Tribhuvan University
Institute of Science and Technology

Improved Dynamic Programming Approach for The Response Time Variability Problem

Dissertation

Submitted to:

Central Department of Computer Science and Information Technology
Kirtipur, Kathmandu, Nepal

In partial fulfillment of the requirements for the Master’s Degree in Computer Science and Information Technology

by

Shiv Raj Pant

April, 2011

Tribhuvan University
Institute of Science and Technology

Improved Dynamic Programming Approach for The Response Time Variability Problem

Dissertation

Submitted to

Central Department of Computer Science and Information Technology
Kirtipur, Kathmandu, Nepal

In partial fulfillment of the requirements for the Master's Degree in Computer Science and Information Technology

by

Shiv Raj Pant

(April, 2011)

Supervisor

Prof. Dr. Shashidhar Ram Joshi

Co-Supervisor
Mr. Bikash Balami

Tribhuvan University
Institute of Science and Technology
Central Department of Computer Science and Information Technology

Student's Declaration

I hereby declare that I am the only author of this work and that no sources other than the listed here have been used in this work.

Shiv Raj pant
Date: 18-04-2011

Supervisor's Recommendation

I hereby recommend that this dissertation prepared under my supervision by **Mr. Shiv Raj Pant** entitled "Improved Dynamic Programming Approach for The Response Time Variability Problem" in partial fulfillment of the requirements for the degree of M. Sc. in Computer Science and Information Technology be processed for the evaluation.

… … … … … … … … …
Prof. Dr. Shashidhar Ram Joshi
Date: 18-04-2011

Tribhuvan University
Institute of Science and Technology
Central Department of Computer Science and Information Technology

LETTER OF APPROVAL

We certify that we have read this dissertation and in our opinion it is satisfactory in the scope and quality as a dissertation in the partial fulfillment for the requirement of Masters Degree in Computer Science and Information Technology.

Date: 18-04-2011

Evaluation Committee

… … …. … … … …
Prof. Dr. Jeevan Jyoti Nakarmi
Acting Head,
Central Department of Computer Science
and Information Technology,
Tribhuvan University, Nepal
(Head)

… … …. … … … …
Prof. Dr. Shashidhar Ram Joshi
Head,
Department of electronics and computer engineering,
Institute of Engineering.
Tribhuvan University, Nepal
(**Supervisor**)

… … …. … … … …
(External Examiner)

… … …. … … … …
(**Internal Examiner**)

ACKNOWLEDGEMENT

I would like to express my sincere and hearty gratitude to my supervisor Prof. Dr. Shashidhar Ram Joshi for guiding me throughout the study. His cooperation, suggestions, constructive comments and keen interest in this study are appreciable. His vigorous efforts made me present this dissertation work in this form.

I would also like to express my sincere gratitude to Prof. Dr. Jeevan Jyoti Nakarmi , for his inspiration and cooperation for this dissertation. I would also like to express my sincere gratitude to Mr. Min B. Khati, Mr. Dinesh Bajracharya, Mr. Arjun Singh Saud, Mr. Jagadish Bhatta, Mr. Bikash Balami, Mr. Yoga Raj Joshi, Mr Nabaraj Paudel, Mr. Bishnu Gautam, and Mr. Ashim Ghising for their invaluable suggestions and help.

Finally, I would like to extend my thanks to all my friends and colleagues and all my well wishers who directly and indirectly helped me during this work.

ABSTRACT

Fair sequences are useful in a variety of manufacturing and computer systems. The concept of fair sequence has emerged independently from scheduling problems of diverse environments, principally from manufacturing, hard real-time systems, operating systems and network environments. There has been a growing interest in scheduling problems where fair sequence is needed. There are various applications where jobs, clients, or products need to be scheduled in such a way that they get their necessary resources at a constant interval, without being too early or too late. The concept of variation in response time has been recently appeared in literature and a lot of research is being carried out in this area.

The problem of variation in the response time is known as Response Time Variability Problem (RTVP). This dissertation includes recent researches regarding the response time variability problem. RTVP is very hard to solve optimally. It has been proved to be NP-hard. Our concern in this dissertation is to find out the optimal sequence of jobs with objective of minimizing the response time variability. Various solutions based on heuristics exist in the literature to fulfill this objective. One of the approaches is the dynamic programming approach. This dissertation work focuses on the dynamic programming approach. Dynamic programming approach is a complete enumeration scheme that minimizes the amount of computation to be done by dividing the problem into series of subproblems. It solves the subproblems until it finds the solution of the original problem. This approach is not supposed to be a practical solution because of the exponential time and space complexity. The main objective of this dissertation is to improve the dynamic programming approach to RTVP to obtain an efficient solution. The dynamic programming approach will be practically improved by applying some heuristic methods. The basic idea behind the improvement is that we need not search the whole state space if we can find that some states do not lead to an optimal solution. Heuristics will be applied to prune the nonoptimal states. Since, the problem is NP-hard, we cannot theoretically reduce exponential complexity to polynomial complexity. But practically, we can apply heuristic methods to modify the algorithm that can solve the larger instances of the problem.

Table of Contents

Chapter 1

Chapter 2

Chapter 3

Chapter 4

Chapter 5

List of Abbreviations

ATM	Asynchronous Transfer Mode
GRASP	Greedy Random Adaptive Search Procedure
JIT	Just-In-Time
MILP	Mixed Integer Linear Programming
PSO	Particle Swarm Optimization
RAM	Random Access Memory
RTV	Response Time Variability
RTVP	Response Time Variability Problem

Chapter 1

1.1 INTRODUCTION

Response time variability problem is a scheduling problem that has recently appeared in the literature with a broad range of real-life applications. This problem occurs whenever events, jobs, clients or products need to be sequenced so as to minimize the variability of time they wait for their next turn in obtaining the necessary resources.

Most modern systems share their resources between different jobs. The jobs define a certain amount of work to be done, for instance the file size to be transmitted to or from a server or the number of cars of a particular model to be produced on a mixed-model assembly line. To ensure fair sharing of common resources between different jobs, this work is divisible in atomic tasks, for instance data blocks or cars. These tasks, in turn, are required to be evenly distributed so that the distance between any two consecutive tasks of the same job is as regular as possible, in other words, ideally constant.

The following are some real-life applications described in [1] :

The Asynchronous Transfer Mode (ATM) networks divide each application (voice, large data file, video) into cells of fixed size so that the application can be preempted after each cell. Furthermore, isochronous applications, for instance voice and video, require that a inter-cell distance in a cell stream be as close to being constant as possible and in the worst case not exceeding some pre-specified value. The latter is to account for limited resources shared with other applications. In fact multimedia systems should avoid presenting video frames too early or too late which would result in jagged motion perceptions.

On a mixed-model, just-in-time assembly line a sequences of different models to produce is sought where each model is distributed as ”evenly” as possible but appears a given number of times to satisfy demand for different models. Consequently, shortages and excessive inventories are reduced.

The stride scheduling is a deterministic scheduling technique where each client is first issued a number of tickets. The resources are then allocated in discrete time slices called quanta. The client to be allocated resources in next quantum is calculated as a certain function of the number of allocations obtained in the past and the number of tickets issued. Here the throughput error and the response time variability are the two main metrics of the schedule obtained.

The RTVP also appears in computer multithreaded systems ([9] and [10]). Multithreaded systems (operating systems, network servers, media-based applications etc) do different tasks to attend to the requests of the client programs that take place concurrently. These systems need to manage the scarce resources in order to service the requests of the clients. For example, multimedia systems must not display video frames too early or too late, because this would produce jagged motion perceptions. Authors of [9], considering that resource rights could be represented by tickets and that each client had its own number of tickets, suggested the RTV metric to evaluate the sequence of resource rights.

Two real-life cases of RTVP applications were reported in the literature. In [16], the study is motivated by the problem faced by the National Broadcasting Company (BNC) of U.S.A., on of the main firms in the television industry. Major advertisers buy to BNC hundreds of time slots to air commercials. The advertisers ask to BNC that the airings of their commercials are evenly spaced as much as possible over the broadcast season. In [12], the author came up with the RTVP while working with a healthcare facility that needed to schedule the collection of waste from waste collection rooms throughout the building. Based on data about how often a waste collector had to visit each room and in view of the fact that different rooms require a different number of visits per shift, the facility manager wanted these visits to occur as regular as possible so that excessive waste would not collect in any room. For instance, if a room needed four visits per eight-hour shift, it should be ideally visited every two hours.

