Chapter 1

Introduction

The response time variability problem (RTVP) is a sequence optimization problem. It occurs in real-life situations in which jobs, clients, products, or events need to be sequenced in order to minimize the variability in the time between two successive points at which they receive their necessary resources.

In the modern system the resources are shared between different jobs. When many jobs request a single resource at a time, then it is necessary to schedule the resource activities in some fair manner so that the jobs can receive the resource that is proportional to its job relative to the computing job. The job is defined as certain amount of work to be done.

The fair sequence (solution) concept has emerged from scheduling problem in different environments [1]. The common aim of scheduling problem is to minimize an objective function. The objective is to minimise the RTV metric value of the solutions. A fair sequence using n symbols must be copied d_i times in the sequence. This fair share of positions allocated to symbol i in a subsequence of length k is proportional to the relative importance of symbol i with respect to the total copies of symbols.

This problem appears in a broad range of real life applications. The following are some real life applications described in [2].

The situation in which the idea of the regular sequence appeared in the sequencing of mixed-model assembly lines at Toyota Motor Corporation under the just-in-time (JIT) production system. Toyota Motor Corporation used the JIT production system, one of the main aim 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 system produces necessary models in necessary quantities at necessary 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.

The RTVP also appears in computer multi-threaded systems [3]. Multi-threaded system performs different tasks of client program that takes place concurrently. These systems need to manage the resources in order to serve the request of n clients.

The Asynchronous Transfer Mode (ATM) defined in [4], networks divide each application (voice, large data file video) into cells of fixed size so that the application can be preempted after each cell. Applications for instance voice and video, requires that a inter-cell distance in a cell stream is 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 to early or too late which would result in jagged motion perceptions.

Another application is stride scheduling, in which the clients are issued the various number of tickets by the resource. The resources are then allocated in discrete time slices called quanta. The client to be allocated resources in next quantum is selected through a certain function of the number of its past allocations and the number of its tickets [5].

The RTVP can be applied to design the sales catalogues, the periodic machine maintenance problem as well as other distance-constrained problems. In distance-constrained scheduling problems the temporal distance between any two consecutive copies of a same task is not longer than the pre-specified distance. Sometimes even a stronger condition is imposed, So that the temporal distance is equal to the pre-specified distance.

The RTVP concept also uses the advertising agency and this application was reported in [6]. This study is motivated by the problem faced by the National Broadcasting Company (NBC). The NBC is one of the leading firms in the US television industry. Major advertisers buy hundreds of time slots to air commercials and often require that NBC space to air their commercials as evenly as possible over the entire broadcast season.

An application of RTVP in a healthcare facility described in [6] is needed to be scheduled the collection of waste materials from trash containers placed in various rooms. Based on the

frequency an employee had to visit each room and the fact that different rooms required a different number of visits per shift, the healthcare facility manager wants these visits to be as regular as possible to avoid excessive waste collecting in any room. For instance, if a room needed four visits per eight-hour shift, it should ideally be visited every two hours.

A resource must be shared between different computing demands that require regular attention, it is important to schedule the access right to the resource in same fair manner. In such a way that the different types of demands share the resource in same manner as mentioned in the above applications. These above mentioned applications are some examples of very common situations in manufacturing and in services, in which the RTVP can be applied. The objective in the RTVP is to minimize variability of the distance between any two consecutive units of the same product, client, job. ie. to have the distances between any two given consecutive units of the same product as constant as possible. The objective of this dissertation focuses primarily on the implement, compare and analyze the metaheuristic solutions to the RTVP. After covering the introduction the rest of the material in this dissertation is organized as follows:

Chapter 2 covers basic concept, literature review, complexity of the RTVP. In section 3.1 the metaheuristic methods for solving RTVP are described. These metaheuristic methods are multi-start, GRASP and PSO. Section 3.2 presents a brief overview of the initial sequences. Chapter 4 describes the results, compare and analysis of the metaheuristic methods to solve RTVP. Finally chapter 5 covers conclusions and future research areas.

Chapter 2

Basic concepts

2.1 The Response Time Variability Problem

The RTVP is the combinatorial optimization problem. The problem has been first solved by Waldspurger and Weihl in 1994 using a method called lottery scheduling [3] and formally formulated in [4] by A. Corominas. Its formulation is as follows. Let n be the number of symbols to be sequenced, and let i be the individual demand. where i is to be copied d_i times in the solution and let D is the total number of copies i.e. summation of the individual demands d_i. Let S be the solution sequence that consists of a circular sequence of copies$(S= S_1S_2\ldots\ldots S_D)$, where S_j is the copy that sequenced in position j of sequence S. for each symbol i in which $d_i>=2$, let t_k^i be the distance between the positions in which the copies $k+1$ and k of same symbol are found. ie. t_k^i is the gap between the consecutive pair of same job. We consider the gap between the two consecutive positions to be equal to 1. the solution sequence is circular, position 1 comes immediately after the last position D. let t_i be the average or constant distance between two consecutive copies of same symbol i. ie $t_i=D/d_i$. The aim is to minimize the metric RTV, which is defined by the following expression.

We define the Response Time Variability for i is as follows.

$$RTV_i = \sum_{i=k}^{d_i} (t_k^i - t_i)^2$$

And the total Response Time Variability is defined as.

$$RTV = \sum_{i=1}^{n} RTVi$$

$$RTV = \sum_{i=1}^{n} \sum_{k=1}^{d_i} (t_k^i - t_i)^2$$

From above,

$$RTV = \sum_{i=1}^{n} \sum_{k=1}^{d_i} (t_k^i - t_i)^2$$

$$= \sum_{i=1}^{n} \sum_{k=1}^{d_i} (t_k^i)^2 + \sum_{i=1}^{n} \sum_{k=1}^{di} (t_i)^2 - \sum_{i=1}^{n} (2.t_i. \sum_{k=1}^{di} t_k^i)$$

$$= \sum_{i=1}^{n} \sum_{k=1}^{d_i} (t_k^i)^2 + \sum_{i=1}^{n} \sum_{k=1}^{di} (t_i)^2 - \sum_{i=1}^{n} (2.t_i. D)$$

Since

$$\sum_{i=1}^{n} \sum_{k=1}^{di} (t_i)^2 \quad and \quad \sum_{i=1}^{n} (2.t_i. D)$$

are constant, the problem of minimizing Response Time Variability is equivalent to minimizing the

$$\sum_{i=1}^{n} \sum_{k=1}^{d_i} (t_k^i)^2 .$$

Thus, the distance between any two consecutive copies of the same symbol should be as regular as possible.

