ISSN 0717-9103

BATCH SIZE DETERMINATION FOR WAFER FABRICATION USING GENETIC ALGORITHMS

NIPA PHOJANAMONGKOLKIJ [1]
OMAR GHRAYEB [2]

Northern Illinois University - USA

ABSTRACT

Processes of a real world manufacturer involve a number of batch-processing operations. Generally, these operations require the decision on batch sizes of products and there is no specific rule to set batch sizes of all products simultaneously so that the sum of the weighted expected cycle times of all products is minimized. The weights represent different levels of importance for products. In this study, we proposed an efficient genetic algorithm to determine the batch size combination that minimizes this sum. The proposed GA was tested on three different data sets, having number of products ranging from 7 to 12. The results show that, for some cases, the proposed GA is effective in locating the global optimum solutions.

Keywords: Batch-processing, Queueing, Genetic Algorithm, Semiconductor manufacturing

[1]Industrial Engineering Department, Northern Illinois University, DeKalb, IL 60115, USA.
e-mail: nipa@ceet.niu.edu
[2]Industrial Engineering Department, Northern Illinois University, DeKalb, IL 60115, USA.
e-mail: ghrayeb@ceet.niu.edu

1. INTRODUCTION

A real world ASIC (Application Specific Integrated Circuit) manufacturer has a diverse market focusing on networking, storage components, telecommunications/wireless, consumer, computer, and storage systems. Processes of the company involve a number of batch-processing operations. Generally, these operations are performed by batch-processing machines, which are capable of processing multiple products at the same time. However, different products are generally not allowed to be in the same batch due to different chemical and operation settings. Service times do not depend upon batch service sizes, but may differ from one product to another. Intuitively, for products with lower arrival rates, it is preferable to use smaller batch sizes; otherwise their waiting times to form batch increases. However, if batch sizes are too small, then the batch-processing machines may run out of capacity. On the other hand, for products with higher arrival rates, it is preferable to use larger batch sizes. If batch sizes are too large, then the waiting times to form batches may increase. Generally, there is no specific rule to set batch sizes of all products simultaneously so that the performance of batch-processing service operations (i.e. cycle time) is improved.

Typically, different products may have different levels of importance (e.g., holding cost, priority, etc.) These importance levels may contribute to even more complex rules of setting simultaneous batch sizes. The company has to make a decision on batch sizes of products. In this study, the goal is to propose an efficient genetic algorithm to determine the batch size combination that minimizes the sum of the weighted expected cycle times of all products. The weights reflect the importance levels of products; the higher the weights, the more important the corresponding products.

2. LITERATURE REVIEW

Previous works on analytical models to determine batch sizes of products include Neuts (1967), Deb and Serfozo (1973), Gurnani et al. (1992), Neuts and Nadarajan (1982), Sim and Templeton (1985), Chandra and Gupta (1997), and Avramidis et al. (1998). In these previous related works, the analytical models are limited to either a single machine or a single product. When multiple products are allowed, they are only permitted to have equal batch sizes. In many real-world problems, however, especially in semiconductor manufacturing, these limitations make the existing models less than satisfactory. The batch-processing workstations may have multiple batch-processing machines and may process multiple products. Each product may require different operating settings (e.g., temperatures, chemical settings, etc.) and thus have to be processed in batches with the same product. Different products may have different level of demands and priorities, and therefore using the same batch size for all products may not be the most effective practice. For these reasons, an analytical model that addresses these issues is needed.

Phojanamongkolkij (2000) and Fowler *et al.* (2002) proposes a batch-processing queueing $G/G^{(bp)}/c$ model to evaluate the performance of multiple machines, multiple products, incompatible products, and unequal batch sizes (that is, different products may have different batch sizes.) She compares her model with a corresponding simulation model and reports that the maximum absolute % difference between these two models is no more than 12% in all the 73 test cases. In addition, the mean and standard deviation are small (4.50% and 3.61%, respectively.) This model is then used along with an intelligent search algorithm (i.e., Genetic Algorithm or GA) to seek for the simultaneous "optimum" batch sizes for all products so that the total expected cycle time (queue time plus service time of any product) at machines is minimized.