Other contexts in which the RTVP can be applied are the design of sales catalogs [16], the periodic machine maintenance problem [15] as well as other distance-constrained problems.

The abovementioned applications are examples of very common situation, in manufacturing and in services, in which a resource must be used successively by different units and it is important to schedule them in such a way that the different types of units share the resource in some fair manner. The RTVP proposes a new universal measure of fairness: to minimize the variability of the distance between any two consecutive units of the same product, event, job, or client; i.e., to have the distances between any two given consecutive units of the same product as constant as possible.

The RTVP has been proved to be NP-hard [1]. Thus, this problem has been mostly solved by means of heuristics and metaheuristic methods. Among various solutions, one is the dynamic programming approach which is formulated in [1]. Due to exponential space and time complexity, dynamic programming approach is not supposed a practical solution. This dissertation proposes an improved dynamic programming approach for the RTVP which will be more efficient.

The remainder of the dissertation is organized as follows. Section 1.2 presents a formal definition of the RTVP. Problem definition along with the introduction of the dynamic programming approach is given in section 2.1. The objectives of the dissertation are listed in section 2.2. Section 2.3 presents a brief review of literature. Section 3.1 and 3.2 describe the design and implementation of the improved dynamic programming approach. Section 4.1 shows the computational results and finally section 5.1 describes the conclusion and future researches.

1.2 The Response Time Variability Problem

The response time variability problem is formulated in [1]. Let n be the number of symbols (jobs), d_i the number of copies to be scheduled of symbol i (i=1…n) and D the total number of copies (equal to $\Sigma_{i=1,\ldots,di}$).

Consider a sequence $S=S_1S_2\ldots.S_D$ of length D where i (a client, a product, or a task; in this dissertation we will use the term "job" most often) occurs exactly d_i times. Such a sequence is called feasible. Here S_j is the copy sequenced in position *j* of sequence *S* and S_1 immediately follows S_D. For any two consecutive occurrences of *i*, we define distance *t* between them as the number of positions that separate them plus 1. So there are d_i distances $t_1^i\ldots\ldots t_{di}^i$ for i.

So we have

$$t_1^i+\ldots\ldots+t_{di}^i=D.$$

The average distance t_i' between the i's equals D/d_i

The response time variability for i is defined as

$$\text{RTVi} = \sum_{1<=j<=di} (t_j^i - t_i')^2$$

The total response time variability is defined as

$$\text{RTV} = \sum_{i=1}^{n} \text{RTV}_i = \sum_{i=1}^{n} \sum_{j=1}^{d_i} (t_j^i - t_i')^2$$

An input to the total response time variability problem is a list of n positive integers $d_1<=d_2<=\ldots\ldots..<=d_n$ (the number of copies of each job). The solution to RTVP is a sequence S of jobs and the objective is to minimize the value of RTV obtained above.

An illustrative example is the following:

Let n=3 with symbols A, B, C. Also consider $d_A=2$, $d_B=2$ and $d_C=4$. Thus D=8, $t_A'=4$, $t_B'=4$ and $t_C'=2$. Then the sequence C A C B C B A C is a solution and has

$$\text{RTV} = ((5-4)^2 + (3-4)^2) + ((2-4)^2 + (6-4)^2) + ((2-2)^2 + (3-2)^2) = 12$$

Chapter 2

2.1 Problem Definition

2.1.1 Complexity

The RTVP is difficult to be solved optimally. It has been proved to be NP-hard. Authors of [1] studied the computational complexity of the RTVP and proved that it is NP-hard. The reduction is from the Periodic Maintenance Scheduling Problem studied by [8]. The Periodic Maintenance Scheduling Problem is defined as follows: Given m machines and integer service intervals $l_1,l_2,\ldots l_m$ such that $\Sigma(1/l_i)< 1$. Does there exist a servicing schedule $S_1,S_2,\ldots S_L$ where L = lcm(l_1, l_2, . . . , l_m) is the least common multiple of $l_1,l_2,\ldots.,l_m$, of these machines in which consecutive servicing of machine i are exactly l_i time slots apart and no more than one machine is serviced in a single time slot ?

The Periodic Maintenance Scheduling Problem has been proved to be NP-complete [8].

2.1.2 The Dynamic Programming Algorithm

RTVP is a combinatorial optimization problem and no polynomial-time algorithm is known for solving it. Various algorithms have been proposed to find the near-to-optimal solution of RTVP. One of the solutions is the dynamic programming approach.

Dynamic programming approach to RTVP is suggested in [1]. This is a straightforward dynamic program formulated as follows:

The state of the dynamic program is represented by a quadruple (f,l,r,d) where,

- f is an n dimensional vector $f = (f_1,f_2,\ldots.,f_n)$, where f_i=0,1,….,D-d_i+1 for i = 1,2,…..,n represents the position of the first copy of job i.
- l is an n dimensional vector $l = (l_1,l_2,\ldots..l_n)$ where l_i=0,d_i+1,…..,D for i = 1,…..n represents the last copy of the job i.
- r is an n dimensional vector $r = (r_1,\ldots..r_n)$ where r_i = 0,1,…..d_i represents the number of copies that remain to be sequenced of job i.
- d is the length of the current sequence , d = 0,1,…..D

Initially f = l = 0, r = $(d_1,d_2,\ldots..d_n)$, and d = 0.

A final state is any state with r = 0 and d = D.

There is a weighted arc from a non-final state (f,l,r,d) to a state (f’,l’,r’,d’) if and only if there is an i such that

$r_i' = r_i$-1>=0,

d’=d+1,

l_i'=d’

The weight of the arc is calculated as follows for d_i>=2,

$$W_{<f,l,r,d><f',l',r',d'>} = \begin{cases} 0, & r_i = d_i\,; \\ (d' - l_i - D/d_i)^2\,, & d_i - 1 >= r_i > 1\,; \\ (d' - l_i - D/d_i)^2 + (D - d' + f_i - D/d_i)^2\,, & r_i = 1 \end{cases}$$

Finally, we connect all final states to a dummy state referred to as the destination. All arcs to the destination have zero weight. The shortest path between the initial state and the destination defines an optimal solution to the total RTV. However, the number of states grow exponentially and practically it is hard to find an optimal solution for an instance of moderate size. According to [1], the time complexity is exponential in D. i.e D^{3n+1}.

2.2 Objective

- To improve the dynamic programming approach for the response time variability problem for obtaining an efficient solution.

- To implement and analyze improved dynamic programming algorithm based on heuristic methods.

2.3 Literature Survey

The RTVP is an optimization sequencing problem which was first reported in [9] and formally formulated in [1]. Since RTVP is NP-hard, research on RTVP is mainly focused on finding the optimal solution for large instances by means of heuristic and metaheuristic procedures. The two-symbol case is optimally solved with a quick algorithm proposed in [1]. For a general case, several solutions based on heuristics and metaheuristics have been proposed.

One of the first situations in which the idea of regular sequence appeared was the sequencing on mixed-model assembly lines at Toyota Motor Corporation under the just-in-time (JIT) production system. Since Toyota popularized the JIT production systems, the problem of sequencing on mixed-model assembly lines has acquired high relevance. One of the main aims of JIT is to eliminate sources of waste and inefficiency. In the case of Toyota, the main source of waste was the production of excessive volumes of stock. To solve this problem, JIT systems produce only the specific models required and in the quantities needed at any given time. In this type of system the units should be scheduled in such a way that the consumption rates of the components in the production process remain constant. Authors of [11] also studied this scheduling problem and considered only the demand rates for the model. The problem proposed in [11] intended to minimize variations in production rate in different models.