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 $t_C=2$.

Consider the sequence C A C B C B A C is a solution and has

RTV = $((5-4)^2 + (3-4)^2) + ((2-4)^2 + (6-4)^2) + ((2-2)^2 + (3-2)^2) = 12$

2.2 Complexity

The RTVP has been proved to be NP-hard in [4]. The solution of the RTVP can be improved. It is a combinatorial problem and no polynomial time algorithm is known yet to solve the RTVP. Authors of [4] studied the computational complexity of the RTVP and proved that it is NP-hard.

The RTVP is NP-hard proved by reducing in to the periodic maintenance scheduling problem. The periodic maintenance scheduling problem is defined as follows. Given M machines and integer service intervals $l_1, l_2, \ldots\ldots, lm$ such that $\sum(1/li)<1$. does there exist a servicing schedule $S_1, S_2, \ldots\ldots, S_L$. Where $L=lcm(l_1, l_2, \ldots\ldots\ldots,lm)$ is the least common multiple of $l_1, l_2, \ldots., lm$ of these machines in which consecutive servicing of machine i are exactly li 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 in [7].

2.3 Literature Survey

The RTVP is an optimization scheduling problem. The solution obtained from the different methods can be improved. The response time variability problem was first reported by Wildpurger and Weihl in 1994 in [3] and formally formulated in [4]. The RTVP has been first time solved in [5] using a method called lottery scheduling. This method is based on generating a solution at random as follows. For each position of the solution, the symbol to be sequenced is chosen at random and the possibility 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 [5] that obtains better results than the lottery scheduling method.

The RTVP is in general NP-hard problem. The polynomial time algorithm is not known yet to solve the real-life application instances. The two-product case can be solved optimally with a polynomial time proposed in [7]. For the other cases authors in [4] proposed a mixed-integer linear programming (MILP) whose practical limit to obtain optimal solutions is 25 copies to be solved. Same authors proposed an improved MILP model and increased the practical limit for obtaining optimal solutions from 25 to 40 copies to be solved.

For solving largest instances, heuristic methods have been proposed in [4]. The bottleneck algorithm was used in [8] to solve the Minmax Product Rate Variation Problem. The two classical parametric methods for solving the apportionment problem called Webster method and Jeffersion method are defined in [8]. Webster's method and Jeffersion's method are parametric

methods. The parametric methods are defined as follows. Let x_{ik} be the number of copies of symbol *i* that have been already sequenced in the solution of length k; the next symbol to be sequenced in position k+1 is $i^* = avg\ max_i \{d_i/x_{ik}+\delta\}$, Where $\delta \in (0,1]$. Webster's and Jefferson's methods are uses a δ value equal to 0.5 and 1, respectively. Same author also describe a random sequence generation method. A new heuristic called insertion was also discussed which is based on grouping symbols into fictitious symbols until only two fictitious symbol remains and then solving optimally using two-product case.

The best exact method to solve RTVP is a MILP which is able to solve optimally instances up to 40 units to be scheduled in a practical time. To overcome from this limitation a branch and bound (B&B) algorithm was proposed in [6]. This algorithm is to increase the size of the instances that can be solved optimally. The proposed B&B algorithm is able to solve larger instances up to 55 units to optimally.

The author of [9] was proposed an aggregation method based on grouping iteratively the symbols with the same number of copies to be sequenced into fictitious symbols and then applying a parametric method.

The simulated annealing (SA) approach has been proposed in [10] to solve the RTVP. SA can be seen as a variant of a local search procedure in which it is allowed moving to a worse solution with small probability. A simple SA-based algorithm is able to improve the results.

Many algorithms are based on metaheuristic schemes and other approaches have also been proposed. Some proposed techniques are:

1 Algorithms based metaheuristics (multi-start, GRASP and PSO).
2 Dynamic programming algorithm.
3 Variable neighborhood search algorithm.
4 Mixed integer linear programming (MILP) method.
5 Tabu search algorithm.
6 Genetic algorithm.

Among these algorithms the Tabu search algorithm and MILP algorithms are supposed to be the best algorithms for small instances. These algorithms cannot give optimal solution for large instances. Thus, the use of heuristic or metaheuristic methods for solving real life RTVP instances is justified. This dissertation work emphasis in the metaheuristic approaches, particularly, multi-start and GRASP method.

Hyper-heuristic algorithms are proposed in [11] to solve the RTVP. Hyper-heuristic algorithms have two classes. The first is based on constructive heuristics, whereas the second uses improvement methods. Hyper-heuristic method is "heuristics to choose heuristics". Hyper-heuristics apply the right heuristic method in the problem solving process. It operates indirectly on the solutions by choosing the heuristic and metaheuristic to be applied. Hyper-heuristic method can be applied to a new problem quickly and cheaply. This method can be divided into two categories: constructive hyper-heuristics and improvement hyper-heuristics.

Chapter 3

The Metaheuristic Methods to Solve RTVP

For solving the RTVP many algorithms are proposed. RTVP is the sequencing problem and no polynomial time algorithm is known yet to solve it. Many algorithms are proposed to find the near-to-optimal solution of RTVP. Authors of [12] discussed the problem and suggested some solutions based on metaheuristic methods. Some of these metaheuristic procedures are: Multi-Start (MS), Greedy Randomized Adaptive Search Procedure (GRASP) and Particle Swarm Optimization (PSO). The MS and GRASP methods are as follows:

3.1 Multi-Start method

The multi-start method is one of metaheuristic procedure for solving the RTVP proposed in [12]. The multi-start metaheuristic is a general scheme that consists of two phases. The first phase obtains an initial solution and in the second phase it improves the obtained initial solution by using the local optimization method and select the best of them.

The pseudocode of the adaptation of the multi-start method is:

1. Let the value of the best solution found be $\overline{z} = \infty$.
2. While (actual time<execution time) do:
3. Get a random initial solution X.
4. Apply the local optimization to X and get X'.
5. If value (X') < $\overline{Z}$, then $\overline{Z}$ = value (X').