However, there is a critical issue that this work fails to consider. That is, there is no consideration of product priorities. Therefore, we propose a more efficient genetic algorithm to determine the simultaneous optimum batch sizes for all products so that the sum of the weighted expected cycle times of all products is minimized. We also include priorities into the objective function of GA. Section 3 provides detailed problem description of the case study. We explain the two GA operators in Section 4. The first operator is from Phojanamongkolkij (2000), and the second operator is from Ghrayeb (2000). Section 5 compares and discusses the results. Finally, we summarize our study and recommend some future work in Section 6.

3. PROBLEM DESCRIPTION OF WAFER FABRICATION

We consider 3 different batch-processing tool sets in the company's process for this study. Data for the studied tool sets are given in Table I. The names of the tool sets are presented in the Tool Set column. Brief descriptions of processes, number of machines, and the service capacities (in lots) of tool sets are given in the second column. The Product column shows products that can be processed by tool sets.

Table I also shows the service time (in hours/batch) by corresponding tool set and arrival rates (in items/month) for each product in the Service time and the Arrival Rate columns, respectively. These batch-processing tool sets process items that come from various upstream steps in the flows. Thus, regardless of the overall inter-release distribution, Poisson arrivals to each batch-processing tool set seem to be reasonable according to Renewal Process (Walrand, 1988 and Lipsky, 1992). The service times in this study are exponentially distributed to represent the worst (standard deviation of service time equal to its mean) scenario. The service times are essentially constant, but they are also assumed to be exponentially distributed in this study to account for unpredictable events, such as operator and tool availability, which may cause the service time to vary greatly. Therefore, findings from this study will provide more generic conclusions on batch-processing policies.

Priority levels (weights) of all products are also given in the Normalized Priority column of Table 1. Priorities in the real system are established based on business conditions. However, in this paper, we randomly generate the priorities from a uniform distribution, and then normalize them so that they all sum to one for a given tool set. The higher the values, the higher the priority levels. Since the company is an ASIC manufacturer, it is unlikely that priority levels of any products are the same.
The goal is to determine processing policies that minimize the sum of the weighted expected cycle times of all products. Mathematically, the objective function can be written as:

$$\min \quad \sum_{p=1}^{P} w_p CT_p \qquad (1)$$

where
P = number of products
w_p = normalized priority value of product p, for $p = 1, 2, \ldots, P$
CT_p = the expected cycle time (the expected waiting time to form a full batch plus the expected waiting time in queue plus the service time) of product p at the tool set, for $p = 1, 2, \ldots, P$

Tool Set	Description, #Machine, (Min, Max) Batch Size (in items)	Product	Service time (in hours/batch) (including load/unload)	Arrival Rate (in items/month)	Normalized Priority	Global Optimum Objective Value from Full Search
SET 2	Diffusion Process 2, 4 machines, (1,6)	C	5.67	16	0.1193	
		D	5.57	16	0.1446	
		E	10.70	16	0.0587	
		F	5.18	16	0.0676	
		G	6.82	16	0.0355	
		H	4.18	483	0.0667	
		I	3.23	495	0.1313	
		J	4.68	91	0.0407	
		K	6.82	495	0.0443	
		L	3.23	12	0.0602	
		M	3.23	12	0.1162	
		N	8.78	32	0.1150	
SET 6	Oxidation Process 1,4 machines, (1,6)	X	11.5	16	0.2501	8.39
		Y	3.9	16	0.0600	
		Z	6.4	160	0.0516	
		AA	6.4	162	0.1934	
		BB	3.8	322	0.1347	
		CC	6.4	8	0.1329	
		DD	3.8	8	0.1772	
SET 7	Oxidation Process 2, 2 machines, (1,6)	FF	14.20	16	0.1731	10.56
		GG	6.60	16	0.1307	
		HH	4.10	162	0.0054	
		II	4.10	322	0.0782	
		JJ	4.10	8	0.1571	
		KK	4.10	8	0.1831	
		MM	4.10	160	0.1834	
		NN	3.90	16	0.0891	

Table 1. Service time (in hours/batch), arrival rates (in items/month), and normalized priority values of products, and the best known solutions

4. GENETIC ALGORITHM OPERATORS

Each chromosome in the GA approach consists of genes that describe a batch size combination. The number of genes in a chromosome is the number of products. For example, a chromosome of (1, 2, 3, 1, 1, 1, 1) for batch size combination of SET6 represents batch size of 1 for X, 2 for Y, 3 for Z, and 1 for AA, ..., and DD. The first GA operator considered is from Phojanamongkolkij (2000) and will be later referred to as GA_I operator. The second operator is from Ghrayeb (2000) and will be later referred to as GA_II operator. The detailed algorithm for each operator is given below.