The RTVP has been first time solved in [9] using a method called lottery scheduling. This method is based on generating a solution at random as follows. For each position of the sequence, the symbol to be sequenced is chosen at random and the probability of each symbol is equal to the number of copies of this symbol that remain to be sequenced divided by the total number of copies that remain to be sequenced. The same authors proposed a greedy heuristic method that they called stride scheduling in [10] that obtains better results than the lottery scheduling method. However, the stride scheduling method is, in fact, the Jefferson method originally designed to solve the apportionment problem.

In [1], five heuristics are proposed to solve the RTVP: the bottleneck algorithm used in [14] to solve the Minmax Product Rate Variation problem, random generation, two classical parametric methods for solving the apportionment problem called Webster method and Jefferson method and a new heuristic called Insertion method by the authors; moreover, a local search procedure is applied to the solutions obtained with the five heuristics. Parametric methods are defined as follows. Let x_{ik} be the number of copies of symbol i that have been already sequenced in the sequence of length k, k = 0, 1, … (asumme $x_{i0} = 0$); the symbol to be sequenced in position k + 1 is $i^* = \arg\max_i \{d_i/(x_{ik}+ \delta)\}$, where $\delta = (0,1]$. . Webster method and Jefferson method are parametric methods that use a δ value equal to 0.5 and 1, respectively. Authors of [13] proposed construction of perfect aggregation to eliminate RTV. More complex algorithms based on metaheuristic schemes and other approaches have also been proposed. Some of the techniques that have been published till date are

- Dynamic Programming Algorithm[1]
- Algorithms based on metaheuristics (multi-start, GRASP and PSO)[3]
- Variable Neighbourhood Search Algorithm[7]
- Tabu Search Algorithm[6]
- Mixed Integer Linear Programming (MILP)[4]
- Genetic Algorithm[5]

The tabu search algorithm and MILP algorithm are supposed to be the best algorithms known. Using MILP approach, instances with 25 to 40 copies of symbols (jobs) can be solved optimally. The disadvantage of the MILP approach is that general software is used to solve the MILP model and it is difficult to take advantage of all characteristics of the problem. Therefore we need some exact algorithm. We will focus on the dynamic programming approach. The dynamic programming approach is not supposed to be the practical solution since it cannot solve the instances of moderate size because of exponential time and space complexity. Heuristics will be applied to improve it.

Chapter 3

3.1 Design of the Improved Algorithm

The naive dynamic programming approach proposed in [1] will be improved as follows: An initial sequence will be created. The value of RTV in initial sequence will be taken as upper bound. In every state with partial sequence, we will calculate the RTV using heuristic methods. This RTV will be taken as lower bound. Based on these bounds, a state will be pruned if its lower bound exceeds the upper bound. The pseudo code of the algorithm is given in section 3.1.1.

3.1.1 Pseudo Code of the Algorithm

```
1. Create initial sequence S using Jefferson’s method or Webster’s method
2. Calculate the value of RTV in S. Let the value be V.
3. for every state Sp with partial sequence do
       apply heuristic to calculate RTV on Sp
       if RTV(Sp)>V then prune the state
```

3.1.2 Details of the Algorithm

3.1.2.1 The Initial Sequence

The methods of creating initial sequence are described in [1] which are Bottleneck (minimum throughput error) sequences, Random sequences, Webster’s sequences, Jefferson’s sequences and Insertion sequences. These methods are heuristic methods to create sequences with near-to-optimal RTV.

- *Webster’s sequences* are obtained by applying the parametric method of apportionment with parameter $\delta = ½$. The sequence is generated as follows. Consider x_{it} , the number of copies of job i in sequence of length t, t = 0,1,….. Assume $x_{i0} = 0$, i = 1,…..,n. The job to be sequenced in position t + 1 is computed as follows:

$$i^* = \arg\max_i \{d_i/(x_{it} + \delta)\}$$

- *Jefferson's sequences* are generated by applying the parametric method of apportionment, described above with δ =1.
- *The bottleneck sequences* can be obtained by solving the bottleneck problem to optimality with the algorithm given in [14].
- *The random sequences* can be obtained by randomizing the bottleneck sequence. The bottleneck sequences can be randomized as follows. For each position x in 1….D, get a random number *ran* in the range 1….D, then swap S[x] with S[ran].

A detailed analysis of all of above methods is described in [1]. All of these methods have comparable results. According to Authors of [1], all of the above mentioned methods have comparable results for small value of n, but Webster and Jefferson methods have relatively poor result for large value of n. We will use Webster's sequences in our implementation. The value of RTV obtained on initial sequence will be taken as upper bound.

3.1.2.2 Finding RTV on Partial Sequences

Heuristics for finding RTV on partial sequences are described in [2]. Consider that a partial sequence has built up and including position k. Given a job, if we compare the cost of allocating a copy of it to position k+1 and the cost of allocating it to position k+2, the difference between the latter and the former can be called the opportunity cost of allocating the copy to position k+2 (instead of allocating it to position k+1). It is reasonable, then, to allocate to position k+1 the job with a greater opportunity cost. As this cost cannot be calculated without an optimizing algorithm, the decision can be taken on the basis of an estimation of it, i.e., the value equal to PLB(i,k+2) – PLB(i,k+1), where PLB(i,k+2) and PLB(i,k+1), are lower bounds on the objective function when a copy of job i is placed at positions k+2 or at position k+1, respectively.

As described in [1], a decomposition vector of D into d_i components can be defined as $\lambda_i = (\lambda_1 \ldots \lambda_{di})$ of d_i positive integers that add up to D and λ_1 >=…..>=λ_{di} . The components of vector λ_i are distances between the d_i copies of job i. Thus the minimum value of RTV for job i, RTV_i , can be obtained when D mod d_i and d_i – D mod d_i components of λ_i are equal to $\lceil D/di \rceil$ and $\lfloor D/di \rfloor$ respectively.

For example, let D=24, n=4, d=(9,8,5,2) and t'=(2.67,3,4.8,12). The decomposition vectors λ_1 = (3,3,3,3,3,3,2,2,2), λ_2 = (3,3,3,3,3,3,3,3), λ_3 = (5,5,5,5,4), and λ_4 = (12,12) provide the minimum values of RTVi (i=1,…,4). A lower bound on the value of RTVi, i.e. RTVLBi, and a lower bound on the value of RTV, i.e. RTVLB, can be defined as follows:

$$RTVLBi = (D \bmod d_i)*\left(\left\lceil D/di \right\rceil - t'_i\right)^2 + (d_i - D \bmod d_i)*\left(\left\lfloor D/di \right\rfloor - t'_i\right)^2 \text{ and}$$

$$RTVLB = \sum_{i=1}^{n} RTVLB_i$$

Hence

$$\begin{aligned} RTVLB &= [6*(3-2.67)^2 + 3*(2-2.67)^2] + [8*(3-3)^2\ ] \\ &\quad + [4*(5-4.8)^2 + 1*(4-4.8)^2] + [2*(12-12)^2\ ] \\ &= 1.18 \end{aligned}$$

But in this case, a lower bound, PLB, is needed for a partial solution, i.e., a solution in which one copy of job has been assigned to each of the first k positions.

A bound for a partial sequence, Sp, can be obtained by adding, for all the jobs with d_i >=2, the sum of RTVps (the value associated with the distances between the copies of the job allocated in [1,…..,k], if any) and RTVrem (a bound corresponding to the assignment of the remaining copies, if any, to the free positions).

Let i be a job, with d_i >=2, whose copies have not all been assigned in the partial solution Sp. Now there are three cases:

- Case 1: No copy of job i has been assigned in the k time slots. In this case, we must distribute D time slots among d_i distances between two copies of job i, guaranteeing that one distance be greater than or equal to k+1.
- Case 2: Only one copy of job i has been assigned to position h (<=k). In this case, we must distribute D time slots among d_i distances, guaranteeing that one distance be greater than or equal to k-h+1 and another be greater than or equal to h.
- Case 3: p copies of job i have been assigned in k time slots, the first in the sequence in position h_f and the last one in the position h_l . In this case, we must distribute D-h_l+h_f

time slots among d_i-p+1 distances, but guarantee that on distance be greater than or equal to k-h_l+1 and another be greater than or equal to h_f . Case 2 can be reduced to Case 3 taking into account that $h_f = h_l = h$ and p = 1. Case 1 can be reduced to case 3 taking into account that $h_f = h_l = h = 0$ and p = 1.