The multi-start algorithm to solve the RTVP is based on a random initial solution and on improving it by means of local optimization procedure. Random solutions are generated as follows. For each position from 1 to D in the solution, a job to be sequenced is chosen at random. The probability of choice of each job is equal to the number of copies of this job that remain to be sequenced divided by the total number of copies that remain to be sequenced.
The local optimization is applied as follows. A local search is performed iteratively in a neighborhood that is generated by interchanging each pair of two consecutive units of the

sequence that represents the current solution. The best solution in the neighborhood is chosen, the optimization ends when no neighboring solution is better than current solution. If the quality of initial solution is poor, the computing time required by local search procedure to find the local optimized solution is increased.

Another form of multi-start algorithm can be obtained by using initial sequence as insertion sequence in [13]. Insertion method is suggested in [14]. In insertion sequence, for more than two products the problem is reduced into two-product. And then solve it by using two-product case method.

Let the demands be $d_1 \leq \ldots\ldots\ldots\ldots \leq d_n$. consider n-1 two case problem.

$$P_{n-1} = (d_{n-1}, d_n),\ p_{n-2} = (d_{n-2}, \sum_{J=n-1}^{n} d_j)\ \ldots\ldots\ldots,\ p_1 = (d_1, \sum_{j=2}^{n} d_j).$$

In each of the problem the first product is the original and second product will be the assumed product, and denoted by the *. Let the sequences $s_{n-1}, s_{n-2}, \ldots\ldots\ldots, s_1$, be the optimal solution. For the given problems they can be obtained by using the two case- problem. The solution is made up of the product j and *. The sequence of the original problem is built recursively by first replacing * in S_1 by S_2 to obtain S_1'. Next * are replaced by S_3 in $S_{1'}$ to obtain the solution S_1''. Sequence S_{n-1} replaces all the remaining * and obtain the final solution. The local optimization is same as in the original multi-start algorithm.

3.2 The Greedy Randomized Adaptive Search Procedure (GRASP) Method

The GRASP metaheuristic method can be considered as a variant of multi-start [12]. In GRASP, the generation of initial solution is obtained by greedy method. In this, random steps are added and choice of elements to be included in the sequence is adaptive. The probability of each job is proportional to the value of an associated index. The algorithm of the GRASP adaptation is

almost same as the multi-start method. The only difference is in constructing the initial solution. For each position from 1 to D, the next job to be sequenced is randomly selected from a list with a probability proportional to the value of its associated index. The associated index suggested in [12] is Webster index.

The Webster sequence is obtained as follows.
Let X_{ik} be the number of units of job i, that have been already sequenced in the sequence of length k, $k=0, 1, \ldots\ldots, d_i$ the number of units of the job i and D the total number of units, the value of the Webster index of product i to be sequenced position k+1 is $d_i/X_{ik}+\delta$.
Here δ is the Webster's parametric metrics and $\delta=1/2$.

Another variation of GRASP can be obtained by using the Jefferson sequence as initial solution. In the Jefferson's sequence the parametric matrices $\delta=1$ is used in [12]. This parameter affects the relative priority of low demand jobs and their position in the sequence. When δ is near to 0, low demand jobs will be positioned earlier in the solution but when δ is near to 1, low demand jobs will be positioned later in the solution.

3.3 Particle Swarm Optimization (PSO)

Particle Swarm Optimization (PSO) metaheuristic algorithm was designed by Kinnedy and Eberhart by establishing an analogy to the social behavior of flocks of birds, when they search for food described in [12]. PSO is population based metaheuristic algorithm which used for solving the optimization scheduling problem such as RTVP. This metaheuristic was designed to optimize continuous functions of real variables. In this method the particles corresponding to the birds, have a position (a feasible solution) and a velocity (the change in their position), and the set of particles form the swarm, which corresponds to the flock. The behavior of a particle is the result of the combination of the following three factors:

1. To continue on the path that it is following.
2. To follw the best solution found and
3. To go to the best position found by the swarm.

The initial sequences are generated as in the multi-start method. The PSO method, iteratively update the sequence and the velocity of each particle as it looks for the optimal solution.

3.4 The Initial Sequences

3.4.1 Two- product case

The two-product case, $n=2$. It shows a solution that minimizes both the total response time variability and the maximum deviation at a same time, which generally is impossible for more than two products. Let d_1 and d_2 are two products with different number of demands. We omit the case $d_1=d_2$ since it is trivial. When solving the two-product case, the first copy of the product with the less number of copies is assigned to the first position and the remaining copies are placed in the sequence $D \bmod d_i$ times with a distance $\lceil D/di \rceil$ to the last position assigned and $d_i - D \bmod d_i$ times with a distance $\lfloor D/di \rfloor$ to the last position assigned [12].

3.4.2 Bottleneck Sequence

The bottleneck sequences can be obtained by solving the bottleneck problem to optimally with the method described in [7].

3.4.3 Random Sequence

The bottleneck sequence S has been randomized as follows. For each position x in the sequence 1….D, get a random number *ran* in the range 1…..D. Then, swap S[x] with S[*ran*] [7].

The detailed analysis of the above methods are described in [7]. All of these methods have comparable results.

Chapter 4

Result and Analysis

4.1 Computational Experiment

The metaheuristic algorithms have been run for 882 different instances, which are grouped into four different categories. Formation of category is based on paper [15]. Category 1 include 162 instances, category 2 include 192 instances, category 3 include 282 instances and category 4 include 246 instances. The corresponding instances are same for every category of different algorithms. The instances in the first category CATEGORY 1 were generated using a random value of D between 25 and 50, and a random value of n between 3 and 15. For the second category CATEGOTY 2, D was between 50 and 100 and number of demands n between 3 and 30; For the third CATEGORY 3, total number of copies i.e. D was from 100 to 200 and number of demands i.e. n between 3 and 65; and finally for the fourth class CATEGORY 4, number of copies are between 200 and 500 and number of demands are between 3 to 150. The instances have been generated by first fixing the total number of copies D and the number of demands n. for all instances and for each type of product $i= 1,..., n,$ a random value of d_i is between 1 and D. the program has been executed to obtained the output of demands among which some of them were executed for several minutes.