4.1. One-Point Crossover operator (GA_I)

Step 1. Use a random number generator to create the first generation with number of chromosomes equal to specified population size.
Step 2. Evaluate the objective function (described below) of each chromosome.
Step 3. Reorder the chromosomes according to the best objective value.
Step 4. If number of generations is less than the specified value, then go to Step 5. Otherwise, go to Step 6.
Step 5. Create the next generation from
  5.1. Cloning (30%): Use the best 30% of chromosomes in the current generation as chromosomes in the next generation.
  5.2. Breeding (60%): From the best 30% of chromosomes, perform pairwise- crossover to reproduce 60% of chromosomes for the next generation. The crossover point is randomly selected.
  5.3. Mutation (10%): From the best 10% of chromosomes, swap two (random) genes in each chromosome to create 10% new chromosomes for the next generation.
  5.4. Start the next generation. Go to Step 2.
Step 6. The GA solution is the best chromosome in the last generation.

This algorithm is coded in Matlab (2001). The best chromosome in this study implies the chromosome that gives the lowest value of the objective function. Cloning allows the best chromosomes to survive to the next generation. Breeding and mutation allow (optimistically) the creation of even better chromosomes from the current best chromosomes. We include the infeasibility of a solution in the objective function. The infeasibility takes place when machine theoretical utilization exceeds 100%. Machine theoretical utilization refers to the arrival rate divided by the service rate. The service rate is determined by the number of machines and assumes only batches of the specific size are formed. The objective function is, therefore, the summation of the weighted expected cycle times of all products (Eq. (1)) and the sum of squares of infeasibility. Thus, the model formulation for the GA is

$$\min \quad \sum_{p=1}^{P} w_p CT_p \quad +(\text{infeasibility})^2$$

where

$$\text{infeasibility} = \begin{cases} 0 & \text{, if \% theoretical utilization} < 100\% \\ \text{\% theoretical utilization} - 100\% & \text{, otherwise} \end{cases}$$

If machine theoretical utilization exceeds 100%, then the term $\sum_{p=1}^{P} w_p CT_p$ (see Phojanamongkolkij (2000) for detailed calculation) is automatically set to be a very large value in the Matlab code and the term (infeasibility)2 is also a very large value. Thus, a chromosome giving an infeasible solution is unlikely to survive to the next generation.

4.2. Partial-Chromosome-Exchange Crossover operator (GA_II)

This algorithm is also coded in Matlab (2001). The following are steps for this algorithm.

Step 1. Use a random number generator to create the first generation with number of chromosomes equal to specified population size.

Step 2. If number of generations is less than the specified value, then go to Step 3. Otherwise, go to Step 6.

Step 3. Perform the following procedures on all chromosomes of the current generation.

- 3.1 Randomly select two parents, excluding the ones (or their associated positions in the generation) already selected, from the current population with equal probability.
- 3.2 Apply crossover operator (explained below) to the two selected parents, which results in two new children.
- 3.3 Put the two parents and the two children together, and pick up the better two according to their objective values. Objective values are evaluated the same way as those of ONE-POINT CROSSOVER operator.
- 3.4 Replace the two parents with the two newly selected individuals.

At the end of Step 3, the current population is the population after crossover operator is performed.

Step 4. Perform the following procedures on 20% of chromosomes of the current population. (This 20% number is randomly selected.)

- 4.1 Randomly select one parent, excluding the ones (or their associated positions in the generation) already selected, from the current population with equal probability.
- 4.2 Apply mutation operator (same as in GA_I operator) to the selected parent, which results in one new child.
- 4.3 Put the parent and the child together, and pick up the better one according to their objective values.
- 4.4 Replace the parent with the one newly selected individual

Step 5. Start the next generation with the current population. Go to Step 2.

Step 6. The GA solution is the best chromosome in the last generation.