Thus, the problem consists of distributing D-h_l+h_f units of distance among d_i-p+1 distances t^i_j (j=1,…., d_i-p+1), taking into account that two distances are lower bounded by k-h_l+1 and h_f, respectively, and the others are lower bounded by 1, with the objective of minimizing a function of the discrepancy between the distances and the average distance t_i'. Thus it is the apportionment problem with lower bounds. For the discrepancy function considered here, the procedure is described in [2] as follows:

```
$t^i_1 = k - h_l +1$
$t^i_2 = h_f$
for j=3 to $d_i$-p+1
        {
          $t^i_j = 1$
          next j
         }
for j = 1 to D-k+p-$d_i$
             {
               Find s* such that $t^i_{s*} = \min_{1<=s<=\max(2,d_i-p+1)} ( t^i_s )$
               $t^i_{s*} = t^i_{s*} + 1$
                next j
             }
```

Example: For the instance n = 4 and d = (9,8,5,2) and the partial solution Sp = (1,3,2,1,1,3,3,3,1,,,,,,,,,,,,,,,), we have:

RTVps = $[1*(4-2.67)^2+1*(3-2.67)^2+1*(1-2.67)^2] + [1*(4-4.8)^2+2*(1-4.8)^2 = 34.187$

Now, applying the procedure described above, the distances (3,3,3,3,2,2), (7,3,3,3,2,2,2,2), (9,9), (12,12) are obtained for job 1, 2, 3 and 4 respectively. The value corresponding to these distances, RTVrem is:

$$RTVrem = [4*(3-2.67)^2+2*(2-2.67)^2] + [1*(7-3)^2+3*(3-3)^2+4*(2-3)^2] +[2*(9-4.8)^2]+[2*(12-12)^2]$$
$$= 56.613$$

Finally, PLB = RTVps + RTVrem = 34.187 + 56.613 = 90.8
When breaking ties, jobs are selected in descending order of i.

3.2 Implementation

3.2.1 Changes in the State representation of Naive Dynamic Program

The state of the naïve dynamic program will be changed as follows: Every state of the dynamic program will be represented by a 5-tuple (f,l,r,d,p), where other symbols have usual meaning except the symbol ‘p’. Here p represents the pruned state. p = 0 represents that the state is not pruned and p = 1 represents that the state has been pruned. The initial state and the final states will have p = 0. Other fields can also be added such as the partial sequence associated with the state and cost of the arc etc to make searching easy.

Dynamic programming approach is an exact algorithm. So, it can be implemented directly in any conventional programming language. Unlike some other approaches, no additional software (e.g. MILP model uses general software) is required. Implementation should be made efficient by using suitable representation for the states (representations that require less memory).

Chapter 4

4.1 Computational results

The computational experiments have been carried out in order to illustrate the improved dynamic programming algorithm. The experiment consists of applying the naïve dynamic programming algorithm and improved dynamic programming algorithm separately to an instance of the problem. All codes have been implemented in C. Both the naïve algorithm and improved algorithm has been implemented and executed on a PC with Intel Pentium 4 (2.26 GHz) processor and 512 MB of RAM.

Instances for this experiment are generated by fixing the total number of units D and number of jobs n, and randomly selecting the number of copies of each job, d_i. In the improved algorithm, the initial sequences have been generated by using Webster's method. We will examine the improvement in terms of the number of states generated by each algorithm.

The experimental results show that the numbers of states are tremendously reduced by the improved algorithm. For n=2 , the improvement is about by 54% on average. For n=3, the states are reduced by 89% on average. For n=4, the states are reduced by 94% on average. For n=5, the states are reduced by 95% on average. These calculations based on reduction in the number of states. The "--" in the tables indicates that the result was not computed within specified time of 2 minutes. All other results were executed within 2 minutes. The following tables show the final results only. The initial sequences generated by Webster's method and their corresponding RTV are tabulated in Appendix A. In the following tables, the first column represents the total demand i.e. total number of copies of each job. The second column represents the input to the algorithm. As described earlier, an input to the total response time variability problem is a list of n positive integers d_1<=d_2<=……..<=d_n (the demand of each job). The third column shows the number of states generated by naïve dynamic program and the fourth column shows the number of states generated by improved dynamic program. The fourth column shows the optimal sequence generated for the corresponding input. The optimal sequence is one with the shortest-weight path from the initial state to the final state. Finally, the last column shows the value of RTV corresponding to the optimal sequence (i.e. the optimal RTV).

Table 1 shows the results obtained for n =2 i.e. two-job case. This table is shown just to illustrate the improved algorithm. This case can be efficiently solved by the simple algorithm proposed in [1], however. So for n = 2 it will be better to use the algorithm proposed in [1]. The efficient solution given in [1] for n = 2 is described in Appendix B. In table 1, various instances of the problem are generated randomly by fixing number of jobs n =2. The number of states are reduced by 54 % on average using improved algorithm.

Table 2 shows the results obtained for n =3. In this case, we observe that the states are reduced by 89% on average. By naïve algorithm, only the instances upto D = 17 were solved within the specified time of 2 minutes. But using improved approach, instances with total demand D =30 were easily solved within the same time. The improvement is better than the case with n =2.

Table 3 and table 4 show the results obtained for n = 4 and n = 5 respectively. For n = 4, the instances upto D = 24 were solved within the given time with improved approach whereas with naïve approach, only instances upto D = 15 were solved. For n = 5, using naïve approach, instances upto D = 12 have been solved and using improved approach the instances upto D = 18 were solved within given time of 2 minutes.

Table 1: Results obtained for n = 2

Number of jobs (n) = 2					
D	Input vector (number of copies of each job)	Number of states generated by naïve algorithm	Number of states generated by improved algorithm	Optimal sequence	Optimal RTV
4	(2,2)	20	16	1,2,1,2	0.00
6	(4,2)	56	30	1,1,2,1,1,2	1.00
6	(3,3)	70	24	1,2,1,2,1,2	0.00
7	(5,2)	84	39	1,1,2,1,1,1,2	1.70
7	(3,4)	126	54	1,2,2,1,2,1,2	1.41
8	(5,3)	210	68	1,1,2,1,2,1,1,2	1.86
8	(6,2)	120	49	1,1,1,2,1,1,1,2	1.33
10	(8,2)	220	73	1,1,1,1,2,1,1,1,1,2	1.50
10	(7,3)	495	101	1,1,1,2,1,1,2,1,1,2	2.38
10	(6,4)	792	137	1,1,2,1,1,2,1,2,1,2	2.33

Table 2: Results obtained for n = 3

Number of jobs (n) = 3					
D	Input vector (number of copies of each job)	Number of states generated by naïve algorithm	Number of states generated by improved algorithm	Optimal sequence	Optimal RTV
7	(3,2,2)	651	227	1,2,3,1,2,1,3	1.66
10	(3,3,4)	13300	659	1,2,3,1,2,3,1,3,2,3	2.33
10	(5.3.2)	8295	785	1,1,3,2,1,2,1,3,1,2	4.66
12	(2,6,4)	46552	937	1,2,3,2,2,3,1,2,3,2,2,3	4.00
12	(2,7,3)	27951	1034	1,2,2,2,3,2,1,2,3,2,2,3	5.42
12	(2,8,2)	11452	1538	1,2,2,2,3,2,1,2,2,2,3,2	2.00
13	(3,8,2)	47125	1473	1,2,2,1,2,3,2,2,1,2,2,3,2	5.04
13	(4,7,2)	88803	425	1,2,2,1,2,3,2,1,2,1,2,3,2	4.10
13	(5,6,2)	120835	1099	1,2,1,2,1,3,2,1,2,1,2,3,2	4.53
13	(5,5,3)	233729	1266	1,2,1,2,3,1,2,1,3,2,1,2,3	3.06

