THE EARLY TARDY SCHEDULING PROBLEM USING JAVA REMOTE METHOD INVOCATION

A Thesis Presented to
The Faculty of the
Fritz J. and Dolores H. Russ
College of Engineering and Technology
Ohio University

In Partial Fulfillment
of the Requirement for the Degree
Master of Science

by
Gopalkrishna Phadke
June, 2001
Acknowledgements

I wish to express my sincere thanks to my advisor Dr. Thomas Keyser for his valuable guidance and assistance during the course of this research. He has encouraged me throughout my study at Ohio University. He has always helped me by taking into consideration my physical problems, which I greatly appreciate.

I would also like to express my gratitude to other faculty members of IMSE department. I would like to thank my friends Satya, Arun, Vinod, Sadhish, Habi and Prasad for all their help.

Last but not the least, I thank my family members for their encouragement and I dedicate my work to my mother Aparna and father Anil.
Table of Contents

List of Figures .................................................................................................................. vii
List of Tables ..................................................................................................................... viii
List of Equations ............................................................................................................... ix

1 Introduction ..................................................................................................................... 1
  1.1 Early Tardy Problem ..................................................................................................... 2
    1.1.1 Industrial Example ............................................................................................... 3
  1.2 Distributed Processing towards Scheduling Problems ......................................... 4
  1.3 Related Work .............................................................................................................. 5
  1.4 Proposed solution ....................................................................................................... 5
    1.4.1 Description ........................................................................................................... 6
    1.4.2 Problem breadth .................................................................................................... 7
    1.4.3 Problem depth ....................................................................................................... 8
    1.4.4 Significance and Objectives ................................................................................9

2 Previous Work ................................................................................................................ 11
  2.1 Early Tardy Scheduling Background ....................................................................... 11
  2.2 Distributed Computing and Scheduling ................................................................. 15
  2.3 Java and Java RMI .................................................................................................... 18
    2.3.1 Remote Method Invocation ................................................................................ 18

3 Methodology .................................................................................................................. 22
  3.1 Problem Structure .................................................................................................... 22
    3.1.1 Node Selection .................................................................................................... 22
3.1.1.1 Precedence Relationships ........................................ 24
3.1.2 Scheduling of Jobs ..................................................... 25
3.2 Classes and Functions Developed .................................... 30
3.3 Data flow ...................................................................... 31
3.4 Data Storage .................................................................. 33
3.5 Design of the Distributed Algorithm ................................. 33
  3.5.1 Client ...................................................................... 33
  3.5.2 Server ...................................................................... 34
3.6 Experimental design ....................................................... 36
  3.6.1 Factors .................................................................... 36
  3.6.2 Design of Experiments ............................................. 37
4 Results ............................................................................. 39
  4.1 Analysis of Variance ...................................................... 39
  4.2 Network Load Effect ..................................................... 44
  4.3 Speed-up ...................................................................... 46
  4.4 RMI vs Sequential Algorithm ........................................ 48
  4.5 Other results .................................................................. 48
5 Conclusions and Future Research ...................................... 50
  5.1 Analysis and Interpretation ............................................ 50
  5.1.1 Distributed Algorithm .............................................. 51
  5.1.2 Statistically Significant Factors ................................ 53
  5.1.3 Statistically Insignificant Factors ............................... 55
5.2 Sequential Algorithm Vs Distributed Algorithm ...........................................56
5.3 Network Load Effect ..................................................................................57
5.4 Recommendations .....................................................................................58
References .......................................................................................................62
Appendix 1: Code for algorithm without RMI ....................................................65
Appendix 2: Code for RMI Algorithm ...............................................................81
Appendix 3: Experimental Results ..................................................................102
List of Figures

Figure 2.1 Functioning of the distributed process ........................................... 19
Figure 2.2 View of the data flow between JVM's ............................................. 19
Figure 3.1 Tree structure of a 3 job problem ................................................... 23
Figure 3.2 Moving a block to find optimal position .................................... 27
Figure 3.3 Unititing two blocks and moving both for optimality ..................... 28
Figure 3.4 Block cannot be moved ................................................................. 28
Figure 3.5 Early job is added to a new block ................................................. 29
Figure 3.6 Interaction between classes and data flow .................................... 32
Figure 3.7 Client Server relationship ............................................................ 35
Figure 4.1 Normal probability plot of residuals before log transform ............... 40
Figure 4.2 Residuals versus fitted values ....................................................... 40
Figure 4.3 Normal Probability Plot after log transforms .................................. 41
Figure 4.4 Residuals versus fitted values after log transform .......................... 41
Figure 4.5 Main Effects Plot - Data Means for time ...................................... 43
Figure 4.6 Interaction plot ............................................................................. 44
Figure 4.7 Speed up diagram ........................................................................ 47
Figure 5.1 Times for 1 client 1 server same machine, random penalties .......... 50
Figure 5.2 Times for 1 Client 1 Server same and different computers ............. 51
Figure 5.3 Times for 1 client 1 server same machine and 1 client 2 servers ...... 52
Figure 5.4 Time RMI and non-RMI algorithm for 7 jobs random penalty ........ 57
Figure 5.5 Times at different intervals of the day ........................................... 58
List of Tables

Table 4.1 ANOVA after the log transform: 95% confidence level ....................... 42
Table 4.2 Times for 1 Client 1 Server same and different computers ............... 45
Table 4.3 Solution times at different time intervals ...................................... 45
Table 4.4 Times for 1 client 1 server same machine and 1 client 2 servers ....... 47
Table 4.5 Solution times and speed up chart .............................................. 47
Table 4.6 Comparison of RMI with Single computer algorithm ..................... 48
Table 4.7 Times for 1 client 1 server same machine ..................................... 49
List of Equations

Equation 3.1 due date tightness (ddt) ................................................................. 38
Equation 3.2 Co-efficient of Variation (Cv) .......................................................... 38
Equation 4.1 Speedup ......................................................................................... 46
1 Introduction

With increasing competition in the world market, companies have to use available resources in the best possible way. Utilizing machines to produce goods optimally is one of the key factors in reducing the cost of the goods. With the use of Just In Time manufacturing (JIT), companies cannot afford late deliveries, nor can they afford to have too much inventory lying in their warehouses. Reducing inventory levels has been an effort of modern manufacturers over the past decade. A balance has to be struck between these two factors. It is possible to minimize the penalties (cost of inventory and penalties due to lost orders) by scheduling jobs in an optimal sequence. Scheduling has become one of the most important activities in the process planning stage. As the number of jobs to be scheduled increases, the number of job sequences also increases. As a result, it becomes more and more difficult to find the best schedule, in the least possible time. Solving these problems requires multi-criteria optimization, which typically implies a larger number of calculations. As the number of machines and jobs increase, the complexity of the problem increases factorially. Increasing the number of jobs increases the number of calculations and the amount of data needed to be stored. It becomes impossible for even the fastest computers, to find an optimal solution because of the memory and computational limitations. In this research, we consider the single machine early tardy problem, with non-symmetric penalties while allowing idle time between consecutive jobs. This is a sub-problem of a scheduling
problem with multiple machines and multiple jobs using multiple criteria/penalties. The Single Machine Early Tardy Problem (SMETP)\(^1\) has been dealt by many researchers in the past [1,2,3,4,5,7,8,9,10,11,12,14]. Their works differ in the assumptions, constraints and algorithms used. In the coming chapters, an explanation of the problem is given and a review of the previous work is elaborated.

### 1.1 Early Tardy Problem

With the advent of JIT, interest in the Single Machine Early Tardy Problem has increased. The Single Machine Early Tardy Problem demands a high degree of computational power to obtain a solution. When the constraints on these problems increase, computations become complex, resulting in increased solution time. For example, consider a case where the jobs are scheduled on a machine with the only constraint being that, the total finishing time of all the jobs should be as small as possible. The total number of job combinations increase factorially, as the number of jobs increase. Additionally, constraints can be applied, such that, if a job finishes late, it will be charged a late penalty and if it finishes before it’s due-date, an earliness penalty will be charged to it. Another example can be, where a job is charged a penalty by the customer for being late and inventory costs are incurred if the job is early. These earliness and tardiness penalties can be possibly different and vary for different

---

\(^1\) SMETP stands for Single Machine Early Tardy Problem
jobs. Thus, the aim changes, to that of finishing jobs in the lowest possible penalty. These added criteria make the problem increasingly complex and computationally expensive. For large number of jobs, the problem becomes so large, that most of the time an optimal solution cannot be reached and a heuristic solution method has to be utilized.

1.1.1 Industrial Example

A steel manufacturer in Spartanburg, SC stamps quarter panels for automobiles and supplies them to major car manufacturers. One production line in this manufacturing plant is devoted to the production of quarter panels. A single machine-press dominates production line operation. This machine dominates, as there is only one dye press combination available, due to the high cost of the dye. In a production line, the utilization of the line as a whole, cannot be more than the efficiency of its slowest or the bottleneck machine. As this machine dominates the whole production line, scheduling of jobs on this machine, has to be given the maximum priority. [6]

One of the car manufacturers insisted on timely delivery and penalized for late deliveries. The car manufacturer also refused early deliveries. It rejected products that arrived more than 2 days ahead of the schedule. Therefore, the stamping company could not deliver products, even if, they were produced ahead of schedule. This caused the manufacturer to invest in additional inventory space to store finished products, ready to be delivered. The added inventory also means locked capital or dead investment. Though there is no
direct earliness penalty from the car manufacturer, the manufacturer had to bear the cost of inventory. The rejection of early deliveries, indirectly levied an earliness penalty. Again, the problem is analogous to a single machine scheduling problem with early/tardy penalties, as the stamping operation alone, determined the throughput and hence the penalties.

1.2 Distributed Processing towards Scheduling Problems

Often optimal solutions cannot be determined to scheduling problem, due to excessive computation time required. Distributed processing can become an alternative, to the solution of the problem. It may be solved faster and at a lower cost by making use of the idle computers on a network using distributed computing. At the first glance, it seems to be an obvious way to solve the problem, but there are some critical issues, that have to be considered. An algorithm should be used in such a way that, it can be broken down into smaller independent problems. These problems will be solved individually on different computers. Additionally, making sub-problems as small as possible minimizes communication between problems. Typically, a main program will monitor and manage all other processes. Communication exists between the various processes, running simultaneously, as they require sharing of a number of values. This creates communication overhead. This inter-computer communication can have a detrimental effect on the solution time for a problem.
Thus, how the problem is partitioned into tasks and how tasks are dispatched to the computers needs to be considered [12].

1.3 Related Work

In the past, work has been done towards this problem. Problems have been solved on single, as well as, multiple computers. The approach and the algorithms to solve these problems differ. Different algorithms and heuristics are proposed to solve the problem. This problem has been dealt, with using varying assumptions, regarding the number of jobs, idle time within the jobs and early and tardy penalties. Research has been done to solve this problem, using distributed computing [12,13,14] and studying different factors, affecting the time to solve the problem. The effort of all the past work is to find an optimal solution to problems, with as many jobs as possible, using the minimum possible time. A study on the effect of due-date tightness and co-efficient of variation is done by Keyser T.K [11]. This past work will be reviewed in detail in the following pages.

1.4 Proposed solution

In past efforts, the programming languages used to solve these problems are Fortran, Pascal, PVM, C and other languages [12,13,14]. These programs are executed on a single computer, as well as, on multiple computers. Java is used to solve the problem in consideration. A branch and bound algorithm with breadth first search technique, is used to solve this problem. The problem
allows different due-dates, unequal penalties and idle times between jobs. The algorithms and methods are explained in detail in the coming pages.

1.4.1 Description

Java is proposed in the current situation. Java is a language developed by Sun Microsystems. Java is chosen because it is a contemporary programming language and it can be applied to distributed computing in a variety of ways including Remote Method Invocation. Additionally, Java is platform independent and the code, once written can be used on any operating system without any changes in the code. Comapared to other languages Java is slower, however, it provides a mechanism (Java Remote Method Invocation), to access idle computer resources, within a network of computers, irrespective of the operating system on the computers, provided access is granted on the target computer.

Java RMI (Remote Method Invocation) is used for running the scheduling algorithm simultaneously on different computers. The scheduling algorithm used, is based on a branch and bound technique which is explained in detail in the coming chapters. By using Java RMI, the much-needed computational power for these computationally expensive problems, can be acquired at a very low cost.

Java enables larger size problems to be solved, via distributed computing. In addition the object-oriented structure of Java, lends itself to using a branch and bound tree structure effectively and adds the flexibility to the code
for future changes. This study reveals the effectiveness of Java for this particular problem. This work solves the single machine early tardy problem via distributed computing using Java RMI. In addition to the above advantages there may be some shortcomings. As Java converts the byte-code to machine code at run time, execution time is sacrificed. This may pose an additional problem in the speed of execution. The following provides an overall view of work that will be done.

1.4.2 Problem breadth

Optimal solutions are very important and they provide a basis for statistical comparisons. This work finds an optimal solution to the Single Machine Early Tardy Problem. The following assumptions are made

- All jobs are available at start of the process.
- There are no breakdowns of machines.
- Early and tardy penalties differ and are different for different jobs: This comes from the fact that a job may be charged a penalty by the customer if late or it may incur an inventory overhead and other earliness penalties.
- Idle time is allowed between jobs.
- A job is processed to its completion once started.
Due-dates and processing times of jobs vary: Due-dates vary as jobs can require completion at different periods of time and processing times for each job will vary, as different operations might be required for each job. These assumptions have affected the solution times in the past studies [7,12].

1.4.3 Problem depth

The effect of different factors on the time to solve the problem is analyzed. The problem is solved using Java as the programming language. First the problem is solved on a single computer and then on multiple computers. Results, regarding the effect of these factors and their interactions on solution time, are presented. The factors considered are

*Number of computers:*

The work can be distributed among a set of computers over the network and thus reducing the time to solve the problem.

*Number of jobs:*

As the number of jobs increase, the number of combinations to be tested, increases factorially which, increases the number of calculations and thus the solution time.
Due-date tightness:
As job due-dates are closer, more and more jobs are scheduled tightly and thus more combinations of the positions of jobs require testing for optimality. This increases the number of calculations required to find an optimal schedule for a particular set of jobs.

Different early and tardy penalties:
With each job having different penalties, a sequence of jobs has to be found, such that, a job with higher penalties (early or tardy) finishes closer to its due-date. At the same time, other job penalties and their positions should also be considered. This increases the number of calculations and thus affects the solution times.

Co-efficient of variation:
Little variation in the due-dates, can increase, the number of calculations and thus affect the time to obtain an optimal solution.

1.4.4 Significance and Objectives

The objective of the study is, to analyze the viability of the distributed computing in case of the SMETP with the stated factors. It also reveals the suitability of Java, for these kind of computationally intensive problems. It is intended to guide future work with suggestions towards the solution of larger size problems. It demonstrates, the feasibility of using multiple platforms over a
network of computers, to solve the problem under consideration. It also provides a basis, for the solution of similar problems, by identifying, which of the factors described above, affect the time required to obtain a solution.

In the following chapters, the problem is dealt with step by step. Chapter 2 provides, previous work in the area of single machine early tardy problem as well as work done, in the distributed computation of this problem. Chapter 3 explains the design of the scheduling algorithm and the way it is applied to distributed computing, using Remote Method Invocation. Chapter 4 presents the results and the factor analysis using a Design of Experiments approach. Chapter 5 presents the conclusions and suggestions toward future research.
2 Previous Work

In the past, work has been done to solve this problem. The approach and the algorithms used to solve to these problems differ. The research is aimed to find an optimal solution to the problems in a minimum possible time. A review of some of the work, which is relevant to the research dealt here and is summarized below.

2.1 Early Tardy Scheduling Background

Baker and Schudder [3] have done a review of the past work on the early tardy scheduling problem. Different models of this problem have been studied. These problems are classified as, the one with common due-dates and with different due-dates. The models in different studies have considered different earliness and tardiness penalties, job dependant penalties, non-linear penalties and job deadlines. According to the survey, the general problem with different due-dates, job dependent penalties and idle times between jobs, needs more attention and new heuristics that can solve the problem efficiently, need to be determined.

Gutjahr [9] presents a branch and bound technique for single machine tardiness problem with job weights. Their observation is, small size problems can be solved, but a medium and large size problem do not yield the optimal solution due to complexity and size of the problem.
Algorithms, such as, dynamic programming and some heuristic procedures have been used to deal with the problem. Chung Lun [5] solved the problem with different earliness and tardiness penalties. A dynamic programming algorithm is used. They propose, that a branch and bound algorithm may be able to find an optimal solution. The general case problem with idle times and different early and tardy penalties is solved using a heuristic, to get a near optimal solution.

Almeida [1] gives a composite heuristic to solve the SMETP problem. The algorithm combines three search techniques, namely, the tabu search, simulated annealing and steepest descent techniques, to get a near optimal solution. The composite algorithm uses three search techniques associated with different neighbourhood definitions in an alternating four-phase procedure. The search technique is guided dynamically. By changing the search strategies, according to the insight gained from the procedure, it is possible to reduce the computing time and improve the solution quality.

Avital Lann and Grur Mosheiov [2] study three different cases of the single machine early tardy problem. They study cases where

- Early and tardy penalties are equal for all jobs.
- Early and tardy penalties are equal for one job.
- Asymmetric early and tardy penalties.

For the first two cases, they use a polynomial time algorithm and dynamic programming algorithm respectively. Optimal solution can be found in the first
two cases but, in the last case, the problem doesn’t yield an optimal solution. A heuristic method, which uses the dynamic programming algorithm, is proposed to solve the problem. The two heuristics suggested, are based on sequence improvement procedure and sequence building procedure respectively. They select one of these heuristics, to solve the problem, depending on the way the due-dates are spread over.

Many researchers have used branch and bound algorithms to deal with this problem. They use different constraint sets to limit the search. Ching-Fang Liaw [4] propose a branch and bound algorithm for a problem with different earliness and tardiness penalties and no idle times within jobs. They prune the tree, using three constraints and thus limit the search. In the first level dominance rules are used, in the second level an adjacent pair wise interchange technique is used to compare the costs when two jobs are interchanged, while at the third level, if the cost of the partial schedule plus the calculated lower bound is more than calculated upper bound, the investigation at that node is terminated. They use a depth first search strategy. From their computational experience, better dominance rules can improve the search results and larger size problems can be solved.

