its SUT form. The second technique permits clever management of the disk accessing for the purpose of computation operation records. By making use of these two techniques, we have the capability of minimizing effectively the DBS solution time. Moreover, since the number of elements of any vertical segment increases excessively during its transformation to the corresponding SUT form, we have proposed a utility for the purpose of allowing an approximate estimation for the number of vertical segments to prevent the memory overflow.

To illustrate the merits of using the DBS technique, we have applied the DBS on the matrix Q obtained from the SRN model of the Kanban manufacturing system. From this model, we have produced 18,400 states to describe the behavior of large-scale models. Also, we have presented the detailed results concerning time and space requirements of the DBS technique. These results show that varying the number of vertical segments leads to 96% reduction in the memory required as compared to that needed by the GE technique, with just only 20% increases in the solution time. This reduction is due to the fact that the GE technique needs 675 Mbytes to solve the state-transition-rate of this model but the DBS technique needs only 60 Mbytes, with 20% increases in the solution time. Finally, we have analyzed the execution processes of the disk management technique to clarify its effectiveness.

References

[1] Mahevas, S. and Rubino, G. "Bound Computation of Dependability and Performance Measures." *IEEE Transactions on Computers*, 50, No.5(2001), 399-413.

[2] Nicola, V. F., Shahabuddin, P. and Nakayama, M. K. "Techniques for Fast Simulation of Models of Highly Dependable Systems." *IEEE Transactions on Reliabilit,* 50, No.3 (2001), 246-64.

[3] Abdallah, H., Hamza, M. and Arabnia, H.R. "Sensitivity Computation of the Expected Accumulated Reward of Stiff Markov Models." *Proceedings of the International Conference on Parallel and Distributed Processing Techniques and Applications*, Monte Carlo Resort, Las Vegas, Nevada, USA, 3 (2000), 1415-1421.

[4] Ramani, S., Trivedi, K.S. and Dasarathy, B. "Performance Analysis of the CORBA Event Service Using Stochastic Reward Nets." *Proceedings of the 19th Symposium on Reliable Distributed Systems*, Nurnberg, Germany (2000), 238-247.

[5] Goseva-Popstojanova, K. and Trivedi, K.S. "Stochastic Modeling Formalisms for Dependability, Performance and Performability." *Lecture Notes in Computer Science,* 1769, Springer-Verlag, (2000), 403-422.

[6] Gribaudo, M., Sereno, M., Horvath, A. and Bobbio, A. "Fluid Stochastic Petri Nets Augmented with Flush-out Arcs: Modelling and Analysis." *Discrete Event Dynamic Systems: Theory & Applications,* 11, No.1-2, (2001), 97-117.

[7] Haverkort, B.R., Boudewijn, R. and Alexander, O. "Steady-state Analysis of Infinite Stochastic Petri Nets: Comparing the Spectral Expansion and the Matrix-Geometric Method." *Proceedings of the 7th International Workshop on Petri Nets and Performance Models,* Saint Malo, France (1997), 36-45.

[8] Hermanns, H., Herzog, U. and Katoen, J-P. "Process Algebra for Performance Evaluation." *Theoretical Computer Science,* 274, No.1-2 (2002), 43-87.

[9] Chiola, G., Franceschinis, G., Gaeta, R. and Ribaudo, M. "Great SPN 1.7: Graphical Editor and Analyzer for Timed and Stochastic Petri Nets." *Performance Evaluation,* 24, No. 1-2 (1995), 47-68.

[10] Buchholz, P. "Hybrid Analysis of SGSPNs with Time-dependent Transition Rates." *Performance Evaluation,* 44, No.1-4 (2001), 187-209.

[11] Deavours, D. D. and Sanders, W. H. "On-The-fly Solution Techniques for Stochastic Petri Nets and Extensions." *IEEE Transactions on Software Engineering,* 24, No.10 (1998), 889-902.

[12] Deavours, D. D. and Sanders, W. H. "An Efficient Disk-based Tool for Solving Large Models." *Performance Evaluation,* 33, No.1-2 (1998), 67-84.