14	(7,4,3)	403326	5567	1,2,1,3,1,2,1,1,2,3,1,2,1,3	5.66
14	(6,5,3)	551838	1180	1,2,1,3,2,1,1,2,3,1,2,1,3,2	4.80
14	(6,6,2)	283712	410	1,2,1,2,1,2,3,1,2,1,2,1,2,3	2.66
14	(5,7,2)	246246	1970	1,2,1,2,2,1,3,2,1,2,2,1,2,3	4.80
15	(7,4,4)	1490203	5990	1,2,1,3,1,2,1,3,1,2,1,3,1,2,3	2.35
15	(6,5,4)	2047123	4822	1,2,1,3,1,2,1,3,2,1,2,3,1,2,3	4.25
15	(5,7,3)	1201409	2969	2,3,1,2,1,2,3,1,2,1,2,3,2,1,2	6.85
15	(5,8,2)	471835	3075	1,2,1,2,1,2,3,2,1,2,2,1,2,3,2	5.37
15	(4,8,3)	774775	2367	1,2,2,1,2,3,2,1,2,3,2,1,2,2,3	5.62
16	(8,4,4)	3039752	9562	1,2,1,3,1,2,1,3,1,2,1,3,1,2,1,3	0.00
16	(7,5,4)	4738734	4356	1,2,1,3,2,1,2,3,1,2,1,3,1,2,1,3	4.22
16	(6,6,4)	5481632	4856	1,2,3,1,2,1,3,2,1,2,3,1,2,1,3,2	2.66
16	(2,9,5)	858858	2397	1,2,2,3,2,2,3,2,1,2,3,2,3,2,2,3	6.35
17	(3,9,5)	4720001	4370	1,2,3,2,2,3,1,2,2,3,2,1,3,2,2,3,2	6.75
17	(4,8,5)	--	1185	1,2,3,2,1,2,3,2,1,2,3,2,1,2,3,2,3	4.82
18	(2,9,7)	--	2102	1,2,3,2,3,2,3,2,3,1,2,3,2,2,3,2,2,3	5.71
18	(3,10,5)	--	2160	1,2,3,2,2,3,1,2,2,3,2,1,2,3,2,2,3,2	6.79
18	(4,10,4)	--	47350	1,2,2,3,2,1,2,2,3,2,1,2,3,2,1,2,3,2	3.60
18	(6,9,3)	--	4460	1,2,1,2,3,1,2,2,1,2,3,1,2,2,1,2,3,2	6.00
20	(8,9,3)	--	5561	1,2,1,2,1,3,2,1,2,1,2,1,3,2,1,2,2,1,3,2	6.22
20	(7,10,3)	--	15551	1,2,1,2,1,2,3,1,2,2,1,2,3,2,1,2,2,1,2,3	7.52
20	(6,10,4)	--	35750	1,2,2,1,2,3,2,1,2,2,3,1,2,1,2,3,2,1,2,3	9.33
20	(5,10,5)	--	54795	1,2,3,2,1,2,3,2,1,2,3,2,1,2,3,2,1,2,3,2	0.00
20	(2,10,8)	--	9787	1,2,3,2,2,3,2,2,3,2,1,3,2,3,2,3,2,3,2,3	6.00
20	(7,8,5)	--	36785	1,2,1,3,2,1,2,3,1,2,1,3,2,1,2,3,2,1,2,3	4.85
22	(7,8,7)	--	5587	1,2,3,1,2,3,1,2,3,1,2,3,1,2,3,1,2,3,2,1,3,2	3.21
22	(7,10,5)	--	4258	1,2,1,3,2,1,2,3,2,1,2,3,2,1,2,3,1,2,2,1,3,2	7.65
22	(6,10,6)	--	11467	1,2,3,1,2,3,1,2,3,2,1,2,3,2,1,2,3,2,1,2,3,2	4.26
22	(5,11,6)	--	134627	1,2,2,3,2,1,2,3,2,1,2,3,2,1,2,3,2,1,3,2,2,3	6.53
22	(2,8,12)	--	13945	1,2,3,2,3,3,2,3,3,2,3,1,3,2,3,3,2,3,2,3,2,3	7.16
22	(4,7,11)	--	61415	1,2,3,2,3,1,2,3,3,2,3,1,2,3,3,2,3,1,3,2,3,3	9.85
24	(4,10,10)	--	15901	2,3,2,3,1,2,3,2,3,2,1,3,2,3,2,3,1,2,3,2,3,2	4.80

				,1,3	
24	(6,8,10)	--	139938	1,2,3,2,1,3,2,3,1,3,2,3,1,2,3,2,1,3,2,3,1,3,2,3	6.40
24	(9,8,7)	--	19316	1,3,2,1,2,3,1,2,1,3,1,2,3,1,2,3,1,2,3,1,2,3,1,2	5.71
24	(7,10,7)	--	422060	1,2,3,2,1,2,3,2,1,2,3,1,2,3,1,2,3,1,2,3,2,1,2,3	5.82
26	(7,12,7)	--	1760098	1,2,3,1,2,3,1,2,3,2,1,2,3,2,1,2,3,2,1,2,3,2,1,2,3,2	4.52
26	(5,12,9)	--	79752	1,2,3,2,1,2,3,2,3,2,1,2,3,2,3,2,1,3,2,3,2,1,2,3,2,3	11.35
26	(2,14,10)	--	68191	2,2,3,2,3,2,3,2,3,1,2,3,2,2,3,2,2,3,2,2,3,2,1,3,2,3	8.11
26	(8,11,7)	--	127250	1,2,1,3,2,1,2,3,1,2,3,1,2,3,2,1,2,3,2,1,2,3,2,1,2,3	7.47
28	(10,11,7)	--	607941	1,2,1,3,2,1,2,3,1,2,1,3,2,1,2,3,1,2,1,3,2,1,2,3,2,1,2,3	6.32
28	(9,8,11)	--	319104	1,2,3,1,2,3,1,2,3,1,2,3,1,2,3,1,3,2,3,1,3,2,3,1,3,2,1,3	7.61
30	(11,8,11)	--	320440	1,3,2,1,3,1,2,3,1,3,2,1,3,1,2,3,1,3,2,1,3,1,2,3,1,3,2,1,3,2	5.86
30	(7,9,14)		291488	1,3,2,3,1,3,2,3,1,3,2,3,1,2,3,2,3,1,3,2,3,1,2,3,3,2,1,3,2,3	9.14

Table 3: Results obtained for n = 4

Number of jobs (n) = 4					
D	Input vector (number of copies of each job)	Number of states generated by naïve algorithm	Number of states generated by improved algorithm	Optimal sequence	Optimal RTV
10	(3,2,2,3)	38849	6570	1,2,4,1,3,4,2,1,4,3	1.33
10	(2,2,2,4)	57450	6507	1,2,4,3,4,1,2,4,3,4	1.00
12	(4,3,3,2)	839806	10878	1,2,1,3,4,1,2,3,1,2,4,3	4.00
12	(5,2,3,2)	515998	21755	1,2,1,3,1,4,1,2,3,1,4,3	5.20
12	(6,2,2,2)	268436	23000	1,2,1,3,1,4,1,2,1,3,1,4	0.00
13	(5,2,3,3)	2219070	27227	1,3,1,2,4,1,3,1,4,2,3,1,4	5.03
13	(6,2,2,3)	1145852	29794	1,1,2,4,1,3,1,4,1,2,1,3,4	4.50
13	(4,4,2,3)	2744867	7232	1,2,3,1,4,2,1,4,2,3,1,2,4	4.66
15	(2,7,4,2)	8733723	63252	1,2,3,2,4,2,3,1,2,3,2,4,2,3,2	2.60
15	(3,8,2,2)	4538239	75867	1,2,2,2,3,1,2,4,2,2,1,2,3,2,1	5.87
15	(4,5,4,2)	--	22402	1,2,3,1,2,3,4,1,2,3,2,1,2,3,4	6.00
18	(4,7,2,5)	--	17906	1,4,2,4,1,2,3,4,2,1,2,4,2,1,2,3,4,2	7.91
18	(5,6,2,5)	--	70135	1,2,4,1,2,3,4,1,2,4,2,1,2,4,3,1,2,4	6.40
18	(3,7,4,4)	--	1164608	1,2,3,2,4,2,3,1,2,4,3,2,1,4,2,3,2,4	5.71
18	(3,8,2,5)	--	75977	1,2,4,2,4,3,1,2,4,2,2,1,4,2,3,2,4,2	10.70
20	(4,7,4,5)	--	817258	1,2,3,4,2,1,2,3,4,2,1,2,3,4,2,1,4,3,2,4	6.85
20	(4,8,3,5)	--	74308	1,2,4,2,1,3,2,4,2,1,2,4,3,2,4,1,2,4,2,3	8.66
20	(4,4,8,4)	--	2830726	1,2,3,4,3,1,2,3,4,3,1,2,3,4,3,1,2,3,4,3	2.00
20	(3,8,3,6)	--	1462605	4,1,2,4,2,3,4,2,1,2,4,3,2,4,1,2,4,2,3,2	4.66
20	(3,8,4,5)	--	128239	2,3,2,4,2,3,1,2,4,2,3,2,4,1,2,4,3,2,4,1	8.66
22	(5,4,6,7)	--	606652	1,2,4,3,1,4,2,3,4,1,3,2,4,3,1,4,3,2,4,1,3,4	8.39
22	(3,6,6,7)	--	148292	1,2,4,3,2,4,3,1,4,2,3,4,2,3,4,1,2,4,3,2,4,3	8.19
22	(2,7,6,7)	--	413692	1,2,3,4,2,3,4,2,3,4,2,1,4,3,2,4,2,3,4,2,3,4	7.04
22	(8,2,6,6)	--	354810	1,3,1,4,2,1,3,4,1,3,4,1,3,4,1,2,3,1,4,1,3,4	8.16
24	(7,8,4,5)	--	696531	1,2,1,3,4,2,1,2,4,3,1,2,1,4,2,3,1,2,4,2,1,3,2,4	10.51