George Li [8] deals with a problem with early and tardy penalties and no idle time. A set of precedence constraints is used to reduce the search and then the problem is broken in two parts and solved using the developed lower bound and upper bound techniques. An upper bound on the problem is found using a
heuristic method. Depending on the tardiness factor, one of forward sequencing or backwards sequencing is decided. A set of 4 precedence rules is applied to limit the search. The first set comprises of rules depending on the ratios of penalties to their processing times. The second test comprises of adjacent pair wise interchange. The third test, a dominance principle of dynamic programming is used and if the node is not yet discarded, then a fourth rule is used to see if it should be fathomed. In the fourth test the cost of partial schedule is added to the lower bound and compared to the upper bound. If it is more than the upper bound, then the node is discarded. The branch and bound algorithm is able to find solution to problems including 50 jobs.

Hirakawa [10] gives a branch and bound algorithm connected directly to Lawler's decomposition. The method, tries to minimize, the total tardiness when idle times within the jobs, are not permitted. The algorithm uses three steps. In the first step, a decomposition pattern is generated, using Lawler's decomposition. Thus, each of these patterns, has, one fixed order job and two sub-problems. If, each sub-problem has more than one job or has tardiness, the problem is decomposed further into sub-problems. In the second step, when all the patterns are decomposed, selection of the best pattern is done. The tardiness of the first pattern evaluated becomes the upper bound. A preemptive performance calculation of tardiness is done, to avoid, unnecessary calculations.
All the work listed above, considers, special cases of the general problem and doesn't consider insertion of idle time between jobs. Fry [7] considers, a more realistic problem, where idle times are permitted within jobs. The procedure presented, tries to minimize the mean absolute lateness (MAL) of the single machine-scheduling problem. The problem considered, selects an optimal sequence of jobs with idle time between consecutive jobs. A branch and bound algorithm is used to solve the problem. The tree structure built is in last-in-first-out manner, which means the job at the root node is the last to be scheduled. An initial solution is calculated, using, pair-wise switching heuristic. The insertion of idle time is done, using an algorithm that uses clusters. Job clusters are shifted ahead in time, depending on the change in the total penalty by doing so. A set of precedence constraints is used, to discard the nodes on the tree and limit the search. The effect of, due-date tightness and coefficient of variation of due-dates on solution times, is studied.

2.2 Distributed Computing and Scheduling

The above research deals with the SMETP problem that is solved, on a single computer. Other research efforts have solved the problem, using distributed computing.

Kudva and Penky [13] have developed a distributed architecture for the branch and bound calculations (DCABB). The processors are functionally equivalent. Each process in the system, executes, the DCABB process
consisting of two sub-processes, called threads. The computation thread performs the task of exploring the tree, while the communication thread acts as an interface between processors on the network. The computation thread has the queue of tasks to be done. Once the computation is done, the result is either discarded or put in to the queue for further processing. The draw back is that, there is no single processor to keep track of the most recent process variables and thus overhead of inter-process communication is increased. There is a processor that is always updated, with the current best solution and other processors are updated when required. The different issues regarding processor connectivity, inter-process communication, workload balancing and network failures are also considered during the design of the system. Problems are solved, on as many as, 32 computers connected by a network. The results show that this approach is quite promising to solve branch and bound problems.

Keyser [12] solves a scheduling problem using the distributed computing approach. Distributed computing is applied to branch and bound algorithm. The branch and bound algorithm is solved using a depth first strategy. At a node, a linear program is used to optimize the model. The whole tree is investigated until all nodes are investigated or fathomed. This basic algorithm is solved in a distributed way, over a set of workstations, which share a parent and child relationship. The parent process keeps track of the global variables, distributes the tasks and communicates with the children. The child processes work on the assigned task and arrive with the solution for the sub-problem. The parent then
decides, whether, to discard the solution, depending on the current best solution. The issues considered are, the effect of number of computers used, size of the problem, the partitioning strategy, allocation strategy and type of scheduling problem solved. The problems are solved on a set of Unix workstations. Algorithms are coded in C in the PVM (Parallel Virtual Machine) environment.

Roucairol [16] summarizes the problems encountered in designing and analyzing parallel algorithms. Problems encountered, while using distributed computing, toward combinatorial problems are discussed. A branch and bound algorithm for distributed computing is proposed. The related problems, such as, load balancing, communication overhead and search strategies are discussed. Theoretical and practical results of distributed computing of large combinatorial problems are given, which suggest, that, this method can be used successfully for the manufacturing scheduling problem.

Mitra [14] solves integer programs using distributed processing. Their study focuses on problems that can be encountered. Longer solution times than sequential branch and bound, can be encountered, because of redundant node investigation or excessive communication overheads. They make use of a two-stage branch and bound algorithm, to solve the problem. The first stage best solution is determined and broadcasted to other processors. In the second stage, branches of a chosen tree are investigated in parallel, to search for any better solution. PVM message passing environment, has been used. A master
and slave-processing scheme is used. The master distributes the work to the slaves, depending on the work in the slave queue. The computational study shows a near-linear speed-up in proving optimality.

2.3 Java and Java RMI

Stal M. and Plasil F. [17] have done a review of different distributed architectures such as RMI, CORBA, and COM/DCOM. Key issues and problems regarding these architectures are summarized. Their paper provides, systematic information, regarding the architecture and explains in detail factors such as remote interface, marshalling, binding, naming and finding a service.

2.3.1 Remote Method Invocation

Comprehensive information of Java and Remote Method Invocation is available at Sun Inc. website [18,19]. This site provides, comprehensive information about the RMI protocol. Information from these sites has been of extreme help, in this research. A brief review and summary of the architectural details and essentials to execute RMI are listed below.

Client Server Communication:

The process of client server communication can be summarized as follows. The Figure 2.1 gives a picture of the whole process. The server instatiates a Remote object and registers it with the rmiregistry running on its machine by calling the method Naming.rebind(). It associates a name with it by which, the client will
look for this object. When the client performs a lookup for the registered object, a stub of this registered object, is returned to the client. The stub acts as a proxy for the server on the client’s machine. A TCP/IP connection is setup, between

![Diagram of RMI System](image)

**Figure 2.1 Functioning of the distributed process**

the client and the server machine by the transport layer of RMI protocol. Figure 2.2 shows the use of TCP/IP connections between JVMs.

![Diagram of data flow between JVMs](image)

**Figure 2.2 View of the data flow between JVM’s**
The client invokes calls on the stub. The arguments to these methods are serialized and sent to the server machine. For this reason, the objects sent as arguments to the remote methods, should implement Serializable interface. These arguments are deserialized by the skeleton on the server machine. The skeleton understands, how to communicate with the stub, across the RMI link. The skeleton communicates with the stub, it reads the parameters for the method call from the link, makes the call to the remote object, accepts the return value and then writes the return value back to the stub.

**Remote Interface:**

This interface is the declaration of all the methods, along with their arguments, that the client can call on the remote object remotely. The purpose of this interface, is to let the client know, the methods it is allowed to invoke on the remote object.

**RMI registry:**

The rmiregistry service runs on the same computer as the server. The server registers a Remote object, whose methods can be invoked by the client, with the rmiregistry. When the client looks up for the remote object with the registry, the rmiregistry sends back object_stub that acts as a server proxy on the client machine. At runtime, the client first looks for the stub in the directories, specified by its classpath.
**Synchronization:**

Data has to be protected against corruption. Corruption is, overwriting data, without a notice to other members, accessing the data. This can be prevented either by using synchronized methods or synchronized blocks. A thread executing a synchronized method or block gets a unique lock over the object whose, synchronized method it is executing and so no other thread can execute any synchronized method at that time.

**Serialization:**

The data structures, across the client and the server, cannot be passed by reference. However, they can be passed by value, as they don’t share the same memory space. As a result, they are serialized, so that, an actual copy of the data can be passed. The data structures passed as arguments to the remote calls, implement `Serializable` interface.

The application of this protocol to distributed computing of the scheduling problem under consideration, is explained in the next section.
3 Methodology

This chapter provides the design of the scheduling algorithm and an explanation of Remote Method Invocation (RMI) as applied to solve the scheduling problem, using distributed computing. The problem solution finds the best possible schedule, such that, the overall penalty (early + tardy) to process a set of jobs on a single machine, is minimized. The set of input values includes the processing time of each job, the due date of each job and the penalties per unit time for each job. The algorithm and the different issues related to it are described below. The source codes for the algorithms are provided in Appendix 1 and 2.

3.1 Problem Structure

Given $n$ number jobs, the jobs can be sequenced in $n!$ ways. To consider the large number of computations, the problem is structured as an $n$-ary tree as shown in the Figure 3.1 This tree structure is solved using a branch and bound algorithm. Typically, the n-ary tree contains the jobs sequences and nodes are selected from the tree, for scheduling. The whole algorithm can be split into two parts, which are explained as follows.

3.1.1 Node Selection

The initial best (may or may not be optimal) solution is calculated by scheduling the jobs in the increasing order of due-dates (EDD – Earliest due
date). The associated penalty with this schedule is the initial best penalty. At
level 0 no jobs are selected. At the next level, a node with minimum penalty is
selected and the remaining jobs (from the set of unscheduled jobs) will be
scheduled one by one after it. Thus, these jobs give rise to separate nodes and

![Tree structure of a 3-job problem](image)

Figure. 3.1 Tree structure of a 3-job problem

a penalty will be associated with each of them. Thus, at any point, the node with
the minimum penalty will be selected. A set of precedence relationships is used
to find the best node to be scheduled, in case, there is a tie among the nodes.
These relationships are discussed in Section 3.1.1.1. If the penalty associated
with a node is greater than the current best solution, then that node along with
its child nodes will not be investigated and will be removed from the task list
(Nodes waiting to be investigated). This process continues, until all nodes with
an inferior bound are eliminated. If for a particular sequence, all the jobs are scheduled and the penalty associated with it is less than the current best penalty, then the penalty associated with the new schedule, will replace the current best penalty and the sequence will replace the current best schedule. Subsequently the nodes with higher penalties than the current best solution are also removed from the task list. The tree will be investigated in this manner until, all nodes in the tree are eliminated or evaluated. A job is selected arbitrarily to break any existing ties.

3.1.1.1 Precedence Relationships

Precedence relationships are used, to limit the search, by evaluating relationships among the adjacent jobs, based on some logical rules when there is no idle time. Consider the following

\[ c_b = \text{completion time of last job scheduled} \]
\[ d_i = \text{due date job } i \]
\[ d_j = \text{due date job } j \]
\[ p_i = \text{due date job } i \]
\[ p_j = \text{due date job } j \]

If \( d_i \leq c_b + p_i \), \( d_j \leq c_b + p_j \) then,

\[ i \text{ will always precede } j \text{ if } p_j \geq p_i \]
\[ j \text{ will always precede } i \text{ if } p_j < p_i \]
In case of different early and tardy penalties, the job sequence will be selected, which will yield lesser penalty. In addition to above constraints, the sequence of jobs in consideration will be checked for the penalty, due to their relative position.

\[ Pen_1 = \text{penalty when } i \rightarrow j \text{ and} \]

\[ Pen_2 = \text{penalty when } j \rightarrow i. \]

The sequence yielding lower penalty between \( pen_1 \) and \( pen_2 \) will be selected.

If amongst two jobs, one job is early and the other job is tardy, then the tardy job always goes before the early job.

These constraints are designed, such that, the total penalty is minimized. These constraints ensure, that jobs are scheduled as close as possible to their due-dates. Checking these conditions, pre-empts, unnecessary sequences from being evaluated and nodes containing the un-investigated sequences are eliminated from the task list.

### 3.1.2 Scheduling of Jobs

This part of the algorithm, schedules jobs sequentially (after a set of scheduled jobs). A node is characterized to have the following properties with it:

- \( \{A\} \) - Set of already scheduled jobs.
- \( \{B\} \) - Set of unscheduled jobs.
- \( S_i \) - Start time of each job in the schedule.
- \( F_i \) - Finish time of each job in the schedule.
• $D_i$ - Due date of each job.

• $P_i$ - Processing time of each job.

• $B_A$ - Block information: sub set of the set of currently scheduled jobs.

• $B_{Aj}$ - Information regarding each block: Total number of jobs in each block.

All this information is stored in a data structure associated with each node.

The following is the nomenclature used:

$S_s$ – Starting time of job to be scheduled  

$P_s$ – Processing time of job to be scheduled  

$D_s$ – Due-date of job to be scheduled  

$F_s$ – Finishing time of job to be scheduled  

$F_b$ – Finishing time of the last job in the latest block scheduled.

$i$ – total number of blocks

The steps to schedule the job in the best possible position are as follows. Jobs are scheduled recursively, using a dynamic programming algorithm, (based on the sequence provided by the branch and bound algorithm), which begins by scheduling the first job in the sequence and concludes when the entire set of jobs is scheduled. Jobs, which are scheduled with no idle time between them, are considered to be a block.

**Case I** - If the finishing time of the job to be scheduled is more than its due-date, then that job is tardy and it is added to the last block created.
\( P_s + F_b \geq D_s \rightarrow \) insert job at the end of block i.

a) Minimum tardiness of the block under consideration is determined. Minimum tardiness is the tardiness of the least tardy job in the block.

b) Block is moved forward by the minimum tardiness value. If, by doing so, the total penalty reduces, the starting and the finishing times of all the jobs in the block, are updated. If the penalty doesn’t reduce, then the position of the current position of the block (prior to moving) is the best position.

c) If by moving the block as in step b the block (Block 2) touches the previous block (Block 1) then both the blocks are united together and a new block (Block 3) is formed. This newly formed block undergoes steps starting from step a. This is shown below in Figures 3.2 and 3.3.

![Figure 3.2 Moving a block to find optimal position](image)

Figure 3.2 Moving a block to find optimal position
d) If at any point, no block can be moved, because the first job in the sequence under consideration, starts at time zero, as shown in Figure 3.4, then, the current position of all the blocks, is the best position.
Case II - If the finishing time of the job to be scheduled is less than its due-date, then create a new block and put this job in it. This is the case where the job to be scheduled is early.

\[ P_s + F_b \leq D_s \rightarrow \text{create new block. } i = i + 1 \]

With its starting time as

\[ S_s = D_s - P_s \]

The position of this new block (job) is already optimum and so need not be moved (refer to Figure 3.5).

![Figure 3.5 Early job is added to a new block](image)

After following either of Case I or Case II new node(s) are created. The process of selecting the node and scheduling the job, goes on until the best solution is reached by investigating the whole tree.
3.2 Classes and Functions Developed

The algorithm as explained above, is coded in JDK 1.2. The code consists of 6 main parts.

*Main/Client:*
This is the main program that finds the best node to be scheduled. The *Main* class is used for the algorithm without RMI, while, the *Client* class is used for the algorithm with RMI.

*Scheduler:*
This class schedules the job in the best position with respect to the parent node. For the RMI algorithm, this class extends *UnicastRemoteObject* and implements *SchedulingInterface*.

*Node:*
This data structure, symbolizes the scheduled node in the tree. This data structure is wrapped in another object of *WrapData* type and sent from the client to the server. It implements *Serializable* interface.
UnscheduledNode:
This data structure holds data for the unscheduled child nodes of a scheduled node. It implements Serializable interface. This data structure is wrapped in WrapData and sent from client to the server.

WrapData:
This class is used while solving the problem on multiple computers. This class wraps UnscheduledNode, Node and the job to be scheduled and sends to the server. It implements Serializable interface.

Server:
This class instantiates itself and registers as an object of the Scheduler class and then listens for any connection requests from the client.

3.3 Data flow
The Miscellaneous class reads the data (processing time, due-dates, early and tardy penalties) from a text file. Main/Client class initializes the process by determining an initial solution and creates the root nodes of the tree. The best node to be scheduled, along with the job to be scheduled, is passed to Scheduler class. Nodes are selected arbitrarily to break any existing ties. A scheduled node is passed back from Scheduler class to Main/Client class. After receiving the new data structure, if the penalty associated with it is less than the
current best solution, then, *Client* class creates the new child nodes (yet to be scheduled) and associates the parent node to them. The number of child nodes created, is equal to the total number of jobs, less the number of jobs scheduled in that branch of the tree. If the associated penalty is more, then that node is no longer investigated and is eliminated. A node is eliminated if the penalty associated with it, is more than the best penalty at the time of evaluation. This node is pruned from the tree. This process continues until the best solution is determined by investigating the whole tree. The Class interaction and process is shown in *Figure 3.6*.
3.4 Data Storage

As discussed earlier, data structures * UnscheduledNode * and * Node * are used to store information about the unscheduled and scheduled nodes respectively. The information about all the current live nodes needs to be stored. These nodes are stored in an array. As the tree is investigated, the number of (nodes in the tree, whose penalty is less than the current best penalty and are not yet investigated) nodes increases, increasing the memory consumed. To make the algorithm efficient, the memory has to be allocated in a judicious way. For this reason, the dead nodes are removed from the memory, when, the array storing these nodes is half full.

3.5 Design of the Distributed Algorithm

Java RMI is used for the distributed computing of this problem. The basic algorithm for (selecting the minimum node and scheduling algorithm) solving this problem on multiple computers remains the same as explained above. Some changes have to be made in the program, to suit, the RMI regulations. The structure of the distributed system is comprised of two parts, a client, which is also the host computer and multiple servers, which process the requests from the client.

3.5.1 Client

A client program makes remote calls. The client creates multiple threads, which communicate with different servers. The client keeps track of the * n-ary *
tree structure and selects appropriate nodes for scheduling. It assigns tasks to different servers. It updates the tree, depending on the solution, that it receives from the server. An object of the class `Client` is instantiated and the initial base nodes are created. Each thread communicates to a separate server. Each thread selects the best node and sends it to the respective server and gets back the newly scheduled node from the server. Threads wanting to read or write data in the tree, wait for the thread, accessing the tree to finish its work. This prevents data corruption. This is taken care of by synchronization. The client arrives at an optimal solution when the task list is empty.

### 3.5.2 Server

The server as a whole can be said to be the component of the distributed system doing the scheduling of jobs. The server is comprised of two parts. The class `Server` instantiates an object of the `Scheduler` class, registers it with the `rmiregistry` and listens for any request from the client. The `Scheduler` class extends `UnicastRemoteObject` and implements the `SchedulingInterface`. The `Scheduler` class, implements all the methods declared in the `SchedulingInterface`. The job of the server is to accept the data sent by the client, schedule the job in the optimum position and return the data structure to the client.

The `Scheduler` class in the case of single computer, when used for RMI has to be changed to `Scheduler1` or `Scheduler2` for every added computer. The reason for this is, when the client calls the method `Naming.lookup()` it gets back
a stub with name as $Scheduler\_stub$ if the stub belongs to $Scheduler$ class. There needs to be a separate stub for each server. Figure 3.7 is the graphical representation of distributed design.