The average initial RTV values (AIRTV), the average optimized RTV values (AORTV) and the average number of iterations required to obtained the optimized sequence (No. of iterations) using in multi-start, $GRASP_{we}$ (GRASP use Webster's sequence as initial solution) and $GRASP_{je}$ (GRASP with use of Jeffersion's sequence as initial solution) metaheuristic algorithms. The experimental result is tabulated in given tables.

Category	Average initial RTV	Average optimal RTV	No. of iterations
Global	137515.75	326	1402
CAT1	890	25	58
CAT2	4837	57	192
CAT3	34050	232	781
CAT4	510286	990	4577

Table 1: Experimental result of multi-start algorithm.

Category	Average initial RTV	Average optimal RTV	No. of iterations
Global	21352.50	292.5	929.25
CAT1	144	38	22
CAT2	1056	80	119
CAT3	5114	315	726
CAT4	79096	737	2850

Table 2: Experimental result of $GRASP_{we}$.

Category	Average initial RTV	Average optimal RTV	No. of iterations
Global	18698.25	244	629
CAT1	143	29	21
CAT2	941	63	95
CAT3	4537	219	322
CAT4	69172	665	2078

Table 2: Experimental result of $GRASP_{je}$.

By analyzing these tables, for all instances in initial value of RTV the $GRASP_{je}$ is 12.43%better than $GRASP_{we}$ and 86.40% better than multi-start. But in optimized RTV value $GRASP_{je}$ is 16.58% better than $GRASP_{we}$ and 25.15% better than multi-start.$GRASP_{je}$ also take the less number of iterations for obtaining the optimized solution i.e. it is faster than other two methods. $GRASP_{je}$ take the 32.31%less number of iterations than $GRASP_{we}$ and 55.13% less number of iterations than multi-start. By analyzing above table, the multi-start algorithm obtains the good averages for small instances (category 1 and category 2) but, poor average results for large instances (category 4). $GRASP_{je}$ and $GRASP_{we}$ gives the better result than multi-start algorithm for large instances (category 4).

4.2 Computational Results

The following different table shows the results obtained from the metaheuristic algorithms to solving the RTVP. For four categories the total number of copies D and the number of input n is fixed and different instances are generated. For each instance the initial RTV value, optimized RTV value and no. of iterations are computed.

Multi-Start method				
Category 1				
N	D	Average initial RTV	Average optimal RTV	No. of iterations
3	25	59	12	10
3	26	92	14	13
4	27	96	16	15
4	28	220	18	22
5	29	208	21	22
5	30	274	18	29
6	31	296	21	36
6	32	373	20	38
7	33	409	20	35
7	34	541	27	43
8	35	572	28	41
8	36	792	49	51
9	37	858	21	57
9	38	1011	20	97
10	39	1972	48	66
10	40	1084	29	78
11	41	1108	43	73
11	42	1446	31	92
12	43	1192	34	69
12	44	1477	21	87

13	45	1391	32	82
13	46	1690	29	98
14	47	1621	23	98
14	48	1798	37	98
15	49	1738	18	93
15	50	1681	24	98
3	50	42	18	14

Category 2				
N	D	Average initial RTV	Average optimal RTV	No. of iterations
3	50	76	18	14
3	51	119	24	15
3	100	171	39	7
4	52	130	28	17
5	53	617	47	63
6	54	577	45	52
7	55	941	63	67
8	56	2107	58	121
9	57	1969	51	126
10	58	2442	55	124
11	59	2828	70	142
12	60	3353	66	158
13	61	3553	61	172
14	62	3719	61	169
15	64	3064	54	146
16	65	3658	49	175
17	67	5529	52	219
18	70	5582	84	234

19	72	5626	85	241
20	73	5056	64	228
21	74	6836	60	271
22	75	6872	67	261
23	76	7679	44	313
24	78	7310	50	288
25	80	8161	46	292
26	85	9551	75	348
27	87	9199	60	317
28	88	8988	41	320
28	90	10121	104	378
29	95	8288	50	305
30	100	11302	52	386
30	98	9353	90	190

Category 3				
N	D	Average initial RTV	Average optimal RTV	No. of iterations
3	100	171	39	18
4	101	489	84	41
5	102	3241	91	154
6	103	1072	118	87
7	104	1356	139	121
8	105	8638	206	159
9	106	6543	140	283
10	108	5229	178	270
11	109	14749	207	352
12	110	13174	185	371
13	111	14663	139	484
14	112	17476	180	544
15	115	18496	233	554
16	120	24746	258	591
17	122	34504	272	764
18	125	28997	1920	2192
19	129	26229	199	743
20	132	36490	268	784
21	133	41537	253	763
22	134	38828	247	851
23	136	41251	302	854
24	139	31654	300	781
25	142	49581	254	999
26	146	41239	239	983
27	148	55287	264	1009
28	150	44451	282	961
29	153	48213	362	987

30	160	56083	337	1122
31	162	59624	260	1224
32	165	53536	160	1163
33	169	53536	160	1163
34	171	56543	200	1193
35	172	53181	176	1139
36	175	47506	144	1127
37	178	45540	148	1094
38	180	43455	165	980
39	182	59586	186	1276
40	183	44981	137	1040
41	185	50876	130	1153
45	189	45619	160	1024
47	192	48033	155	1004
50	194	60561	277	1245
53	196	64345	463	1342
55	198	6787	132	1196
60	199	77484	116	1268
65	200	64432	141	1128
3	200	771	115	60

Category 4				
N	D	Average initial RTV	Average optimal RTV	No. of iterations
3	200	771	115	60
4	210	125	42	15
5	220	61125	494	290
6	225	5595	540	301
7	230	2962	334	220
8	240	41362	805	741
18	290	60870	2281	2192
25	320	50234	2445	2271
35	335	268438	1077	4439
48	359	299257	707	4823
65	200	64432	141	1128
65	370	53753	325	5889
72	379	649814	809	5823
78	388	797601	1242	7419
83	392	818188	701	6884
95	400	940530	806	7743
100	409	915885	568	7188
120	462	1469352	536	10530
140	494	1626431	626	10902
148	498	1362236	433	8845
150	500	1227050	433	8417