24	(6,9,4,5)	--	5343628	1,3,2,4,2,1,2,3,4,2,1,2,4,1,3,2,1,4,2,3,1,2,4,2	10.80
24	(10,5,4,5)	--	3905563	1,3,2,1,4,1,2,3,1,4,1,2,1,3,4,1,2,1,4,3,1,2,1,4	4.00
24	(9,7,5,3)	--	1491107	1,2,1,3,4,1,2,1,3,1,2,4,1,3,2,1,2,3,1,4,2,1,3,2	12.51
24	(10,6,3,5)	--	196490	1,1,2,4,3,1,2,1,4,1,2,1,3,4,1,2,1,4,1,2,3,1,4,2	9.20
24	(11,5,4,4)	--	--	--	

Table 4: Results obtained for n = 5

Number of jobs (n) = 5					
D	Input vector (number of copies of each job)	Number of states generated by naïve algorithm	Number of states generated by improved algorithm	Optimal sequence	Optimal RTV
12	(2,3,3,2,2)	4855966	225354	1,2,4,3,5,2,1,3,4,2,5,3	0.00
12	(2,4,2,2,2)	3680930	174458	1,2,3,4,2,5,1,2,3,4,2,5	0.00
13	(3,3,3,2,2)	--	294574	1,2,4,3,1,2,5,3,1,4,2,3,5	3.00
13	(4,2,3,2,2)	--	324069	2,3,1,4,5,1,3,2,1,4,3,5,1	2.51
14	(4,3,3,2,2)	--	474747	2,4,3,1,2,5,3,1,4,2,1,3,5,1	2.33
14	(5,2,3,2,2)	--	1173899	1,2,1,3,4,1,5,1,3,2,1,4,3,5	5.46
14	(3,3,3,3,2)	--	123687	1,2,3,4,1,2,5,3,4,1,2,3,4,5	2.66
15	(3,2,3,4,3)	--	596162	1,2,3,4,5,1,3,4,2,5,1,4,3,5,4	5.25
15	(2,2,3,5,3)	--	1370138	1,3,4,2,5,4,3,1,4,5,2,3,4,5,4	5.00
15	(6,2,2,2,3)	--	1814285	1,2,1,3,5,1,4,1,2,5,1,3,1,4,5	3.00
16	(6,3,2,2,3)	--	2994087	1,2,1,3,5,1,2,4,1,5,1,3,2,1,5,4	2.66
16	(5,4,2,2,3)	--	1462874	1,2,1,3,5,1,2,4,1,5,2,3,1,2,5,4	5.46
16	(4,4,3,2,3)	--	763656	1,2,3,1,5,2,4,3,1,2,5,1,3,2,4,5	5.33
17	(4,4,4,2,3)	--	371550	1,2,3,4,1,5,2,3,1,2,5,3,4,1,2,3,5	5.41
17	(4,4,3,3,3)	--	--	--	--
18	(4,5,4,2,3)	--	1534327	1,2,3,4,1,2,5,3,1,2,3,5,4,2,1,3,2,5	9.20
18	(4,5,3,3,3)	--	--	--	--

Chapter 5

5.1 Conclusions and future research

The Response Time Variability Problem, recently defined in the literature, is a scheduling problem with a broad range of real-life applications that is very difficult to solve optimally. Several approaches have been proposed in the literature for solving the RTVP. One of the approaches is the dynamic programming approach. We studied the various properties of the response time variability problem. The dynamic programming approach for solving RTVP was studied. The dynamic programming approach gives the optimal results but, as the problem is NP-hard, this approach can not solve the problem instances of large sizes.

The objective of this dissertation work was to increase the size of the instances that can be solved optimally using dynamic programming. Some heuristics were applied on the naïve dynamic programming approach to improve it. By doing so, we were able to solve the larger instances of the problem. Computational results show that we were able to solve the instances which were not solved by naïve approach. But still, the size of instances that can be solved by our improved dynamic program is small. Further improvements can be done in this approach to solve larger instances of the problem.

There are some ways to proceed:

- Better heuristics can be researched to calculate the tighter upper and lower bounds.
- Equivalent states can be identified in the state space. For example, the sequences 1,2,1,2 and 2,1,2,1 are equivalent with equal value of RTV. If such states are identified, more states can be pruned.

The abovementioned ideas can further improve the dynamic programming approach. Besides these, another line of research could be the use of hyper-heuristics to solve RTVP. Hyper-heuristics are an emerging methodology in search and optimization. Hyper-heuristic methods choose dynamically the most suitable (meta)heuristic among a set of them according to the state of the search of the solution.

References:

[1] A. Corominas et. al., W. Kubiak, N. M. Palli. “Response time variability”. *Journal of Scheduling*, Vol. 10, pp. 97-110, Jan. 2007

[2] A. Corominas et. al., “Heuristics for the Response time variability problem”, Tech. Rep. IOC-DT-P-2009-03, Universitat Politecnica de Calalunya, EOLI, Dec. 2009

[3] A. Corominas et. al., “Solving the Response Time Variability Problem by means of metaheuristics”, *Special Issues of Frontiers in artificial intelligence and applications on Artificial Intellegence Research and Development.* Vol. 146, pp. 187-194, 2006.

[4] A. Corominas et. al., “Mathematical Programming Modeling of the Response Time Variability Problem”, *European Journal of Operational Research*, Vol. 200, pp. 347-357, 2010

[5] A. Garcia-Villoria, and R. Pastor , “Solving the Response Time Variability Problem by Means of a Genetic Algorithm”, *European Journal of Operational Research*, Vol. 202, pp. 320-327, 2010.

[6] A. Corominas et. al., “Using Tabu Search for the Response Time Variability Problem”, *3rd International Conference on Industrial Engineering and Industrial Management (CIO 2009)*, Barcelona and Terrassa, Spain, 2009.

[7] A. Corominas et. al., “Solving the Response Time Variability Problem by means of a Variable Neighbourhood Search Algorithm”, *13th IFAC Symposium of Information Control Problems in Manufacturing (INCOM 2009)*, Moscow, Russia, 2009

[8] A. Bar-Noy et al, “Minimizing service and operation costs of periodic scheduling”, *Mathematics of Operations Research*, Vol. 27, pp. 518-544, 2002.