Figure 3.7 Client Server relationship
3.6 Experimental design

The data gathered by experimentation at different levels of factors, is analyzed using Design of Experiments. It is used to identify the statistically significant factors and their mutual interactions with respect to solution times. The factors and their levels are discussed in detail in the coming pages. The data obtained is statistically analyzed, using a p-value of 0.05. In addition to the results of DOE, comparison of running the client and server on same machines, as well as different machines, is tested.

3.6.1 Factors

The effect of following factors on the solution times is analyzed. Each factor is tested with three levels.

*Number of jobs (3 levels):*

These levels are such, that, problems size ranges from small to large (5, 6 and 7 jobs) and are based on available resources.

*Due-date tightness (3 levels):*

The due date tightness levels used are 0.3, 0.5 and 0.7 which represent, extremely dispersed data sets, to extremely tight data sets.
Early and tardy penalties (3 levels):
The three levels are equal penalties, tardy being twice as important as early and random penalties ranging from 1 to 4.

Co-efficient of variation of due-dates (3 levels):
The co-efficient of variation levels used are 0.3, 0.5 and 0.7, which represent tightly to loosely dispersed due-dates.

Number of computers (3 levels):
After examining the pilot studies, these levels (1, 2 and 4 computers) are used. In case of the host computer i.e. the computer that runs the client, also has a server running on it. The computers used, are selected to be as homogeneous as possible.

3.6.2 Design of Experiments

The experiments are conducted over a set of PC’s with following configuration: CyberMax Pentium III processor at 500MHz, 128MB RAM, Windows NT version 4. The computer has a PCI Fast Etherenet Adapter, which means, it can hadle data above 10 Mbps, though, the network can handle data upto 10Mbps. The algorithms are coded using JDK 1.2. The data is analyzed using Minitab Version 12. The data is generated using an Excel Macro. The data is generated in sets, so that, each set of data has a specific due-date
tightness and coefficient of variation. The formulas [11] used for the generation of data are as follows.

\[
\text{due date tightness (ddt)} = 1 - \left( \frac{\sum_{i=1}^{n} d_i}{\sum_{i=1}^{n} p_i} \right)
\]  

(3.1)

\[
\text{co-efficient of variation (Cv)} = \frac{\left[ \sum_{i=1}^{n} (d_i - \bar{d})^2 / n \right]^{1/2}}{\bar{d}}
\]  

(3.2)

where

\( d_i = \text{due date of job } i \)

\( p_i = \text{processing time of job } i \)

\( n = \text{number of jobs to be scheduled} \)

Three replicates are taken at each level. Therefore, \( 3 \times 3^5 = 3 \times 243 = 729 \) experiments are conducted. The individual results are summarized in Appendix 3. In addition, a set of 243 experiments with 1 client and 1 server on separate machines and a set of 243 experiments without using RMI are conducted. A final additional set of experiments is conducted every 3 hours of the day and solution times are recorded. This experiment is conducted, to determine, if network load has a significant effect on solution time.
4 Results

4.1 Analysis of Variance

An analysis of variance on the gathered data is performed. Two graphs in Figure 4.1 and Figure 4.2 illustrate the normal probability plot and the plot of residuals versus the fitted values. These are plotted to check the assumptions of normality and homogeneity required by ANOVA. Interpretation of the two graphs is explained in brief.

Normal Probability Plot (NPP):
If the residuals follow a normal distribution, then, this plot follows a straight line. In visualizing the straight line, more emphasis is placed, on the central values of the plot, than, on the extremes [15].

Plot of residuals versus the fitted values:
If this plot shows a random distribution, then, the assumption of homogenous variance can be said to be satisfied [15].

For the data under consideration, the NPP (Figure 4.1) and the plot of residuals versus the fitted values (Figure 4.2) are shown. The NPP clearly shows that the graph doesn’t follow a straight line. Graphing of the residuals versus fitted values shows, a bell shape pattern, suggesting a non-homogeneous variance
and thus a transform is necessary to continue the analysis and determine if an
ANOVA is valid.

Figure 4.1 Normal probability plot of residuals before log transform

Figure 4.2 Residuals versus fitted values
A "ln" (log to the natural base) transform is performed on the data and then an ANOVA test is performed. The normal probability plot (Figure 4.3) closely follows a straight line and shows, that the normality assumption is not violated. The plot of residuals (Figure 4.4) is random and does not show any significant trend and thus suggests homogeneous variance.

Figure 4.3 Normal Probability Plot after log transforms

Figure 4.4 Residuals versus fitted values after log transform
The transformed data is used for the further analysis. The ANOVA in Table 4.1 shows the statistically significant factors, along with the statistically significant interactions. The main effects plot is shown in Figure 4.5.

Table 4.1 ANOVA after the log transform: 95% confidence level

<table>
<thead>
<tr>
<th>Source</th>
<th>Degrees of Freedom</th>
<th>Sums of Squared</th>
<th>Mean Square</th>
<th>F value</th>
<th>P value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Number of comps</td>
<td>2</td>
<td>3.2071</td>
<td>1.6036</td>
<td>35.60</td>
<td>0</td>
</tr>
<tr>
<td>Number of jobs</td>
<td>2</td>
<td>690.8365</td>
<td>345.4182</td>
<td>7668.89</td>
<td>0</td>
</tr>
<tr>
<td>Penalty</td>
<td>2</td>
<td>3.1791</td>
<td>1.5895</td>
<td>35.29</td>
<td>0</td>
</tr>
<tr>
<td>ddt</td>
<td>2</td>
<td>132.7191</td>
<td>66.3596</td>
<td>1473.30</td>
<td>0</td>
</tr>
<tr>
<td>Cv</td>
<td>2</td>
<td>13.2785</td>
<td>6.6392</td>
<td>147.40</td>
<td>0</td>
</tr>
<tr>
<td>Comp* job</td>
<td>4</td>
<td>0.0653</td>
<td>0.0163</td>
<td>0.36</td>
<td>0.835</td>
</tr>
<tr>
<td>Comp* Penalty</td>
<td>4</td>
<td>0.0665</td>
<td>0.0166</td>
<td>0.37</td>
<td>0.831</td>
</tr>
<tr>
<td>Comp* ddt</td>
<td>4</td>
<td>0.1493</td>
<td>0.0373</td>
<td>0.83</td>
<td>0.507</td>
</tr>
<tr>
<td>Comp* Cv</td>
<td>4</td>
<td>0.0556</td>
<td>0.0139</td>
<td>0.31</td>
<td>0.872</td>
</tr>
<tr>
<td>Job* Penalty</td>
<td>4</td>
<td>0.5992</td>
<td>0.1498</td>
<td>3.33</td>
<td>0.01</td>
</tr>
<tr>
<td>Job* ddt</td>
<td>4</td>
<td>27.657</td>
<td>6.9142</td>
<td>153.51</td>
<td>0</td>
</tr>
<tr>
<td>Job* Cv</td>
<td>4</td>
<td>2.744</td>
<td>0.6861</td>
<td>15.23</td>
<td>0</td>
</tr>
<tr>
<td>Penalty* ddt</td>
<td>4</td>
<td>6.1097</td>
<td>1.5274</td>
<td>33.91</td>
<td>0</td>
</tr>
<tr>
<td>Penalty* Cv</td>
<td>4</td>
<td>0.2124</td>
<td>0.0531</td>
<td>1.18</td>
<td>0.319</td>
</tr>
<tr>
<td>ddt* Cv</td>
<td>4</td>
<td>9.3926</td>
<td>2.3481</td>
<td>52.13</td>
<td>0</td>
</tr>
<tr>
<td>Comp* job* Penalty</td>
<td>8</td>
<td>0.0926</td>
<td>0.0116</td>
<td>0.26</td>
<td>0.979</td>
</tr>
<tr>
<td>Comp* job* ddt</td>
<td>8</td>
<td>0.1497</td>
<td>0.0187</td>
<td>0.42</td>
<td>0.912</td>
</tr>
<tr>
<td>Comp* job* Cv</td>
<td>8</td>
<td>0.2056</td>
<td>0.0257</td>
<td>0.57</td>
<td>0.802</td>
</tr>
<tr>
<td>Comp* Penalty* ddt</td>
<td>8</td>
<td>0.1064</td>
<td>0.0136</td>
<td>0.3</td>
<td>0.966</td>
</tr>
<tr>
<td>Comp* Penalty* Cv</td>
<td>8</td>
<td>0.0498</td>
<td>0.0062</td>
<td>0.14</td>
<td>0.997</td>
</tr>
<tr>
<td>Comp* ddt* Cv</td>
<td>8</td>
<td>0.0529</td>
<td>0.0066</td>
<td>0.15</td>
<td>0.997</td>
</tr>
<tr>
<td>Job* Penalty* ddt</td>
<td>8</td>
<td>1.5315</td>
<td>0.1914</td>
<td>4.25</td>
<td>0</td>
</tr>
<tr>
<td>Job* Penalty* Cv</td>
<td>8</td>
<td>0.2837</td>
<td>0.0355</td>
<td>0.79</td>
<td>0.614</td>
</tr>
<tr>
<td>Job* ddt* Cv</td>
<td>8</td>
<td>4.6998</td>
<td>0.5875</td>
<td>13.04</td>
<td>0</td>
</tr>
<tr>
<td>Penalty* ddt* Cv</td>
<td>8</td>
<td>1.5612</td>
<td>0.1952</td>
<td>4.33</td>
<td>0</td>
</tr>
<tr>
<td>Error</td>
<td>598</td>
<td>26.9348</td>
<td>0.045</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Total</td>
<td>728</td>
<td>925.9424</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
ANOVA results show that number of computers, number of jobs, early and tardy penalties, due-date tightness and coefficient of variation are statistically significant and affect the time taken to find a solution. Figure 4.6 shows the plot for the interaction between factors. The interactions \textit{job-penalty, job-ddt, job-Cv, Penalty-ddt} and \textit{ddt-Cv} are statistically significant. This can be inferred, from the non-parallelism between lines, for the respective factor interactions.
4.2 Network Load Effect

Apart from the DOE analysis, data is analyzed, using Excel tools to investigate, if any other factors affect the solution time, which are not apparent from the DOE analysis. Table 4.2 shows the times for 1 client 1 server combination, when executed on same computer and on 2 different computers. This comparison helps in determining the effect of data transfer over the network, on solution time.

Figure 4.6 Interaction plot
Table 4.2 Times for 1 Client 1 Server same and different computers

<table>
<thead>
<tr>
<th>levels</th>
<th>same computer</th>
<th>different computer</th>
</tr>
</thead>
<tbody>
<tr>
<td>.3ddt 0.3Cv</td>
<td>8.65</td>
<td>8.82</td>
</tr>
<tr>
<td>.3ddt 0.5Cv</td>
<td>6.95</td>
<td>11.00</td>
</tr>
<tr>
<td>.3ddt 0.7Cv</td>
<td>7.33</td>
<td>9.94</td>
</tr>
<tr>
<td>0.5ddt 0.3Cv</td>
<td>17.57</td>
<td>33.27</td>
</tr>
<tr>
<td>0.5ddt 0.5Cv</td>
<td>15.57</td>
<td>23.76</td>
</tr>
<tr>
<td>0.5ddt 0.7Cv</td>
<td>9.41</td>
<td>11.83</td>
</tr>
<tr>
<td>0.7ddt 0.3Cv</td>
<td>44.93</td>
<td>63.98</td>
</tr>
<tr>
<td>0.7ddt 0.5Cv</td>
<td>30.88</td>
<td>53.64</td>
</tr>
<tr>
<td>0.7ddt 0.7Cv</td>
<td>21.94</td>
<td>45.59</td>
</tr>
</tbody>
</table>

To find the effect of network traffic, 3 replicates of a set of data with 7 jobs, random penalties, 0.7 ddt and 0.3 Cv are executed every 3 hrs. Table 4.3 shows the solution time at different time intervals. These experiments were conducted on weekdays during the quarter, 2 weeks prior to the finals week. This ensured the network had typical daily traffic due to students and faculty working in the Stocker Center, thus attempting to eliminate biased results.

Table 4.3 Solution times at different time intervals

<table>
<thead>
<tr>
<th>Jobs ddt=0.7 cv=0.3</th>
<th>Time</th>
<th>1&lt;sup&gt;st&lt;/sup&gt;</th>
<th>2&lt;sup&gt;nd&lt;/sup&gt;</th>
<th>3&lt;sup&gt;rd&lt;/sup&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>11Am</td>
<td>79.955</td>
<td>43.528</td>
<td>58.304</td>
</tr>
<tr>
<td></td>
<td>2PM</td>
<td>78.073</td>
<td>41.68</td>
<td>57.503</td>
</tr>
<tr>
<td></td>
<td>5Pm</td>
<td>72.554</td>
<td>42.441</td>
<td>59.676</td>
</tr>
<tr>
<td></td>
<td>8PM</td>
<td>76.15</td>
<td>43.152</td>
<td>57.243</td>
</tr>
<tr>
<td></td>
<td>11PM</td>
<td>76.1</td>
<td>42.9</td>
<td>57.113</td>
</tr>
</tbody>
</table>
4.3 Speed-up

Though, the main effects plot shows the way, in which, number of computers affect the solution time, comparison of solution time is done for 1 client 1 server on same computer and 1 client 2 servers configuration. Table 4.4 presents this data. This comparison magnifies the results as seen in the main effects plot. A speedup diagram helps in determining the magnitude of this effect very easily. Table 4.5 shows the solution times for a 7 jobs problem with tardiness penalty being twice as important as earliness penalty. Figure 4.7 shows speedups for data in Table 4.5. Speed-up is calculated by using the following formula [12].

\[
\text{Speedup} = \frac{T_{\text{single}}}{T_{\text{multiple}}} \tag{4.1}
\]

where

\(T_{\text{single}}\) = Solution Time for sequential algorithm

\(T_{\text{multiple}}\) = Solution time for a n computer algorithm
Table 4.4 Times for 1 client 1 server same machine and 1 client 2 servers

<table>
<thead>
<tr>
<th>Levels</th>
<th>1 Client 1 Server Same Comp</th>
<th>1 Client 2 Servers</th>
</tr>
</thead>
<tbody>
<tr>
<td>.3ddt 0.3Cv</td>
<td>6.32</td>
<td>5.42</td>
</tr>
<tr>
<td>.3ddt 0.5Cv</td>
<td>7.11</td>
<td>6.20</td>
</tr>
<tr>
<td>.3ddt 0.7Cv</td>
<td>6.68</td>
<td>5.76</td>
</tr>
<tr>
<td>0.5ddt 0.3Cv</td>
<td>33.67</td>
<td>28.72</td>
</tr>
<tr>
<td>0.5ddt 0.5Cv</td>
<td>25.20</td>
<td>19.07</td>
</tr>
<tr>
<td>0.5ddt 0.7Cv</td>
<td>9.79</td>
<td>8.77</td>
</tr>
<tr>
<td>0.7ddt 0.3Cv</td>
<td>58.05</td>
<td>52.73</td>
</tr>
<tr>
<td>0.7ddt 0.5Cv</td>
<td>52.93</td>
<td>48.27</td>
</tr>
<tr>
<td>0.7ddt 0.7Cv</td>
<td>47.41</td>
<td>32.29</td>
</tr>
</tbody>
</table>

Table 4.5 Solution times and speed up chart

<table>
<thead>
<tr>
<th>Level</th>
<th>Solution Time</th>
<th>Speed-up</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1 Client 1 Server</td>
<td>1 Client 2 Servers</td>
</tr>
<tr>
<td>.3ddt 0.3Cv</td>
<td>6.32</td>
<td>5.42</td>
</tr>
<tr>
<td>0.5ddt 0.5Cv</td>
<td>25.20</td>
<td>19.07</td>
</tr>
<tr>
<td>0.7ddt 0.7Cv</td>
<td>47.41</td>
<td>32.29</td>
</tr>
</tbody>
</table>

Figure 4.7 Speed up diagram
4.4 RMI vs Sequential Algorithm

Results from RMI algorithm are compared to that, from the non-RMI algorithm. A comparison chart is shown in Table 4.6. This comparison will help in determining, advantages/disadvantages of RMI and changes that need to be made.

Table 4.6 Comparison of RMI with Single computer algorithm

<table>
<thead>
<tr>
<th>Levels</th>
<th>7 jobs with RMI on same machine</th>
<th>Without RMI</th>
</tr>
</thead>
<tbody>
<tr>
<td>.3 dtd 0.3 Cv</td>
<td>8.6</td>
<td>0.3</td>
</tr>
<tr>
<td>.3 dtd 0.5 Cv</td>
<td>6.9</td>
<td>0.3</td>
</tr>
<tr>
<td>.3 dtd 0.7 Cv</td>
<td>7.3</td>
<td>0.3</td>
</tr>
<tr>
<td>0.5 dtd 0.3 Cv</td>
<td>17.6</td>
<td>0.9</td>
</tr>
<tr>
<td>0.5 dtd 0.5 Cv</td>
<td>15.6</td>
<td>0.4</td>
</tr>
<tr>
<td>0.5 dtd 0.7 Cv</td>
<td>9.4</td>
<td>1.0</td>
</tr>
<tr>
<td>0.7 dtd 0.3 Cv</td>
<td>44.9</td>
<td>2.9</td>
</tr>
<tr>
<td>0.7 dtd 0.5 Cv</td>
<td>30.9</td>
<td>2.0</td>
</tr>
<tr>
<td>0.7 dtd 0.7 Cv</td>
<td>21.9</td>
<td>1.1</td>
</tr>
</tbody>
</table>

4.5 Other results

To magnify the results, as seen in the main effects plot, a separate comparison of solution time, is done for 5, 6 and 7 jobs. Table 4.7 presents these times. This comparison amplifies the effect of due-date tightness and coefficient of variation.
Table 4.7 Times for 1 client 1 server same machine

<table>
<thead>
<tr>
<th>Levels</th>
<th>7 jobs</th>
<th>6 jobs</th>
<th>5 jobs</th>
</tr>
</thead>
<tbody>
<tr>
<td>.3ddt 0.3Cv</td>
<td>8.65</td>
<td>3.67</td>
<td>1.23</td>
</tr>
<tr>
<td>.3ddt 0.5Cv</td>
<td>6.95</td>
<td>4.72</td>
<td>1.25</td>
</tr>
<tr>
<td>.3ddt 0.7Cv</td>
<td>7.33</td>
<td>3.52</td>
<td>1.38</td>
</tr>
<tr>
<td>0.5ddt 0.3Cv</td>
<td>17.57</td>
<td>4.56</td>
<td>1.45</td>
</tr>
<tr>
<td>0.5ddt 0.5Cv</td>
<td>15.57</td>
<td>4.14</td>
<td>1.34</td>
</tr>
<tr>
<td>0.5ddt 0.7Cv</td>
<td>9.41</td>
<td>4.04</td>
<td>1.09</td>
</tr>
<tr>
<td>0.7ddt 0.3Cv</td>
<td>44.93</td>
<td>7.01</td>
<td>1.87</td>
</tr>
<tr>
<td>0.7ddt 0.5Cv</td>
<td>30.88</td>
<td>6.38</td>
<td>1.93</td>
</tr>
<tr>
<td>0.7ddt 0.7Cv</td>
<td>21.94</td>
<td>4.34</td>
<td>1.68</td>
</tr>
</tbody>
</table>
5 Conclusions and Future Research

5.1 Analysis and Interpretation

The main effects plot in Figure 4.5 on page 42, shows the aggregated effect of the individual factors. This plot shows the trend, in which, each factor affects the solution time, irrespective of their levels. A prominent effect of number of jobs can be seen as explained in Chapter 1. It is obvious that, as the number of jobs increases, the time increases factorially. The main effect plot makes clear the accelerated effect of due-date tightness and the inverse effects co-efficient of variation. Figure 5.1 presents a graph for the data tabulated in Table 4.7 and shows these results in detail.