$GRASP_{we}$ method				
Category 1				
N	D	Average initial RTV	Average optimal RTV	No. of iterations
3	25	10	6	2
3	26	9	9	1
3	50	16	15	1
4	27	16	13	4
5	29	25	17	4
5	30	33	26	4
6	31	38	23	7
6	32	46	29	7
7	33	48	29	8
7	34	45	33	8
8	35	65	42	9
8	36	72	41	11
9	37	99	40	19
9	38	99	56	15
10	39	126	61	21
10	40	114	72	12
11	41	188	51	32
11	42	227	36	39
12	43	211	49	36
12	44	178	91	18
13	45	252	53	38
13	46	264	46	46
14	47	347	52	58
14	48	382	52	55
15	49	506	30	71
15	50	457	37	71

4	28	16	16	1

Category 2				
N	D	Average initial RTV	Average optimal RTV	No. of iterations
3	50	16	15	1
3	51	17	15	2
3	100	26	24	1
4	52	31	23	3
5	53	72	35	7
6	54	62	33	11
7	55	70	43	13
8	56	75	59	6
9	57	135	60	21
10	58	124	82	17
11	59	142	89	21
12	60	105	118	25
13	61	281	85	50
14	62	319	90	57
15	64	387	107	54
16	65	442	129	66
17	67	576	74	104
18	70	678	169	86
19	72	723	90	140
20	73	949	123	133
21	74	1267	74	170
22	75	1391	69	167
23	76	1688	48	193
24	78	1944	92	187

25	80	2148	55	220
26	85	2439	63	442
27	87	2438	72	256
28	88	2721	102	257
28	90	2660	74	260
29	95	2966	145	266
30	98	3460	117	291
30	100	3451	194	277

Category 3				
N	D	Average initial RTV	Average optimal RTV	No. of iterations
3	100	26	24	1
4	101	47	39	4
5	102	74	60	7
6	103	110	54	24
7	104	120	90	8
8	105	119	94	11
9	106	159	105	22
10	108	219	127	33
11	109	209	244	26
12	110	253	177	42
13	111	300	198	33
14	112	333	180	43
15	115	411	427	58
16	120	363	248	56
17	122	683	286	83
18	125	599	343	61
19	129	670	350	92

20	132	706	386	76
21	133	909	502	85
22	134	1011	471	134
23	136	1079	523	123
24	139	1411	416	565
25	142	1589	704	163
26	146	1746	421	280
27	148	1902	583	268
28	150	2038	665	233
29	153	2256	542	340
30	160	2754	566	406
31	162	2945	699	394
32	165	3525	449	491
33	169	3500	528	475
34	171	4794	378	558
35	172	4982	319	5677
36	175	6394	274	698
37	178	7408	184	754
38	180	8051	237	717
39	182	7538	310	746
40	183	9727	232	812
41	185	10769	154	869
45	189	13923	203	11165
47	192	15102	300	902
50	194	15556	381	1005
53	196	19062	393	1050
55	198	20302	230	1209
60	199	25194	222	1323
65	200	34371	381	1278

Category 4				
N	D	Average initial RTV	Average optimal RTV	No. of iterations
3	200	57	49	5
4	210	47	29	8
5	220	160	134	12
6	225	198	140	23
7	230	252	104	52
8	240	300	173	53
18	290	2030	933	241
25	320	5241	1003	639
35	335	6227	1150	1024
48	359	15832	1026	1981
65	200	34371	381	1278
65	370	23015	1253	2315
72	379	33271	2820	2994
78	388	37883	1714	3421
83	392	50256	3601	3539
95	400	79400	971	4684
100	409	98204	1269	4880
120	462	179585	1732	6843
140	494	301352	1082	8558
148	498	374087	599	8786
150	500	419248	750	8516

$GRASP_{je}$ method				
Category 1				
N	D	Average initial RTV	Average optimal RTV	No. of iterations
3	25	9	7	1
3	26	9	9	0
3	50	16	16	0
4	27	17	11	4
4	28	18	15	1
5	29	26	16	6
5	30	29	16	6
6	31	46	22	9
6	32	47	16	13
7	33	54	23	9
7	34	65	35	13
8	35	65	27	15
8	36	88	36	19
9	37	100	38	19
9	38	105	52	14
10	39	177	41	25
10	40	146	38	24
11	41	195	35	32
11	42	223	41	36
12	43	220	36	33
12	44	269	28	36
13	45	206	37	27
13	46	302	39	44
14	47	401	44	50
14	48	301	26	41
15	49	356	35	46

15	50	373	42	47

Category 2				
N	D	Average initial RTV	Average optimal RTV	No. of iterations
3	50	16	16	0
3	51	19	15	3
3	100	24	22	1
4	52	29	17	5
5	53	48	32	8
6	54	68	25	9
7	55	70	52	17
8	56	102	46	18
9	57	155	50	30
10	58	143	84	17
11	59	197	79	30
12	60	299	68	43
13	61	403	91	117
14	62	401	69	65
15	64	361	61	50
16	65	442	62	58
17	67	669	74	84
18	70	871	90	102
19	72	876	80	101
20	73	837	72	122
21	74	1214	71	132
22	75	1216	90	124
23	76	1663	73	162
24	78	1560	66	149

25	80	1510	49	142
26	85	2230	64	204
27	87	2103	126	170
28	88	2347	48	202
28	90	2444	88	212
29	95	2379	70	206
30	98	2809	89	226
30	100	2606	70	234

Category 3				
N	D	Average initial RTV	Average optimal RTV	No. of iterations
3	100	24	22	1
4	101	51	33	8
5	102	75	53	9
6	103	107	49	21
7	104	130	66	20
8	105	161	99	24
9	106	187	109	27
10	108	248	110	37
11	109	749	182	103
12	110	377	190	37
13	111	470	189	61
14	112	567	167	87
15	115	562	180	89
16	120	824	243	90
17	122	1004	228	129
18	125	1082	180	144
19	129	1046	233	151