[9] C. A. Waldspurger and W. E. Weihl, “Lottery Scheduling: Flexible Proportional-Share Resource Management”, *First USENIX Symposium on Operating System Design and Implementation*,1994

[10] C. A. Waldspurger, W. E. Weihl, “Stride Scheduling: Deterministic Proportional-Share Resource Management”, Tech. Rep. MIT/LCS/TM-528, Massachusetts Institute of Technology. MIT Laboratory for Computer Science, 1995.

[11] J. Miltenburg, “Level Schedules for Mixed-Model Assembly Lines in Just-In-Time Production Systems”. *Management science*, Vol. 35, pp. 192-207, 1989.

[12] J. W. Herrmann, "Generating Cyclic Fair Sequences Using Aggregation and Stride Scheduling". Tech. Rep. TR 2007-12, university of Maryland, USA, 2007. Available at http://hdl.handle.net/1903/7082.

[13] J. W. Herrmann, "Using Aggregation to Reduce Response Time Variability in Cyclic Fair Sequences", Tech. rep. 2008-29, University of Maryland, USA, 2008.

[14] N. Moreno, "Solving the Product Rate Variation Problem of Large Dimensions as an Assignment Problem", Doctoral Thesis, DOE, UPC ,2002.

[15] S. Anily et. al., "The Scheduling of Maintenance Service", *Discrete Applied Mathematics*, Vol 82, pp. 27-42, 1998.

[16] S. Bollapragada et. al., "Scheduling Commercial Videotapes in Broadcast Television", *Operations Research*, Vol. 52, pp. 679-689, 2004.

Appendix A

Initial sequences obtained using Webster's method

n= 2

D	Input vector (number of copies of each job)	Sequence generated by Webster's method	RTV
4	(2,2)	2,1,2,1	0.00
6	(4,2)	1,2,1,1,2,1	1.00
6	(3,3)	2,1,2,1,2,1	0.00
7	(5,2)	1,2,1,1,1,2,1	1.70
7	(3,4)	2,1,2,1,2,1,2	1.41
8	(5,3)	1,2,1,2,1,1,2,1	1.86
8	(6,2)	1,2,1,1,1,2,1,1	1.33
10	(8,2)	1,1,2,1,1,1,1,2,1,1	1.50
10	(7,3)	1,2,1,1,2,1,1,1,2,1	2.38
10	(6,4)	1,2,1,2,1,1,2,1,2,1	2.33

n= 3

D	Input vector (number of copies of each job)	Sequence generated by Webster's method	RTV
7	(3,2,2)	1,3,2,1,3,2,1	3.66
10	(3,3,4)	3,2,1,3,2,1,3,2,1,3	4.33
10	(5.3.2)	1,2,3,1,2,1,1,3,2,1	4.66
12	(2,6,4)	2,3,2,1,3,2,2,3,2,1,3,2	4.00
12	(2,7,3)	2,3,2,1,2,3,2,2,1,2,3,2	5.42
12	(2,8,2)	2,2,3,1,2,2,2,2,3,1,2,2	6.00
13	(3,8,2)	2,1,2,3,2,2,1,2,2,3,2,1,2	5.04
13	(4,7,2)	2,1,2,3,2,1,2,1,2,3,2,1,2	4.10
13	(5,6,2)	2,1,3,2,1,2,1,2,1,3,2,1,2	4.53
13	(5,5,3)	2,1,3,2,1,3,2,1,2,1,3,2,1	5.06

14	(7,4,3)	1,2,3,1,1,2,3,1,2,1,1,3,2,1	7.66
14	(6,5,3)	1,2,3,1,2,1,3,2,1,2,1,3,2,1	4.80
14	(6,6,2)	2,1,3,2,1,2,1,2,1,3,2,1,2,1	2.66
14	(5,7,2)	2,1,2,3,1,2,2,1,2,1,3,2,1,2	4.80
15	(7,4,4)	1,3,2,1,1,3,2,1,3,2,1,1,3,2,1	8.35
15	(6,5,4)	1,2,3,1,2,3,1,2,1,3,2,1,3,2,1	6.25
15	(5,7,3)	2,1,3,2,1,2,3,2,1,2,1,2,3,1,2	6.85
15	(5,8,2)	2,1,2,3,1,2,2,1,2,2,1,3,2,1,2	5.37
15	(4,8,3)	2,1,3,2,2,1,2,3,2,1,2,2,3,1,2	5.62
16	(8,4,4)	1,3,2,1,1,3,2,1,1,3,2,1,1,3,2,1	8.00
16	(7,5,4)	1,2,3,1,2,1,3,2,1,3,1,2,1,3,2,1	6.22
16	(6,6,4)	2,1,3,2,1,3,2,1,2,1,3,2,1,3,2,1	6.66
16	(2,9,5)	2,3,2,1,2,3,2,3,2,2,3,2,1,2,3,2	6.35
17	(3,9,5)	2,3,2,1,2,3,2,3,2,1,2,3,2,2,1,3,2	6.75
17	(4,8,5)	2,3,1,2,3,2,1,2,3,2,1,2,3,2,1,3,2	4.82
18	(2,9,7)	2,3,2,3,1,2,3,2,3,2,2,3,2,1,3,2,3,2	5.71
18	(3,10,5)	2,3,2,1,2,3,2,2,3,1,2,2,3,2,1,2,3,2	6.79
18	(4,10,4)	2,3,1,2,2,2,3,1,2,2,3,1,2,2,2,3,1,2	11.60
18	(6,9,3)	2,1,3,2,1,2,2,1,3,2,1,2,2,1,3,2,1,2	6.00
20	(8,9,3)	2,1,3,2,1,2,1,2,1,3,2,1,2,1,2,1,3,2,1,2	6.22
20	(7,10,3)	2,1,2,3,1,2,2,1,2,3,1,2,1,2,2,1,3,2,1,2	7.52
20	(6,10,4)	2,1,3,2,2,1,2,3,1,2,2,1,3,2,2,1,2,3,1,2	9.33
20	(5,10,5)	2,3,1,2,2,3,1,2,2,3,1,2,2,3,1,2,2,3,1,2	10.00
20	(2,10,8)	2,3,2,3,2,1,3,2,3,2,2,3,2,3,2,1,3,2,3,2	6.00
20	(7,8,5)	2,1,3,2,1,3,2,1,2,3,1,2,1,2,3,1,2,3,1,2	8.85
22	(7,8,7)	2,3,1,2,3,1,2,3,1,2,3,1,2,3,1,2,3,1,2,3,1,2	5.21
22	(7,10,5)	2,1,3,2,1,2,3,2,1,2,3,1,2,1,2,3,2,1,2,3,1,2	7.65
22	(6,10,6)	2,3,1,2,3,2,1,2,3,1,2,2,3,1,2,3,2,1,2,3,1,2	8.26
22	(5,11,6)	2,3,1,2,2,3,1,2,2,3,2,1,3,2,2,1,3,2,2,1,3,2	12.53
22	(2,8,12)	3,2,3,2,3,1,3,2,3,2,3,3,2,3,2,3,1,3,2,3,2,3	7.16
22	(4,7,11)	3,2,1,3,2,3,3,2,1,3,3,2,3,1,2,3,3,2,3,1,2,3	9.85
24	(4,10,10)	3,2,1,3,2,3,2,3,2,1,3,2,3,2,1,3,2,3,2,3,2,1,3,2	8.79