Crossover

To exploit the potential of the current generation, we use crossover to generate new generation that, hopefully, retains good features from the current generation. The crossover operator can be summarized as follows:

1. Decide on the value of *Block*, which is the length of the partial chromosomes to be exchanged between the two parents to create offspring.
2. Randomly pick a position in parent *P1*. The second position will be equal to the first position plus *Block*. Then, *Partial Chromosome 1* is formed with the genes between and including the first and second positions.
3. To form *Partial Chromosome* 2, repeat step 2 for parent *P2*. Figure 9 shows children *C1* and *C2* generated by this method.
4. Exchange the partial chromosomes, as shown in Figure 1, to generate children *C1* and *C2*.

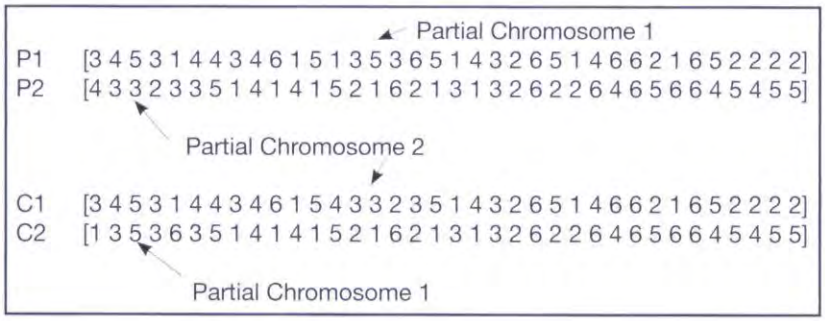

Figure 1. Exchange partial chromosomes.

5. COMPUTATIONAL RESULTS

In this study we set population size to be 50 and 100. For GA_II operator, *Block* parameter is set to be one-sixth of total number of genes in chromosomes. If the value is not integer, we always round the length up to the nearest integer. Therefore, for SET2, SET6, and SET7 *Block* parameters are 2 (12/6), 2 (7/6), and 2 (8/6), respectively. This one-sixth value is selected randomly.

To have a rigouros comparison between the two GA operators, we use Full Search (FS) to obtain the global optimum objective values for two data sets (SET6 and SET7). For FS method, we evaluate the objective value of all batch size combinations. For example in case of SET6, all batch size combinations of X, Y, ..., and DD include (1, 1, 1, 1, 1, 1, 1), (1, 1, 1, 1, 1, 1, 2), ..., (6, 6, 6, 6, 6, 6, 5), and (6, 6, 6, 6, 6, 6, 6). Then we find the batch size combination that provides the minimum sum of the weighted expected cycle times of all products, and this combination will be the global optimum solution. The reason for obtaining the global optimum solution only for SET6 and SET 7 is that it would take extremely long computational time in case of SET2 (Phojanamongkolkij, 2000 and Fowler *et al.*, 2002) . The global optimum solution for these data sets is reported in the last column of Table I.

Figures 2, 3, and 4 show the best convergence curves for SET2, SET6, and SET7, respectively. The curves labeled with GA_II_50 and GA_II_100 represent the curves obtained from GA_II operator with population size of 50 and 100, respectively. Similar interpretation applies for the curves labeled with GA_I_50 and GA_I_100, but both curves are obtained from GA_I operator. Each curve in all figures is obtained from running the GA three times with the same parameter settings, and then selecting the best curve among the three to represent each curve in all figures.

As we can see from Figure 3, the GA_II operator converged to the global optimal solution for SET6 for both population size cases. Actually, it converged to the global optimal in the three runs. On the other hand the GA_I operator converged to the expected cycle time of 8.42 hours when the population size was 100 and to 8.669 hours when the population size was 50. For the other runs, it converged to higher values. From the same figure, we also notice that the GA_I operator converged at a very early stage, actually, it converged at the 5th generation when the population size was 100 and at the 6*th* generation when the population size was 50. This indicates that the GA_I operator was trapped in a local optimum very early. If we look at Figure 4, we draw the same conclutions as in the case of SET 6. Therefore, we can conlcude that the GA_II operator was successful for both data sets and outperformed the GA_I operator.

For SET2, as shown in Figure 2, the GA_II operator converged to a solution with objective value of 9.204 hours while the GA_I operator converged to 10.732 and 10.058 when the population size was 50 and 100, respectively. Again, the GA_I operator showed a premature convergence compared with the GA_II operator. Therefore, we believe that it is fair to conclude that, in general, the GA_II operator outperformed the GA_I operator and is recommended to be used to find the batch size combination.