20	132	1394	260	150
21	133	1568	289	166
22	134	1924	267	232
23	136	2058	325	205
24	139	2325	314	265
25	142	2723	374	236
26	146	2409	323	277
27	148	2946	330	292
28	150	3150	282	322
29	153	3563	428	341
30	160	3989	307	419
31	162	4205	384	413
32	165	4791	280	454
33	169	5182	266	467
34	171	5503	346	450
35	172	5677	259	514
36	175	5984	315	487
37	178	6372	265	547
38	180	6596	223	527
39	182	7537	278	600
40	183	8584	192	622
41	185	8657	195	639
45	189	11166	173	688
47	192	11711	186	637
50	194	12869	376	691
53	196	14478	341	767
55	198	14740	264	762
60	199	16392	142	778
65	200	22524	176	834

Category 4				
N	D	Average initial RTV	Average optimal RTV	No. of iterations
3	200	53	48	4
4	210	47	29	8
5	220	168	124	19
6	225	191	116	31
7	230	232	111	44
8	240	290	176	42
18	290	1514	489	216
25	320	4789	772	450
35	335	7800	987	870
48	359	18017	2423	1429
65	200	2252	176	834
65	370	34395	1757	1952
72	379	43405	817	2435
78	388	56087	1175	2568
83	392	64562	813	2906
95	400	92981	921	3400
100	409	100340	604	3600
120	462	176803	823	5033
140	494	261247	854	6036
148	498	278589	453	5741
150	500	308841	469	5970

Chapter 5

Conclusions and Future Research

The response time variability problem is an NP-hard scheduling problem. This scheduling problem arises in a variety of real-life applications including mixed-model assembly lines, multi-threaded computer systems, periodic machine maintenance and waste collection. In the RTVP, the aim is to minimize variability in the distances between any two consecutive copies of the same symbol. i.e. to distribute the symbols as regular as possible. Several algorithms have been proposed in the literature for solving the RTVP. Since it is an NP-hard problem, metaheuristic methods are needed for solving real life problems. A computational experiment was done and its result show that on average the $GRASP_{je}$ is better than $GRASP_{we}$ and multi-start. But for small instances multi-start is better metaheuristic method for solving RTVP. In addition the $GRASP_{je}$ method has a stable behavior for small, medium and large instances.

Future research may focus on improving the result to solve the RTVP is adding the simulated annealing algorithms and variable neighborhood search hybrid algorithms as low-level heuristics in the metaheuristics.

References

[1] S. Salhi, A. Garcia-Villoria,(2011). An adaptive search for the ResponseTimeVariability Problem. *Journal of the Operational Research Society*, 0, 1-9.

[2] A. Corominas, A. Garcia-Villoria, R. Pastor,(2011). Metaheuristic algorithmms hybridized with variable neighbourhood search for solving response time variability problem. *Institute of Industrial and Control Engineering (IOC),U*niversitat Politecnuca de Catalunga *(UPC)*, Barcelona, Spain.

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

[4] A. Corominas et. al., W. Kubiak, N. M. Palli,(2007). Response time variability. *Journal of Sched uling*, 10, 97-110.

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

[6] A. Garcia-Villoria, A. Corominas, X. Delorme, A. Dolgui, W. Kubiak, R. Pastor,(2012). A branch and bound algorithm for the Response time variability problem. *Journal of Scheduling*.

[7] A. Corominas, W. Kubik, N. M. Palli,(2004). Response time variability , IOC-DT- 2004- 08.

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

[9] J.W.Herrmann,(2008). Using Aggregation to Reduce Response Time Variability in Cycle Fair Sequences. Tech. Rep. University of Maryland, USA.

[10] A. Corominas, A. Garcia-Villoria, R. Pastor,(2010). A new metaheuristic procedure for improving the solution of the response time variability problem. S*cheduling and Sequencing*.

[11] A.Garcia-Villoria, S. Salhi, A. Corominas, R.Pastor,(2011). Hyper-Heuristic approaches for the response time variability problem. *European Journal of Operational Research*, 211, 160-169.

[12] A. Corominas et. al.,(2006). Solving the Response Time Variability Problem by means of metaheuristics. *Special Issues of Frontiers in artificial intelligence and Applications onArtificial Intellegence Research and Development.* 146, 187-194.

[13] N. K. Ray,(2012). Improved Multi-Start method for solving The Response Time Variability Problem. *Master's dissertations,* Computer science and IT, T.U..

[14] A. Corominas, W. Kubiak and R. Pastor,(2009). Heuristics for the response time variability problem. *Institute of industrial and control engineering,* IOC-DT-P-2009-03.

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

Appendices

Code for GRASP$_{we}$

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
#include<alloc.h>
#include<stdlib.h>

#define n 150
#define D 500
float rtv(int *,float *);
int max(float *);
float webster(float *);
int min(float *);
int sequence[D];
void main()
{
float rt;
float d[n];
float e[n];
int i,j,m,k=0,temp,t=1;
int seq1[D];
int seq2[D];
float rem[n];
int temparray[D][D];
float rtvs[D];
float current_rtv;
int x;
int xxx=0;
```

```
printf("Input vector");
for(i=0;i<n;i++)
 scanf("%f",&d[i]);
 rt=webster(d);
 printf("\nRTV=%f\n",rt);
 current_rtv=rt;
 for(j=0;j<D;j++)
 seq1[j]= sequence[j];
 while(t==1)
    {
 printf("\n \n\n");
 for(i=0;i<D-1;i++)
      {
 for(j=0;j<D;j++)
 seq2[j]= seq1[j];
 temp=seq2[i];
 seq2[i]=seq2[i+1];
 seq2[i+1]=temp;
 rtvs[i]=rtv(seq2,d);
 for(j=0;j<D;j++)
 temparray[i][j]= seq2[j];
 printf("\n ");
 for(j=0;j<D;j++)
 printf(" %d ",seq2[j]);
 printf("   :rtv=  %f",rtv(seq2,d));*/
      }

 for(j=0;j<D;j++)
 seq2[j]= seq1[j];
```

```
temp=seq2[0];
   seq2[0]=seq2[D-1];
   seq2[D-1]=temp;
   rtvs[D-1]=rtv(seq2,d);
   printf("\n ");
   for(j=0;j<D;j++)
   printf(" %d ",seq2[j]);
   printf("   :rtv=  %f",rtv(seq2,d));
   printf("\n  ");
   for(j=0;j<D;j++)
   temparray[D-1][j]= seq2[j];
    x=min(rtvs);
   if(rtvs[x]<current_rtv)
     {
   current_rtv=rtvs[x];

   for(j=0;j<D;j++)
   seq1[j]= temparray[x][j];
      }

 else
   {
 t=0;

 printf("\n optimized sequence: ");

 for(i=0;i<D;i++)
 printf(" %d ",seq1[i]);
 printf("\n  optimized rtv = %f",current_rtv);
 printf("\n Iterations=%d", xxx);
   }
```