24	(6,8,10)	3,2,1,3,2,3,1,2,3,1,2,3,3,2,1,3,2,3,1,2,3,1,2,3	10.40
24	(9,8,7)	1,2,3,1,2,3,1,2,3,1,2,3,1,2,1,3,2,1,3,2,1,3,2,1	7.71
24	(7,10,7)	2,3,1,2,3,1,2,2,3,1,2,3,1,2,3,1,2,2,3,1,2,3,1,2	11.82
26	(7,12,7)	2,3,1,2,2,3,1,2,3,1,2,2,3,1,2,2,3,1,2,3,1,2,2,3,1,2	14.52
26	(5,12,9)	2,3,1,2,3,2,3,2,1,2,3,2,3,1,2,3,2,1,2,3,2,3,2,1,3,2	11.35
26	(2,14,10)	2,3,2,3,2,3,2,1,2,3,2,3,2,2,3,2,3,2,3,2,1,2,3,2,3,2	8.11
26	(8,11,7)	2,1,3,2,1,3,2,1,2,3,2,1,3,2,1,2,3,2,1,2,3,1,2,3,1,2	9.47
28	(10,11,7)	2,1,3,2,1,3,2,1,2,1,3,2,1,3,2,1,2,3,1,2,1,2,3,1,2,3,1,2	12.32
28	(9,8,11)	3,1,2,3,1,2,3,1,2,3,1,3,2,3,1,2,3,1,3,2,1,3,2,1,3,2,1,3	9.61
30	(11,8,11)	3,1,2,3,1,2,3,1,2,3,1,3,1,2,3,1,2,3,1,3,1,2,3,1,2,3,1,2,3,1	11.86
30	(7,9,14)	3,2,1,3,2,3,1,3,2,3,1,2,3,3,2,1,3,3,2,1,3,2,3,1,3,2,3,1,2,3	11.14

n= 4

D	Input vector (number of copies of each job)	Sequence generated by Webster’s method	RTV
10	(3,2,2,3)	4,1,3,2,4,1,3,2,4,1	9.33
10	(2,2,2,4)	4,3,2,1,4,4,3,2,1,4	9.00
12	(4,3,3,2)	1,3,2,4,1,3,2,1,4,3,2,1	8.00
12	(5,2,3,2)	1,3,4,2,1,3,1,1,4,2,3,1	11.20
12	(6,2,2,2)	1,4,3,2,1,1,1,4,3,2,1,1	12.00
13	(5,2,3,3)	1,4,3,2,1,4,3,1,1,2,4,3,1	11.03
13	(6,2,2,3)	1,4,3,2,1,1,4,1,3,2,1,4,1	10.50
13	(4,4,2,3)	2,1,4,3,2,1,4,2,1,3,4,2,1	6.66
15	(2,7,4,2)	2,3,2,4,1,2,3,2,3,2,4,1,2,3,2	10.60
15	(3,8,2,2)	2,1,2,4,3,2,2,1,2,2,4,3,2,1,2	11.87
15	(4,5,4,2)	2,3,1,4,2,3,1,2,3,1,2,4,3,1,2	8.00
18	(4,7,2,5)	2,4,1,2,3,4,2,1,4,2,1,2,4,3,2,1,4,2	7.91
18	(5,6,2,5)	2,4,1,3,2,4,1,2,4,1,2,4,1,3,2,4,1,2	10.39
18	(3,7,4,4)	2,4,3,1,2,2,4,3,2,1,4,3,2,2,1,4,3,2	17.71

18	(3,8,2,5)	2,4,1,2,3,4,2,2,4,1,2,2,4,3,2,1,4,2	10.69
20	(4,7,4,5)	2,4,3,1,2,4,2,3,1,4,2,3,1,2,4,2,3,1,4,2	14.85
20	(4,8,3,5)	2,4,1,3,2,4,2,1,2,4,3,2,1,2,4,2,3,1,4,2	10.66
20	(4,4,8,4)	3,4,2,1,3,3,4,2,1,3,3,4,2,1,3,3,4,2,1,3	18.00
20	(3,8,3,6)	2,4,3,1,2,4,2,4,2,3,1,2,4,2,4,2,3,1,4,2	16.66
20	(3,8,4,5)	2,4,3,1,2,4,2,3,2,4,1,2,3,2,4,2,1,3,4,2	10.66
22	(5,4,6,7)	4,3,1,2,4,3,1,4,2,3,4,1,3,2,4,1,3,4,2,1,3,4	12.39
22	(3,6,6,7)	4,3,2,1,4,3,2,4,3,2,4,1,3,2,4,3,2,4,1,3,2,4	10.19
22	(2,7,6,7)	4,2,3,4,2,3,1,4,2,3,4,2,3,4,2,3,1,4,2,3,4,2	11.04
22	(8,2,6,6)	1,4,3,1,4,3,2,1,4,3,1,1,4,3,1,4,3,2,1,4,3,1	12.16
24	(7,8,4,5)	2,1,4,3,2,1,4,2,1,3,2,4,1,2,3,1,2,4,1,2,3,4,1,2	12.51
24	(6,9,4,5)	2,1,4,3,2,1,2,4,3,2,1,4,2,1,2,3,4,2,1,2,3,4,1,2	16.80
24	(10,5,4,5)	1,4,2,3,1,1,4,2,1,3,1,4,2,1,3,1,4,2,1,1,3,4,2,1	16.00
24	(9,7,5,3)	1,2,3,4,1,2,1,3,2,1,4,3,2,1,1,2,3,1,2,4,1,3,2,1	14.51
24	(10,6,3,5)	1,2,4,1,3,2,1,4,1,2,1,4,3,1,2,1,4,2,1,3,1,4,2,1	11.20

n = 5

D	Input vector (number of copies of each job)	Sequence generated by Webster's method	RTV
12	(2,3,3,2,2)	3,2,5,4,1,3,2,5,4,1,3,2	18.00
12	(2,4,2,2,2)	2,5,4,3,1,2,2,5,4,3,1,2	16.00
13	(3,3,3,2,2)	3,2,1,5,4,3,2,1,5,4,3,2,1	17.00
13	(4,2,3,2,2)	1,3,5,4,2,1,3,1,5,4,2,3,1	16.91
14	(4,3,3,2,2)	1,3,2,5,4,1,3,2,1,5,4,3,2,1	16.33
14	(5,2,3,2,2)	1,3,5,4,2,1,3,1,1,5,4,2,3,1	21.46
14	(3,3,3,3,2)	4,3,2,1,5,4,3,2,1,5,4,3,2,1	10.66
15	(3,2,3,4,3)	4,5,3,1,2,4,5,3,1,4,2,5,3,1,4	15.25
15	(2,2,3,5,3)	4,5,3,2,1,4,5,4,3,4,2,1,5,3,4	19.00
15	(6,2,2,2,3)	1,5,4,3,2,1,1,5,1,4,3,2,1,5,1	21.00
16	(6,3,2,2,3)	1,5,2,4,3,1,1,5,2,1,4,3,1,5,2,1	20.66

16	(5,4,2,2,3)	1,2,5,4,3,1,2,5,1,2,1,4,3,5,2,1	17.46
16	(4,4,3,2,3)	2,1,5,3,4,2,1,5,3,2,1,4,5,3,2,1	15.33
17	(4,4,4,2,3)	3,2,1,5,4,3,2,1,5,3,2,1,4,5,3,2,1	11.41
17	(4,4,3,3,3)	2,1,5,4,3,2,1,5,4,3,2,1,5,4,3,2,1	21.50
18	(4,5,4,2,3)	2,3,1,5,4,2,3,1,5,2,3,1,2,4,5,3,1,2	15.20
18	(4,5,3,3,3)	2,1,5,4,3,2,1,5,4,3,2,1,2,5,4,3,1,2	24.20

Appendix B

The solution for n = 2

The problem instance for n = 2 can be solved by a quick procedure described in [1]. The sequence minimizing the RTV for two jobs is quite easy to obtain as follows: Let $d_1<d_2$. We omit the case $d_1 = d_2$ since it is trivial. If we can find a solution with distances $\lceil D/d_1 \rceil$ and $\lfloor D/d_1 \rfloor$ for job 1, and $\lceil D/d_2 \rceil$ and $\lfloor D/d_2 \rfloor$ for job 2, then this solution will be optimal. Such a solution is always possible since $\lfloor D/d_1 \rfloor$ >= 2 and 2> D/d_2 >1. Therefore starting a sequence with job 1 and then sequencing any consecutive copy of 1 at a distance either $\lceil D/d_1 \rceil$ or $\lfloor D/d_1 \rfloor$ (the number of times each distance is used are (D mod d_i) and (d_i – D mod d_i) respectively) from the last one will produce the sequence where empty positions are separated by at most a single copy of job 1. This allows us to fit in job 2 in the empty positions ensuring the desired distances for job 2. The resulting sequence will minimize RTV. The details of the algorithm with proofs is given in [1].