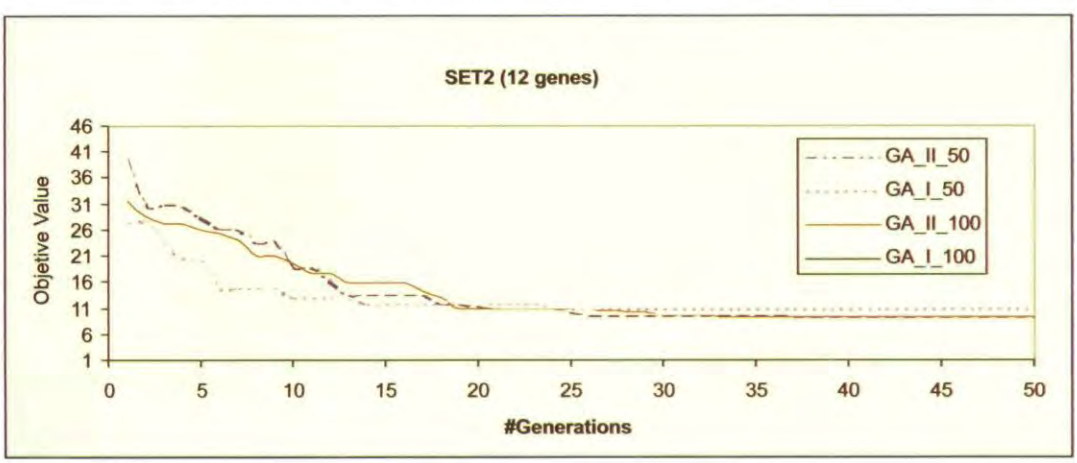

Figure 2. Convergence graph for SET2 data set.

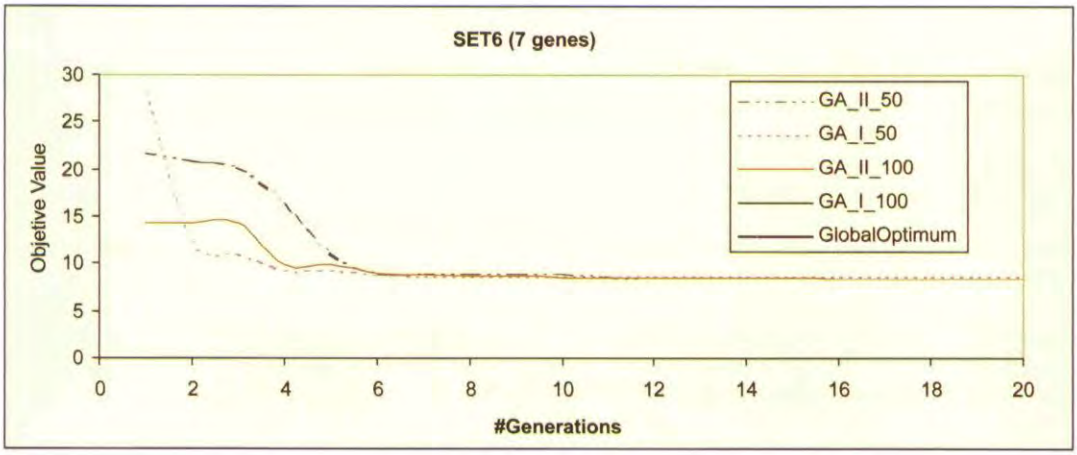

Figure 3. Convergence graph for SET6 data set.