Figure 5.1 Times for 1 client 1 server same machine, random penalties
5.1.1 Distributed Algorithm

From the comparison of times of the configuration with 1 client 1 server on same parent computer, to that, on different computers, it can be seen that the configuration with 1 client 1 server on same computer yields lower solution times. Figure 5.2 gives the sample plot of times for a 7 jobs problem with random penalties tabulated in Table 4.2. The reduction in time is apparent for readings with due-date tightness 0.7 and a coefficient of variation 0.3, 0.5 and 0.7 respectively. These data sets are difficult to solve and require a larger number of computations. This implies, more data has to be sent over the network and transfer of data over the network, takes considerable amount of time. In the first configuration, (client and server on same computer) the data between the client and server is not transferred over the network, resulting in lesser solution times. (Recall Figure 2.2).

Figure 5.2 Times for 1 Client 1 Server same and different computers
This fact makes the second configuration slower, as compared to the first one. Looking at the main effects plot (Figure 4.5) it can be seen that, as the number of computers changes from one to two, (i.e. from 1 client 1 server to 1 client 2 server) there is a drop in time required for the solution by almost 15%. Recall Figure 2.2. When the client and the server are running on the same computer, data transfer is faster as explained earlier. For 1 client 2 servers configuration, data between the client and server has to be transferred over the network, as one server is running on a separate computer. In spite of this addition, there is a considerable reduction in solution time, which is reflected in the main effects plot in Figure 4.5 and in Figure 5.3 This shows that, addition of another computer to the process, has improved the solution times. For the 1 client and 4 servers

Figure 5.3 Times for 1 client 1 server same machine and 1 client 2 servers
configuration, there is not any significant reduction by increasing the number of computers, which can be seen from the Figure 4.5. The time required is less, than the time, required when the client and server are on the same machine, but there is only a small reduction, as compared to level with 1 client and 2 servers. In the case of 1 client and 4 servers, the client computer has an added overhead of communicating with multiple computers. This factor nullifies the effect of adding extra computers for computation. Additionally, each time the data is sent to the computers, it has to be serialized which slows down the process.

5.1.2 Statistically Significant Factors

The effect of other statistically significant factors, as indicated by the ANOVA in Table 4.1 is explained as follows

*Due-date tightness (ddt):*  
Tighter due-dates, increase the number of jobs, being scheduled back to back and hence, more iterations are required, to come to an optimal solution, refer Figure 5.1. For the seven jobs problem, as the due-date tightness changes from 0.3 to 0.5 to 0.7 (i.e. for every 3 readings) the times go higher. This shows that, higher due-date tightness increases solution times. The same trend can be also seen in the main effects plot in Figure 4.5.
**Co-efficient of variation (Cv):**

Smaller value of co-efficient of variation suggests, tighter due-dates and thus increases the number of calculations required. This is a result of the increasing number of combinations, needed to be checked and pruning in the branch and bound algorithm looses power. Refer Figure 5.1. For any due-date tightness level, as the co-efficient of variation changes from 0.3, 0.5 to 0.7 (i.e. the due-dates are farther apart) the solution times decrease. The same can be deduced from the main effects plot of Co-efficient of variation in Figure 4.5. As the co-efficient of variation increases, the solution times decrease, thus suggesting an inverse variation.

**Early and tardy penalties:**

As each job has different penalties, a sequence of jobs has to be found, such that, a job with higher penalties (early or tardy) finishes closer to its due-date. At the same time, other job penalties and their relative positions need to be considered. This increases the number of calculations and thus affects the solution times.

**Interactions between number of jobs and penalties, ddt and Cv:**

The effect of penalties, due-date tightness and co-efficient of variation is already explained. The combined effect of these factors can be seen in Figure 5.1 and the interaction plot in Figure 4.6. The effect of the individual factors (penalties,
due-date tightness and co-efficient of variation) is amplified, as the number of jobs increase, which makes the problem size large. This can be seen, as the lines are more skewed at 7 jobs level in the interaction plot and also in Figure 5.3. The unevenness in the curve is more prominent at 7 jobs level, as compared to 6 jobs or 5 jobs level.

*Interaction between ddt and Cv:*  
This is the most interesting of the interactions. The precedence relationships become weaker at certain combinations of these factors and more nodes in the tree need to be investigated. The increase in the number of combination to be checked, nullifies the effect of precedence constraints. Refer to the interaction plot in Figure 4.6. The effect of the interaction is less prominent at lower levels of due-date tightness, whereas, at higher levels of due-date tightness a slight change in the co-efficient of variation does reduce the solution times.

*Interaction between ddt and the early/tardy penalties:*  
From the interaction plot, it can be observed, that the skewness of the plot is prominent for data sets, which are difficult to solve. At a higher level of due-date tightness and the effect of penalties is more prominent.

### 5.1.3 Statistically Insignificant Factors

The interactions between number of computers and other factors are statistically insignificant. The interaction between number of jobs and number of
computers is expected to be significant. The ANOVA contradicts this assumption.

Bigger size problems are expected to improve the efficiency of the client, as bigger problems take more time, to get scheduled at the server. It is expected, that, as the number of jobs will increase, this interaction will be significant as bigger size problem will be able to magnify the effect.

5.2 Sequential Algorithm Vs Distributed Algorithm

The solution times for the single computer algorithm (without Remote Method Invocation) are smaller, than the solution times for the distributed algorithm, (with Remote Method Invocation) for the same data sets. Comparisons of these times can be seen in Table 4.6. Figure 5.4 shows the times for data sets with 7 jobs and random penalty executed on single computer without RMI and with RMI (1 client 1 server on same computer). In the case of the first configuration, the data structures are referenced within the same memory space and thus, access to these structures and data is faster. In the second case, a separate copy of data structures has to be passed from the client to the server, as both client and the server run separate JVM’s and have physically different memory space. For this reason, data has to be serialized and de-serialized, which is time consuming. Additionally, data has to be sent over the network, which is another factor, which increases the time.
5.3 Network Load Effect

A data set is run every 3hrs from 11am to 11pm. The purpose of this experiment is to detect the effect of network traffic/load on solution time. It is clear that, solution times are affected, as data travels over the network. There is no exact knowledge of the traffic over the network and it is assumed, that solution times are affected, relative to traffic on the network during these intervals. The intervals are chosen, such that, the number of students working varied within the Stocker Center. The number of students is more during 11am and 2pm interval, so it is assumed, that the network traffic would be more during this time. The traffic is assumed to go down from 5 pm to 11pm interval. Table 4.3 gives the solution times recorded at different intervals. Figure 5.5 shows graphical representation of these times.
Though there is no exact data about the network traffic, solution times do change in accordance with the network traffic assumption. It is clear from Figure 5.5, that solution time is highest at 11 am when maximum number of students are working and lowest at 5 pm as students and faculty take a break from the days work. At around 8pm, students do return to the labs, which increases the network traffic and it is reflected in the solution time. These results validate the assumption that, increase in the network traffic, increases the solution time.

5.4 Recommendations

Remote method invocation has been used in the following research, to evaluate the suitability towards a computationally hard problem. The RMI protocol is slow, as data has to pass through different levels of the protocol as
seen in Figure 2.1. In case of algorithm using RMI, the solution time reduces as the number of computers increases from 1 to 2, but for any further increase, there is not any significant reduction in time, as the host computer reaches its maximum efficiency with increasing number of servers. However, it would be useful to design a distributed algorithm, which would solve problem, while minimizing communication.

The comparison of single computer algorithm with the RMI algorithm in Section 5.2 illustrates, that the single computer algorithm is efficient. Though for extremely large problems (problems with more than 15 jobs) the RMI algorithm may be able to reach an optimal solution. Problems of this magnitude slow down the processing power of the computer, as considerably high amount of data needs to be stored and at times optimal solution cannot be reached, as the computer runs out of memory.

As a part of the future work alterations, data can be sent in groups of more than 1 node. The tree can also be broken in such a way, that the client works on upper levels of the tree, while the servers share the branches below, equally. For example, each server will investigate a particular branch, along with all the sub-branches, propagating from this main branch. This approach might make the client wait for the servers, in other words "starve the client" though it can help distributing the network communication overhead.

Additionally, dynamic load balancing approach can be applied. The client will distribute work among the available servers, depending on the load on each
server. Either the client has to keep track of the load on each server or the server has to notify the client, whenever the tasks in the server queue are depleted. This can result in the added communication overhead, but will ensure maximum utilization of the servers.

For improving the speed of execution, instead of using the RMI protocol, direct socket connections between the client and the servers can be tried. The client machine has to be used efficiently. An answer to this can be writing the client program in a language like C or C++, which is faster than Java. The servers will be written in Java, so as to achieve platform independency.

The work shows that, improvements can be made in the distributed computing approach. There is a huge potential for improvements as faster networks, networks that can handle huge amount of data and faster versions of Java are developed. In this work JDK 1.2 has been used. With the coming version of JDK 1.3 which is claimed to be up to 5 times faster than JDK 1.2, and the future versions still faster, the speed of execution will be less of a problem and will enhance, the results of distributed computing. Additionally, other protocols, such as, RMI-IIOP can be used to increase the speed of execution.

In the current research a set of homogeneous computers (computers with same operating system) has been used. Java is platform independent and hence, in the future research, computers with different operating systems can be used. By doing so, faster computers with varied operating systems can be
used. Care has to be taken that, the faster computers do more work and slower computers do not form the bottleneck.

A web application to solve this problem can be developed. An applet/servlet can be used to get the problem parameters from the user. The data can then be processed at the backend on faster computers, using networks with higher bandwidth. This will allow the user, to take advantage of much needed processing power (possibly world wide) at an affordable cost.

The approach can also be used to solve other computationally intensive problems, like the traveling salesman and resource allocation problems. These problems have the same structure, as the manufacturing scheduling problem and thus this work can be extended, to solve these problems. The approach can also be applied Artificial Intelligence problems, where the best solution has to be determined, depending on a set of constraints.
References


Appendix 1: Code for algorithm without RMI

```java
import java.io.*;
import java.lang.*;
import java.awt.*;
import java.util.*;

class Thesis for the single computer algorithm without RMI
{                           
Public Variables*/
    int CONSTANT=7;/* tells the number of jobs*/
    int NODESIZE=15000;/*initial size of the node array*/
    int PENALTYISGREATER=-1;
    int JOBSAREFINISHED=1;
    int PROCEEDAHEAD=0;
    int numberofnodes=CONSTANT;
    int duedate[] = new int[CONSTANT+1];
    int process[] = new int[CONSTANT+1];
    int early[] = new int[CONSTANT+1];
    int tardy[] = new int[CONSTANT+1];
    int totalnumberofnodeobjects=0;
    int totalnumberofnodes=0;
    Node updatenode[] = new Node[CONSTANT];
    Auxilliary aux=new Auxillary();
    BestSolution bestsolutionobject=new BestSolution();
    Miscellaneous aux1=new Miscellaneous();
    NodeClass nodeobject[] = new NodeClass[NODESIZE];
    Node node[] = new Node[NODESIZE];
    NodeClass updatenodeobject[] = new NodeClass[1];
    NodeClass object;
    Node newnode;
    double time;
    double timel;
    double time2;

public class Thesis {
    /*Public Variables*/
    int CONSTANT=7;/* tells the number of jobs*/
    int NODESIZE=15000;/*initial size of the node array*/
    int PENALTYISGREATER=-1;
    int JOBSAREFINISHED=1;
    int PROCEEDAHEAD=0;
    int numberofnodes=CONSTANT;
    int duedate[] = new int[CONSTANT+1];
    int process[] = new int[CONSTANT+1];
    int early[] = new int[CONSTANT+1];
    int tardy[] = new int[CONSTANT+1];
    int totalnumberofnodeobjects=0;
    int totalnumberofnodes=0;
    Node updatenode[] = new Node[CONSTANT];
    Auxilliary aux=new Auxillary();
    BestSolution bestsolutionobject=new BestSolution();
    Miscellaneous aux1=new Miscellaneous();
    NodeClass nodeobject[] = new NodeClass[NODESIZE];
    Node node[] = new Node[NODESIZE];
    NodeClass updatenodeobject[] = new NodeClass[1];
    NodeClass object;
    Node newnode;
    double time;
    double timel;
    double time2;
    /*******************************************************************************************/
    Default Constructor
    /*******************************************************************************************/
    public Thesis()
    {
    }
    /*******************************************************************************************/
    The main function instantiates an object of class Thesis
    /*******************************************************************************************/
    public static void main(String args[])
    {
        Thesis frame=new Thesis();
        frame.start();
    }
    /*******************************************************************************************/
    The start method first calls all the essential methods in the routine
    Creates nodes with initial values.
    /*******************************************************************************************/
    public void start()
    {
        timel=(double)System.currentTimeMillis();
        aux1.readFile("C:\gopal\single\t4.txt");
        populate();
        for(int x=1;x<=CONSTANT;x++)
        {   
            nodeobject[x]=new NodeClass();
        }
    }
```
totalnumberofnodes = totalnumberofnodes+1;
node[x]=new Node();
totalnumberofnodeobjects = totalnumberofnodeobjects+1;
node[x].number=1;
node[x].nodeobjectnumber=x;
node[x].penalty=0;
node[x].jobsequence[1]=x;
nodeobject[x].job[1]=x;
constructNodeObject(nodeobject[x]);
}
schedule();
}

**********************************************************************************
Populates the arrays for the processing time, due dates, early and tardy penalties
**********************************************************************************
public void populate()
{
    int i=1;
    for (int x=1;x<=CONSTANT;x++)
    {
        process[x]=auxl.input[i];
i=i+1;
        duedate[x]=auxl.input[i];
i=i+1;
        early[x]=auxl.input[i];
i=i+1;
        tardy[x]=auxl.input[i];
i=i+1;
    }
}

**********************************************************************************
Puts the initial values in the Node array which represents the task list
**********************************************************************************
public void constructNodeObject( NodeClass nC)
{
    int i=1;
    for (int x=1;x<=CONSTANT;x++)
    {
        nC.jprocess[x]=auxl.input[i];
i=i+1;
        nC.jdue[x]=auxl.input[i];
i=i+1;
        nC.jearly[x]=auxl.input[i];
i=i+1;
        nC.jtardy[x]=auxl.input[i];
i=i+1;
        nC.objectpenalty=0;
        nC.totalnumberofjobsscheduled=0;
    }
}

**********************************************************************************
Finds the minimum node and the related objectnumber
**********************************************************************************
public int findMinimumNode(Node n[])
{
    int x=1;
    int pen;
    int returnnodenumber;
    /*finds the first non null node in the array*/
    while(true)
    {
        if(node[x]!=null)
        {
            returnnodenumber=x;
        }
    }
}
pen = node[x].penalty;
break;
}
x = x + 1;
}
while (true)
{
if (n[x] != null)
{
if (n[x].penalty < pen)
{
return nodedenumber = x;
pen = n[x].penalty;
}
}
x = x + 1;
if (x > total number of nodes) // number of nodes
{
break;
}
}
return return nodedenumber;

/******************************************************************************************
Creates a separate copy of the argument passed
******************************************************************************************/
public NodeClass copy(NodeClass x)
{
NodeClass return NodeClass = new NodeClass(x);
return return NodeClass;

/******************************************************************************************
Creates a separate copy of the argument passed
******************************************************************************************/
public Node copy(Node m)
{
Node return Node = new Node(m);
return return Node;

/******************************************************************************************
Returns the initial arrangement according to the EDD
******************************************************************************************/
public void set_initial_solution()
{
/* checks that a particular jobs duedates are greater how many other jobs duedates
and then sets the job in that position */
int edd[] = new int[CONSTANT + 1];
int seq[] = new int[CONSTANT + 1];
int bstart[] = new int[CONSTANT + 1];
int bfinish[] = new int[CONSTANT + 1];
int pen_of_initial = 0;
int equal jobs = 0;
for (int i = 1; i <= CONSTANT; i++)
{
int count = 0;
for (int j = 1; j <= CONSTANT; j++)
{
if (duedate[i] > duedate[j])
{
count = count + 1;
}
if (duedate[i] < duedate[j])
{
count = count;
if (duedate[i] == duedate[j] && i <= j)
{
    count = count + 1;
}
edd[count] = duedate[i];
seq[count] = i;
}
bfinish[0] = 0;
/*set the start times according to the duedates*/
for (int i = 1; i < CONSTANT; i++)
{
    bstart[seq[i]] = bfinish[seq[i-1]];
    bfinish[seq[i]] = bstart[seq[i]] + process[seq[i]];
}
/*calculate the penalty*/
for (int i = 1; i < CONSTANT; i++)
{
    int temp;
    temp = bfinish[seq[i]] - duedate[seq[i]];
    if (temp <= 0)
    {
        penofinitial = penofinitial - (temp * early[seq[i]]);
    }
    if (temp > 0)
    {
        penofinitial = penofinitial + (temp * tardy[seq[i]]);
    }
}
/*set all the values in the best solution class*/
System.arraycopy(bstart, 0, bestsolutionobject.beststart, 0, CONSTANT + 1);
System.arraycopy(bfinish, 0, bestsolutionobject.bestfinish, 0, CONSTANT + 1);
System.arraycopy(seq, 0, bestsolutionobject.bestjobsequence, 0, CONSTANT + 1);
bstart = null;
bfinish = null;
edd = null;
seq = null;
}