```
   xxx++;
     }
getch();

   }
float webster(float *x)
   {
int s[D];
int d1[n];
int length,j,m;
float temp[n];              //Webster index
float r;
int i;
for(i=0;i<n;i++)            //temporary vector to keep track of
 d1[i]=0;                  //no. of sequenced copies
for(length=0;length<D;length++)
     {
 for(j=0;j<n;j++)
               {
if(d1[j]<=x[j])
temp[j]= (x[j])/(d1[j]+0.5);
               }
m=max(temp);                        //return position of max element
sequence[length]=m+1;
d1[m]=d1[m]+1;
     }
printf("\n\n  initial sequence: ");
for(i=0;i<D;i++)
printf("  %d",sequence[i]);
r=rtv(sequence,x);
```

```
return(r);
}
float rtv(int *x,float *y)              //To calculate RTV on initial sequence
   {
int i,j,k,l,m,p,q=0;
int distances[D];
float avg[n];
float rt;
for(i=0;i<D;i++)
distances[i]=0;
p=0;
rt=0;
                                        //calculate average distances for each symbol
for(i=0;i<n;i++)
avg[i]=D/y[i];
printf("\n\n  Average distances: ");
for(i=0;i<n;i++)
printf("  %f",avg[i]);*/
                                       //calculate actual distances between copies of symbols
for(i=0;i<n;i++)
              {
l=0;
for(j=0;j<y[i];j++)
                {
m=1;
while(x[l]!=i+1)
l=(l+1)%D;
k=(l+1)%D;
while(x[k]!=x[l])
                  {
 m++;
```

```
k=(k+1)%D;
                }
distances[p]=m;
l=k;
p++;
        }
           }
printf("\n\n distances: ");
for(i=0;i<D;i++)
printf("%d  ",distances[i]);   */
for(i=0;i<n;i++)
      {
for(j=0;j<y[i];j++)
       {
 rt=rt+((distances[q]-(D/y[i]))*(distances[q]-(D/y[i])));
 q++ ;
       }
      }
return rt;
   }
int max(float *x)
   {
int i,j=0;
float m;
m=x[0];
for(i=1;i<n;i++)
     {
if(x[i]>=m)
              {
m=x[i];
j=i;
```

```
}
        }
return j;
   }
int min(float *x)
   {
int i,j=0;
float m;
m=x[0];
for(i=1;i<D;i++)
     {
 if(x[i]<m)
              {
m=x[i];
j=i;
              }
     }
return j;
  }
```

Code for $GRASP_{je}$

```
#include<stdio.h>
#include<conio.h>
#include<math.h>
#include<alloc.h>
#include<stdlib.h>
#define n 150
#define D 500
float rtv(int *,float *);
int max(float *);
float webster(float *);
int min(float *);
int sequence[D];
void main()
  {
float rt;
float d[n];
float e[n];
int i,j,m,k=0,temp,t=1;
int seq1[D];
int seq2[D];
float rem[n];
int temparray[D][D];
float rtvs[D];
float current_rtv;
int x;
int xxx=0;
printf("Input vector");
for(i=0;i<n;i++)
scanf("%f",&d[i]);
 rt=webster(d);
```

```
printf("\nRTV=%f\n",rt);
current_rtv=rt;
for(j=0;j<D;j++)
seq1[j]= sequence[j];
while(t==1)
      {
 printf("\n \n\n");
 for(i=0;i<D-1;i++)
        {
 for(j=0;j<D;j++)
seq2[j]= seq1[j];
temp=seq2[i];
seq2[i]=seq2[i+1];
seq2[i+1]=temp;
rtvs[i]=rtv(seq2,d);
for(j=0;j<D;j++)
temparray[i][j]= seq2[j];
 printf("\n ");
 for(j=0;j<D;j++)
 printf(" %d ",seq2[j]);
 printf("    :rtv=  %f",rtv(seq2,d)); */
        }
for(j=0;j<D;j++)
seq2[j]= seq1[j];
temp=seq2[0];
seq2[0]=seq2[D-1];
seq2[D-1]=temp;
rtvs[D-1]=rtv(seq2,d);
 printf("\n ");
 for(j=0;j<D;j++)
printf(" %d ",seq2[j]);
```

```
printf("   :rtv=  %f",rtv(seq2,d));
printf("\n  ");  */
for(j=0;j<D;j++)
temparray[D-1][j]= seq2[j];
x=min(rtvs);
if(rtvs[x]<current_rtv)
        {
current_rtv=rtvs[x];
for(j=0;j<D;j++)
seq1[j]= temparray[x][j];
         }
else
      {
t=0;
printf("\n optimized sequence: ");

for(i=0;i<D;i++)
printf(" %d ",seq1[i]);
printf("\n  optimized rtv = %f",current_rtv);
printf("\n iterations: %d",xxx);
       }
xxx++;
     }
getch();
   }
float webster(float *x)
   {
int s[D];
int d1[n];
int length,j,m;
float temp[n];              //Webster index
```

```
float r;
int i;
for(i=0;i<n;i++)          //temporary vector to keep track of
d1[i]=0;                  //no. of sequenced copies
for(length=0;length<D;length++)
     {
for(j=0;j<n;j++)
               {
if(d1[j]<=x[j])
temp[j]= (x[j])/(d1[j]+1);
               }
m=max(temp);                    //return position of max element
sequence[length]=m+1;
d1[m]=d1[m]+1;
     }
printf("\n\n  initial sequence: ");
for(i=0;i<D;i++)
printf("  %d",sequence[i]);
r=rtv(sequence,x);
return(r);
   }
float rtv(int *x,float *y)            //To calculate RTV on initial sequence
   {
int i,j,k,l,m,p,q=0;
int distances[D];
float avg[n];
float rt;
 for(i=0;i<D;i++)
 distances[i]=0;
p=0;
rt=0;
```