[13] Qureshi, M.A., Sanders, W.H., Van Moorsel, A.P.A. and German R. "Algorithms for the Generation of State-level Representations of Stochastic Activity Networks with General Reward Structures." *IEEE Transactions on Software Engineering,* 22, No.9 (1996), 603-614.

[14] Buchholz, P. and Kemper, P. "Quantifying the Dynamic Behavior of Process Algebras." *Proceedings of the Joint PAPM-PROBMIV Workshop*, Aachen, Germany: Springer- Verlag, LNCS 2165, 184-199.

[15] Buchholz, P. and Kemper, P. "Compact Representations of Probability Distributions in Analysis of Superposed Gspns." *Proceedings of the 9th International Workshop on Petri Nets and Performance Models,* Aachen, Germany: IEEE CS Press, (2001), 81-90.

[16] German, R. "Iterative Analysis of Markov Regenerative Models." *Performance Evaluation,* 44, No.1-4 (2001), 51-72.

[17] Plateau, B., Stewart, W. J. and Silva, M. "Numerical Solution of Markov Chains." *Proceedings of the 3rd International Conference on Numerical Solution of Markov Chains*. Spain: Prensas Universitarias de Zaragoza, (1999), 435-448.

[18] Stewart, W. J. *Introduction to the Numerical Solution of Markov Chains*. Princeton University Press,

[19] Buchholz, P., Ciardo, G., Kemper, P. and Donatelli, S. "Complexity of Memory-efficient Kronecker Operations with Applications to the Solution of Markov Models." *INFORMS Journal on Computing,* 13, No. 3 (2000), 203-222.

[20] Hirel, C., Trivedi, K.S. and Tuffin, B. "SPNP Version 6.0." *Lecture Notes in Computer Science* 1786, Springer-Verlag (2000), 354-357.

[21] Lindemann, C. "DSPNexpress: A Software Package for the Efficient Solution of Deterministic and Stochastic Petri Nets." *Performance Evaluation,* 22, No.1-2 (1995), 3-21.

[22] Donatelli, S. and Kemper, P. "Integrating Synchronization with Priority Into a Kronecker Representation." *Performance Evaluation,* 44, No.1-4 (2001), 73-96.

تكنيك جديد معتمدا علي القرص الصلب لحل مشكلة كبر صيغ النمذجة العشوائية

سمير م. كريم و و. س. الكيلاني*

قسم هندسة النظم والحاسبات، كلية الهندسة، جامعة الأزهر، القاهرة،

**قسم تقنيات المعلومات، كلية الحاسب والمعلومات،*

جامعة المنوفية، شبين الكوم؛ جمهورية مصر العربية

(قدّم للنشر في ٢٠٠١/١٠/١٤م؛ وقبل للنشر في ٢٠٠٢/٢/١١م)

ملخص البحث. صيغ النمذجه العشوائية مثل شبكه بتري العشوائية، وشبكه بتري العامة العشوائية، وشبكات بتري المكافأة العشوائية يمكن أن تستخدم لعمل نمذجة وتقويم للسلوك الديناميكي للحاسبات العملية الواقعية. متى تم استنتاج حالات ماركوف من هذا النموذج العشوائي، فإنه لحلها يتم رياضيًا وضعها في صورة مصفوفة عدد عناصر يزداد زيادة رهيبة تصل إلى مئات الآلاف. لحل مثل هذه المشكلة نقدم في هذا البحث تكنيك جديد نطلق عليه "انقسام المصفوفات مستخدما إمكانات القرص الصلب." في هذا التكنيك المقترح تم تقسيم المصفوفات إلى عدد من الأقسام العمودية وتم استخدام إمكانات القرص الصلب لتخزينها. وبهذه الطريقة تم تقليل حجم الذاكرة اللازمة لتخزين المصفوفات. وفي هذا الصدد تم تصميم ثلاثة برامج لتساعدنا في عملية تقسيم المصفوفات وتخزينها على القرص الصلب بكفاءة عالية. ولتوضيح مدي كفاءة التكنيك المقترح، تم تطبيقه على نموذج تم تصميمه بشبكة بتري المكافأة العشوائية لكي يمثل السلوك الديناميكي لإحدى الحاسبات العملية الواقعية التي تمدنا بمئات الآلاف من حالات ماركوف.