/******************************************************************************/
Updates the Node array, jobs next to be scheduled and creates a new Node
******************************************************************************/
public int createNewNodes(Node update_node, NodeClass update_nodeobject, int jobscheduled)
{
    if (update_nodeobject.objectpenalty > bestsolutionobject.bestpenalty)
    {
        return PENALTYISGREATER;
    }
    if (update_node.numberofjobs >= CONSTANT)
    {
        return JOBSAREFINISHED;
    }
    /*takes care if the number of jobs in the node is more than the actual number of jobs*/
    else
    {
        int j;
        updatenodeobject[0] = update_nodeobject;
        /*updating the sequence in the object*/
        updatenodeobject[0].job[updatenodeobject.totalnumberofjobsscheduled + 1] = jobscheduled;
        updatenodeobject[0].totalnumberofjobsscheduled =
update_nodeobject.totalnumberofjobsscheduled+1;

int temp=CONSTANT-update_node.numberofjobs;
for(int x=1;x<=temp;x++)
|
updatenodeobject[0].childnodes[x]=totalnumberofnodes+x;
|
updatenodeobject[0].totalnumberofchildnodes=temp;
System.arraycopy(updatenodeobject, 0, nodeobject,
totalnumberofnodeobjects+1, 1);

totalnumberofnodeobjects = totalnumberofnodeobjects+1;
for(j=1;j<=temp;j++)
/*this block is for assigning the values to the new updating node*/
|
| updatenode[j]=new Node();
| updatenode[j]=copy(update_node);
| updatenode[j].numberofjobs=(update_node.numberofjobs)+1;
| updatenode[j].nodeobjectnumber=totalnumberofnodeobjects;
| updatenode[j].penalty=update_nodeobject.objectpenalty;
|
j=1;
for (int k=1;k<=CONSTANT;k++)
|
| if(isKthereInJobSequence(updatenode[j].jobsequence,k) ==false)
|
| updatenode[j].jobsequence[update_node.numberofjobs+1]=k;
| j=j+1;
| if(j==temp+1)
| break;
| else{
| continue;
| }
System.arraycopy(updatenode,1,node, totalnumberofnodes+1,
temp);//updatenode.length-1);

totalnumberofnodes=totalnumberofnodes+temp;
update_nodeobject=null;
updatenodeobject[0]=null;
return PROCEEDAHEAD;
|
|
******************************************************************************
Checks if a particular number is there in the array passed
******************************************************************************
private boolean isKthereInJobSequence(int sequence[],int K)
|
| boolean returnval=false;
| for (int i=1;i<=CONSTANT;i++)
| |
| if(sequence[i]==K)
| |
| returnval=true;
| break;
| }
| return returnval;
|
******************************************************************************
Returns a value which specifies which job is supposed to be scheduled in the object
******************************************************************************
public int findJobToSchedule(Node fnode)
```java
private void setToBestSolution(NodeClass nBest, int minnode) {
    System.arraycopy(object.jfinish, 0, bestsolutionobject.bestfinish, 0, object.jfinish.length);
    System.arraycopy(object.jstart, 0, bestsolutionobject.beststart, 0, object.jstart.length);
    System.arraycopy(object.job, 0, bestsolutionobject.bestjobsequence, 0, object.job.length);
    bestsolutionobject.bestpenalty = object.objectpenalty;
    bestsolutionobject.nnum = minnode;
}
```

```java
public void schedule() {
    int returnofcreatenewnodes;
    int min_node = 0;
    int jobtoschedule = 0;
    setInitialSolution(); /*set the initial solution based on EDD*/
    while (true) {
        if (allNodesAreNull(node) == true) {
            break;
        }
        min_node = findMinimumNode(node); /*returns the nodeobjectnumber for the minimum penalty node*/
        int tp = min_node;
        newnode = copy(node[min_node]);
        int number = newnode.nodeobjectnumber;
        object = copy(nodeobject[number]); /*making a copy of the nodeobject*/
        min_node = findMinimumNodeWithPreConStratints(min_node, newnode, node, object);
        newnode = copy(node[min_node]);
        node[min_node] = null; /*removing the node and making it null*/
        /*finding which job is to be scheduled next in this object*/
        jobtoschedule = findJobToSchedule(newnode);
        aux.scheduleJob(object, jobtoschedule);
        returnofcreatenewnodes = createNewNodes(newnode, object, jobtoschedule);
        if (returnofcreatenewnodes == PENALTYISGREATER) {
            //should not add any new node in nodearray and nullify that node*/
        } else if (returnofcreatenewnodes == JOBSAREFINISHED) {
            //check with the optimum node at present
            if (object.objectpenalty < bestsolutionobject.bestpenalty) {
                object.job[object.totalnumberofjobsscheduled + 1] = jobtoschedule;
                setToBestSolution(object, min_node);
            }
        }
    }
```
if(totalNumberofNodeObjects==10000)
{
    cleanNodeObjects();
}
}
bestSolutionObject.printBestSolution();
time2=(double)System.currentTimeMillis();
System.out.println("time taken for the optimal solution "+(time2-time1)/1000+" seconds");

/***************************************************************************/
Function to find if all the nodes in the tree are null
***************************************************************************/
public boolean allNodesAreNull(Node nN[])
{
    boolean returnisnull=true;
    int j=1;
    while(j<=totalNumberOfNodes)
    {
        if(nN[j]!=null)
        {
            returnisnull=false;
            break;
        }
        j=j+1;
    }
    return returnisnull;
}

/***************************************************************************/
Cleans the node objects which are not pointed to by any node
***************************************************************************/
public void cleanNodeObjects()
{
    for(int x=CONSTANT+1;x<=totalNumberOfNodeObjects;x++)
    {
        if(nodeObject[x]==null)
        {
            break;
        }
        boolean flag=false;
        for(int y=1;y<=nodeObject[x].totalNumberOfChildNodes;y++)
        {
            if(node[nodeObject[x].childNodes[y]]!=null)
            {
                flag=true;
                break;
            }
        }
        if(flag==false)
        {
            nodeObject[x]=null;
        }
    }
}

/***************************************************************************/
Finds the minimum node based on the precedence constraints
***************************************************************************/
public int findMinimumNodeWithPreConStratints(int mymin_node,Node mynewnode,Node mynode[],NodeClass myobject)
{
    int counter;
}
boolean myflag;
int numberofchilds;
int lastcompletiontime;
int jobtobescheduled;
numberofchilds=myobject.totalnumberofchildnodes;
int child[] = new int[numberofchilds+1];
int proccesetimes[] = new int[numberofchilds+1];
int duedateofchilds[] = new int[numberofchilds+1];
int tardyofchilds[] = new int[numberofchilds+1];
lastcompletiontime=myobject.jfinish[myobject.block[myobject.m][myobject.n]];
counter=0;
for (int i=1; i<=numberofchilds; i++)
{
    /*creating a sub copy of the jobs in all sister(same parent object) nodes*/
    if (mynode[myobject.childnodes[i]] !=null)
    {
        counter=counter+1;
        child[counter]=myobject.childnodes[i];
        proccesetimes[counter]=this.process[mynode[child[counter]]].jobsequence[mynode[child[counter]].numberofjobs];
        duedateofchilds[counter]=this.duedate[mynode[child[counter]]].jobsequence[mynode[child[counter]].numberofjobs];
        tardyofchilds[counter]=this.tardy[mynode[child[counter]]].jobsequence[mynode[child[counter]].numberofjobs];
    }
    for (int i=1; i<=counter; i++)
    {
        int j=0;
        myflag=true;
        jobtobescheduled=mynode[child[i]].jobsequence[mynode[child[i]].numberofjobs];
        /*tardy*/
        if (lastcompletiontime+this.process[jobtobescheduled]>this.duedate[jobtobescheduled])
        {
            for (j=1; j<=counter; j++)
            {
                int pen1=(lastcompletiontime+this.process[jobtobescheduled]-
                this.duedate[jobtobescheduled])*tardy[jobtobescheduled];
                int pen2=(lastcompletiontime+this.process[jobtobescheduled]+processetimes[j]-
                duedateofchilds[j])*tardyofchilds[j];
                int pen3=(lastcompletiontime+processetimes[j]-
                duedateofchilds[j])*tardyofchilds[j];
                int pen4=(lastcompletiontime+this.process[jobtobescheduled]+processetimes[j]-
                duedateofchilds[j])*tardy[jobtobescheduled];

                if (pen1+pen2+pen3+pen4 & pen2>0 & pen3>0)
                //process[jobtobescheduled]<processetimes[j])
                {
                    myflag=false;
                }
            }
        }
    }
    if (myflag==true & j==counter)
mymin_node = child[i];
break;
}
return mymin_node;
}

 bisheriges Minimum wird gefunden, und derjenige Knoten wird von dem Array 'child' ausgewählt, der durch 'min_node' bezeichnet wird. Die Methode wird hierbei nach unten rekursiv aufgerufen und endet, indem der zurückgegebene Knoten als 'mymin_node' ausgegeben wird.

*************
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
Class has the data for the Best solution
*****************************************************************************

class BestSolution
{
    int CONSTANT = 7;
    int beststart[] = new int[CONSTANT + 1];
    int bestfinish[] = new int[CONSTANT + 1];
    int bestjobsequence[] = new int[CONSTANT + 1];
    int bestpenalty;
    int nnum;

    public BestSolution()
    {
    }

    public void printBestSolution()
    {
        System.out.println("Best Job sequence is ");
        for (int x = 1; x <= CONSTANT; x++)
        {
            System.out.println(String.valueOf(bestjobsequence[x]) + "+
                    String.valueOf(beststart[bestjobsequence[x]]));
            System.out.println("Best Job sequence penalty is ");
            System.out.println(bestpenalty);
        }
    }
}

import java.lang.*;
import java.util.*;
import java.io.*;

*****************************************************************************
* * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * * *
Class does the actual scheduling of a job in the optimum position
*****************************************************************************

public class Auxiliary
{
    private int CONSTANT = 7;

    // Schedule the job in a given place
    public void scheduleJob(NodeClass objectnode, int job_to_schedule)
    {
        // has to put the job in a proper block
        if (objectnode.jdue[job_to_schedule] - objectnode.jprocess[job_to_schedule] <=
             objectnode.jfinish[objectnode.job[objectnode.totalnumberofjobsscheduled]])
        {
            if (objectnode.totalnumberofjobsscheduled == 0)
            {
                objectnode.jstart[job_to_schedule] =
                objectnode.jdue[job_to_schedule] - objectnode.jprocess[job_to_schedule];
            }
            if (objectnode.jstart[job_to_schedule] < 0)
            {
                objectnode.jstart[job_to_schedule] = 0;
            }
        }
    }
else
{
    objectnode.jstart[job_to_schedule]=
        objectnode.jfinish[objectnode.job[objectnode.totalnumberofjobsscheduled]];
    objectnode.jfinish[job_to_schedule]=
        objectnode.jstart[job_to_schedule] +
        objectnode.jprocess[job_to_schedule];
}

/* adds the number of jobs in the block*/
objectnode.n=objectnode.n+1;
objectnode.block[objectnode.m][objectnode.n]=job_to_schedule;
objectnode.numjobs[objectnode.m]=objectnode.n;

/* have to create another block and insert the new job in to it in another case if the job is early*/
if(objectnode.jdue[job_to_schedule]-objectnode.jprocess[job_to_schedule]>
    objectnode.jfinish[objectnode.job[objectnode.totalnumberofjobsscheduled]])
{
    if(objectnode.m!=1 | objectnode.n >=1)
    {
        objectnode.m=objectnode.m+1;
    }
    objectnode.block[objectnode.m][l]=job_to_schedule;
    objectnode.numjobs[objectnode.m]=1;
    objectnode.n=1;
    objectnode.jstart[job_to_schedule]=objectnode.jdue[job_to_schedule]-
        objectnode.jprocess[job_to_schedule];
    objectnode.jfinish[job_to_schedule]=objectnode.jdue[job_to_schedule];
}

while(true)
{
    int status= moveBlock(objectnode.m, objectnode.n, objectnode.numjobs, 
        objectnode.block, objectnode.jstart, objectnode.jfinish, 
        objectnode.jdue, objectnode.jearly, 
        objectnode.jtardy, objectnode.blockpenalty);

    if(status==1)/"case where min tardy is ZERO"/
    {
        break;
    }

    /*case is true only for the first block in the sequence*/
    if(status==2)
    {
        break;
    }

    /*difference in the two(start and finish) jobs in different jobs is ZERO*/
    if (status==3)
    {
        /*unites the two blocks*/
        uniteBlock(objectnode.m, objectnode.numjobs, 
            objectnode.block, objectnode.blockpenalty);

        int temp=objectnode.numjobs[objectnode.m];
        objectnode.numjobs[objectnode.m]=0;
        temp=temp+objectnode.numjobs[objectnode.m-1];
        objectnode.m = objectnode.m-1;
        objectnode.numjobs[objectnode.m]=temp;
objectnode.n=objectnode.numjobs[objectnode.m];

/*penalty becomes greater and the current position is the best*/
if(status==4)
{
  break;
}

/*calculate the total penalty*/
int temppenalty=0;
for(int i=1;i<=objectnode.m;i++)
{
  temppenalty=temppenalty+objectnode.blockpenalty[i];
}
objectnode.objectpenalty=temppenalty;

Moves the block to the optimum position
************************************************************************************/
public int moveBlock(int bl,int jb,int numjobblk[],int j_block[][],int j_start[],
int j_finish[],int j_due[],int early[],int tardy[],int block_penalty[])
{
  int stat=0;
  int prevpenalty;
  int mintardy;
  int newpenalty;
  int diff;

  /*keep track of jobs in the block*/
  int jobblk[]=new int[jb+1];
  /* keeps the new start time and finishing time after shifting the block*/
  int newstart[]=new int[CONSTANT+1];
  int newfinish[]=new int[CONSTANT+1];
  /* assigns jobs from the block to a variable*/
  for(int i=1;i<=jb;i++)
  {
    jobblk[i]=j_block[bl][i];
  }
  while(true)
  {
    prevpenalty=this.calculatePenalty(jobblk, j_finish, j_due, early, tardy);
    /* this finds the min tardy value*/
    mintardy=0;
    for(int i=1;i<=jb;i++)
    {
      if((j_finish[jobblk[i]]-j_due[jobblk[i]]) >=0)
      {
        /*checks if the job is tardy*/
        if(mintardy==0 & (j_finish[jobblk[i]]-j_due[jobblk[i]])>0)
        {
          mintardy=j_finish[jobblk[i]]-j_due[jobblk[i]];
        }
        if((j_finish[jobblk[i]]-j_due[jobblk[i]])< mintardy & mintardy > 0)
        {
          mintardy=j_finish[jobblk[i]]-j_due[jobblk[i]];
        }
      }
    }
    /*case where min tardy is ZERO*/
    if(mintardy==0)
    {
      block_penalty[bl]=prevpenalty;
      stat=1;
      break;
    }
    /*calculate difference in the finishing time of the last job
in the previous block and start time of the first job in the present block*/
int lastblk=numjobblk[bl-1];
if((bl-1)==0)
{
    diff=j_start[jobblk[1]]-0;
}
else
    diff=j_start[jobblk[1]]-j_finish[j_block[bl-1][lastblk]];
/*case where the block cannot be move any further as it already starts at zero*/
/*true only in the case of first block*/
if(diff<=0 & (j_start[jobblk[1]]==0) & bl==1)
{
    block_penalty[bl]=prevpenalty;
    stat=2;
    break;
}
/*case were the block touches the previous block*/
if (diff<=0)
{
    block_penalty[bl]=prevpenalty;
    stat=3;
    break;
}
if (diff<=mintardy)
{
    mintardy=diff;
}
/* determine the new starting and finishing times according to the min tardy*/
for (int k=1;k<=jb;k++)
{
    newstart[jobblk[k]]=j_start[jobblk[k]]-mintardy;
    newfinish[jobblk[k]]=j_finish[jobblk[k]]-mintardy;
}
/*calculates the penalty with respect to the shifted times*/
newpenalty= this.calculatePenalty(jobblk,newfinish,j_due,early,tardy);
/*block should not be moved and it is at the optimum position*/
if(newpenalty >= prevpenalty)
{
    block_penalty[bl]=prevpenalty;
    /*int retu=conv.giveMessage("penalty is greater", 1);*/
    stat=4;
    break;
}
/*update the new times of the block as they are move to the new optimum position*/
if (newpenalty<prevpenalty)
{
    for(int i=1;i<=jb;i++)
    {
        j_start[jobblk[i]]=newstart[jobblk[i]];
        j_finish[jobblk[i]]=newfinish[jobblk[i]];
        block_penalty[bl]=newpenalty;
    }
}
return stat;
}
************************************************************************************
Calculate penalty for a given set of jobs
*******************************************************************************/
public int calculatePenalty(int j[],int f[],int d[],int e[],int t[])

int time;
int penalty=0;
for(int i=1;i<j.length;i++)
{
    if((time=(f[j[i]]-d[j[i]])>=0)
    penalty=penalty+time*t[j[i]];
    else
    penalty=penalty-time*e[j[i]];
}
return penalty;

/************************************************************************************
Unites two block together and creates a single new block
************************************************************************************/
public void uniteBlock(int num,int jobsinBlock[],int blocknum[][],int blockPenalty[])
{
    int a=jobsinBlock[num];
    int b=jobsinBlock[num-1];
    int newblock[]=new int[a+b+1];
    for(int i=1;i<=b ;i++)
    {
        newblock[i]=blocknum[num-1][i];
    }
    int j=b+1;
    for(int i=1;i<=a ;i++)
    {
        newblock[j]=blocknum[num][i];
        blocknum[num][i]=0;
        j=j+1;
    }
    for(int jobi=1;jobi<=(a + b);jobi++)
    {
        blocknum[num-1][jobi]=newblock[jobi];
    }
    blockPenalty[num-1]=blockPenalty[num-1]+blockPenalty[num];
    blockPenalty[num]=0;
}
import java.lang.*;
import java.io.*;