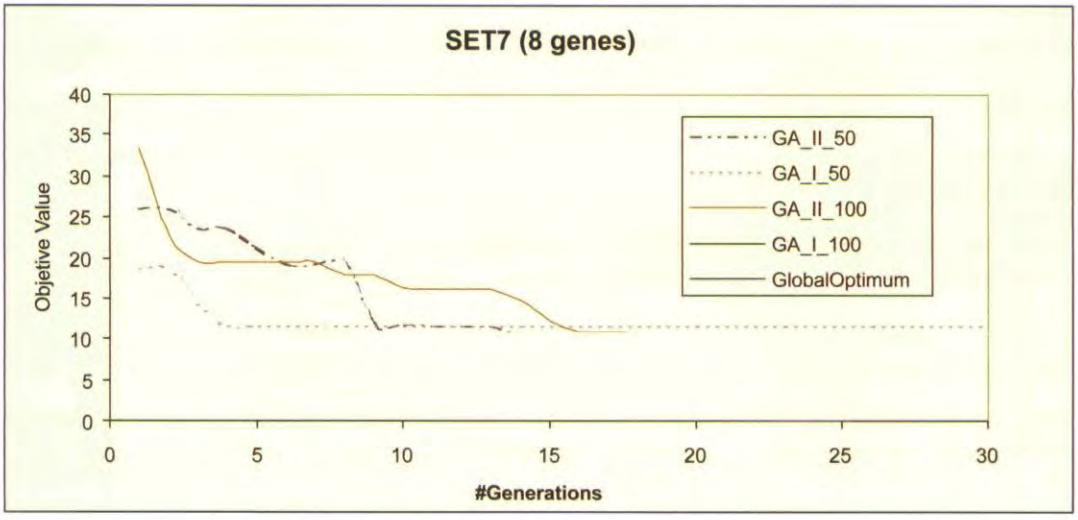

Figure 4. Convergence graph for SET7 data set.

6. CONCLUSIONS

Processes of a real world ASIC (Application Specific Integrated Circuit) manufacturer involve a number of batch-processing operations. Generally, these operations require the decision on batch sizes of products and there is no specific rule to set batch sizes of all products simultaneously so that the cycle time of batch-processing service operations is improved. Typically, different products may have different levels of importance. These importance levels may contribute to even more complex rules of setting simultaneous batch sizes. In this study, we proposed an efficient genetic algorithm to determine the batch size combination that minimizes the sum of the weighted expected cycle times of all products. The proposed GA was tested on three different data sets collected from the company and its performance was compared with the Full Search on two data sets (one with 7 products, and the other with 8 products) and with one-point crossover GA. As the results showed, the proposed GA located the global optimal solution for the two data sets and outperformed the one-point crossover GA for the third data set having 12 products. We recommend using the suggested simultaneous batch sizes as the minimum batch sizes to start the operation at the batch-processing machines.

7. REFERENCES

- Avramidis, A.N., Healy, K.J., and Uzsoy, R. (1998) Control of a batch-processing machine: a computational approach. *International Journal of Production Research*, **36 (11)**, 3167-3181.

- Chandra, P. and Gupta, S. (1997) Managing Batch Processors to Reduce Lead Time in a Semiconductor Packaging Line. *International Journal of Production Research*, **35 (3)**, 611-633.

- Deb, R.K. and Serfozo, R.F. (1973) Optimal Control of Batch Service Queues. *Advanced Applied Probability*, **5**, 340-361.

- Fowler, J.W., Phojanamongkolkij, N., Cochran, J.K., and Montgomery, D.C. (2002) Optimal Batching in a Wafer Fabrication Facility Using a Multiproduct G/G/c Model with Batch Processing. *International Journal of Production Research*, **40 (2)**, 275-292.

- Ghrayeb, O. (2000) *Solving Job-Shop Scheduling Problem with Fuzzy Durations Using Genetic Algorithms*, Doctoral Dissertation, New Mexico State University.

- Gurnani, H., Anupindi, R., and Akella, R. (1992) Control of Batch Processing Systems in Semiconductor Wafer Fabrication Facilities. *IEEE Transactions on Semiconductor Manufacturing*, **5 (4)**, 319-328.

- Lipsky, L. (1992) *Queueing Theory, A Linear Algebraic Approach*, Macmillan Publishing Company.

- Matlab, Version 5.3.0 (1999), The MathWorks Inc.

- Neuts, M.F. (1967) A General Class of Bulk Queues with Poisson Input. *Annals of Mathematical Statistics*, **38**, 759-770.

- Neuts, M.F. and Nadarajan, R. (1982) A Multiserver Queue with Threshold for the Acceptance of Customers into Service. *Operations Research*, **30 (5)**, 948-960.

- Phojanamongkolkij, N. (2000) *Analytical Models of Batch Processing for Optimal Design of Semiconductor Manufacturing*, PhD Dissertation, Arizona State University.

- Sim S.H. and Templeton, J.G.C. (1985) Steady State Results for the M/M(a,b)/c Batch-Service System. *European Journal of Operations Research*, **21**, 260-267.

- Walrand, J. (1988) *An Introduction to Queueing Networks*, Prentice Hall, Inc.