```
                        //calculate average distances for each symbol
for(i=0;i<n;i++)
avg[i]=D/y[i];
printf("\n\n  Average distances: ");
for(i=0;i<n;i++)
printf("  %f",avg[i]);*/
                                //calculate actual distances between copies of symbols
for(i=0;i<n;i++)
            {
l=0;
for(j=0;j<y[i];j++)
             {
m=1;
while(x[l]!=i+1)
l=(l+1)%D;
k=(l+1)%D;
while(x[k]!=x[l])
              {
m++;
k=(k+1)%D;
              }
distances[p]=m;
l=k;
p++;
             }
            }
printf("\n\n distances: ");
for(i=0;i<D;i++)
printf("%d  ",distances[i]);
for(i=0;i<n;i++)
    {
```

```
for(j=0;j<y[i];j++)
        {
rt=rt+((distances[q]-(D/y[i]))*(distances[q]-(D/y[i])));
q++ ;
        }
       }
return rt;
   }
int max(float *x)
   {
int i,j=0;
float m;
m=x[0];
for(i=1;i<n;i++)
     {
if(x[i]>=m)
               {
m=x[i];
j=i;
               }
     }
return j;
   }
int min(float *x)
   {
int i,j=0;
float m;
m=x[0];
for(i=1;i<D;i++)
                {
if(x[i]<m)
```

```
            {
 m=x[i];
j=i;
            }
     }
return j;
   }
```

Code for multi-start method

```
#include<stdio.h>
 #include<conio.h>
 #include<math.h>
 #include<alloc.h>
 #include<stdlib.h>
#define n 150
#define D 500
    float rtv(int *,float *);
    int max(float *);
    float rtv(int *,float *);
    int min(float *);

  void main()
  {
  float rt;
  float d[n];
  float e[n];
  int i,j,m,k=0,temp,t=1;
  int sequence[D];
```

```
  int seq1[D];
  int seq2[D];
  float rem[n];
  int remaning = D;
  int temparray[D][D];
  float rtvs[D];
  float current_rtv;
  int x;
  int xxx;
  printf("Input vector");
  for(i=0;i<n;i++)
 scanf("%f",&d[i]);
 for(i=0;i<n;i++)
e[i]= d[i];
 for ( i=0;i<D; i++)
                        {
 for (j=0;j<n;j++)
            {
 rem[j]=e[j]/remaning;
         }
 m=max(rem);
 sequence[i]=m+1;
 remaning=remaning-1;
e[m]=e[m]-1;
      }
 rt=rtv(sequence,d);
 printf("\n Initial sequence: ");
 for(i=0;i<D;i++)
 printf(" %d ",sequence[i]);printf("\nRTV=%f\n",rt);
    current_rtv=rt;
    for(j=0;j<D;j++)
```

```
 seq1[j]= sequence[j];
  while(t==1)
  {
 printf("\n \n\n");
for(i=0;i<D-1;i++)
    {
for(j=0;j<D;j++)
seq2[j]= seq1[j];
temp=seq2[i];
seq2[i]=seq2[i+1];
seq2[i+1]=temp;
rtvs[i]=rtv(seq2,d);
for(j=0;j<D;j++)
temparray[i][j]= seq2[j];
 printf("\n ");
 for(j=0;j<D;j++)
 printf(" %d ",seq2[j]);
 printf("   :rtv=  %f",rtv(seq2,d)); */
    }
 for(j=0;j<D;j++)
 seq2[j]= seq1[j];
 temp=seq2[0];
 seq2[0]=seq2[D-1];
 seq2[D-1]=temp;
 rtvs[D-1]=rtv(seq2,d);
 printf("\n ");
 for(j=0;j<D;j++)
 printf(" %d ",seq2[j]);
 printf("   :rtv=  %f",rtv(seq2,d));

 printf("\n  ");
```

```
 for(j=0;j<D;j++)
 temparray[D-1][j]= seq2[j];
 x=min(rtvs);
 if(rtvs[x]<current_rtv)
    {
 current_rtv=rtvs[x];
 for(j=0;j<D;j++)
 seq1[j]= temparray[x][j];
     }
 else
   {
 t=0;
 printf("\n optimized sequence: ");
 for(i=0;i<D;i++)
 printf(" %d ",seq1[i]);
 printf("\n  optimized rtv = %f",current_rtv);
 printf("\n iterations: %d",xxx);
   }
xxx++;
  }
getch();

 }
float rtv(int *x,float *y)            //To calculate RTV on initial sequence
 {
int i,j,k,l,m,p,q=0;
int distances[D];
float avg[n];
float rt;
for(i=0;i<D;i++)
distances[i]=0;
```

```
p=0;
rt=0;
                                    //calculate average distances for each symbol

for(i=0;i<n;i++)
avg[i]=D/y[i];
printf("\n\n  Average distances: ");
for(i=0;i<n;i++)
 printf("  %f",avg[i]);*/
                                        //calculate actual distances between copies of symbols
 for(i=0;i<n;i++)
            {
 l=0;
 for(j=0;j<y[i];j++)
              {
  m=1;
  while(x[l]!=i+1)
  l=(l+1)%D;
  k=(l+1)%D;
  while(x[k]!=x[l])
                {
  m++;
  k=(k+1)%D;
                }
  distances[p]=m;
   l=k;
   p++;
              }
            }
   printf("\n\n distances: ");
   for(i=0;i<D;i++)
```

```
  printf("%d  ",distances[i]);   */
  for(i=0;i<n;i++)
   {
  for(j=0;j<y[i];j++)
     {
  rt=rt+((distances[q]-(D/y[i]))*(distances[q]-(D/y[i])));
  q++ ;
     }
   }
 return rt;
}

 int max(float *x)
{
 int i,j=0;
 float m;
 m=x[0];
 for(i=1;i<n;i++)
  {
if(x[i]>=m)
            {
 m=x[i];
 j=i;
            }
  }
 return j;
}
int min(float *x)
 {
int i,j=0;
float m;
```

```
m=x[0];
for(i=1;i<D;i++)
  {
 if(x[i]<m)
            {
 m=x[i];
  j=i;
            }
  }
  return j;
 }
```