/************************************************************************************
Class represents the data structure for a scheduled node
************************************************************************************/
public class NodeClass extends Object
{
    int SIZE=7;
    int totalnumberofjobsscheduled;
    int job[]=new int[SIZE+1];
    int jstart[]=new int[SIZE+1];
    int jdue[]=new int[SIZE+1];
    int jprocess[]=new int[SIZE+1];
    int jfinish[]=new int[SIZE+1];
    int jearly[]=new int[SIZE+1];
    int jtardy[]=new int[SIZE+1];
    int childnodes[]=new int[SIZE+1];/*to keep track of the child nodes*/
    int totalnumberofchildnodes;
    int objectpenalty;
    int m=1;
    int n=0;
    int block[][]=new int[SIZE+1][SIZE+1];
    int numjobs[]=new int[SIZE+1];
    int blockpenalty[]=new int[SIZE+1];

    /************************************************************************************
Default constructor
************************************************************************************/
public NodeClass()
{
}

/************************************************************************************
Copy constructor
************************************************************************************/
public NodeClass(NodeClass x)
{
    //copy constructor
    SIZE=x.SIZE;/*fixed*/
    System.arraycopy(x.job,0,job,0,SIZE+1);/* updated bycreateNewNodes in Thesis*/
    System.arraycopy(x.jstart,0,jstart,0,SIZE+1);/*changed by auxillary*/
    System.arraycopy(x.jdue,0,jdue,0,SIZE+1);/*fixed*/
    System.arraycopy(x.jprocess,0,jprocess,0,SIZE+1);/*fixed*/
    System.arraycopy(x.jfinish,0,jfinish,0,SIZE+1);/*auxillary*/
    System.arraycopy(x.jearly,0,jearly,0,SIZE+1);/*fixed*/
    System.arraycopy(x.j tardy,0,j tardy,0,SIZE+1);/*fixed*/
    System.arraycopy(x.numjobs,0,numjobs,0,SIZE+1);/*auxillary*/
    System.arraycopy(x.blockpenalty,0,blockpenalty,0,SIZE+1);/*auxillary*/
    objectpenalty=x.objectpenalty; /**
    totalnumberofjobsscheduled=x.totalnumberofjobsscheduled;
    m=x.m;/*auxillary*/
    n=x.n;/*auxillary*/
    for(int y=1;y<=SIZE;y++)
    {                
        block[y]=(int[])x.block[y].clone(); /*auxillary*/
    }
    totalnumberofchildnodes=x.totalnumberofchildnodes;
    System.arraycopy(x.childnodes,0,childnodes,0,SIZE+1);
}
}
import java.lang.*;
import java.io.*;

public class Node
{
    int constant=7;//has to be changed with number of jobs
    int numberofjobs;
    int nodeobjectnumber;
    int penalty;

    Default constructor
    public Node()
    {
    }

    Copy constructor
    public Node(Node x)
    {
        numberofjobs=x.numberofjobs;
        nodeobjectnumber=x.nodeobjectnumber;
        penalty=x.penalty;
        System.arraycopy(x.jobsequence, 0, jobsequence, 0, constant+l);
    }
}
import java.lang.*;
import java.util.*;
import java.io.*;

public class Miscellaneous
{
    public int input[]=new int[100];
    private int i;

    public void readFile(String arg)
    {
        try
        {
            FileReader reader = new FileReader(arg);
            BufferedReader buf=new BufferedReader(reader);
            while (true)
            {
                line=buf.readLine();
                if (line==null)
                {
                    break;
                }
                input[]=Integer.parseInt(line);  
                j=j+1;
            }
            catch (IOException ioErr){
        }
    }
}
Appendix 2: Code for RMI Algorithm

```java
import java.io.IOException;
import java.io.Serializable;
import java.rmi.RemoteException;
import java.rmi.RMISecurityManager;
import java.rmi.*;
import java.io.*;
import java.lang.*;
import java.awt. *
import java.util.*;
import java.lang.InterruptedException;

//***********************
//Class for distributed computing using RMI
//***********************
public class ClientThesis implements Runnable
{
    int CONSTANT=5; /* tells the number of jobs*/
    int NODESIZE=15000; /*initial size of the node array*/
    int NUMBEROFCOMPUTERS=1;
    int PENALTYSGREATER=-1;
    int JOBSAREFINISHED=1;
    int PROCEEDAHEAD=0;
    int numberofnodes=CONSTANT;
    int duedate[]=new int[CONSTANT+1];
    int process[]=new int[CONSTANT+1];
    int early[]=new int[CONSTANT+1];
    int tardy[]=new int[CONSTANT+1];
    int totalnumberofnodeobjects=0;
    int totalnumberofnodes=0;
    Node updatenode[]=new Node[CONSTANT];
    //Auxiliary aux=new Auxiliary(); /*have to make change here*/
    BestSolution bestsolutionobject=new BestSolution();
    Miscellaneous aux1=new Miscellaneous();
    NodeClass nodeobject[]=new NodeClass[NODESIZE];
    Node node[]=new Node[NODESIZE];
    NodeClass updatenodeobject[]=new NodeClass[1];
    NodeClass object;
    Node newnode;
    double time;
    double time1;
    double time2;
    Object obj[]=new Object[NUMBEROFCOMPUTERS+1];
    /*stores the stubs downloaded dynamically from the servers*/
    public int servicenumber=1;
    public Thread thread[]=new Thread[NUMBEROFCOMPUTERS+1];
    WrapData buffer[]=new WrapData[NUMBEROFCOMPUTERS+1];
    public ClientThesis()throws IOException
    {
        obj[1]=new Object();
        try
        {
            //finding the registry
            obj[1]=Naming.lookup("rmi://pirogue/Scheduler1");
            System.out.println("I got " + obj[1].toString());
        }catch (Exception e){
            System.out.println("Cannot find");
            e.printStackTrace();
            System.exit(1);
        }
        try
        {
        }
    }
    public void run()
    {
        try
        {
            //Code for RMI algorithm
        }
    }
}
```
System.out.println("i got " + obj[2].toString());
} catch (Exception e) {
    System.out.println("Cannot find " + e.getMessage());
    e.printStackTrace();
    System.exit(1);
} try {
    System.out.println("i got " + obj[3].toString());
} catch (Exception e) {
    System.out.println("Cannot find " + e.getMessage());
    //e.printStackTrace();
    System.exit(1);
} try {
    System.out.println("i got " + obj[4].toString());
} catch (Exception e) {
    System.out.println("Cannot find " + e.getMessage());
    //e.printStackTrace();
    System.exit(1);
}
this.start();

/***********************************************************************************
Main: Instantiates an object of Client Thesis
***********************************************************************************/
public static void main(String args[]) {
    ClientThesis thesis = null;
    System.out.println("I have started..... here in main");
    try {
        thesis = new ClientThesis();
    } catch (IOException e) {
        System.out.println("The Problem in client is .." + e.getMessage());
    }
    while (true) {
        try {
            Thread.sleep(1000000000);
        } catch (InterruptedException e) {
        }
    }
}

/***********************************************************************************
The start method first calls all the essential methods in the routine to create the
initial set of data.
***********************************************************************************/
public void start() {
    timel = (double) System.currentTimeMillis();
    auxl.readFile("c:\gopal\client\t4.txt");
    System.out.println("before populate");
    populate();
    for (int x = 1; x <= CONSTANT; x++) {
        nodeobject[x] = new NodeClass();
        totalnumberofnodeobjects = totalnumberofnodeobjects + 1;
        node[x] = new Node();
        totalnumberofnodes = totalnumberofnodes + 1;
        node[x].numberofjobs = 1;
        node[x].nodeobjectnumber = x;
node[x].penalty=0;
node[x].jobsequence[l]=x;
nodeobject[x].job[l]=x;
constructNodeObject(nodeobject[x]);
}
setInitialSolution();
男主set the initial solution based on EDD .******Change has been made here*/
createThreads(); //  have to start the threads here
}

Populates the arrays for the processing time, due dates, early and tardy penalties
*******************************************************************************/
public void populate()
{
  int i=l;
  for (int x=l;x<=CONSTANT;x++)
  {
    process[x]=auxl.input[i];
i=i+l;
dueDate[x]=auxl.input[i];
i=i+l;
early[x]=auxl.input[i];
i=i+l;
tardy[x]=auxl.input[i];
i=i+l;
  }
}

Puts the initial values in the Node array which represents the task list
*******************************************************************************/
public void constructNodeObject(NodeClass nC)
{
  int i=l;
  for (int x=l;x<=CONSTANT;x++)
  {
    nC.jprocess[x]=auxl.input[i];
i=i+l;
nC.jdue[x]=auxl.input[i];
i=i+l;
nC.jearly[x]=auxl.input[i];
i=i+l;
nC.jtardy[x]=auxl.input[i];
i=i+l;
nC.objectpenalty=0;
nC.totalnumberofjobsscheduled=0;
  }
}

This creates the different threads which will run during the process
*******************************************************************************/
public void createThreads()
{
  for(int x=l;x<=NUMBEROFCOMPUTERS;x++)
  {
    thread[x]=new Thread(this,String.valueOf(x));
    System.out.println("Starting thread "+thread[x].toString());
    thread[x].start();
  }
}

Finds the minimum node and the realted objectnumber
*******************************************************************************/
public int findMinimumNode(Node n[])
{
```java
int x=1;
int pen;
int returnnodenumber;
//finds the first non null node in the array
while(true)
{
  if(node[x]!=null)
  {
    returnnodenumber=x;
    pen= node[x].penalty;
    break;
  }
  x=x+1;
}
while(true)
{
  if(n[x]!=null)
  {
    if(n[x].penalty<pen)
    {
      returnnodenumber=x;
      pen=n[x].penalty;
    }
  }
  x=x+1;
  if(x>totalnumberofnodes)//numberofnodes
  {
    break;
  }
}
return returnnodenumber;

/******************************************************************************
Creates a separate copy of the argument passed
***********************************************************************************/
public NodeClass copy(NodeClass x)
{
  NodeClass returnNodeClass=new NodeClass(x);
  return returnNodeClass;
}

/******************************************************************************
Creates a separate copy of the argument passed
***********************************************************************************/
public Node copy(Node m)
{
  Node returnNode=new Node(m);
  return returnNode;
}

/******************************************************************************
Returns the initial arrangement according to the EDD
***********************************************************************************/
public void setlnitialSolution()
{
  /*checks that a particular jobs duedates are greater how many other jobs
   duedates and then sets the job in that position */
  int edd[] = new int[CONSTANT+1];
  int seq[] = new int[CONSTANT+1];
  int bstart[] = new int[CONSTANT+1];
  int bfinish[] = new int[CONSTANT+1];
  int penofinitial = 0;
  int equaljobs = 0;
  for(int i=1;i<=CONSTANT;i++)
  {
    //
  }
```
int count=0;
for(int j=1;j<=CONSTANT;j++)
{
  if (duedate[i]>duedate[j])
  {
    count=count+1;
  }
  if (duedate[i]<duedate[j])
  {
    count=count;
  }
  if (duedate[i]==duedate[j] & i<=j)
  {
    count=count+1;
  }
}
edd[count]=duedate[i];
seq[count]= i;
}
bfinish[0]=0;
/*set the start times according to the duedates*/
for(int i=1;i<=CONSTANT;i++)
{
    bstart[seq[i]]=bfinish[seq[i-1]];
    bfinish[seq[i]]=bstart[seq[i]]+process[seq[i]];
}
/*calculate the penalty*/
for (int i=1;i<=CONSTANT;i++)
{
    int temp;
    temp=bfinish[seq[i]]-duedate[seq[i]];
    if (temp<0)
    {
        penofinitial=penofinitial-(temp*early[seq[i]]);
    }
    if (temp>=0)
    {
        penofinitial=penofinitial+(temp*tardy[seq[i]]);
    }
}
/*set all the values in the best solution class*/
System.arraycopy(bstart,0,bestsolutionobject.beststart,0,CONSTANT-1);
System.arraycopy(bfinish,0,bestsolutionobject.bestfinish,0,CONSTANT+1);
System.arraycopy(seq,0,bestsolutionobject.bestjobsequence,0,CONSTANT+1);
bestsolutionobject.bestpenalty=penofinitial;
bstart=null;
bfinish=null;
edd=null;
seq=null;
}

************************************************************************************
Updates the Node array,jobs next to be scheduled and creates a new Node
************************************************************************************
public int createNewNodes(Node update_node, NodeClass update_nodeobject,
int jobscheduled)
{
    if(update_nodeobject.objectpenalty >bestsolutionobject.bestpenalty)
    {
        return PENALTYISGREATER;
    }
    if (update_node.numberofjobs>=CONSTANT)
return JOBSAREFINISHED;

/*takes care if the number of jobs in the node is more than the actual number of jobs*/
else
{
    int j;
    updatenodeobject[0]=update_nodeobject;
    /*updating the sequence in the object*/
    updatenodeobject[0].job[update_nodeobject.totalnumberofjobsscheduled+1]=
    jobscheduled;
    updatenodeobject[0].totalnumberofjobsscheduled=
    update_nodeobject.totalnumberofjobsscheduled+1;
    int temp=CONSTANT-update_node.numberofjobs;
    for(int x=1;x<=temp;x++)
    {
        updatenodeobject[0].childnodes[x]=totalnumberofnodes+x;
    }
    updatenodeobject[0].totalnumberofchildnodes=temp;
    System.arraycopy(updatenodeobject, 0, nodeobject, totalnumberofnodeobjects+1, 1);
    totalnumberofnodeobjects = totalnumberofnodeobjects+1;
    for(j=1;j<=temp;j++)
    /*this block is for assigning the values to the new updating node*/
    {
        updatenode[j]=new Node();
        updatenode[j]=copy(update_node);
        updatenode[j].numberofjobs=(update_node.numberofjobs)+1;
        updatenode[j].nodeobjectnumber=totalnumberofnodeobjects;
        updatenode[j].penalty=update_nodeobject.objectpenalty;
    }
    j=1;
    for (int k=1;k<=CONSTANT;k++)
    {
        if(isKthereInJobSequence(updatenode[j].jobsequence,k) ==false)
        {
            updatenode[j].jobsequence[update_node.numberofjobs+1]=k;
            j=j+1;
            if(j==temp+1)
                break;
        }
        else
        {
            continue;
        }
    }
    /*have to update the childnodes*/
    /*other variables will be automatically updated*/
    System.arraycopy(updatenode,1,node, totalnumberofnodes+1, temp);
    totalnumberofnodes=totalnumberofnodes+temp;
    update_nodeobject=null;
    updatenodeobject[0]=null;
    return PROCEEDAHEAD;
}

**************************************************************************
Checks if a particular number is there in the array passed
**************************************************************************
private boolean isKthereInJobSequence(int sequence[],int K)
{
    boolean returnval=false;
    for (int i=1;i<=CONSTANT;i++)
    {
        if(sequence[i]==K)
```java
returnval=true;
break;
}
}
return returnval;

/************************************************************************************
Returns a value which specifies which job is supposed to be scheduled in the object
**************************************************************************************/
public int findJobToSchedule(Node fnode)
{
  int numofjobs;
  int returnjobtoschedule;
  numofjobs=fnode.numberofjobs;
  returnjobtoschedule=fnode.jobsequence[numofjobs];
  return returnjobtoschedule;
}

/************************************************************************************
Sets the values in the current object as the values in the best solution
**************************************************************************************/
private void setToBestSolution(NodeClass nBest, int minnode)
{
  System.arraycopy(object.jfinish,0,bestsolutionobject.bestfinish,
                  0,object.jfinish.length);
  System.arraycopy(object.jstart,0,bestsolutionobject.beststart,
                  0,object.jstart.length);
  System.arraycopy(object.job,0,bestsolutionobject.bestjobsequence,
                  0,object.job.length);
  bestsolutionobject.bestpenalty=object.objectpenalty;
  bestsolutionobject.nnum=minnode;
}

/*******************************************************************************
Implementation of the Runable interface's run method
*******************************************************************************/
public void run()
{
  //originally schedule()
  int returnofcreatenewnodes;
  int min_node=0;
  int jobtoSchedule=0;
  WrapData wrapdata=null;
  int threadnameint;
  String threadname=new String();
  /*set the initial solution based on EDD*/
  while(true)
  {
    if(allNodesAreNull(node)==true)
    {
      break;
    }
    synchronized(this)
    {
      min_node=findMinimumNode(node);
      /*returns the nodeobjectnumber for the minimum penalty node*/
      int tp=min_node;
      newnode=copy(node[min_node]);
      int number=newnode.nodeobjectnumber;
      object=copy(nodeobject[number]);
      /*making a copy of the nodeobject*/
```
min_node=
    findMinimumNodeWithPreConStratints(min_node, newnode, node, object);

newnode=copy(node[min_node]);
node[min_node]=null; /*removing the node and making it null*/

jobtoschedule=findJobTOSchedule(newnode);
/*creates an object that is to be sent over the network*/
wrapdata=new WrapData(object, newnode, jobtoschedule, min_node);
/*selects the proper thread at run time*/
threadname=Thread.currentThread().getName();
threadnameint=Integer.parseInt(threadname);
try
    {
        /*call on the remote method*/
        ((SchedulingInterface)obj[threadnameint]).scheduleJob(wrapdata);
        wrapdata=null;
    }
catch (RemoteException e)
    {
        System.out.println("cannot call method scheduleJob" + 
            e.getMessage());
    }
}
try
    {
        Thread.sleep(l);
    }
catch (InterruptedException e)
    {
        System.out.println("here the Thread is broken");
    }
/*have to get the value of the object variable*/
try
    {
        buffer=Integer.parseInt(Thread.currentThread().getName())= 
            ((SchedulingInterface)obj,Integer.parseInt(Thread.currentThread().
                getName())).getObject();
    }
catch (RemoteException e)
        System.out.println("Cannot get the return object" + e.getMessage());
}
synchronized(this)
    {
        wrapdata=buffer=Integer.parseInt(Thread.currentThread().getName());
        returnofcreatenewnodes=createNewNodes(wrapdata.t_newnode,
            wrapdata.t_object, wrapdata.t_jobtoschedule);
        if (returnofcreatenewnodes==PENALTYISGREATER)
            {
                wrapdata=null;
                /*should not add any new node in nodearray and nullify that node*/
            }
        if (returnofcreatenewnodes==JOBSAREFINISHED)
            {
                //should check with the optimum node at present
                if (wrapdata.t_object.objectpenalty<bestsolutionobject.bestpenalty)
                    {
                        object=wrapdata.t_object;
                        object.job[object.totalnumberofjobscheduled+1]=jobtoschedule;
                        setToBestSolution(wrapdata.t_object, wrapdata.t_min_node);
                    }
                wrapdata=null;
            }
        if (totalnumberofnodeobjects==10000 | totalnumberofnodeobjects==20000 | 
            totalnumberofnodeobjects==50000 | totalnumberofnodeobjects==70000 )
    
        synchronized(this)
```java
{  
cleanNodeObjects();  
}
}
synchronized(this)
{
  bestsolutionobject.printBestSolution();
time2=(double)System.currentTimeMillis();
System.out.println("time taken for the optimal solution "+
(time2-timel)/1000+" seconds");
  System.exit(1);
}

//************************************************************************************
// Checks if any nodes are yet to be scheduled
//**************************************************************************************/
public boolean allNodesAreNull(Node nN[])
{
  boolean returnisnull=true;
  int j=1;
  while(j<=totalnumberofnodes)
  {
    if(nN[j]!=null)
    {  
      returnisnull=false;
      break;
    }
    j=j+1;
  }
  return returnisnull
}

//************************************************************************************
// Cleans the Nodeobjects which are not pointed to by any node
//**************************************************************************************/
public void cleanNodeObjects()
{
  for(int x=CONSTANT+1;x<=totalnumberofnodeobjects;x++)
  {
    if(nodeobject[x]==null)
    {
      break;
    }
    boolean flag=false;
    for(int y=1;y<=nodeobject[x].totalnumberofchildnodes;y++)
    {
      if(node[nodeobject[x].childnodes[y]]!=null)
      {  
        flag=true;
        break;
      }
    }
    if(flag==false)
    {
      nodeobject[x]=null;
    }
  }
}
```
Finds the minimum node based on the precedence constraints
************************************************************************************/
public int findMinimumNodeWithPreConStratints(int mymin_node, Node mynewnode,
Node mynode[]), NodeClass myobject)
{
    int counter;
    boolean myflag;
    int numberofchilds;
    int lastcompletiontime;
    int jobtobescheduled;
    numberofchilds=myobject.totalnumberofchildnodes;
    int child[]=new int[numberofchilds+1];
    int processtimes[]=new int[numberofchilds+1];
    int duedateofchilds[]=new int[numberofchilds+1];
    int tardyofchilds[]=new int[numberofchilds+1];
    lastcompletiontime=myobject.jfinish[myobject.block[myobject.m][myobject.n]);
    counter=0;
    for (int i=1;i<=numberofchilds;i++)
    {
        if(mynode[myobject.childnodes[i]]!=null)
        {
            counter=counter+1;
            child[counter]=myobject.childnodes[i];
            processtimes[counter]=this.process[mynode[child[counter]].
                jobsequence[mynode[child[counter]].numberofjobs]];
            duedateofchilds[counter]=this.duedate[mynode[child[counter]].
                jobsequence[mynode[child[counter]].numberofjobs]];
            tardyofchilds[counter]=this.tardy[mynode[child[counter]].
                jobsequence[mynode[child[counter]].numberofjobs]];
        }
    }
    for(int i=1;i<=counter;i++)
    {
        int j=0;
        jobtobescheduled=mynode[child[i]].jobsequence[mynode[child[i]].numberofjobs];
        myflag=true;
        if(lastcompletiontime+this.process[jobtobescheduled]>
            this.duedate[jobtobescheduled])
        {
            for(j=1;j<=counter;j++)
            {
                int pen1=(lastcompletiontime+this.process[jobtobescheduled] -
                    this.duedate[jobtobescheduled])*tardy[jobtobescheduled];
                int pen2=(lastcompletiontime+
                    this.process[jobtobescheduled]+processtimes[j] -
                    duedateofchilds[j])*tardyofchilds[j];
                int pen3=(lastcompletiontime+processtimes[j] -
                    duedateofchilds[j])*tardyofchilds[j];
                int pen4=(lastcompletiontime+this.process[jobtobescheduled] +
                    processtimes[j] -
                    duedate[jobtobescheduled])*tardy[jobtobescheduled];
                if(pen1+pen2 > pen3+pen4 & pen2>0 &
                    pen3>0)//process[jobtobescheduled]<processtimes[j])
                {
                    myflag=false;
                }
            }
        }
    }
    if(myflag==true & j==counter)
mymin_node=child[i];
break;
}
}
return mymin_node;
}

/**************************************************************************************
Class for the best solution
***************************************************************************************/
class BestSolution
{
    int CONSTANT=5;
    int beststart[]=new int[CONSTANT+1];
    int bestfinish[]=new int[CONSTANT+1];
    int bestjobsequence[]=new int[CONSTANT+1];
    int bestpenalty;
    int nnum;
    public BestSolution()
    {
    }
    public void printBestSolution()
    {
        System.out.println("best Job sequence is ");
        for(int x=1;x<=CONSTANT;x++)
        {
            System.out.println(String.valueOf(bestjobsequence[x])+" "+String.valueOf(beststart[bestjobsequence[x]]));
        }
        System.out.println("best Job sequence penalty is ");
        System.out.println(bestpenalty);
        //System.exit(1);
    }
}
import java.lang.*;
import java.io.*;

/******************************************************************************
Class represents the data structure for a scheduled node
******************************************************************************/
public class NodeClass extends Object implements Serializable{
    int SIZE=7;
    int totalnumberofjobsscheduled;
    int job[]=new int[SIZE+1];
    int jstart[]=new int[SIZE+1];
    int jdue[]=new int[SIZE+1];
    int jprocess[]=new int[SIZE+1];
    int jfinish[]=new int[SIZE+1];
    int jearly[]=new int[SIZE+1];
    int jtardy[]=new int[SIZE+1];
    int chilndnodes[]=new int[SIZE+1];/*to keep track of the child nodes*/
    int totalnumberofchildnodes;
    int objectpenalty;
    int m=1;
    int n=0;
    int block[]=new int[SIZE+1][SIZE+1];
    int numjobs[]=new int[SIZE+1];
    int blockpenalty[]=new int[SIZE+1];

    //**************************************************************************
    Default constructor
    **************************************************************************/
    public NodeClass()
    {
    }

    //**************************************************************************
    Copy constructor
    **************************************************************************/
    public NodeClass(NodeClass x)
    {
        SIZE=x.SIZE;/*fixed*/
        System.arraycopy(x.job,O,job,O,SIZE+1);/* updated bycreateNewNodes in Thesis*/
        System.arraycopy(x.jstart,O,jstart,O,SIZE+1);/*changed by auxillary*/
        System.arraycopy(x.jdue,O,jdue,O,SIZE+1);/*fixed*/
        System.arraycopy(x.jprocess,O,jprocess,O,SIZE+1);/*fixed*/
        System.arraycopy(x.jfinish,O,jfinish,O,SIZE+1);/*auxillary*/
        System.arraycopy(x.jearly,O,jearly,O,SIZE+1);/*fixed*/
        System.arraycopy(x.jtardy,O,jtardy,O,SIZE+1);/*fixed*/
        System.arraycopy(x.numjobs,O,numjobs,O,SIZE+1);/*auxillary*/
        System.arraycopy(x.childrennodes,O,childrennodes,O,SIZE+1);/*auxillary*/
        objectpenalty=x.objectpenalty;/**/
        totalnumberofjobsscheduled=x.totalnumberofjobsscheduled;
        m=x.m;"auxillary"/
        n=x.n;"auxillary"/
        for(int y=1;y<=SIZE;y++)
            {block[y]=(int[])x.block[y].clone(); /*auxillary*/
            }
        totalnumberofchildnodes=x.totalnumberofchildnodes;
        System.arraycopy(x.childrennodes,O,childrennodes,O,SIZE+1);
    }
}
import java.lang.*;
import java.io.*;

/******************************************************************************
This class represents the unscheduled nodes
******************************************************************************
public class Node implements Serializable
{
    int constant=7;//has to be changed with number of jobs
    int numberOfJobs;
    int jobSequence[] = new int[constant+1];
    int nodeObjectNumber;
    int penalty;

    /******************************************************************************
    Default constructor
    ******************************************************************************/
    public Node()
    {
    }

    /******************************************************************************
    Copy constructor
    ******************************************************************************/
    public Node(Node x)
    {
        numberOfJobs = x.numberOfJobs;
        nodeObjectNumber = x.nodeObjectNumber;
        penalty = x.penalty;
        System.arraycopy(x.jobSequence, 0, jobSequence, 0, constant+1);
    }
}
import java.lang.*;
import java.io.Serializable;
import java.util.*;

/***********************************************************************************
Class for sending the data from the client to the server
***********************************************************************************/
public class WrapData implements Serializable {
    public NodeClass t_object;
    public Node t_newnode;
    public int t_jobtoschedule;
    public int t_min_node;

    public WrapData(NodeClass t_object, Node t_newnode, int t_jobtoschedule, int t_min_node) {
        this.t_object = t_object;
        this.t_newnode = t_newnode;
        this.t_jobtoschedule = t_jobtoschedule;
        this.t_min_node = t_min_node;
    }
}
}
import java.rmi.*;
import java.io.Serializable;
public interface SchedulingInterface extends Remote
{
    public void scheduleJob(WrapData wd) throws RemoteException;
    public WrapData getObject() throws RemoteException;
}
import java.io.IOException;
import java.io.Serializable;
import java.rmi.RemoteException;
import java.rmi.RMISecurityManager;
import java.rmi.*;
import java.io.Serializable;
import java.util.*;

public class ServerThesis implements Runnable
{
    /***********************************************************************************
    Constructor-binds the object to the rmi registry
    ***********************************************************************************/
    public ServerThesis() throws IOException
    {
        Auxillary1 bindobject=null;
        String url="pirogue";
        try{
            bindobject=new Auxillary1();
            }catch (RemoteException e){
                System.out.println("cannot create bind object.."+e.getMessage());
            }
        try{
            Naming.rebind("rmi://"+url+"/Scheduler1",bindobject);
            System.out.println("bound sucessfully");
        }catch (Exception e){
            System.out.println("error in binding..."+e.getMessage());
        }
    }

    /***********************************************************************************
    Implementation of the run method which keeps the server running
    ***********************************************************************************/
    public void run()
    {
        while(true)
        {
            try
            {
                Thread.sleep(1000000);
            }catch (InterruptedException ex){
            }
        }
    }

    /***********************************************************************************
    The main method -instantiates the server
    ***********************************************************************************/
    public static void main(String args[])
    {
        try
        {
            ServerThesis service=new ServerThesis();
            new Thread(service).start();
        }catch (IOException e)
        {
            System.out.println("Couldn't create Service"+e.getMessage());
        }
    }
}
import java.lang.*;
import java.util.*;
import java.io.*;
import java.io.Serializable;
import java.rmi.*;
import java.rmi.Remote;
import java.rmi.server.*;

public class Auxillaryl extends UnicastRemoteObject implements Schedulinglnterface,Serializable{
    private int CONSTANT=5;
    private NodeClass objectToReturn=null;
    private Node r_node=null;
    private int r_jobscheduled;
    private int r_min_node;

    /***********************************************************************
    The constructor for Auxillaryl class
    ***********************************************************************/
    public Auxillaryl() throws RemoteException
    {
        super();
    }

    /***********************************************************************
    Schedule the job in a given place
    ***********************************************************************/
    public void scheduleJob(WrapData wd)throws RemoteException
    {
        this.r_node=wd.t_newnode;
        this.r_jobscheduled=wd.t_jobtoschedule;
        this.r_min_node=wd.t_min_node;
        NodeClass objectnode=wd.t_object;
        int jobtoschedule=wd.t_jobtoschedule;
        //has to put the job in a proper block
        if (objectnode.jdue[objectnode.job[objectnode.totalnumberofjobsscheduled]]-
            objectnode.jprocess[objectnode.job[objectnode.totalnumberofjobsscheduled]])
        {
            if(objectnode.totalnumberofjobsscheduled==0)
            {
                objectnode.jstart[objectnode.job[objectnode.totalnumberofjobsscheduled]]=
                    objectnode.jdue[objectnode.job[objectnode.totalnumberofjobsscheduled]]-
                    objectnode.jprocess[objectnode.job[objectnode.totalnumberofjobsscheduled]];
                if(objectnode.jstart[objectnode.job[objectnode.totalnumberofjobsscheduled]]<0)
                {
                    objectnode.jstart[objectnode.job[objectnode.totalnumberofjobsscheduled]]=0;
                }
                objectnode.jfinish[objectnode.job[objectnode.totalnumberofjobsscheduled]]=
                    objectnode.jstart[objectnode.job[objectnode.totalnumberofjobsscheduled]]+
                    objectnode.jprocess[objectnode.job[objectnode.totalnumberofjobsscheduled]];
            }
            else
            {
                objectnode.jstart[objectnode.job[objectnode.totalnumberofjobsscheduled]]=
                    objectnode.jfinish[objectnode.job[objectnode.totalnumberofjobsscheduled]];
                objectnode.jfinish[objectnode.job[objectnode.totalnumberofjobsscheduled]]=
                    objectnode.jstart[objectnode.job[objectnode.totalnumberofjobsscheduled]]+
                    objectnode.jprocess[objectnode.job[objectnode.totalnumberofjobsscheduled]];
            }
        }
        /*adds the number of jobs in the block*/
        objectnode.n=objectnode.n+1;
        objectnode.block[objectnode.m][objectnode.n]=jobtoschedule;
        objectnode.numjobs[objectnode.m]=objectnode.n;
    }
}
// have to create another block and insert the new job into it in another case if the job is early

if (objectnode.jdue[job_to_schedule] - objectnode.jprocess[job_to_schedule] >
    objectnode.jfinish[objectnode.job[objectnode.totalnumberofjobsscheduled]])
{
    if(objectnode.m!=1 || objectnode.n >=1)
    {
        objectnode.m=objectnode.m+1;
    }
    objectnode.block[objectnode.m][1]=job_to_schedule;
    objectnode.numjobs[objectnode.m]=1;
    objectnode.n=1;
    objectnode.jstart[job_to_schedule]=objectnode.jdue[job_to_schedule] -
    objectnode.jprocess[job_to_schedule];
    objectnode.jfinish[job_to_schedule]=objectnode.jdue[job_to_schedule];
}
while(true){
    int status=moveBlock(objectnode.m,objectnode.n,objectnode.numjobs,
        objectnode.block,objectnode.jstart,objectnode.
        jfinish,objectnode.jdue,objectnode.jearly,
        objectnode.jtardy,objectnode.blockpenalty);
    if(status==1)/*case where min tardy is ZERO*/
    {
        break;
    }
    if (status==2)
    /*case where the job in the (first) block cannot be moved as it
    starts at ZERO*/
    {
        /*case is true only for the first block in the sequence*/
        break;
    }
    if (status==3)
    /*difference in the two(start and finish) jobs in different jobs is
    ZERO*/
    {
        /*unites the two blocks*/
        uniteBlock(objectnode.m,objectnode.numjobs,
            objectnode.block,objectnode.blockpenalty);
        int temp=objectnode.numjobs[objectnode.m];
        objectnode.numjobs[objectnode.m]=0;
        temp=temp+objectnode.numjobs[objectnode.m-1];
        objectnode.m = objectnode.m-1;
        /*updates the total number of jobs in the block*/
        objectnode.n=objectnode.numjobs[objectnode.m];
    }
    if(status==4)
    /*penalty becomes greater and the current position is the best*/
    {
        break;
    }
}
// have to calculate the total penalty
int temppenalty=0;
for(int i=1;i<=objectnode.m;i++)
{
    temppenalty=temppenalty+objectnode.blockpenalty[i];
}
objectnode.objectpenalty=temppenalty;
public WrapData getObject() throws RemoteException
{
    return new WrapData(this.objectToReturn, this.r_node, this.r_jobscheduled, this.r_min_node);
}

public int moveBlock(int bl, int jb, int numjobblk[], int j_block[][], int j_start[],
    int j_finish[], int j_due[], int early[], int tardy[], int block_penalty[])

int stat=0;
int prevpenalty;
int mintardy;
int newpenalty;
int diff;
/*keep track of jobs in the block*/
int jobblk[] = new int[jb+l];
/* keeps the new start time and finishing time after shifting the block*/
int newstart[] = new int[CONSTANT+1];
int newfinish[] = new int[CONSTANT+1];

/* assigns jobs from the block to a variable*/
for(int i=1;i<=jb;i++)
{
    jobblk[i]=j_block[bl][i];
}
while(true)
{
    prevpenalty=this.calculatePenalty(jobblk, j_finish, j_due, early, tardy);
    /* this finds the min tardy value*/
    mintardy=0;
    for(int i=1;i<=jb;i++)
    {
        if((j_finish[jobblk[i]]-j_due[jobblk[i]]) >=0)
        {
            /*checks if the job is tardy*/
            if(mintardy==0 & (j_finish[jobblk[i]]-j_due[jobblk[i]])>0)
            {
                mintardy=j_finish[jobblk[i]]-j_due[jobblk[i]];
            }
            if((j_finish[jobblk[i]]-j_due[jobblk[i]])< mintardy & mintardy > 0)
            {
                mintardy=j_finish[jobblk[i]]-j_due[jobblk[i]];
            }
        }
    }
    /*case where min tardy is ZERO*/
    if(mintardy==0)
    {
        block_penalty[bl]=prevpenalty;
        stat=1;
        break;
    }
    /*calculate difference in the finishing time of the last job
    in the previous block and start time of the first job in the present block*/
int lastblk = numjobblk[bl-1];
if ((bl-1) == 0)
{
    diff = j_start[jobblk[1]] - 0;
}
else
    diff = j_start[jobblk[1]] - j_finish[j_block[bl-1][lastblk]];
/*case where the block cannot be move any further as it already starts at zero*/
/*true only in the case of first block*/
if (diff <= 0 & (j_start[jobblk[1]] == 0) & bl == 1)
{
    block_penalty[bl] = prevpenalty;
    stat = 2;
    break;
}
/*case were the block touches the previous block*/
if (diff <= 0)
{
    block_penalty[bl] = prevpenalty;
    stat = 3;
    break;
}
if (diff <= mintardy)
{
    mintardy = diff;
}
/*determine the new starting and finishing times according to the min tardy*/
for (int k = l; k <= jb; k++)
{
    newstart[jobblk[k]] = j_start[jobblk[k]] - mintardy;
    newfinish[jobblk[k]] = j_finish[jobblk[k]] - mintardy;
}
/*calculates the penalty with respect to the shifted times*/
newpenalty = this.calculatePenalty(jobblk, newfinish, j_due, early, tardy);
/*block should not be moved and it is at the optimum position*/
if (newpenalty >= prevpenalty)
{
    block_penalty[bl] = prevpenalty;
    stat = 4;
    break;
}
/*update the new times of the block as they are move to the new optimum position*/
if (newpenalty < prevpenalty)
{
    for (int i = l; i <= jb; i++)
    {
        j_start[jobblk[i]] = newstart[jobblk[i]];
        j_finish[jobblk[i]] = newfinish[jobblk[i]];
        block_penalty[bl] = newpenalty;
    }
}
return stat;

/******************************************************************************
Calculate a penalty of given set of jobs
******************************************************************************
public int calculatePenalty(int j[], int f[], int d[], int e[], int t[])
{
    int time;
    int penalty = 0;
    /* remainder */
for(int i=1;i<j.length;i++)
{
    if((time=(f[j[i]]-d[j[i]]))>=0)
        penalty=penalty+time*t[j[i]];
    else
        penalty=penalty-time*e[j[i]];
}
return penalty;

/**
 * Unites two blocks together to create a single new block
 */
public void uniteBlock(int num,int jobsinBlock[],int blocknum[],int blockPenalty[])
{
    int a=jobsinBlock[num];
    int b=jobsinBlock[num-1];
    int newblock[]=new int[a+b+1];
    for(int i=1;i<=b ;i++)
    {
        newblock[i]=blocknum[num-1][i];
    }
    int j=b+1;
    for(int i=1;i<=a ;i++)
    {
        newblock[j]=blocknum[num][i];
        blocknum[num][i]=0;
        j=j+1;
    }
    for(int jobi=1;jobi<=(a + b);jobi++)
    {
        blocknum[num-1][jobi]=newblock[jobi];
    }
    blockPenalty[num-1]=blockPenalty[num-1]+blockPenalty[num];
    blockPenalty[num]=0;
}
Appendix 3: Experimental Results

The following key can be used to identify the data points.

<table>
<thead>
<tr>
<th>Levels</th>
<th># of computers</th>
<th># of jobs</th>
<th>Penalty</th>
<th>ddt</th>
<th>Cv</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>4</td>
<td>7</td>
<td>Random</td>
<td>0.7</td>
<td>0.7</td>
</tr>
<tr>
<td>0</td>
<td>2</td>
<td>6</td>
<td>T=2E</td>
<td>0.5</td>
<td>0.5</td>
</tr>
<tr>
<td>-1</td>
<td>1</td>
<td>5</td>
<td>Equal</td>
<td>0.3</td>
<td>0.3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th># of computers</th>
<th># of jobs</th>
<th>Penalty</th>
<th>ddt</th>
<th>Cv</th>
<th>Time in sec</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>1.742</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>1.301</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>1.142</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>0</td>
<td>1.061</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>0</td>
<td>1.081</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>0</td>
<td>1.141</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>1.251</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>1.242</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>1.281</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>0</td>
<td>1.713</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>0</td>
<td>-1</td>
<td>1.812</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>0</td>
<td>-1</td>
<td>1.773</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>1.112</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>1.302</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>1.342</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>1.102</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>1.112</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>1.001</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>2.153</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>2.263</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>2.073</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>1.883</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>2.664</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>2.624</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>1.832</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>1.922</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>1.592</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>1.302</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>1.327</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>1.081</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>1.061</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>1.091</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>1.152</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>0</td>
<td>-1</td>
<td>1</td>
<td>1.192</td>
</tr>
<tr>
<td>----</td>
<td>----</td>
<td>---</td>
<td>----</td>
<td>---</td>
<td>------</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>0</td>
<td>-1</td>
<td>1</td>
<td>1.212</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>0</td>
<td>-1</td>
<td>1</td>
<td>1.192</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>1.903</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>1.822</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>1.822</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1.472</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1.592</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1.503</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1.212</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1.091</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1.092</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>-1</td>
<td>2.213</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>-1</td>
<td>2.274</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>-1</td>
<td>2.203</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1.973</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2.794</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2.854</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1.963</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2.003</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1.682</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>1.282</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>1.252</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>1.142</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>0</td>
<td>1.091</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>0</td>
<td>1.191</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>0</td>
<td>1.462</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>1.362</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>1.182</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>1.583</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>1.792</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>1.332</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>1.212</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1.532</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1.202</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1.272</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1.102</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1.041</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1.132</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>1.993</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>1.813</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>1.803</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1.582</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>2.163</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>2.043</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>-1</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>-1</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>-1</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>-1</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>0</td>
</tr>
<tr>
<td>-1</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>0</td>
</tr>
<tr>
<td>-1</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>0</td>
</tr>
<tr>
<td>-1</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
</tr>
<tr>
<td>-1</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
</tr>
<tr>
<td>-1</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
</tr>
<tr>
<td>-1</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>-1</td>
<td>1</td>
</tr>
<tr>
<td>-1</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>-1</td>
<td>1</td>
</tr>
<tr>
<td>-1</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>-1</td>
<td>1</td>
</tr>
<tr>
<td>-1</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>-1</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>-1</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>-1</td>
<td>0</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>1</td>
</tr>
<tr>
<td>-1</td>
<td>0</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>1</td>
</tr>
<tr>
<td>-1</td>
<td>0</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>1</td>
</tr>
<tr>
<td>-1</td>
<td>0</td>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>-1</td>
<td>0</td>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>-1</td>
<td>0</td>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>-1</td>
<td>0</td>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>-1</td>
<td>0</td>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>-1</td>
<td>0</td>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>0</td>
</tr>
<tr>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>3.295</td>
</tr>
<tr>
<td>-----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>-------</td>
</tr>
<tr>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>3.686</td>
</tr>
<tr>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>6.63</td>
</tr>
<tr>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>-1</td>
<td>9.203</td>
</tr>
<tr>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>-1</td>
<td>9.003</td>
</tr>
<tr>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>-1</td>
<td>9.984</td>
</tr>
<tr>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>11.076</td>
</tr>
<tr>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>10.696</td>
</tr>
<tr>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>7.241</td>
</tr>
<tr>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>7.04</td>
</tr>
<tr>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>6.419</td>
</tr>
<tr>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>3.626</td>
</tr>
<tr>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>3.525</td>
</tr>
<tr>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>3.846</td>
</tr>
<tr>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>-1</td>
<td>0</td>
<td>5.157</td>
</tr>
<tr>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>-1</td>
<td>0</td>
<td>3.435</td>
</tr>
<tr>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>-1</td>
<td>0</td>
<td>5.656</td>
</tr>
<tr>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>3.605</td>
</tr>
<tr>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>3.886</td>
</tr>
<tr>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>3.065</td>
</tr>
<tr>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>3.685</td>
</tr>
<tr>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>4.797</td>
</tr>
<tr>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>5.198</td>
</tr>
<tr>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>5.167</td>
</tr>
<tr>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>4.567</td>
</tr>
<tr>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>2.684</td>
</tr>
<tr>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>3.084</td>
</tr>
<tr>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>3.596</td>
</tr>
<tr>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>5.428</td>
</tr>
<tr>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>7.03</td>
</tr>
<tr>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>5.448</td>
</tr>
<tr>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>8.562</td>
</tr>
<tr>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>5.869</td>
</tr>
<tr>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>5.438</td>
</tr>
<tr>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>7.821</td>
</tr>
<tr>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>5.127</td>
</tr>
<tr>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>5.317</td>
</tr>
<tr>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2.583</td>
</tr>
<tr>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>5.949</td>
</tr>
<tr>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>6.97</td>
</tr>
<tr>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>5.237</td>
</tr>
<tr>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>7.792</td>
</tr>
<tr>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>9.013</td>
</tr>
<tr>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>7.37</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
</tr>
<tr>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
</tr>
<tr>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
</tr>
<tr>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>0</td>
<td>-1</td>
<td>1</td>
</tr>
<tr>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>0</td>
<td>-1</td>
<td>1</td>
</tr>
<tr>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>0</td>
<td>-1</td>
<td>1</td>
</tr>
<tr>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>1</td>
</tr>
<tr>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>1</td>
</tr>
<tr>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
</tr>
<tr>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
</tr>
<tr>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
</tr>
<tr>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>1</td>
</tr>
<tr>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>1</td>
</tr>
<tr>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>1</td>
</tr>
<tr>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>-1</td>
<td>1</td>
</tr>
<tr>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>-1</td>
<td>1</td>
</tr>
<tr>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>-1</td>
<td>1</td>
</tr>
<tr>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>Column 1</td>
<td>Column 2</td>
<td>Column 3</td>
<td>Column 4</td>
<td>Column 5</td>
<td>Column 6</td>
</tr>
<tr>
<td>---------</td>
<td>---------</td>
<td>---------</td>
<td>---------</td>
<td>---------</td>
<td>---------</td>
</tr>
<tr>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>29.172</td>
</tr>
<tr>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>53.066</td>
</tr>
<tr>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>59.986</td>
</tr>
<tr>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>12.097</td>
</tr>
<tr>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>8.523</td>
</tr>
<tr>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>5.328</td>
</tr>
<tr>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>0</td>
<td>6.619</td>
</tr>
<tr>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>0</td>
<td>7.371</td>
</tr>
<tr>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>7.2</td>
</tr>
<tr>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>5.779</td>
</tr>
<tr>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>9.012</td>
</tr>
<tr>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>16.784</td>
</tr>
<tr>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>17.064</td>
</tr>
<tr>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>18.867</td>
</tr>
<tr>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>14.631</td>
</tr>
<tr>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>16.294</td>
</tr>
<tr>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>15.792</td>
</tr>
<tr>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>6.049</td>
</tr>
<tr>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>7.661</td>
</tr>
<tr>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>14.521</td>
</tr>
<tr>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>48.51</td>
</tr>
<tr>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>26.709</td>
</tr>
<tr>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>59.565</td>
</tr>
<tr>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>26.939</td>
</tr>
<tr>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>38.646</td>
</tr>
<tr>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>27.069</td>
</tr>
<tr>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>27.549</td>
</tr>
<tr>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>9.774</td>
</tr>
<tr>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>28.491</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>1.252</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>1.081</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>0.921</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>0</td>
<td>0.841</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>0</td>
<td>0.891</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>0</td>
<td>0.932</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>1.051</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>1.042</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>1.072</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>0</td>
<td>1.443</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>0</td>
<td>1.492</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>0</td>
<td>1.442</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>1.091</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>1.172</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>0.941</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>0.931</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>0.841</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>1.742</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>1.833</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>1.703</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>1.562</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>2.053</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>2.043</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>1.512</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>1.602</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>1.352</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>1.101</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>1.072</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>0.941</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>0.911</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>0.932</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>0.992</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>-1</td>
<td>1</td>
<td>1.021</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>-1</td>
<td>1</td>
<td>1.021</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>-1</td>
<td>1</td>
<td>1.001</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>1.522</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>1.522</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>1.513</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1.212</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1.292</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1.252</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1.032</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0.91</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0.992</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>-1</td>
<td>1.833</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>-1</td>
<td>1.912</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>-1</td>
<td>1.823</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1.882</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2.153</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2.083</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1.623</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1.692</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1.402</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>0.991</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>1.041</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>1.022</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>0</td>
<td>1.021</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>0</td>
<td>0.942</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>0</td>
<td>1.111</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>1.092</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>1.081</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>1.052</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>1.452</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>1.642</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1.101</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1.001</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1.252</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0.962</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0.932</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0.901</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>1.632</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>1.692</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>1.622</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1.682</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1.703</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1.732</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1.583</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1.752</td>
</tr>
<tr>
<td>0</td>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1.342</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>3.886</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>1.693</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>2.424</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>0</td>
<td>2.013</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>0</td>
<td>1.923</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>0</td>
<td>2.784</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>2.233</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>3.355</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>2.554</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>-1</td>
<td>4.446</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>-1</td>
<td>4.907</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>-1</td>
<td>4.606</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>3.495</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>3.545</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>4.176</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>2.164</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>2.544</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>4.747</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>6.87</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>7.1</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>7.851</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>9.153</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>9.904</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>8.552</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>3.115</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>3.274</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>4.376</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>5.087</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>5.018</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>5.097</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>4.386</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>5.918</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>6.159</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>4.316</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>4.607</td>
</tr>
<tr>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2.403</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>5.498</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>5.938</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>4.606</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>14.691</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>16.453</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>27.049</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>8.362</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>11.907</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>17.935</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>4.467</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>5.528</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>12.888</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>51.424</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>31.115</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>45.756</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>33.759</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>51.944</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>29.663</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>19.157</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>16.053</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>34.78</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>6.028</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>5.518</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>4.727</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>5.848</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>7.531</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>5.217</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>-1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>-1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>-1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>-1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>-1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>-1</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
</tr>
<tr>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
</tr>
<tr>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
</tr>
<tr>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>-1</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>1</td>
<td>-1</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>1</td>
<td>-1</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
</tr>
<tr>
<td>1</td>
<td>-1</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>-1</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>-1</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>-1</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>-1</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>-1</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>-1</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>-1</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>-1</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
</tr>
<tr>
<td>1</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td></td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>-----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>----</td>
<td>-----</td>
</tr>
<tr>
<td>1</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1.002</td>
</tr>
<tr>
<td>1</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0.881</td>
</tr>
<tr>
<td>1</td>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>-1</td>
<td>0.891</td>
</tr>
<tr>
<td>1</td>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>-1</td>
<td>2.103</td>
</tr>
<tr>
<td>1</td>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>-1</td>
<td>2.133</td>
</tr>
<tr>
<td>1</td>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>-1</td>
<td>2.043</td>
</tr>
<tr>
<td>1</td>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2.193</td>
</tr>
<tr>
<td>1</td>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>2.043</td>
</tr>
<tr>
<td>1</td>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2.123</td>
</tr>
<tr>
<td>1</td>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2.113</td>
</tr>
<tr>
<td>1</td>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>2.294</td>
</tr>
<tr>
<td>1</td>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1.472</td>
</tr>
<tr>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>1.061</td>
</tr>
<tr>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>1.092</td>
</tr>
<tr>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>1.041</td>
</tr>
<tr>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>0</td>
<td>0.971</td>
</tr>
<tr>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>0</td>
<td>0.922</td>
</tr>
<tr>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>0</td>
<td>0.851</td>
</tr>
<tr>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>1.091</td>
</tr>
<tr>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>1.082</td>
</tr>
<tr>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>1.052</td>
</tr>
<tr>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>1.282</td>
</tr>
<tr>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>1.352</td>
</tr>
<tr>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>1.883</td>
</tr>
<tr>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>1.052</td>
</tr>
<tr>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1.062</td>
</tr>
<tr>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1.142</td>
</tr>
<tr>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1.052</td>
</tr>
<tr>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1.051</td>
</tr>
<tr>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0.902</td>
</tr>
<tr>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>1.943</td>
</tr>
<tr>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>2.333</td>
</tr>
<tr>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>1.712</td>
</tr>
<tr>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1.592</td>
</tr>
<tr>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>2.123</td>
</tr>
<tr>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>2.063</td>
</tr>
<tr>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1.512</td>
</tr>
<tr>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1.362</td>
</tr>
<tr>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1.192</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>3.274</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>1.913</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>3.345</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>0</td>
<td>2.253</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>0</td>
<td>2.264</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>0</td>
<td>2.885</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>2.654</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>3.365</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>2.714</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>-1</td>
<td>4.506</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>-1</td>
<td>4.997</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>-1</td>
<td>4.716</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>3.235</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>3.234</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>4.066</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>2.634</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>2.934</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>4.677</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>7.571</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>7.491</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>9.824</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>9.814</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>9.814</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>9.644</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>5.688</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>5.808</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>5.217</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>3.545</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>2.273</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>3.404</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>1.913</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>2.013</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>2.824</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>1</td>
<td>2.744</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>1</td>
<td>3.374</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>1</td>
<td>2.473</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>5.788</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>6.109</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>6.018</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3.445</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>3.656</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>5.158</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2.994</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>3.114</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>5.918</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>-1</td>
<td>9.143</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>-1</td>
<td>9.384</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>-1</td>
<td>8.963</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>9.924</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>9.754</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>9.153</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>6.69</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>6.44</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>5.738</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>3.025</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>3.706</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>2.704</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>-1</td>
<td>0</td>
<td>2.925</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>-1</td>
<td>0</td>
<td>2.964</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>-1</td>
<td>0</td>
<td>3.155</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>2.944</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>3.224</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>2.704</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>3.005</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>2.347</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>2.834</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>3.786</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>3.135</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>3.045</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>2.85</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>3.345</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>5.488</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>6.54</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>5.678</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>6.149</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>5.709</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>5.428</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>6.269</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>3.675</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>5.198</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>3.044</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>5.388</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>6.309</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>-1</td>
<td>4.516</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>0</td>
<td>7.22</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>0</td>
<td>8.122</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>0</td>
<td>8.122</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>0</td>
<td>6.83</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>6.709</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>6.069</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>1</td>
<td>10.215</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>0</td>
<td>-1</td>
<td>15.313</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>0</td>
<td>-1</td>
<td>17.385</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>0</td>
<td>-1</td>
<td>27.34</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>9.754</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>12.288</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>0</td>
<td>0</td>
<td>18.978</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>4.787</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>5.929</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>0</td>
<td>1</td>
<td>13.81</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>55.029</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>-1</td>
<td>34.329</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>35.601</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>0</td>
<td>56.151</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>25.487</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>17.045</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>1</td>
<td>39.617</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>6.98</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>5.939</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>-1</td>
<td>4.716</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>6.209</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>8.482</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>0</td>
<td>5.488</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>1</td>
<td>6.299</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>1</td>
<td>5.027</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>1</td>
<td>7.341</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>26.117</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>27.83</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>-1</td>
<td>38.916</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>14.05</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>18.356</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>28.872</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>5.118</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>5.047</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>17.125</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>-1</td>
<td>72.554</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>-1</td>
<td>42.441</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>-1</td>
<td>59.676</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>41.159</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>70.211</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>40.378</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>27.55</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>25.277</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>43.372</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>7.171</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>7.707</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>-1</td>
<td>5.696</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>0</td>
<td>8.212</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>0</td>
<td>11.697</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>0</td>
<td>7.711</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
<td>---</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>7.44</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>5.668</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>1</td>
<td>8.051</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>15.482</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>14.12</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>-1</td>
<td>25.377</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>11.767</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>12.288</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>13.579</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>4.326</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>5.017</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>8.642</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>53.407</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>29.092</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>-1</td>
<td>42.652</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>28.701</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>32.096</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>40.168</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>25.197</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>15.162</td>
</tr>
<tr>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>17.385</td>
</tr>
</tbody>
</table>